Numéro d'ordre: **289**
Numéro attribué par la bibliothèque: **04ENSL0 289**

THÈSE

en vue d'obtenir le grade de
Docteur de l'École Normale Supérieure de Lyon.
spécialité informatique – Laboratoire de l'Informatique et du Parallélisme
École doctorale de Mathématique et Informatique Fondamentale
présentée et soutenue publiquement le 8 novembre 2004 par

Étienne Lozes

Expressivité des Logiques d'Espaces

Jury

Pierre-Louis Curien rapporteur
Stephan Merz rapporteur
Pierre Lescanne directeur (École Normale Supérieure de Lyon)
Luís Monteiro directeur (Universidade Nova de Lisboa)
Davide Sangiorgi examinateur
Luca Cardelli examinateur
Daniel Hirschkoff encadrant
Luís Caires encadrant

Table des matières

Remerciements

Cette thèse a été effectuée en cotutelle entre l'École Normale Supérieure de Lyon et l'Universidad Nova de Lisboa. Je tiens tout d'abord à remercier ces deux administrations respectives pour avoir approuvé cette collaboration, et je remercie le *projet européen PROFUNDIS* et la *région Rhône-Alpes* (bourses Eurodoc) pour avoir financé les déplacements occasionnés.

Durant cette thèse, j'ai bénéficié de l'aide et du soutien de nombreux chercheurs. *Pierre Lescanne* a accepté de diriger cette thèse sur Lyon et m'a fait bénéficier de sa longue expérience sur le λ-calcul au cours de riches discussions. *Luis Monteiro* était le directeur de thèse lisboète ; il a suivi activement mes travaux et m'a fait part de nombreuses références intéressantes, en particulier sur la notion de précompacité.

Je souhaite remercier par ailleurs *Stephan Merz* et *Pierre-Louis Curien* pour avoir accepté de rapporter le présent manuscrit. Leurs remarques appropriées ont permis à ce manuscrit de trouver une forme notablement mieux aboutie.

Je remercie enfin *Luca Cardelli*, initiateur de nombreux concepts étudiés dans cette thèse, *Peter O'Hearn*, *Cristiano Calcagno*, et surtout *Hongseok Yang* pour leurs travaux remarquables et les discussions dont ils m'ont fait bénéficier, *Jean-Marc Talbot* pour sa visite à Lyon et les discussions sur le calcul des Ambients, et *Giorgio Ghelli* pour des discussions sur la logique des Ambients statiques.

Je tiens plus particulièrement à remercier trois personnes. *Davide Sangiorgi* a œuvré de façon décisive pour cette thèse. Les grandes orientations de la thèse lui son dûes, et grâce à nos discussions, au suivi constant de l'avancement de mon travail, et surtout son enthousiasme, j'ai pu mener à bien les projets que nous élaborions. *Luis Caires* a encadré ma thèse a Lisbonne. J'ai vécu avec lui les espoirs et les doutes suscités par les logiques spatiales, il a activement guidé les développement les plus récents de ce travail, et a surtout, par sa gentillesse, fait de mon séjour à Lisbonne une période riche et créative. Enfin c'est surtout à *Daniel Hirschkoff*, que je dois le plus pour avoir accepté la difficile tâche de m'encadrer à Lyon, et d'y avoir mis pour cela les plus belles qualités.

Spécial Dédicace

au principinho pour son coup de marteau, au grande del Chile pour son maté, à la blik maisonnée pour ses coups de rosé, à Satchmo pour la stéréo, à Fanfan pour les coups de vents, à Nourdine pour avoir usé ma sourdine, à ma super zainana pour ce qui regarde que moi, à David pour sa sérénité dans mon bolide, à Maria-João para um poema, ao Ugo para Caetano Veloso, à Claire pour ses cerises et aux enfants pour les dessins, à Romain pour le tango argentin, à Benjamin et Pierre-Yves pour les balades en scooter, à Pierric pour ne pas m'avoir démis une épaule avant la rédaction, à "Braco" pour son sax et ses 2 Hertz, à Nathalie pour un voyage au soleil couchant et à Samia pour un autre au soleil levant, à Pomodoro pour ses bons plans flunch, à Linda pour mes slips secs, à Céline pour ses maillots de bain humides, à Iris pour ma valise en Inde, à Stéphane pour nos talents de déménageurs, à Karin pour un carnet de voyages, à Mette pour un rêve de flémingite, à Elia pour un petit rien de rose, à Guillaume pour m'avoir appris à mettre sans risque la tête à l'envers, à Thierry pour ses soirées punky chez Yop, à Taos-Hélène pour une escale à Port-Cros, à Jérémie pour un théorème de cuisine, à M. Gostiaux pour de biens beaux livres, à Mme Gostiaux pour de bien bonnes confitures, à Bertrand pour sa pipe de pépé, ...

Spécial dédicace à Tonio pour les photos, à Françoise pour les pots de graisse d'oie, à André pour ses livres et ses dessins, à Jean pour la technologie numérique et à Denise pour les recettes de champignons.

Introduction

Je te vends une mobylette,
en plus elle est très chouette.
Je te vends un gros bidet,
en plus il est pas laid.
Chanson de collégien.

Des systèmes mobiles

Par *système mobile*, on entend un ensemble d'agents qui évoluent dans un milieu selon des règles primitives simples, avec la spécificité de modifier la structure de ce milieu. On peut par exemple songer à des réseaux de communication reconfigurables (informatiques, neuroniques, sociologiques,...), où c'est le tissu des échanges entre agents qui évolue, ou encore à un domaine de sites perméables au mouvement d'autres sites (mobilité de code et d'unités de calculs, biologie macrocellulaire,...) pour lesquels c'est la hiérarchie des agents qui évolue. Sous ces deux exemples se retrouvent deux familles d'algèbres de processus, celle du π-calcul pour les réseaux reconfigurables, et celle des ambients pour les sites en mouvement.

L'objet des algèbres de processus est de fournir une représentation de tels systèmes dans un formalisme algébrique abstrait directement dédié aux systèmes mobiles, et cela dans diverses perspectives. La première est héritée du λ-calcul, et considère que ces formalismes sont des bases saines pour élaborer des paradigmes de programmation. Ces idées sont par ailleurs justifiées par les nets progrès des langages de programmation dans la sécurisation du code, par le biais de diverses méthodes automatisées, notamment la théorie des types, et les langages de spécification formels. L'autre perspective est celle d'un cadre général pour l'étude de ces systèmes, qui échappent à la plupart des modèles antérieurs de système distribués dans lesquels on est contraint de faire l'hypothèse d'une topologie de l'espace des agents non évolutive [1].

Extensionalité et intensionalité

Deux niveaux de description sont nécessaires pour la vérification de systèmes : le niveau intensionnel, qui est celui de la mécanique interne du système, et le niveau extensionnel, qui en donne la fonctionalité, l'effet produit sur un envi-

[1]On peut citer, entre autres, les automates cellulaires, les réseaux de Petri, ou CCS.

ronnement donné. Ces deux niveaux de description se complètent dans l'analyse d'un système.

Les algèbres de processus permettent de décrire les systèmes mobiles à ces deux niveaux uniquement par des processus algébriques et la notion de bisimulation, mais cette approche induit des descriptions extensionnelles complexes. Les logiques modales fournissent en revanche des spécifications extensionnelles partielles et sont plus commodes à manipuler à ce niveau descriptif.

Les logiques spatiales au début de la thèse

A la fin des années 90 apparaissent de nouvelles logiques pour la spécification de processus algébriques : les logiques spatiales. Pour ces logiques, l'extensionalité n'est pas primordiale, il s'agit avant tout de pouvoir énoncer simplement les reconfigurations spatiales internes à un système mobile, et non plus de le regarder du point de vue des possibilités d'interactions qu'il offre avec l'extérieur.

Pour cela, elles introduisent de nouvelles formes de connecteurs logiques qui prennent en compte directement la compositionnalité de l'espace et y assoient de nouveaux principes de raisonnement. Ces connecteurs sont inspirés de multiples travaux récents, portant sur la *bunched logic* [POY04], le π-calcul [Cai99], ou encore la quantification générique [GP99]. Les logiques spatiales pour les algèbres de processus sont dans leur présentation relativement exploratoires, au sens ou de multiples connecteurs sont proposés de façon systématisée, sans souci de minimalité.

La bunched logic est essentiellement aux origines de la *logique séparante*. Cette logique spatiale n'est pas un langage d'assertion pour les algèbres de processus, mais sert de fondement à une sémantique dénotationnelle des langages impératifs manipulant des pointeurs. Pour la structure spatiale considérée par cette logique, l'opération de composition n'est donc pas la mise en parallèle de processus, mais l'assemblage sans recouvrement de portions de l'espace mémoire.

Au début de cette thèse, les logiques spatiales commençaient tout juste à être étudiées et la spécificité et l'intérêt de leur approche restaient et reste encore à comprendre.

Pourquoi l'expressivité

Lorsqu'apparait une nouvelle logique, on doit trouver des moyens de l'évaluer. Le thème fédérateur choisi dans cette thèse pour l'évaluation des logiques spatiales est celui de l'expressivité. Les logiques spatiales sont avant tout des langages d'assertions, basés sur la notion de satisfaction d'une propriété logique $\mathcal{A}$ dans un processus P, notée $P \models \mathcal{A}$. Dans ce cadre, de nombreux problèmes d'expressivité peuvent être considérés :

- quel est le *pouvoir distinctif* des logiques spatiales ? quelle granularité d'observation la logique induit-elle sur les processus ?
- quel est l'*enrichissement expressif* présenté par ces logiques par rapport à celles préexistantes (logiques classiques et modales) ? comment se comparent les classes de propriétés qu'elles permettent d'énoncer, leur *pouvoir expressif* ?

– quels connecteurs sont essentiels ? les logiques spatiales admettent-elles des redondances ou sont-elles *minimales* ?
– jusqu'à quel point leur richesse d'expressivité n'entrave pas leur automatisation ? en particulier, existe-t-il des procédures de décision pour les problèmes de satisfaction et de validité ? dans le cas négatif, quelles sont les causes de l'*indécidabilité* de ces problèmes ? quelles restrictions permettent de mécaniser la vérification de systèmes à l'aide de ces logiques ?
– peut-on structurer les preuves de propriétés énoncées dans ces logiques ? admettent-elles des *systèmes d'inférence* et sont-ils complets ?

Toutes ces questions sont importantes lors de l'introduction d'une nouvelle logique. D'elles dépendent l'utilisation pratique de la logique. Par exemple, le pouvoir distinctif et la structure de preuve sont fondamentaux avant toute spécification : le pouvoir distinctif guide le spécificateur dans le niveau de précision qu'il doit atteindre quand il modélise son système par un processus ; le système d'inférence est quant à lui le garde fou théorique qui garantit la justesse de son raisonnement lorsqu'il établit la preuve de son système.

Enfin, l'expressivité est énoncée comme un thème fédérateur pour toutes ces questions, et ce essentiellement parce qu'elles sont intimement liées. Par exemple, une trop grande richesse dans le pouvoir expressif peut rendre difficile certains problèmes de décision. La réduction à des logiques minimales présente en revanche un moyen de simplification des problèmes de vérification.

Contributions de cette thèse

Cette thèse répond dans une certaine mesure à chacune des questions posées précédemment. Pour ce faire, des propriétés connues pour les logiques standard (mise sous forme prénexe, élimination des quantificateurs, formules caractéristiques, double négation, ...), sont reformulées dans le cadre des logiques spatiales (chapitres 4 et 8). Les implications de ces propriétés sont par ailleurs le plus souvent relativement différentes de celles auxquelles on s'attendrait : par exemple, dans le cas de la logique classique, l'élimination constructive des quantificateurs induit la décidabilité de la logique, avec éventuellement d'autres conséquences sur la complexité des problèmes de décision associés, alors que rien de semblable n'a lieu avec les logiques spatiales.

De nombreux résultats en théorie de la mobilité, en particulier du π-calcul, ont en revanche guidé ces travaux et ont fourni des éclaircissements et des éléments de justification à certains résultats. Ainsi, par exemple, les travaux portant sur la caractérisation de la bisimilarité par la congruence à barbe offrent une interprétation a posteriori de l'expressibilité des logiques modales dans les logiques spatiales (chapitre 6), alors que les techniques de preuves nécessaires pour établir ces deux résultats se recoupent très superficiellement. Le calcul des Ambients quant à lui connaissait une théorie assez récente au début de la thèse, et même si certaines idées antérieures ont pu être adaptées à ce calcul, des techniques complètement nouvelles et qui lui sont propres ont été développées sur le calcul des Ambients, ce qui a par ailleurs permis d'en illustrer la spécificité par rapport au π-calcul (chapitre 7).

Les techniques et les résultats mis en oeuvre dans cette thèse présentent donc des antécédents à la fois dans la théorie standard de la logique et dans la théorie de la concurrence, mais restent sous certains aspects tout à fait originaux, et on ne saurait dire jusqu'à quel point, sans doute assez spécifiques aux logiques spatiales. De nombreux liens sont établis entre les divers problèmes d'expressivité mentionnés précédemment. Un concept fort unit particulièrement les problèmes de la caractérisation du pouvoir distinctif et de la minimalité de la logique, c'est celui de bisimilarité intensionnelle (chapitre 7).

Concernant le pouvoir distinctif des logiques spatiales, on établit une très fine granularité d'observation pour ces logiques. En effet, alors que pour les logiques modales deux processus logiquement équivalents sont extensionnellement équivalents, la logique spatiale est plus distinctive et les processus logiquement équivalents sont intensionnellement équivalents.

L'apport expressif des logiques spatiales est illustré par la comparaison de leur pouvoir expressif et de celui des logiques qu'elles étendent ou dont elles sont des variations. Pour la logique séparante, on donne à la fois un résultat d'équivalence expressive avec la logique classique dans un cas restreint et un contre-exemple à la généralisation de ce résultat. Pour la logique spatiale du π-calcul, on établit une expressivité strictement plus forte que pour la logique de Hennessy-Milner.

La question de la minimalité dans les logiques spatiales est abordée de diverses façons. D'une part il est proposé une logique spatiale unifiée pour les algèbres de processus, obtenue comme un fragment de logiques spatiales existantes en supprimant les connecteurs spécifiques à l'algèbre de processus considérée. On montre alors que la logique unifiée exprime complètement la logique spatiale du π-calcul et dans une certaine mesure celle des Ambients. D'autre part, on établit un procédé de minimalisation par élimination des adjoints pour une logique spatiale statique, et on montre un contre-exemple à sa généralisation à une logique plus riche. Enfin, on établit un autre procédé de minimalisation par élimination des quantificateurs, qui s'adapte à toutes les logiques spatiales pour la concurrence. On tire enfin de ces résultats d'expressivité certaines conséquences sur la décidabilité de quelques problèmes de décision relatifs à ces logiques.

Comment lire ce manuscrit

Ce manuscrit reprend essentiellement les résultats publiés au cours de cette thèse [HLS02, HLS03, Loz03, Loz04b, CL04, Loz04a], en les réorganisant pour en faire apparaître les grandes idées directives. Tous les chapitres sont donc intimenent liés.

La première partie introduit les objets fondamentaux étudiés par la suite. Le chapitre 1 introduit brièvement quelques algèbres de processus (CCS, π-calcul, Ambients) dans une perspective historique. Au chapitre 2, on étudie la notion de comportement, intimement liée aux notions de *bisimilarité* et de logique modale. Le chapitre 3 introduit les logiques spatiales, d'une part pour les algèbres de processus vues précédemment, d'autres part pour des structures statiques (tas et pile, documents XML). Enfin le chapitre 4 donne une illustration de la spécificité des logiques spatiales sur certaines propriétés structurelles comme la dualité, la mise sous forme prénexe, ou les règles d'inférence.

La deuxième partie présente les contributions originales de cette thèse. Elles peuvent être lues de façon plus ou moins indépendante :

– le chapitre 5 est autonome, même s'il donne une introduction et un éclairage sur la démarche des chapitres suivants. On y étudie quelques problèmes d'expressivité et de minimalité de la logique séparante, dont en particulier un résultat d'équivalence expressive avec la logique classique ;
– le chapitre 6 illustre les *jeux contextuels* dérivables dans les logiques spatiales, qui sont immédiatement utilisés au chapitre 7 pour établir l'intensionalité des logiques spatiales, en particulier sur le cas du calcul des Ambients ;
– le chapitre 8 illustre des techniques de minimalisation de logiques spatiales, pour des structures statiques puis pour des structures dynamiques.
– le chapitre 9 reprend les résultats de tous les chapitres précédents et les examine du point de vue de la décidabilité des problèmes qu'ils soulèvent.

Les annexes reprennent les preuves esquissées dans le corps de la thèse avec davantage de détails. Elles sont tirées de publications internationales et sont donc rédigées en anglais. Outre des détails omis dans le corps de la thèse, elles présentent parfois des résultats légèrement plus forts que ceux énoncés dans la partie en français.

L'annexe A reprend ainsi les jeux contextuels du chapitre 6 dans le cadre plus général du calcul des Ambients avec communication. L'annexe B reprend dans le détail la preuve de l'intensionalité des logiques spatiales aperçues au chapitre 7, ainsi que la définition de machines de Turing en Ambients présentée au chapitre 9. L'annexe C reprend la première partie du chapitre 8 sur l'élimination des adjoints ainsi que l'ensemble du chapitre 5. L'annexe D reprend enfin la seconde partie du chapitre 8 sur l'élimination des quantificateurs ainsi que la preuve de l'indécidabilité des logiques spatiales pour la concurrence évoquée au chapitre 9.

Introdução

Esta tese estuda certos formalismos introduzidos recentemente para a verificação de propriedades de processos distribuídos móveis. Com a introdução de novos modelos de computação móvel, tornou-se mais clara a necessidade de recorrer a linguagens de especificação incluindo *observações espaciais*, estes novos formalismos lógicos designam-se correntemente por "lógicas espaciais".

Estas lógicas podem ser encaradas como surgindo no seguimento do desenvolvimento de formalismos bem conhecidos para a verificação de sistemas concorrentes (as lógicas modais) e para a caracterização de propriedades de espaços de recursos (a "bunched logic"), mas também possuem características muito particulares, relacionadas com os seus objectivos próprios. São muito recentes, sendo que os primeiros trabalhos sobre lógicas espaciais datam do fim dos anos 90, pelo que eram pouco conhecidas na altura em que os trabalhos que conduziram a esta tese se iniciaram.

Quando uma nova lógica é introduzida, é necessário avaliar em que medida responde às motivações que conduziram à sua definição. A orientação seguida nesta tese é pois estudar as lógicas espaciais do ponto de vista da sua expressividade. Muitos problemas interessantes pertencem a esta temática :

- O poder expressivo das lógicas (*que propriedades podem o não ser enunciadas ?*)
- O poder de separação das lógicas (*até que ponto a lógica pode distinguir entre elementos diferentes do modelo ?*)
- A minimalidade (*Que operações da lógica são realmente necessárias para exprimir todas as propriedades enunciáveis ou distinguir todos os processos que se podem distinguir ?*)
- Qual é a complexidade das formulas lógicas, em particular, que problemas associados são computáveis : *Existem algoritmos para decidir se um certo sistema satisfaz certa especificação ? Em caso negativo, quais são as causas essenciais da indecidabilidade ? E quais são os fragmentos para os quais existem procedimentos de decisão ?* Este é o chamado problema da verificação duma formula relativamente a um sistema, a ele associado considera-se também o problema da valididade : *dada um certa formula, é esta satisfeita por todos os sistemas, ou existe um sistema que a não realiza ?*
- Finalmente, note-se que a expressividade dos vários modelos de computação móvel não era ainda completamente conhecida no início deste trabalho, pelo que se podem colocar ainda questões a este nível sobre o poder computacional dos vários modelos.

Para estudar estas questões, que são de natureza essencialmente semântica, são usadas não só técnicas bastante "standard" usadas em lógica, mas também uma parte substancial da teoria da concorrência, assim como técnicas sem dúvida bastante específicas das lógicas espaciais.

Esta tese fornece respostas às questões acima mencionadas em múltiplos contextos.

A primeira lógica considerada é a chamada lógica da separação, que se compara, em termos de expressividade, à lógica clássica. Mais concretamente, demonstra-se que, em certo sentido, estas lógicas são igualmente expressivas, o que mostra que em alguns casos a lógica da separação não é minimal.

De seguida, considera-se a utilização de especificações espaciais como meio de exprimir observações comportamentais dos processos. O modo segundo o qual aquelas observações são derivadas é designado então por "jogos contextuais", dada a observação do comportamento de processo ser realisada através da introdução de um outro processo no contexto. Estes jogos contextuais são completamente gerais, e são ilustrados em vários modelos de cálculos.

Após definir tais observações, torna-se possível estudar o poder de separação das lógicas espaciais quando interpretada sobre as álgebras de processos. Mostra-se então que estas lógicas são intensionais, no sentido em que atingem um nível de descrição dos sistemas muito mais fino que as lógicas modais clássicas.

De seguida, estuda-se até que ponto é necessário todo o formalismo tal como definido para se obter uma tal expressividade. Trata-se pois de considerar o problema já mencionado de minimalidade das lógicas. Dois resultados de minimalidade são estão derivados, o primeiro para uma lógica espacial estática interpretada sobre modelos que representam documentos em formato XML, e um outro que se aplica às lógicas espaciais dinâmicas em geral.

Embora todos os resultados mencionados terem sido inicialmente motivados por questões de expressividade, alguns têm consequências importantes sobre certos problemas de decisão, os quais são considerados de forma integrada no fim da tese.

Como ler esta dissertação

A parte principal da tese está escrita em francês, e apresenta de forma bastante didática os vários resultados. Tais resultados foram em grande parte publicados em revistas internacionais em inglês [HLS02, HLS03, Loz03, Loz04b, CL04, Loz04a], sendo que uma adaptação de tais artigos se apresenta nos anexos da tese, também em inglês. Essa parte é complementar do corpo principal escrito em francês, porque nele se discutem as demonstrações em detalhe, e se precisam alguns resultados que foram apenas mencionados de forma breve anteriormente.

Os primeiros capítulos introduzem as álgebras de processos (capítulo 1) e as lógicas modais para a concorrência (capítulo 2). Os capítulos seguintes introduzem várias lógicas espaciais (capítulo 3) assim como algumas propriedades estruturais das mesmas (capítulo 4).

Os capítulos seguintes são sempre contribuições originais da tese. O capítulo 5 é uma introdução aos diversos problemas de expressividade que se colocam no caso específico da lógica da separação. O capítulo 6 define "jogos contextuais" atrás referidos, e estabelece uma comparação entre as lógicas espaciais e as lógicas modais. O capítulo 7 desenvolve uma caracterisação da intensionalidade das lógicas espaciais, concentrando-se sobre o caso da lógica dos ambientes com a modalidade temporal fraca, que é o caso mais geral e mais difícil. O capítulo 8 desenvolve dois métodos de minimização de lógicas espaciais no caso estático e no caso dinâmico respectivamente. O capítulo 9 apresenta algumas consequências dos resultados de expressividade em alguns problemas de decisão.

O anexo A apresenta jogos contextuais apropriados para o caso dos ambientes. O anexo B apresenta a caracterização da intensionalidade da lógica dos ambientes com vários dos detalhes omitidos no capítulo 7, assim como um resultado de indecidabilidade para o cálculo dos ambientes (capítulo 9). O anexo C apresenta a técnica de minimização para as lógicas espaciais estáticas (capítulos 5 e 8), e o anexo D apresenta outra técnica de minimização para lógicas dinâmicas (capítulo 8), assim como um resultado de indecidabilidade para as lógicas espaciais para a concorrência (capítulo 9).

Première partie

Chapitre 1

Algèbres de processus

Les mots *mobilité* et *concurrence*, galvaudés par les économistes, par les politiciens, même par certains informaticiens, recouvrent des sujets et des analyses multiples, qui pour la plupart ont façonné les derniers progrès de notre société contemporaine. Pour ce qui va suivre, on se méfiera toutefois : ces mots prennent dans cette thèse une signification très particulière. D'une part ils s'appliquent au domaine informatique que l'on va préciser par la suite, et d'autre part ils sont issus pour l'informatique de leur traduction littérale de l'anglais, et ont sans doute par cette opération perdu une part de leur signification originale.

Le mot mobilité a par ailleurs pris une ampleur inattendue avec la modélisation de communications par téléphone portable dans le π-calcul[OP92]. Mais par systèmes mobiles et concurrents, on entend représenter essentiellement des systèmes multi-agents distribués, qui communiquent, collaborent ou protègent leurs ressources individuelles selon les cas considérés. Les algèbres de processus restent des modèles très abstraits et très généraux, et touchent donc des problèmes présents à des niveaux scientifiques et d'ingénierie aussi divers que variés, et il serait sans doute dommage de vouloir restreindre leur champ d'application à des problèmes au goût du jour.

Une représentation algébrique de tels systèmes est essentielle. D'une part, elle fournit un paradigme simple et précis pour la programmation, la spécification, ou la schématisation de systèmes distribués qui sont source d'inspiration pour de nombreux informaticiens. D'autre part, elle permet d'élaborer une théorie mathématique solide et des outils informatiques exacts pour vérifier des propriétés de sécurité dans ces systèmes. Enfin, la recherche de fondements algébriques à des phénomènes le plus souvent partiellement expliqués au niveau informel enrichit le niveau descriptif informel lui-même, en ce qu'elle fait apparaître de nouveaux concepts (asynchronisme, choix multiples, localité...) qui aident le non-algébriste à préciser ses idées.

Dans ce chapitre, on fait un bref historique des diverses algèbres de processus qui ont été proposées ces vingt cinq dernières années. On présente tout d'abord le calcul CCS, introduit par Milner [Mil89], puis le π-calcul qui rajoute la notion de communication à celle de synchronisation présente en CCS, enfin la notion de localité qui rend explicite la notion d'agent, telle qu'elle a été définie dans le calcul des Mobile Ambients.

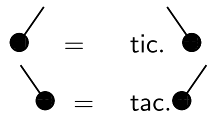
1.1 CCS ou la synchronisation

1.1.1 Les automates

Un automate est une entité active individuelle, un élément atomique d'un système évolutif. Cette notion se confond assez bien avec celle d'agent dans un système distribué. Un automate est reconfigurable, peut prendre divers états selon son interaction avec l'environnement ou son évolution interne.

Les automates sont essentiellement présents dans la culture scientifique informatique pour leur intérêt dans la caractérisation de classes de langages : rationnels, algébriques, décidables. Dans ce cadre l'environnement sur lequel agit l'automate est un mot. Dans les systèmes distribués, les automates sont dans un environnement d'automates et les notions de mots et de langage reconnu perdent leur importance. La représentation des automates est aussi assez différente. En général, un automate est représenté par un graphe avec des arêtes orientées étiquetées par des mots. Dans les algèbres de processus, on donne une représentation syntaxique de l'automate : un automate est la donnée d'un ensemble d'états et pour chacun d'eux d'une équation qui traduit les évolutions possibles vers d'autres états. Les étiquettes sont appelées *actions*, elles appartiennent à un ensemble infini arbitraire que l'on notera Act ; on note $X = \alpha.Y$ le fait que de l'état X l'automate puisse passer à l'état Y en effectuant l'action α. Lorsque plusieurs actions sont possibles, on utilise l'opérateur de *choix* $+$. $\mathbf{0}$ représente l'état inactif.

Exemple 1.1.1 *On représente une pendule avec balancier par un automate à deux états et pouvant effectuer deux actions* tic *et* tac*. C'est un automate fini déterministe :*



On représente un pêcheur par un automate à quatre états (tendu, déçu, ému, repu), et trois actions jetteligne, reprendsligne, *et* mangepoisson *; la pêche est non déterministe, la même action de reprendre la ligne peut tout aussi bien mettre le pêcheur dans un état ému que déçu.*

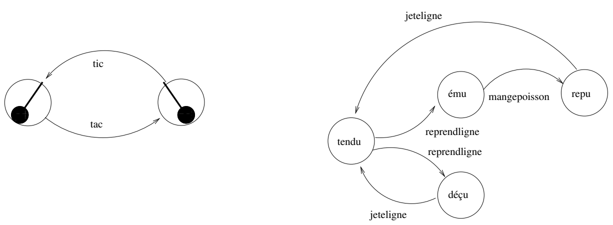


FIG. 1.1 – Automates du balancier et du pêcheur

$$
\begin{array}{rcl}
\mathit{tendu} & = & \mathsf{reprendsligne}.\mathit{d\acute{e}cu} \;+\; \mathsf{reprendsligne}.\mathit{\acute{e}mu} \\
\mathit{\acute{e}mu} & = & \mathsf{mangepoisson}.\mathit{repu} \\
\mathit{repu} & = & \mathsf{jetteligne}.\mathit{tendu} \\
\mathit{d\acute{e}\c{c}u} & = & \mathsf{jetteligne}.\mathit{tendu}
\end{array}
$$

1.1.2 Processus et interaction

Les automates sont amenés à agir en parallèle dans un même système. On introduit donc un nouvel opérateur, noté $|$, pour la *composition parallèle* de deux automates. Le système composite $A|B$, au même titre qu'un automate, peut effectuer des actions et changer d'état. Du point de vue du langage reconnu, on obtient le langage des entrelacements libres des langages reconnus par A et B. On veut néanmoins exprimer la notion de synchronisation entre deux composants. Ainsi, on considère une action particulière τ pour la synchronisation, et une notion de conjugaison sur les actions, soit une involution $\mathsf{Act}{\rightarrow}\mathsf{Act}$. Si A par l'action α passe à l'état A', et si B par l'action conjuguée $\overline{\alpha}$ passe à l'état B', alors le système $A|B$ peut effectuer l'action de synchronisation τ et passer dans l'état $A'|B'$.

Exemple 1.1.2 *On considère le système* feu de circulation/voiture*, où le feu de circulation synchronise le passage d'une voiture* $\mathsf{Vroum} \stackrel{\text{def}}{=} pv.\mathsf{Vroum}$ *qui cherche en permanence à passer (action pv). Le feu change d'états par transitions interne, et selon son état laisse ou non passer la voiture :*

$$\begin{aligned}
\mathsf{FV} &\stackrel{\text{def}}{=} \overline{pv}.\mathsf{FV} \ + \ \tau.\mathsf{FO} \\
\mathsf{FO} &\stackrel{\text{def}}{=} \overline{pv}.\mathsf{FO} \ + \ \tau.\mathsf{FR} \\
\mathsf{FR} &\stackrel{\text{def}}{=} \tau.\mathsf{FV}
\end{aligned}$$

Les actions du système composite $A|B$ sont donc des actions de A sans effet sur B, des actions de B sans effet sur A, ou des actions de synchronisation qui affectent à la fois A et B. La façon la plus claire d'exprimer ces phénomènes est de définir un système de transitions étiquetées, comme on le verra au prochain chapitre. Pour le moment, on va donner une vision complètement algébrique des processus.

1.1.3 Soupe de processus

On a donc donné les constructions essentielles d'un calcul de synchronisation : variables et équations récursives, préfixage par une action, choix non-déterministe, composition parallèle, et inactivité. On appelle ainsi processus un terme P issu de la grammaire syntaxique suivante[1]

$$\begin{array}{rlll}
P & ::= & \mathbf{0} & \text{(neutre)} \\
 & & \mid X & \text{(variable d'état)} \\
 & & \mid \mathbf{rec}\ X.P & \text{(équation récursive)} \\
 & & \mid \alpha.P & \text{(préfixe d'action)} \\
 & & \mid P+P & \text{(choix non déterministe)} \\
 & & \mid P|P & \text{(composition parallèle)}
\end{array}$$

Exemple 1.1.3 *Les automates sont des processus. Ainsi, le pêcheur dans l'état tendu est représenté par le processus*

$$\mathbf{rec}\ \text{tendu}.\ \big(\mathsf{reprendligne}.\mathsf{jetteligne}.\text{tendu} \ + \ \mathsf{reprendligne}.\mathsf{mangepoisson}.\mathbf{0}\ \big)$$

[1] Notons qu'à cette étape, on ne distingue plus les processus des automates. Cette généralisation est très commode mais perd la notion explicite d'agent au niveau du calcul. La notion d'agent est "dans les yeux du spécificateur" qui stratifie son système en divers sous-systèmes et raisonne de façon modulaire ; c'est seulement par l'opérateur de restriction, que l'on omet pour le moment, que l'on pourra expliciter l'interface d'un processus.

On identifie différentes écritures d'un même processus. Les opérateurs de choix et de composition parallèle sont en effet fondamentalement commutatifs, et le processus inactif **0** est élément neutre pour ces opérateurs. On considère donc les processus non pas à un niveau syntaxique strict, mais on s'autorise à identifier par une relation dite de *congruence structurelle* $\equiv$ des processus structurellement équivalents. La congruence structurelle est ainsi définie comme la plus petite relation d'équivalence congruente satisfaisant les axiomes suivants :

$$\begin{array}{c} P|\mathbf{0} \ \equiv \ P \qquad P+\mathbf{0} \ \equiv \ P \qquad \mathbf{rec}\ X.P \ \equiv \ P\{^{\mathbf{rec}\ X.P}/X\} \\ P|Q \ \equiv \ Q|P \qquad P+Q \ \equiv \ Q+P \\ P|(Q|R) \ \equiv \ (P|Q)|R \qquad P+(Q+R) \ \equiv \ (P+Q)+R \end{array}$$

On parle alors de *soupe* de processus[2], dans laquelle les processus évoluent dans un même milieu réactif.

On se focalise sur l'évolution du système par synchronisations internes, et on représente cette évolution par une relation dite *de réduction* $\longrightarrow$: ainsi la réduction $P \longrightarrow Q$ correspond à une synchronisation interne à P qui évolue vers Q, (action τ), et peut se définir comme la plus petite relation vérifiant les axiomes[3] :

$$\alpha.P+R \mid \overline{\alpha}.Q+S \ \longrightarrow \ P|Q \qquad \frac{P \ \longrightarrow \ P'}{P|Q \ \longrightarrow \ P'|Q}.$$

Notation : On notera $\Rightarrow$ la relation $\longrightarrow^*$, clôture réflexive transitive de $\longrightarrow$, et on adoptera la même notation pour les autres relations de réduction que l'on considèrera par la suite.

Par la suite, on appellera CCS fini sans choix, ou par abus simplement CCS, le fragment du calcul suivant :

$$P \ ::= \ \mathbf{0} \ \big| \ \alpha.P \ \big| \ P|P$$

On supprime d'une part la récursion et d'autre part le choix. On adopte cette restriction essentiellement dans un souci de simplicité ; on perd d'ailleurs assez peu de l'expressivité du calcul. D'une part on aura la possibilité de considérer des processus arbitrairement grands et ainsi de simuler un comportement récursif fini, d'autre part même sans le choix, le non déterminisme reste présent dans le calcul, par exemple dans le processus $\alpha.P|\overline{\alpha}.Q_1|\overline{\alpha}.Q_2$.

On considèrera par ailleurs des *permutations d'actions* : une permutation d'actions est une bijection $\sigma : \mathsf{Act} \to \mathsf{Act}$ telle que $\sigma(\overline{\alpha}) = \overline{\sigma(\alpha)}$

Le calcul actuel permet de représenter des problèmes de synchronisation entre processus ; cela reste un calcul très simple, mais son expressivité est limitée, en particulier la diffusion d'actions dans le système est extrêmement élémentaire : toute action est perceptible à l'échelle du système entier. On va voir maintenant qu'en parlant de canaux et de communications, on peut donner des descriptions plus fines des systèmes distribués.

[2] L'origine de cette appellation vient de l'article fondateur [BB90]. Cet article fait effectivement naître la notion d'algèbre de processus comme calcul algébrique avec règle de réduction, alors que les présentations antérieures considéraient la synchronisation comme une action au même titre que les autres.

[3] On se place ici en choix gardé, sinon la définition de $\longrightarrow$ nécessite un système de transitions étiquetées (cf. prochain chapitre).

1.2 Un calcul de canaux : le π-calcul

Jusqu'à présent, on a parlé de synchronisation entre deux parties sans véritablement en expliciter le mécanisme. En pratique, les synchronisations se font par le biais de communications, soit directement entre les deux parties, soit en passant par un arbitre (verrous, barrières de synchronisation). Dans tous les cas, la communication est supportée par un *médium*. On peut ainsi interpréter le processus $\alpha.P$ comme un agent qui cherche à se synchroniser en utilisant le médium de communication α. On est alors amené à considérer un réseau d'agents reliés entre eux par des média de communications. Les modèles classiques de réseaux s'appuient sur des graphes, mais l'originalité du π-calcul est d'en donner une représentation algébrique[4].

1.2.1 Des synchronisations aux communications

De façon surprenante, une seule opération est nécessaire : il suffit de dire qui a le droit d'utiliser un médium pour fixer clairement la topologie du réseau. Pour cela, on introduit la notion de *restriction* : dans le processus $(\nu n)P$, le médium n est connu seulement par P (on dira qu'il est privé), et ne peut être utilisé pour se synchroniser qu'à l'intérieur de P.

Exemple 1.2.1 *Dans le processus $n.P|\overline{n}.Q|\overline{n}.R$, P peut se synchroniser aussi bien avec Q qu'avec R. En revanche, dans le processus $(\nu n)\big(n.P|\overline{n}.Q\big)\ |\overline{n}.R$, P ne peut plus se synchroniser qu'avec Q.*

Le fondement syntaxique de la restriction est la notion de lieur : les références à n qui apparaissent dans la portée d'un (νn) n'ont pas la même signification que les références à n qui apparaissent en dehors. Si les deux coexistent, on utilise une règle d'alpha conversion $(\nu n)P \equiv (\nu n')P\{^{n'}/n\}$ pour lever éventuellement l'ambiguïté, et on ajoute par ailleurs comme règle structurelle l'*extension de restriction* (en anglais *scope extrusion*)

$$\big((\nu n)P\big)\ |\ Q \quad \equiv \quad (\nu n)\big(P|Q\big) \qquad \text{lorsque } Q \text{ ignore } n.$$

Par α-conversion et extension de restriction, tous les agents sont ainsi structurellement en parallèle malgré la privauté de certains média :

Exemple 1.2.2 *Par les égalités suivantes*

$$\big((\nu n)n.P\big)\ |\ \overline{n}.Q \quad \equiv \quad \big((\nu n')n'.P\big)\ |\ \overline{n}.Q \quad \equiv \quad (\nu n')\big(n'.P|\overline{n}.Q)$$

on peut considérer P et Q comme deux agents en parallèle sans possibilité de synchronisation.

Pour parler complètement de communication, il faut aussi tenir compte du contenu de cette communication. Dans les modèles de calcul, les objets manipulés par des agents sont parfois appelés *valeurs*, comme résultat d'un calcul antérieur. Il a ainsi été proposé une version de CCS où au cours de leurs synchronisations les agents échangent des valeurs [Mil89].

[4]La traduction de la définition algébrique en termes de graphe a été par ailleurs abondamment étudiée [Hir97, DL03].

Une idée intéressante est de confondre sous l'appellation de *canal* les deux notions de médium et de de valeur[5]. Ceci permet de représenter des protocoles de communication par l'entremise d'un tiers : par exemple, pour communiquer avec C, A demande tout d'abord à B un canal connu par C, B transmet à A un tel canal, puis A communique avec C par ce nouveau canal. De cette façon, B établit une nouvelle possibilité de communication entre A et C et modifie la topologie du réseau de canaux. De façon plus générale, la communication de canaux sur des canaux permet de voir le calcul comme la reconfiguration du réseau de communication entre agents, et on parle alors de *mobilité*.

Dans une communication enfin, les deux parties jouent des rôles dysymétriques. On a d'une part une émission d'un message b sur le canal a, notée $\overline{a}\langle b\rangle.P$, d'autre part la réception d'un message sur le canal a, notée $a(x).P$. Ici le x figure l'abstraction du contenu du message avant réception. Cette abstraction se concrétise au moment de la communication par une simple substitution :

$$a(x).P \mid \overline{a}\langle b\rangle.Q \quad\longrightarrow\quad P\{^b/x\} \mid Q$$

Ainsi le processus $a(x).\overline{x}\langle b\rangle$ attend sur le canal a un canal inconnu x sur lequel il transmettra ensuite b. La réduction suivante

$$a(n).P \mid (\nu m)\overline{a}\langle m\rangle.Q \quad\longrightarrow\quad (\nu m).\big(P\{^m/n\}|Q\big)$$

traduit effectivement la mise en communication de P et Q par l'entrée du canal privé m dans P.

On va maintenant préciser formellement ces idées.

1.2.2 Définition du π-calcul

On considère un ensemble infini de *noms* Na (on adoptera cette terminologie pour parler de ce que l'on a désigné jusque là comme des canaux), que l'on note $a, b, c, \ldots$.

Définition 1.2.3 *On appelle* processus *du π-calcul (sans choix) les termes issus de la grammaire suivante :*

$$\begin{array}{rlll} P & ::= & \mathbf{0} & \textit{(neutre)} \\ & & \mid a(n).P & \textit{(réception)} \\ & & \mid \overline{a}\langle b\rangle.P & \textit{(émission)} \\ & & \mid P|P & \textit{(composition parallèle)} \\ & & \mid\, !P & \textit{(réplication)} \\ & & \mid (\nu n)P & \textit{(restriction)} \end{array}$$

On note $\mathsf{fn}(P)$ l'ensemble des noms libres de P, soit l'ensemble défini par induction sur P par

$$\mathsf{fn}(\mathbf{0}) = \emptyset \quad \mathsf{fn}(a(n).P) = (\mathsf{fn}(P) - \{n\}) \cup \{a\} \quad \mathsf{fn}(\overline{a}\langle b\rangle.P) = \mathsf{fn}(P) \cup \{a, b\}$$
$$\mathsf{fn}(P|Q) = \mathsf{fn}(P) \cup \mathsf{fn}(Q) \quad \mathsf{fn}(!P) = \mathsf{fn}(P) \quad \mathsf{fn}((\nu n)P) = \mathsf{fn}(P) - \{n\}.$$

[5]On peut considérer que le λ-calcul procède de la même façon en confondant la valeur et le manipulateur de valeur (fonction).

On définit la relation de congruence structurelle $\equiv$ comme la plus petite relation de congruence sur les processus qui vérifie les axiomes :

$$
\begin{array}{c}
P|\mathbf{0} \;\equiv\; P \quad \big((\nu n)P\big)|Q \;\equiv\; (\nu n)\big(P|Q\big) \quad \textit{pour } n \notin \mathsf{fn}(Q) \\
(\nu n)\mathbf{0} \;\equiv\; \mathbf{0} \quad (\nu n)P \;\equiv\; (\nu n')\big(P\{^{n'}/n\}\big) \quad P|Q \;\equiv\; Q|P \quad (P|Q)|R \;\equiv\; P|(Q|R) \\
!P \;\equiv !P|P \quad !\mathbf{0} \;\equiv \mathbf{0} \quad !(P|Q) \;\equiv !P|!Q \quad !!P \;\equiv !P
\end{array}
$$

Le seul opérateur que l'on n'a pas introduit jusqu'ici est l'opérateur de réplication. C'est un opérateur infinitaire qui s'inspire du connecteur "bien sûr" en logique linéaire et exprime la persistance d'un processus. Il remplace de façon élégante la récursion pour créer des processus avec des réductions infinies.

Pour parler de réduction, on introduit donc la relation $\longrightarrow$ qui exprime le mécanisme de communication :

Définition 1.2.4 *On appelle relation de réduction la plus petite relation $\longrightarrow$ qui vérifie l'axiomatisation suivante :*

$$
\frac{}{a(n).P \mid \overline{a}\langle b\rangle.Q \;\longrightarrow\; P\{^{b}/n\} \mid Q}
\qquad
\frac{P \;\longrightarrow\; P'}{P \mid Q \;\longrightarrow\; P' \mid Q}
$$

$$
\frac{P \;\longrightarrow\; P'}{(\nu n)P \;\longrightarrow\; (\nu n)P'}
\qquad
\frac{P \;\equiv\; P_1 \quad P_1 \;\longrightarrow\; P_2 \quad P_2 \;\equiv\; P'}{P \;\longrightarrow\; P'}
$$

On mentionne brièvement le résultat suivant : le π-calcul permet de représenter le λ-calcul en appel par nom ou par valeur, et une notion de types sur les canaux permet une interprétation du λ-calcul typé ; pour plus de détail, voir [SW01].

1.2.3 Quelques dialectes π

Afin d'illustrer l'expressivité de cette algèbre de processus, on la compare maintenant à plusieurs variantes qui en ont été proposées.

Une première variation consiste à communiquer non pas un mais un nombre arbitraires de canaux en une étape de communication. On parle alors de communication *polyadique* (et le π-calcul défini précédemment est appelé, par opposition, *monadique*). Formellement, cela revient à remplacer la règle de communication précédente par

$$a(n_1,\ldots,n_k).P \mid \overline{a}\langle b_1,\ldots,b_k\rangle.Q \quad\longrightarrow\quad P\{b_1/n_1,\ldots,b_k/n_k\} \mid Q$$

Pour représenter une communication polyadique en calcul monadique, on la décompose en plusieurs communications. Toutefois, remplacer l'émission $\overline{a}\langle b_1,\ldots,b_k\rangle.Q$ par $\overline{a}\langle b_1\rangle\ldots\overline{a}\langle b_k\rangle.Q$ ne garantit pas que le même agent recevra chacun des b_i. Pour garantir l'atomicité de la communication, l'émetteur commence donc par créer un canal de communication privé qu'il révèle à un seul partenaire, puis émet sur ce canal privé :

$$[\![\overline{a}\langle b_1,\ldots,b_k\rangle.Q]\!] \quad=\quad (\nu w)\ \overline{a}\langle w\rangle.\overline{w}\langle b\rangle_1\ldots\overline{w}\langle b\rangle_k.[\![Q]\!]$$

De même un récepteur attend tout d'abord un canal de communication avant de lire les noms transmis :

$$[\![a(n_1,\ldots,n_k).P]\!] \quad=\quad a(w).w(n_1)\ldots w(n_k).[\![P]\!] \qquad (w \notin \mathsf{fn}(P))$$

Cette traduction préserve le comportement du processus[6], c'est à dire comme on le verra au chapitre suivant son interprétation extensionnelle. En particulier, cette transformation fournit une traduction du calcul CCS dans le π-calcul, puisque ce premier correspond au π-calcul polyadique dans lequel les n-uplets de noms communiqués sont toujours vides.

Une autre variante intéressante est de considérer des communications *asynchrones*. Dans la synchronisation $a(n).P \mid \overline{a}\langle b\rangle.Q \quad\longrightarrow\quad P\{b/n\} \mid Q$, le processus Q doit attendre la fin de la communication pour pouvoir s'exécuter, et reçoit implicitement l'information que la communication s'est achevée. C'est cette propriété que l'on veut écarter des communications asynchrones. Pour cela, l'idée originale de Honda, Tokoro [HT91] et Boudol [Bou92] est de considérer le fragment suivant du π-calcul :

Définition 1.2.5 *On appelle π-calcul asynchrone le fragment du π-calcul de la définition 1.2.3 obtenu par la grammaire suivante :*

$$P \ ::= \ \mathbf{0} \ \big| \ P|P \ \big| \ (\nu n)P \ \big| \ \big| \ !P \ \big| \ a(n).P \ \big| \ \overline{a}\langle b\rangle.\mathbf{0}$$

On définit de même le π-calcul polyadique asynchrone.

Il est intéressant de comparer l'expressivité du calcul synchrone et du calcul asynchrone. Pour représenter des communications synchrones à partir des communications asynchrones, on peut utiliser l'encodage polyadique[7] suivant :

$$\begin{array}{rcl} [\![\overline{a}\langle b\rangle.P]\!] & = & (\nu n)\big(\overline{a}\langle b,n\rangle \mid n.[\![P]\!] \\ [\![a(x).P]\!] & = & a(x,n).\big(\overline{n} \mid [\![P]\!]\big) \end{array}$$

De même que pour l'encodage précédent, cette transformation conserve en partie la sémantique extensionnelle du processus.

Toutefois, il est intéressant de noter que l'asynchronisme se traduit aussi par une perte d'expressivité dans certains cas. Ainsi, si l'on considère un π-calcul avec un opérateur de choix $+$ similaire à celui présenté précédemment, on peut encoder un choix gardé homogène (des processus de la forme $\sum_i \alpha_i.P_i$ avec les α soit tous émetteurs, soit tous récepteurs), mais pas le choix mixte (voir [SW01, Pal97]).

Le passage de la notion de synchronisation à celle de la communication a donc permis de définir un calcul simple, d'une riche expressivité, et capable de représenter des modèles de calcul (λ-calcul) aussi bien que de modéliser à un niveau abstrait des problèmes de réalisation de langages concurrents (polyadicité, asynchronisme,...). On a toutefois en partie perdu la notion d'agent en tant qu'automate ; pour le moment, on ne sait si par agent du calcul on doit entendre un processus gardé $\alpha.P$ ou bien un processus insécable (par exemple $(\nu n)\big(\overline{a}\langle n\rangle.\mathbf{0}|n(x).P)\big)$. C'est en partie dans ce souci qu'une nouvelle classe d'algèbres de processus a été introduite, où la notion fondamentale de canal est remplacée par celle de localité.

1.3 Un calcul d'agents : les Mobile Ambients

Dans le cadre du π-calcul, on a parlé de *mobilité* pour les canaux de communication. Toutefois, à cause de la règle de congruence structurelle $(\nu n)\big(P \mid Q\big) \ \equiv \ (\nu n)P \mid Q$

[6] Quoique en partie seulement, voir [SW01].

[7] On peut par ailleurs encoder le calcul polyadique asynchrone à partir du calcul monadique asynchrone, cf [SW01].

lorsque $n \notin \mathsf{fn}(Q)$, les canaux privés ne sont attachés à des agents en particulier que du point de vue du spécificateur, mais aucun opérateur syntaxique ne vient préciser la localisation des canaux.

On part maintenant d'un point de vue différent [CG98]. Les spécifications de systèmes que l'on souhaite définir doivent prendre en compte une description explicite des possibilités d'interaction entre agents, et par ailleurs de la notion d'agent. Pour cela, on parle de localité, ou d'*ambient*, comme unité de calcul atomique. La topologie des localités s'exprime par des rapports hiérarchiques, et on n'a plus une soupe mais une arborescence de processus. Les interactions concernent alors des processus voisins au sens de cette topologie et se traduisent par une reconfiguration locale de la hiérarchie des processus par un mécanisme de vases communicants.

On présente tout d'abord le modèle algébrique d'arbres, appelé par la suite Ambients statiques, qui n'est pas encore une algèbre de processus, puis on développe l'algèbre des Mobile Ambients basée sur ce paradigme.

1.3.1 Un calcul d'arbres

On donne ici succinctement une présentation algébrique des arbres étiquetés, finis, non ordonnés, avec une racine distinguée. On considère à nouveau un ensemble infini de *noms* Na que l'on note $a, b, c, \ldots$:

Définition 1.3.1 *On appelle Ambients statiques les termes issus de la grammaire suivante :*

$$\begin{array}{lll} P ::= & \mathbf{0} & \textit{(neutre)} \\ & \mid n[P] & \textit{(ambient)} \\ & \mid P|P & \textit{(composition parallèle)} \\ & \mid (\nu n)P & \textit{(restriction)} \end{array}$$

On note $\mathsf{fn}(P)$ l'ensemble des noms libres de P, soit l'ensemble défini par induction sur P par

$$\mathsf{fn}(\mathbf{0}) = \emptyset \quad \mathsf{fn}(n[P]) = \mathsf{fn}(P) \cup \{n\}$$
$$\mathsf{fn}(P|Q) = \mathsf{fn}(P) \cup \mathsf{fn}(Q) \quad \mathsf{fn}((\nu n)P) = \mathsf{fn}(P) - \{n\}.$$

On définit la relation de congruence structurelle $\equiv$ comme la plus petite relation de congruence sur ces termes qui vérifie les axiomes :

$$P|\mathbf{0} \;\equiv\; P \quad \big((\nu n)P\big)|Q \;\equiv\; (\nu n)\big(P|Q\big) \quad \textit{pour } n \notin \mathsf{fn}(P)$$
$$(\nu n)\mathbf{0} \;\equiv\; \mathbf{0} \quad (\nu n)P \;\equiv\; (\nu n')\big(P\{^{n'}/n\}\big) \quad P|Q \;\equiv\; Q|P \quad (P|Q)|R \;\equiv\; P|(Q|R)$$
$$n[(\nu m)P] \;\equiv\; (\nu m)n[P] \quad (m \neq n)$$

On note la présence de la restriction de noms $(\nu n)P$. Sans cette construction, les termes s'interprètent directement comme des arbres étiquetés, mais si l'on cherche un modèle mathématique des ambients statiques, on peut penser à des arbres avec des étiquettes de deux sortes. Cette algèbre a été proposée comme modèle de données semi-structurées, en particulier de documents au format XML [CG01a], et dans ce modèle la restriction de nom représente un “lien symbolique” entre deux localités ; on reparlera de ce modèle avec les logiques spatiales au chapitre 3.

1.3.2 Mouvements et communications

On étend maintenant le modèle précédent par des instructions de mouvement qui permettent à l'agent qui les exécute de se déplacer dans son entourage d'agents, ainsi que de supprimer un ambient existant. On omet la restriction de nom, présente dans de nombreuses présentations du calcul [CG98], essentiellement dans un souci de simplicité quant aux preuves qui seront développées pour cette algèbre de processus. On omet aussi les communications, que l'on discute un peu après.

Définition 1.3.2 *On appelle* Mobile Ambients publics *(MA) les termes issus de la grammaire suivante :*

$$
\begin{array}{rcll}
P & ::= & \mathbf{0} & \textit{(neutre)} \\
 & & \mid n[P] & \textit{(ambient)} \\
 & & \mid P|P & \textit{(composition parallèle)} \\
 & & \mid\, !P & \textit{(réplication)} \\
 & & \mid \mathsf{in}\, n.P & \textit{(capabilité d'entrée)} \\
 & & \mid \mathsf{out}\, n.P & \textit{(capabilité de sortie)} \\
 & & \mid \mathsf{open}\, n.P & \textit{(capabilité d'ouverture)}
\end{array}
$$

On note $\mathsf{fn}(P)$ *l'ensemble des noms de* P*, soit l'ensemble défini par induction sur* P *par*

$$
\begin{array}{c}
\mathsf{fn}(\mathbf{0}) = \emptyset \quad \mathsf{fn}(n[P]) = \mathsf{fn}(P) \cup \{n\} \\
\mathsf{fn}(P|Q) = \mathsf{fn}(P) \cup \mathsf{fn}(Q) \quad \mathsf{fn}(\mathsf{in}\, nP) = \mathsf{fn}(\mathsf{out}\, nP) = \mathsf{fn}(\mathsf{open}\, nP) = \mathsf{fn}(P) \cup \{n\}.
\end{array}
$$

On définit la relation de congruence structurelle $\equiv$ *comme la plus petite relation de congruence sur ces termes qui vérifie les axiomes :*

$$
\begin{array}{c}
P|\mathbf{0} \;\equiv\; P \quad P|Q \;\equiv\; Q|P \quad (P|Q)|R \;\equiv\; P|(Q|R) \\
!P \;\equiv\; !P|P \quad !\mathbf{0} \;\equiv\; \mathbf{0} \quad !!P \;\equiv\; !P \quad !(P|Q) \;\equiv\; !P|!Q
\end{array}
$$

Les instructions de mouvement d'entrée et de sortie sont exécutées par un agent, et entraînent le mouvement de cet agent. En revanche, l'instruction d'ouverture dissout un ambient du même niveau. Ces instructions peuvent s'exécuter à n'importe quel niveau de l'arbre : d'un point de vue formel, les seuls constructeurs bloquants pour $\longrightarrow$ (appelés "gardes") sont les capabilités $\mathsf{in}\, n.-$, $\mathsf{out}\, n.-$ et $\mathsf{open}\, n.-$. Par la suite, on adopte la notation cap pour désigner de façon générique l'un quelconque de ces constructeurs.

Définition 1.3.3 *On appelle relation de réduction la plus petite relation* $\longrightarrow$ *qui vérifie l'axiomatisation suivante :*

$$
\frac{}{a[\mathsf{in}\, n.P \mid Q] \mid n[R] \;\longrightarrow\; n[a[P \mid Q] \mid R]} \qquad \frac{}{n[a[\mathsf{out}\, n.P \mid Q] \mid R] \;\longrightarrow\; a[P \mid Q] \mid n[R]}
$$

$$
\frac{}{\mathsf{open}\, n.P \mid n[Q] \;\longrightarrow\; P \mid Q} \qquad \frac{P \;\longrightarrow\; P'}{P \mid Q \;\longrightarrow\; P' \mid Q}
$$

$$
\frac{P \;\longrightarrow\; P'}{(\nu n)P \;\longrightarrow\; (\nu n)P'} \qquad \frac{P \;\longrightarrow\; P'}{n[P] \;\longrightarrow\; n[P']}
$$

$$
\frac{P \;\equiv\; P_1 \quad P_1 \;\longrightarrow\; P_2 \quad P_2 \;\equiv\; P'}{P \;\longrightarrow\; P'}
$$

Au cours d'une réduction, une seule instruction est donc exécutée, mais deux ou trois parties sont impliquées dans la synchronisation. Cette situation, différente de celle des autres algèbres de processus, est problématique à de nombreux points de vue ; ainsi des variantes synchrones de ce calcul ont été proposées[8]. D'autres points ont fait l'objet de nombreuses discussions, comme le statut des messages, la capabilité détenue par l'objet ou le sujet du mouvement, etc. Dans [DP04], une justification du jeu d'instructions est donnée par une propriété de symétrie interne du calcul.

Communications et Ambients
La présentation originale du calcul des Ambients inclut une notion de communication. Par la suite, on n'abordera qu'à de rares occasions la question des communications dans ce calcul[9].

Les communications dans les Mobile Ambients ont lieu à l'intérieur d'un Ambient[10]. De ce fait, l'ambient joue le rôle du canal de synchronisation en π-calcul, et seul compte alors le message M transmis dans la communication. On rajoute donc les constructeurs $(x)P$ pour la réception, et $\langle M\rangle.P$ pour l'émission. La règle de communication pour $\longrightarrow$ est alors simplement

$$(x)P \mid \langle M\rangle.Q \quad\longrightarrow\quad P\{^M/x\} \mid Q.$$

Dans la présentation originale du calcul [CG98], les messages peuvent être aussi bien des noms de localité que des séquences de capabilités, et les émissions sont toujours asynchrones.

1.4 Propriétés de processus

On conclut cette présentation des algèbres de processus par la présentation de diverses propriétés des processus. Ces propriétés sont utilisées dans de multiples preuves que l'on développe dans les chapitres suivants.

Dans une algèbre de processus, on appelle *garde*, ou capabilité, les constructeurs qui sont bloquants, c'est à dire qui ne sont pas congruents pour la relation de réduction $\longrightarrow$. Pour le π-calcul, les gardes sont ainsi les constructeurs $a(n).-$ et $\overline{a}\langle b\rangle.-$. Les gardes garantissent la séquentialité des instructions qu'elles représentent. On peut alors mesurer le *degré de séquentialité* d'un processus comme l'imbrication maximale de gardes qu'il contient. Plus formellement

Définition 1.4.1 *Dans CCS, on appelle degré de séquentialité d'un processus P l'entier noté $\mathsf{ds}(P)$ et défini par*

$$\mathsf{ds}(\mathbf{0}) \;=\; 0 \qquad \mathsf{ds}(P|Q) \;=\; \max \mathsf{ds}(P), \mathsf{ds}(Q) \qquad \mathsf{ds}(\alpha).P \;=\; 1 + \mathsf{ds}(P)$$

De même, pour les Mobile Ambients, le degré de séquentialité d'un processus P est l'entier noté $\mathsf{ds}(P)$ défini par

$$\begin{gathered}\mathsf{ds}(\mathbf{0}) \;=\; 0 \qquad \mathsf{ds}(P|Q) \;=\; \max \mathsf{ds}(P), \mathsf{ds}(Q) \qquad \mathsf{ds}(!P) \;=\; \mathsf{ds}(P)\\ \mathsf{ds}(n[P]) \;=\; \mathsf{ds}(P) \qquad \mathsf{ds}(\mathsf{cap}.P) \;=\; 1 + \mathsf{ds}(P)\end{gathered}$$

[8] Voir par exemple [LS00a, TZH02].

[9] Ceci à nouveau par souci de simplicité. En revanche, les résultats sur les Ambients présentés en annexe tiennent compte des communications. On y trouvera du reste leur définition précise.

[10] Là aussi, de nombreuses variantes existent.

Pour une capabilité cap des Mobile Ambients, on note $P \stackrel{\mathsf{cap}}{\rightarrow} Q$ lorsque $P \equiv \mathsf{cap}.P_1|P_2$ et $Q \equiv P_1|P_2$ (cette notation s'explique par la notion de transition étiquetée que l'on abordera au prochain chapitre). Le résultat suivant énonce la monotonie du degré de séquentialité pour la réduction et les actions :

Lemme 1.4.2 *Pour tous processus P, Q du calcul des Mobile Ambients*[11]*, si $P \stackrel{\mathsf{cap}}{\rightarrow} Q$ pour un certain cap, ou si $P \longrightarrow Q$, alors $\mathsf{ds}(P) \geq \mathsf{ds}(Q)$.*

On définit par ailleurs $\pi_k(P)$ la troncature de P à profondeur k. Cette notion ne nous sera utile que dans le contexte de CCS :

Définition 1.4.3 *Soit P un processus de CCS, et k un entier positif. On note $\pi_k(P)$ le processus défini par*

$$\pi_0(P) \; = \; \mathbf{0}$$

$$\pi_{i+1}(\mathbf{0}) \; = \; \mathbf{0} \qquad \pi_{i+1}(P|Q) \; = \; \pi_{i+1}(P)|\pi_{i+1}(Q) \qquad \pi_{i+1}(\alpha.P) \; = \; \alpha.\pi_i(P)$$

La partie du terme effacée par troncature ne joue en effet un rôle qu'après un nombre suffisant de réductions. Ainsi, on peut montrer par induction que pour tous $k' \leq k$, $P \longrightarrow^{k'} Q$ ssi $\pi_k(P) \longrightarrow^{k'} \pi_k(Q)$.

1.5 Conclusion

Ce premier chapitre a donc permis de présenter divers modèles algébriques de calculs d'interactions. Ils illustrent les notions de concurrence et de mobilité par des constructions algébriques originales, et fournissent un formalisme puissant pour la spécification de systèmes et de protocoles réels. On va maintenant développer pour ces algèbres une notion de comportement, ainsi que les bases logiques originellement employées pour leur vérification.

[11] On peut bien sûr étendre ce résultat aux autres calculs

Chapitre 2

Extensionalité

La connaissance des objets qui nous entourent est à la fois d'origine mécaniste et expérimentale. On est parfois l'auteur de ces objets, et on les connaît par leur mécanique interne que l'on a par exemple soit-même créée. C'est le niveau de connaissance le plus avancé que l'on puisse avoir de l'objet, car même si l'on n'en a pas encore compris complètement le comportement, on sait synthétiser l'objet et faire des expériences avec celui-ci. Ce niveau de connaissance est dit *intensionnel.* Dans d'autres cas, on ne connaît pas la mécanique interne, mais l'abondance de cet objet permet d'en faire de multiples expériences et d'en donner une description comportementale complète. On ne peut donc en deviner la mécanique interne que jusqu'à un certain point. C'est le niveau de connaissance dit *extensionnel.*

Pour les algèbres de processus, le niveau de description intensionnel correspond à l'implémentation du système, sa description purement syntaxique, et deux processus sont intensionnellement équivalents si ils ont la même implémentation, à l'équivalence structurelle $\equiv$ près. En revanche, le niveau de description extensionnel dépend de l'observation que l'on sait faire du système. L'idée essentielle de Milner [Mil89] consiste à considérer les expérimentateurs du processus comme des processus eux-mêmes. Cette idée donne lieu à deux définitions de l'équivalence extensionnelle.

La première définition est extrêmement opérationnelle. On définit une notion d'observation primitive sur les processus, appelée *barbe*, et on définit comme observationnellement équivalents des processus qui, *dans n'importe quel contexte de test*, exhibent les mêmes barbes. Cette notion est ainsi appelée *congruence à barbe.*

La deuxième définition de l'équivalence extensionnelle est plus subtile. Elle s'abstrait du processus testeur, et ne retient de lui que sa possibilité de tester des actions sur le processus examiné. A chaque action correspond une transition d'état, et une nouvelle étape de test. Pour représenter l'ensemble des expériences possibles, on dresse un graphe de transitions du système, et c'est sur cette notion que l'on définit l'équivalence extensionnelle. On a recours pour cela à la notion de *bisimulation*, deux graphes de transitions étant équivalents ssi ils savent simuler leurs transitions l'un l'autre.

Dans une spécification de processus, il n'est toutefois pas toujours nécessaire d'en décrire tout le comportement, mais seules certaines propriétés de sécurité (vivacité, disponibilité,...). On utilise alors un langage descriptif pour ces propriétés. Le langage introduit par Hennessy-Milner [HM85] est une logique modale qui voit les systèmes de transition étiquetées comme des modèles de Kripke. Cette logique est la logique extensionnelle canonique pour les algèbres de processus. Des logiques extensionnelles moins connues ont été proposées, basées entre autre sur la logique de la relevance et la logique linéaire[Dam88].

Dans ce chapitre, on introduit tout d'abord l'équivalence comportementale pour CCS ; ceci conduit aux systèmes de transition étiquetées et à la notion de bisimilarité. On étend ensuite ces idées au π-calcul, et l'on compare dans ce cas la notion de bisimilarité à celle de congruence à barbe. On définit par ailleurs la congruence à barbe pour les Ambients. On aborde enfin succinctement les travaux de Mads Dam sur l'interprétation de la logique linéaire et de la logique de la relevance.

2.1 Equivalence comportementale en CCS

Dans cette section on s'intéresse à l'algèbre de processus CCS, avec l'opérateur de choix $P+Q$. On définit le comportement du processus comme un graphe de transitions étiquetées par les actions que peut réaliser le processus pour interagir avec son environnement. Puis on définit plusieurs équivalences comportementales entre processus basées sur leur graphe de transition.

2.1.1 Graphes de transitions étiquetées

On définit ici deux notions de transitions entre états. Un processus évolue par réalisation d'une action. On distingue parmi les actions celles qui sont visibles, et destinées à se synchroniser avec l'environnement, et celles qui sont internes au système, les actions τ, qui lui permettent d'évoluer par une synchronisation interne.

La première notion de transition, notée $P \xrightarrow{\alpha} P'$, est dite forte, et correspond à l'observation du processus P vers le processus P' par l'action α sans qu'aucune synchronisation interne supplémentaire ne soit autorisée. La deuxième notion de transition, notée $P \stackrel{\alpha}{\Longrightarrow} P'$, est dite faible, et correspond à l'évolution par l'action α modulo d'éventuelles synchronisations internes.

Définition 2.1.1 *On note $P \xrightarrow{\alpha} P'$, la plus petite relation qui vérifie les règles suivantes :*

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$
$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

On rappelle que l'on note $\Rightarrow$ la clôture réflexive transitive de $\rightarrow$.

Définition 2.1.2 *Pour $\alpha \neq \tau$, on note $P \stackrel{\alpha}{\Longrightarrow} P'$ si il existe des entiers $n_1, n_2 \geq 0$ tels que $P \xrightarrow{\tau}^{n_1} \xrightarrow{\alpha} \xrightarrow{\tau}^{n_2}$, et on note $P \stackrel{\tau}{\Longrightarrow} P'$ si $P \Rightarrow P'$.*

2.1.2 Bisimilarité

On définit la notion de bisimilarité entre deux processus. A chaque type de transition, forte ou faible, correspond une bisimilarité.

Définition 2.1.3 *On appelle bisimilarité forte la plus grande relation symétrique $\sim$ telle que pour tout processus P, Q, si $P \ \sim \ Q$, on ait :*
pour tout α, P' tels que $P \xrightarrow{\alpha} P'$, il existe Q' tel que $Q \xrightarrow{\alpha} Q'$ et $P' \sim Q'$.
On appelle bisimilarité faible la plus grande relation symétrique $\approx$ telle que pour tout processus P, Q, si $P \ \approx \ Q$, on ait :
pour tout α, P' tels que $P \xrightarrow{\alpha} P'$, il existe Q' tel que $Q \stackrel{\alpha}{\Longrightarrow} Q'$ et $P' \approx Q'$.

Cette définition a un sens, elle est fondée par une coinduction [Mil89], et peut aussi s'exprimer comme le plus grand point fixe d'une fonctionnelle. On vérifie aisément que les bisimilarités définissent des relations d'équivalence entre processus.

On donne maintenant une brève illustration de l'intérêt de la bisimilarité dans la spécification de systèmes concurrents :

Exemple 2.1.4 *On reprend ici le processus de feu de circulation de l'exemple 1.1.2, et on en précise l'implémentation comme un processus affichage qui stoppe ou laisse passer les voitures, et une horloge qui décide du changement d'affichage :*

$$\begin{aligned} \mathsf{AFV} &\stackrel{\text{def}}{=} \overline{pv}.\mathsf{AFV} \ + \ \overline{tic}.\mathsf{AFO} \\ \mathsf{AFO} &\stackrel{\text{def}}{=} \overline{pv}.\mathsf{AFV} \ + \ \overline{tic}.\mathsf{AFR} \\ \mathsf{AFV} &\stackrel{\text{def}}{=} \overline{tic}.\mathsf{AFV} \\ \mathsf{CLOCK} &\stackrel{\text{def}}{=} tic.\mathsf{CLOCK} \end{aligned}$$

On considère alors le processus $(\nu tic)\big(\mathsf{AFV}|\mathsf{CLOCK}\big)$, c'est à dire le système composé des deux processus AFV et CLOCK en parallèle, avec l'action tic comme action de synchronisation interne, donc non observable pour la bisimilarité. On vérifie alors que cette implémentation vérifie sa spécification de l'exemple 1.1.2 du fait que

$$\mathsf{FV} \ \sim \ (\nu tic)\big(\mathsf{AFV}|\mathsf{CLOCK}\big)$$

et de même pour $\approx$. Cet exemple est relativement simple du fait que la reconfiguration interne se fait en une seule étape de synchronisation (sur tic). En général, on ne peut prouver l'équivalence que pour la bisimilarité faible $\approx$, les synchronisations internes n'étant pas comptées avec leur multiplicité dans la spécification.

Les bisimilarités sont donc utilisées pour comparer l'implémentation d'un processus à sa spécification. On souhaite donner une spécification modulaire d'un système en divers sous-systèmes vérifiés séparément, puis remplacer leur implémentation par leur spécification dans le système final (le système feu/voiture dans les exemples 1.1.2 et 2.1.4). Pour cela, il faut que la bisimilarité entre processus soit préservée par la mise en contexte. On doit pour cela vérifier que l'équivalence entre processus considérée est une congruence (on verra au chapitre 7 une autre application de la propriété de congruence). C'est effectivement le cas pour $\sim$, mais pas pour $\approx$. En effet, $\alpha.\mathbf{0} \approx \tau.\alpha.\mathbf{0}$, mais $\beta.\mathbf{0} + \alpha.\mathbf{0} \ \not\approx \ \beta.\mathbf{0} + \tau.\alpha.\mathbf{0}$ (voir [Mil89]).

On a donc vu comment décrire l'équivalence comportementale entre processus de CCS par la bisimilarité entre graphes de transitions. On reprend maintenant ces idées dans le cadre du π-calcul.

2.2 Équivalences comportementales en π-calcul et en Ambients

De même que pour CCS, on peut définir une notion de graphe de transitions étiquetées pour les processus du π-calcul et définir des bisimilarités fortes et faibles. On se propose par ailleurs d'introduire une autre notion d'équivalence comportementale, appelée congruence à barbe, pour laquelle le test du processus est réalisé par un autre processus mis en interaction avec celui-ci. C'est aussi cette notion d'équivalence comportementale que l'on définit sur le calcul des Ambients.

2.2.1 Graphe de transitions étiquetées et bisimilarité

On considère ici le π-calcul monadique défini au chapitre 1, et on donne une notion de graphe de transition étiqueté pour les processus. Pour cela, on commence par définir les actions des processus. Par rapport aux capabilités, on ajoute une action d'extrusion :

$$\begin{array}{rlll} \alpha & ::= & ab & \text{réception} \\ & & \mid \overline{a}b & \text{émission} \\ & & \mid \overline{a}(b) & \text{extrusion} \\ & & \mid \tau & \text{synchronisation} \end{array}$$

et on note
- $\mathsf{bn}(\alpha)$ l'ensemble de noms défini par $\mathsf{bn}(a(b)) = \{b\}$, et $\mathsf{bn}(\alpha) = \emptyset$ sinon,
- $n(\alpha)$ l'ensemble de nom défini par $n(\tau) = \emptyset$ et $n(ab) = n(\overline{a}b) = n(\overline{a}(b)) = \{a, b\}$.

On utilise maintenant ces actions pour étiqueter les transitions d'un processus :

Définition 2.2.1 *On appelle transition*[1] *la plus petite relation $P \xrightarrow{\alpha} P'$ vérifiant :*

$$\frac{}{a(n).P \xrightarrow{ab} P\{^b/_n\}} \qquad \frac{}{\overline{a}\langle b\rangle.P \xrightarrow{\overline{a}b} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}(\mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset) \qquad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} !P \mid P'}$$

$$\frac{P \xrightarrow{\overline{a}b} P'}{(\nu b)P \xrightarrow{\overline{a}(b)} P'}(b \neq a) \qquad \frac{P \xrightarrow{\alpha} P'}{(\nu c)P \xrightarrow{\alpha} (\nu c)P'}(c \notin n(\alpha))$$

$$\frac{P \xrightarrow{ab} P' \quad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{ab} P' \quad Q \xrightarrow{\overline{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} (\nu b)\big(P' \mid Q'\big)}(b \notin \mathsf{fn}(P))$$

On peut citer les résultats suivants qui permettent de relier la vision “soupe de processus” du chapitre 1 à celle de transition étiquetée :
- si $P \equiv\xrightarrow{\alpha} P'$ alors $P =\xrightarrow{\alpha}\equiv P'$, pour $=$ l'égalité syntaxique à α-conversion près ;
- $P \longrightarrow P'$ ssi $P =\xrightarrow{\tau}\equiv P'$.

On note à nouveau $P \stackrel{\alpha}{\Longrightarrow} P'$ si $P \Rightarrow\xrightarrow{\alpha}\Rightarrow P'$. En utilisant ces transitions étiquetées, on définit les bisimilarités fortes et faibles entre processus du π-calcul de la même façon que pour CCS :

[1] On parle ici de transition *anticipée* (*early*), mais il existe aussi la transition *tardive* (*late*), cf. [SW01]

Définition 2.2.2 *La bisimilarité forte est la plus grande relation symétrique $\sim$ telle que pour tous processus P, Q, si $P \sim Q$, alors*
pour tout α, P' tels que $P \xrightarrow{\alpha} P'$, il existe Q' tel que $Q \xrightarrow{\alpha} Q'$ et $P' \sim Q'$.
La bisimilarité faible est la plus grande relation symétrique $\approx$ telle que pour tous processus P, Q, si $P \approx Q$, alors
pour tout α, P' tels que $P \xrightarrow{\alpha} P'$, il existe Q' tel que $Q \stackrel{\alpha}{\Longrightarrow} Q'$ et $P' \approx Q'$.

Les relations $\sim$ et $\approx$ sont des relations d'équivalence, congruentes pour tous les opérateurs sauf la réception[2]. Ceci permet à nouveau de faire des spécifications modulaires de systèmes distribués, les opérateurs d'assemblage de sous-systèmes étant $-|-$ et $(\nu n)-$. On donne maintenant une autre caractérisation de ces équivalences comportementales.

2.2.2 Congruence à barbes

La bisimilarité est un outil pratique pour vérifier que deux processus ont le même comportement. Toutefois, la notion d'équivalence observationnnelle est relative à celle d'un observateur. On a choisi jusque là d'observer le graphe de transition d'un processus "de l'extérieur", par son graphe de transitions étiquetées. On se propose maintenant de faire jouer à un processus testeur le rôle de l'observateur, ou en d'autres termes d'internaliser la notion d'observation dans le calcul. On doit pour cela tout d'abord définir les règles d'un jeu d'intéraction entre un processus testé et un processus observateur. Pour cela, on commence par définir les *barbes* d'un processus, c'est à dire les possibilités de synchronisation immédiates qu'il offre.

Définition 2.2.3 *Une barbe μ est soit un nom a, soit un nom conjugué $\overline{a}$. On dit que P présente la barbe forte μ, noté $P \downarrow_\mu$ si*
- *pour $\mu = a$, il existe b, P' tels que $P \xrightarrow{ab} P'$.*
- *pour $\mu = \overline{a}$, il existe b, P' tels que $P \xrightarrow{\overline{a}b} P'$ ou $P \xrightarrow{a(b)} P'$.*

On définit la relation "P présente la barbe faible μ", notée $P \Downarrow_\mu$, de la même façon pour les transitions faibles.

On utilise maintenant la notion de barbe pour définir le jeu entre les processus testeur et testé en terme de barbes :

Définition 2.2.4 *On appelle bisimilarité à barbe forte la plus grande relation symétrique $\dot{\sim}$ telle que pour tous processus P, Q, si $P \dot{\sim} Q$, on vérifie*
- *pour tout μ, si $P \downarrow_\mu$, alors $Q \downarrow_\mu$;*
- *pour tout P' tel que $P \longrightarrow P'$, il existe Q' tel que $Q \longrightarrow Q'$ et $P' \dot{\sim} Q'$.*

On définit de même la bisimilarité à barbe faible comme la plus grande relation symétrique $\dot{\approx}$ telle que pour tous processus P, Q, si $P \dot{\approx} Q$, on vérifie
- *pour tout μ, si $P \downarrow_\mu$, alors $Q \Downarrow_\mu$;*
- *pour tout P' tel que $P \longrightarrow P'$, il existe Q' tel que $Q \Rightarrow Q'$ et $P' \dot{\approx} Q'$.*

La bisimilarité à barbe définit un cadre de jeu entre processus testeur et processus observé. Deux processus observés, s'ils sont mis chacun en présence d'un

[2]En fait, les contre-exemples présentés dans [SW01] font intervenir l'opérateur de choix. La bisimilarité close par substitution, dite *entière*, est en revanche une congruence.

testeur quelconque, doivent être bisimilaires à barbes s'ils ont le même comportement observable. On définit donc l'équivalence comportementale proprement dite, appelée congruence à barbe, comme la plus grande relation de congruence sans réception incluse dans $\dot{\sim}$:

Définition 2.2.5 *On dit qu'un contexte de processus* C *est* sans réception *si le trou* [.] *n'apparaît pas sous un préfixe de réception.*
On dit que P est fortement congruent à barbe sans réception *avec Q, noté $P \simeq Q$, si $\mathsf{C}[P] \;\dot{\sim}\; \mathsf{C}[Q]$ pour tout contexte sans réception.*
On définit de même la congruence à barbe faible, notée $\cong$.

Par rapport à la bisimilarité basée sur le graphe de transition, la bisimilarité à barbe propose donc une notion d'observation du point de vue d'un processus expérimentateur, ce qui représente précisément la notion de comportement que l'on souhaite exprimer. Le résultat suivant fournit donc la justification théorique du choix de la bisimilarité comme équivalence comportementale :

Théorème 2.2.6 ([SW01]) *Pour tous processus P, Q, $P \sim Q$ ssi $P \simeq Q$.*

Ce résultat se prouve en définissant une stratification de la bisimilarité forte $\sim$: on note $\sim_i$ le i-ème approximant de la fonctionnelle dont $\sim$ est le plus grand point fixe, autrement dit $\sim_0$ est la relation totale, et $P \sim_{i+1} Q$ est la plus grande relation symétrique pour laquelle $P \xrightarrow{\alpha} P'$ implique $Q \xrightarrow{\alpha}\sim_i P'$. On pose alors

$$\sim_\omega \stackrel{\text{def}}{=} \bigcap_{i\geq 0} \sim_i$$

et on montre que $\simeq\; = \;\sim_\omega$. L'égalité entre la bisimilarité et son approximant d'ordinal ω est alors vérifiée pour un graphe de transition étiqueté dans lequel chaque noeud est à branchement fini[3]. Par la suite, on dira qu'un processus ayant un tel graphe de transition vérifie la condition de *finitude sémantique* (*image-finiteness* en anglais). Si pour le cas fort cette condition est réalisée sur tout processus, ce n'est plus vrai dans le cas faible. On ne sait donc prouver l'équivalence entre bisimilarité et congruence à barbe dans le cas faible que sous cette hypothèse.

Théorème 2.2.7 ([SW01]) *Pour tous processus P, Q vérifiant la condition de finitude sémantique, $P \approx Q$ ssi $P \cong Q$.*

Pour mieux comprendre le besoin de cette hypothèse de finitude sémantique, on peut considérer les graphes de transitions décrits par

$$a^+ \stackrel{\text{def}}{=} \sum_{n\geq 1} \underbrace{a \ldots a}_{n \text{ fois}} \quad \text{et} \quad a^\omega \stackrel{\text{def}}{=} a.\mathbf{0} + a.a^\omega).$$

Alors $a^+ \sim_\omega a^\omega$, mais $a^+ \not\sim a^\omega$.

Congruence à barbe pour les Mobile Ambients

La dynamique du calcul des Mobile Ambients se prête assez peu à une expression sous forme de τ-transition pour un système de transitions étiquetées. D'une part, les actions de mouvement n'admettent pas véritablement de coaction associée. D'autre part, un mouvement implique le transport non pas d'un nom mais d'un processus tout entier[4].

[3] on peut même étendre aux graphes à branchement finis modulo bisimilarité

[4] Pour exprimer cela, Zappa-Nardelli et Merro ont proposé une notion de système de transition étiqueté à l'ordre supérieur, dans lequel les étiquettes contiennent des processus [MN03].

Il n'est donc pas naturel de définir une équivalence comportementale entre Mobile Ambients sous forme de bisimilarité. En revanche, on peut définir une équivalence comportementale comme une congruence à barbe :

Définition 2.2.8 *Dans le calcul des Mobile Ambients, on dit que P présente la barbe (faible) n, noté $P \Downarrow_n$, si il existe P', P'' tels que $P \Rightarrow n[P'] \mid P''$. La bisimilarité à barbe est la plus grande relation symétrique $\dot{\approx}$ telle que pour tous P, Q, si $P \dot{\approx} Q$, on vérifie*
- *pour tout P' tel que $P \Rightarrow P'$, il existe Q' tel que $Q \Rightarrow Q'$ et $P' \dot{\approx} Q'$;*
- *pour tout n, si $P \Downarrow_n$, alors $Q \Downarrow_n$.*

L'équivalence comportementale entre processus est la plus grande relation $P \approx Q$ telle que pour tout contexte de processus C, $\mathsf{C}[P] \dot{\approx} \mathsf{C}[Q]$.

On a donc défini plusieurs notions d'équivalence comportementale entre processus, soit d'une façon plutôt dénotationnelle en termes de bisimilarité sur des graphes de transition, soit d'une façon plutôt opérationnelle par le jeu contextuel du processus testé avec un environnement d'exécution. Ce dernier type d'équivalence est assez peu pratique à établir de manière effective entre deux processus, mais fournit une justification théorique de la bisimilarité.

L'intérêt d'une notion d'équivalence comportementale apparaît dans les problèmes de spécification et vérification modulaire d'un système : si P est un processus qui correspond à un module d'un système plus complexe $\mathcal{S}$, et si P' et un processus qui représente la spécification de ce module, autrement dit son comportement vis-à-vis de l'extérieur sans donner de détails sur les synchronisations internes, les processus P et P' seront prouvés comportementalement équivalents, et dans la vérification du système $\mathcal{S}$, on pourra remplacer l'implémentation du module P par sa spécification P', et étudier le système simplifié $\mathcal{S}\{^{P'}/P\}$.

Dans ce cadre, P' représente une spécification *complète* de P. Dans de nombreux cas toutefois, une spécification partielle du processus est suffisante. Pour cela, on développe un langage de spécification logique dans lequel on peut énoncer des propriétés extensionnelles d'un processus sans en donner sa sémantique extensionnelle complète.

2.3 Logiques extensionnelles

On s'intéresse donc maintenant à définir un langage d'assertions fondé sur des logiques standard, et on souhaite utiliser ce langage pour décrire des propriétés extensionnelles des processus. De façon plus précise, il s'agit de définir, étant donnés un processus P et une assertion $\mathcal{A}$, la véracité de l'assertion $\mathcal{A}$ pour le processus P, notée $P \models \mathcal{A}$.

Dans un premier temps, on présente la première logique extensionnelle proposée par Hennessy-Milner [HM85] dans le cadre de CCS. Cette logique, basée sur la logique modale, caractérise précisément l'extensionnalité dans un sens que l'on détaille ici. Dans un second temps, on présente la logique extensionnelle proposée par Mads Dam dans l'un de ses travaux [Dam88]. Cette logique étend, sous certaines hypothèses la logique linéaire [Gir87], et dans une variante la logique de la relevance [Urq72], et permet un raisonnement modulaire sur les processus.

2.3.1 Logiques modales de Hennessy-Milner

On définit donc tout d'abord la logique modale de Hennessy-Milner. C'est une extension de la logique classique avec pour chaque action α une modalité $\langle\alpha\rangle$ ou $\langle\langle\alpha\rangle\rangle$ suivant la sémantique, forte ou faible, considérée. Ainsi les assertions, ou formules, sont données par la grammaire :

$$\begin{array}{rcl} \mathcal{A} & ::= & \top \;\big|\; \mathcal{A}_1 \wedge \mathcal{A}_2 \;\big|\; \neg\mathcal{A} \\ & \big| & \langle\alpha\rangle\mathcal{A} \quad \text{(modalité forte)} \\ & \big| & \langle\langle\alpha\rangle\rangle\mathcal{A} \quad \text{(modalité faible)} \end{array}$$

La relation de satisfaction $P \models \mathcal{A}$ est définie de façon naturelle par l'interprétation du graphe de transitions étiquetées de P comme un modèle de Kripke :

Définition 2.3.1 $\models$ *est la plus petite relation telle que :*
- $P \models \top$ *pour tout processus* P *;*
- $P \models \mathcal{A}_1 \wedge \mathcal{A}_2$ *si* $P \models \mathcal{A}_1$ *et* $P \models \mathcal{A}_2$ *;*
- $P \models \neg\mathcal{A}$ *si* $P \not\models \mathcal{A}$.
- $P \models \langle\alpha\rangle\mathcal{A}$ *(resp.* $P \models \langle\langle\alpha\rangle\rangle\mathcal{A}$*) si il existe* P' *tel que* $P \xrightarrow{\alpha} P'$ *et* $P' \models \mathcal{A}$ *(resp. il existe* P' *tel que* $P \overset{\alpha}{\Longrightarrow} P'$ *et* $P' \models \mathcal{A}$*).*

$\langle\alpha\rangle\mathcal{A}$ est dite modalité de possibilité ; on peut par ailleurs définir par dualité la modalité de nécessité $[\alpha]\mathcal{A}$, telle que $P \models [\alpha]\mathcal{A}$ ssi pour tout P' tel que $P \xrightarrow{\alpha} P'$, $P' \models \mathcal{A}$. Les formules sont des spécifications partielles du comportement. En effet, la formule $\langle\alpha\rangle\top$ est satisfaite par les processus $\alpha.\mathbf{0}$ et $\alpha.\mathbf{0}|\beta.\mathbf{0}$, qui, bien qu'ayant une barbe commune, n'en ont pas moins des comportements différents.

On note $P =_L Q$ si les processus P, Q vérifient exactement les mêmes formules ; on dira que P, Q sont logiquement équivalents.

On vérifie immédiatement que cette logique est extensionnelle au sens de la propriété de correction suivante :

Proposition 2.3.2 $\sim$ *(resp.* $\approx$*) induit* $=_L$ *pour l'équivalence logique forte (resp. faible), autrement dit si* $P \sim Q$ *et* $P \models \mathcal{A}$*, alors* $Q \models \mathcal{A}$*.*

En effet, la satisfaction $\models$ est définie à partir du graphe de transition de P, qui est le même que celui Q.

On peut se poser la question de l'implication inverse, soit la complétude de la bisimilarité pour la logique. Ce résultat n'est vrai que pour l'approximant d'ordre ω, noté précédemment $\sim_\omega$, et on a donc la complétude modulo une hypothèse de finitude sémantique [GS86] ; une autre complétude est possible si l'on se donne une conjonction généralisée $\bigwedge_{i\in I} \mathcal{A}_i$ [SI94].

Une autre technique pour établir la complétude de la bisimilarité consiste à définir la *formule caractéristique* $\mathcal{F}_P$ d'un processus P, sous certaines conditions sur P. Cette formule caractérise exactement les processus bisimilaires à P, et fournit donc la preuve que les processus logiquement équivalents sont bisimilaires. Par correction, elle possède finalement la propriété suivante :

$$\forall Q, \qquad Q \models \mathcal{F}_P \quad \text{ssi} \quad Q =_L P \quad \text{ssi} \quad P \sim Q$$

La logique de Hennessy-Milner est donc purement extensionnelle, et permet de donner une spécification partielle des processus qu'elle considère. Bien que simple, son expressivité est relativement conséquente, comme le montre par exemple l'encodage d'une logique à la Hoare pour le langage de programmation impératif basé sur CCS (voir [Mil89]). De nombreux outils automatiques permettent de vérifier en pratique les spécifications dans ces logiques [VM94].

En revanche, cette logique ne permet pas très clairement de développer une spécification modulaire de processus. Une idée élégante a été proposée par Mads Dam pour permettre le raisonnement modulaire dans une logique extensionnelle.

2.3.2 Les travaux de Mads Dam

Supposons que l'on étudie le système $P|Q$, dans lequel Q est un module qui satisfait une spécification $\mathcal{A}$, et on veut montrer que $P|Q$ satisfait la spécification $\mathcal{B}$. Quelle spécification doit-on établir sur P pour cela ? Et comment composer ces deux spécifications pour obtenir $\mathcal{B}$?

Une solution est de voir P comme une fonction, ou plutôt un serveur, qui en présence d'un client satisfaisant la spécification $\mathcal{A}$ produit un système qui vérifie $\mathcal{B}$. Dans cette idée-là, Mads Dam introduit le connecteur $\multimap$ défini par

$$P \models \mathcal{A} \multimap \mathcal{B} \quad \text{si} \quad \text{pour tout } Q, \text{ si } Q \models \mathcal{A} \text{ alors } P \times Q \models \mathcal{B}$$

où $\times$ est la composition de processus, que l'on a notée $|$ auparavant[5]. Afin d'obtenir un support à la spécification modulaire, Dam considère donc un tel connecteur $\multimap$. Par adjonction, il définit le connecteur $\circ$, qu'il appelle fusion, pour lequel

$$\mathcal{A}_1 \multimap \mathcal{A}_2 \multimap \mathcal{B} \quad = \quad (\mathcal{A}_1 \circ \mathcal{A}_2) \multimap \mathcal{B}.$$

De ce fait, il obtient la définition suivante pour la fusion

$$P \models \mathcal{A} \circ \mathcal{B} \quad \text{si} \quad \text{il existe } P_1, P_2 \text{ t.q. } P \sim P_1|P_2,\ P_1 \models \mathcal{A}_1 \text{ et } P_2 \models \mathcal{A}_2.$$

Pour fonder sa logique, il remarque que l'algèbre de processus qu'il considère, ordonnée par l'ordre de simulation [Dam88], a une structure de *frame*[6], ce qui fournit un modèle de la logique linéaire positive LL^+ [Gir87], et sous certaines hypothèses de la logique de la relevance positive RL^+ [Urq72]. Sa logique prend en compte des modalités d'action à la Hennessy-Milner, et de ce fait est pleinement extensionnelle, au sens ou l'on retrouve la propriété

$$P =_L Q \quad \text{ssi} \quad P \sim Q.$$

Les idées de Mads Dam sont extrêmement intéressantes et mériteraient sans doute une étude plus approfondie. Plus ou moins consciemment, les logiques spatiales que l'on va voir au chapitre suivant se sont inspirées de ces idées.

On peut citer au moins deux critiques quant aux travaux de Mads Dam. D'une part, l'algèbre de processus utilisée pour définir la logique est assez spécifique, et relativement différente de CCS et du π-calcul, ce qui rend les constructions de [Dam88] difficilement adaptables à ces dernières algèbres de processus. D'autre part, le connecteur $\circ$ suggère que le processus que l'on considère est un composé, ce qui est une notion plutôt intensionnelle, et on est forcé de

[5] Cette notation est celle utilisée par Milner et reprise par Dam, elle représente la composition dans le calcul CCS synchrone [Mil89, Dam88]

[6] $\times$ est la composition compatible avec l'ordre, et $+$ la borne inférieure.

“rattraper” cette intensionalité en définissant la logique modulo $\sim$, alors qu'à ce stade il pourrait sembler plus naturel de la définir modulo $\equiv$[7].

Le pas de l'intensionalité assumée sera franchi quelques années plus tard par Luis Caires [Cai99], puis Cardelli et Gordon [CG00], et donnera lieu aux logiques spatiales. On retiendra que de nombreuses idées originales des logiques spatiales sont présentes dans les travaux de Dam[8].

[7] C'est l'approche choisie dans [Win85]. Pour préserver l'extensionalité de la logique, l'utilisation de $\circ$ est toutefois restreinte aux formules gauches pour $\multimap$.

[8] En particulier, on notera dans [Dam88] la mention d'un résultat d'élimination de $\multimap$ que l'on retrouvera de façon un peu plus précise au chapitre 8.

Chapitre 3

Logiques spatiales

L'approche tarskienne de l'étude de la logique (classique) est de comprendre une catégorie d'objets mathématiques en considérant le formalisme nécessaire à sa description. La logique, plus qu'un cadre de raisonnement, est dans cette approche l'outil descriptif d'une structure.

La notion de structure est suffisamment vague pour fourmiller d'instanciations possibles, en particulier dans le domaine informatique. Dans la suite, nous prendrons des exemples dans les réseaux de communication, les bases de données, ou encore le vraie et le tas de l'espace mémoire d'un programme. Pour abstraire, on pourrait parler d'une façon générale d'*espace de ressources*. Un espace de ressources est un peu comme un ensemble d'objets (agents, champs de données, cases mémoires), mais peut contenir plusieurs fois le même objet sans savoir distinguer deux copies ; on est donc plus proche d'un multiensemble. Il existe aussi des relations entre ces objets, hiérarchiques, topologiques, éventuellement temporelles. Il serait arbitraire de chercher à donner une définition formelle de ce que sont les espaces de ressources prenant en compte les différents exemples que l'on développera par la suite, mais on retiendra ces aspects[1].

Les relations entre les objets d'une structure sont parfaitement étudiées par la logique classique avec prédicats. Néanmoins, afin de faciliter le raisonnement, on peut chercher à isoler certaines ressources d'une partie de leur contexte, celui-ci "polluant" les observations que l'on souhaite faire sur la ressource. C'est la notion de *raisonnement localisé* qui est le point de départ de la logique spatiale. Afin d'exploiter un raisonnement localisé, la logique doit aussi inclure une forme d'*importation d'hypothèses* locales à l'échelle globale. La logique spatiale développe une extension de la logique classique qui répond à ces besoins. En plus de la conjonction classique, elle dispose d'une conjonction séparante : la structure satisfait "A et séparément B" si elle est la réunion de deux sous-structures, l'une satisfaisant A et l'autre B [2]. En plus de l'implication classique, elle dispose d'une implication spatiale qui traduit l'introduction spatiale d'hypothèses : une

[1] Le lecteur intéressé regardera les modèles proposés par O'Hearn et Pym pour la bunched logic [POY04]

[2] On notera que cette forme de conjonction met en défaut le principe de non contradiction selon lequel "A et non A" est impossible. En effet, il est tout à fait cohérent d'affirmer "A est vrai (ici) et séparément A est faux (là)". Ce type de phénomène est à mettre en relation avec les recherches menées essentiellement par les philosophes sur les logiques "paraconsistantes"

structure satisfait “A implique spatialement B” si lorsque j'étends ma structure par une autre structure qui satisfait A, la structure étendue résultante satisfait B.

On présente maintenant les différentes logiques spatiales étudiées par la suite.

3.1 Logique séparante

3.1.1 Motivations

Suite aux travaux de Tony Hoare, l'approche classique de la preuve de programmes impératifs est basée sur les notions de précondition et postcondition, et de programme comme transformateur d'état. Le triplet de Hoare $\{Pre\}\ \mathsf{P}\ \{Post\}$ signifie ainsi que pour tout état initial σ qui vérifie la précondition Pre, l'état final σ' après exécution du programme P satisfait la postcondition $Post$. Dans le cas le plus simple de langage de programmation, l'instruction d'affectation admet une axiomatisation élémentaire à l'aide du tripet

$$\big\{A[e/\mathsf{x}]\big\}\ \ \mathsf{x} := e\ \ \big\{A\big\}$$

appelé raisonnement arrière (*backward reasoning*).

Cette axiomatisation n'est plus valable lorsque le langage de programmation permet de manipuler des pointeurs, puisqu'alors d'autres variables que x peuvent dénoter la même adresse mémoire. Le problème de l'axiomatisation des langages à pointeurs est resté ouvert pendant plusieurs décennies, et on peut dire qu'une solution satisfaisante est actuellement élaborée par Reynolds et O'Hearn [Rey02]. Leur observation essentielle est que la logique classique n'est pas suffisante comme langage d'assertion dans lequel formuler les pré/postconditions, pour raisonner localement sur la partie de la mémoire à considérer. Ils développent donc la logique séparante (*separation logic*) avec laquelle ils étudient un langage impératif canonique incluant des pointeurs. Pour ne citer qu'un des nombreux exemples du bénéfice que l'on tire de ce langage d'assertion, l'axiomatisation de l'affectation d'un pointeur s'écrit alors (voir plus bas exemple 3.1.3)

$$\big\{\mathsf{x} \mapsto -\ *\ (\mathsf{x} \mapsto e\ \ -\!\!*\ \ A)\big\}\ \ [\mathsf{x}] := e\ \ \big\{A\big\}$$

faisant de la substitution implicite précédente une substitution explicite par retrait puis rajout dans la mémoire de la cellule indexée par x (et éventuellement d'autres variables). Sur cet exemple que l'on reprendra bientôt, $*$ est la conjonction séparante, $-\!\!*$ l'implication spatiale, et $[\mathsf{x}] := e$ l'affectation de la valeur dénotée par l'expression e à la case mémoire pointée depuis x.

Dans une preuve de programme, appliquer la règle de consistance à un jugement $\{A\}\ P\ \{\mathcal{B}\}$ consiste à laisser de coté les assertions portant sur la partie mémoire à laquelle le programme n'accède pas. Dans un langage sans pointeur, déterminer quelle est cette partie mémoire inutilisée revient simplement à considérer quelles sont les variables mentionnées par le programme P. Cette observation mène à la règle suivante

$$\frac{\{\phi\}\ c\ \{\psi\} \quad \mathsf{fv}(H) \cap \mathsf{fv}(c) = \emptyset}{\{\phi \wedge H\}\ c\ \{\psi \wedge H\}}$$

Pour un langage avec pointeurs, il est plus difficile de localiser la partie mémoire accessible par le programme et la formule logique, en particulier on ne sait pas a priori si les adresses accédées par les variables de H ne sont pas déjà accédées par d'autres variables dans ϕ. A nouveau, la logique séparante propose une reformulation de cette règle à l'aide de la conjonction séparante :

$$\frac{\{\phi\}\ c\ \{\psi\} \quad \mathsf{fv}(H) \cap \mathsf{fv}(c) = \emptyset}{\{\phi * H\}\ c\ \{\psi * H\}}$$

Pour d'autres exemples et une définition précise du langage de programmation considéré, on se référera à l'article introductif de Reynolds [Rey02].

3.1.2 Définitions

On va maintenant définir formellement la structure d'espace mémoire. Un espace mémoire comporte deux parties distinctes, appelées pile et tas. La pile est allouée initialement au moment de la déclaration des variables au sein de la routine en cours d'exécution, tandis que le tas est alloué dynamiquement par l'exécution du programme. Pour nous, la pile correspond donc à la relation qui existe entre une variable du programme et son adresse mémoire, et le tas est un ensemble modifiable de cellules mémoire contenant des valeurs. Les valeurs elles-mêmes peuvent être des résultats de calcul (entier, booléen, réel) ou des adresses d'autres cases mémoire (pointeurs).

Définition 3.1.1 (pile, tas, état mémoire) *Soit* VProg *l'ensemble des variables de programmation,* Addr *l'ensemble (infini) des adresses mémoire, et* Val *l'ensemble des valeurs, que l'on désignera par les lettres*[3] $\mu, \nu, \ldots$ *. avec en particulier* $\mathsf{Addr} \subseteq \mathbb{N} \subseteq \mathsf{Val}$*, ainsi qu'une valeur particulière* nil*. On appelle* pile *une fonction* $s : \mathsf{VProg} \to \mathsf{Addr}$*, et* pile *une fonction partielle* $h : \mathsf{Addr} \rightharpoonup_{fin} \mathsf{Val}$ *de domaine fini. Un* état *est un couple* (s, h)*.*

Si deux tas h, h' sont de domaines disjoints ($\mathsf{dom}(h) \cap \mathsf{dom}(h)' = \emptyset$), ce que l'on notera $h \perp h'$, on considérera la pile réunion $h * h'$, de domaine $\mathsf{dom}(h) \cup \mathsf{dom}(h)'$, et défini par prolongement de h et h'. On notera aussi $a \mapsto \nu$ le tas h de domaine $\{a\}$ définie par $h(a) = \nu$.

La logique séparante est le langage d'assertion basé sur ce modèle de mémoire dont les formules sont définies par la grammaire suivante :

$$\begin{array}{lcl} e & ::= & \mathsf{x} \mid i \mid \mathsf{nil} \mid e + e \\ \mathcal{A} & ::= & \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \mathsf{emp} \mid \mathcal{A} * \mathcal{A} \mid \mathcal{A} -\!\!* \mathcal{A} \mid e \mapsto e \mid \exists i.\mathcal{A} \end{array}$$

où i désigne une variable logique, prise dans un ensemble infini VLog disjoint de VProg. Les *expressions* e forment l'ensemble Exp et les *formules* - ou *assertions* - $\mathcal{A}$ forment l'ensemble $\mathcal{L}^{\exists}_{Sep}$. Pour ces assertions, $\mathcal{A}_1 * \mathcal{A}_2$ représente la conjonction séparante discutée précédemment, $\mathcal{A}_1 -\!\!* \mathcal{A}_2$ représente l'implication spatiale, $\mathcal{A}_1 \wedge \mathcal{A}_2$ et $\neg\mathcal{A}_1$ reprennent les connecteurs de la logique classique, emp représente l'espace mémoire vide, et $x \mapsto e$ représente une cellule mémoire indexée par x. Plus formellement, on définit la relation de satisfaction $s, h \models_v \mathcal{A}$ entre un état (s, h) et une assertion $\mathcal{A}$ sous la valuation $v : \mathsf{VLog} \to \mathsf{Addr}$ ainsi :

[3] Attention ! On notera v pour une valuation.

Définition 3.1.2 (Satisfaction en $\mathcal{L}^{\exists}_{Sep}$) *On note $s, h \models_v \mathcal{A}$ la relation définie par induction sur $\mathcal{A}$ comme suit :*

$$
\begin{array}{lll}
s,h \models_v & \mathcal{A}_1 \wedge \mathcal{A}_2 & si \;\; s,h \models_v \mathcal{A}_1 \;\; et \; s,h \models_v \mathcal{A}_2 \\
s,h \models_v & \neg\, \mathcal{A} & si \;\; s,h \not\models_v \mathcal{A} \\
s,h \models_v & \mathsf{emp} & si \;\; \mathsf{dom}(h) = \emptyset \\
s,h \models_v & \mathcal{A}_1 * \mathcal{A}_2 & si\ il\ existe \;\; h_1, h_2 \; tels\ que\ h = h_1 * h_2, \;\; s,h_1 \models_v \mathcal{A}_1 \;\; et\ s,h_2 \models_v \mathcal{A}_2 \\
s,h \models_v & \mathcal{A}_1 \mathrel{-\!\!*} \mathcal{A}_2 & si\ pour\ tout\ h', \;\; h' \perp h \;\; et\ s,h' \models_v \mathcal{A}_1 \;\; implique\ s, h * h' \models_v \mathcal{A}_2 \\
s,h \models_v & e \mapsto e' & si\ a = [\![e]\!](s,v) \in \mathsf{Addr}, \;\; \mathsf{dom}(h) = \{a\}, \;\; et\ h(a) = [\![e']\!](s,v) \\
s,h \models_v & \exists i.\mathcal{A} & si\ il\ existe\ \nu \in \mathsf{Val}\ tel\ que\ s,h \models_{v,i \mapsto \nu} \mathcal{A}
\end{array}
$$

avec $[\![\mathsf{x}]\!](s,v) = s(\mathsf{x})$, $[\![i]\!](s,v) = v(i)$, $[\![\mathsf{nil}]\!](s,v) = \mathsf{nil}$ et $[\![e+e']\!](s,v) = [\![e]\!](s,v) + [\![e']\!](s,v)$. On notera $s,h \models \mathcal{A}$ si pour toute valuation v, $s,h \models_v \mathcal{A}$.

On notera $\mathcal{L}^{\exists}_{Sep}$ l'ensemble des assertions ainsi définies, et $\mathcal{L}_{Sep}$ le sous-ensemble des assertions sans quantificateur.
Convention d'écriture : on omettra parfois certaines parenthèses et on adoptera la règle de précédence syntaxique suivante : les connecteurs classiques sont considérés comme les moins liants, puis viennent les connecteurs spatiaux, puis (éventuellement) les autres connecteurs. Ainsi la formule $\mathcal{A}_1 * \mathcal{A}_2 \wedge \mathcal{A}_3$ doit se lire $(\mathcal{A}_1 * \mathcal{A}_2) \wedge \mathcal{A}_3$. Pour les connecteurs $\rightarrow$ et $\mathrel{-\!\!*}$, on adopte la convention habituelle de noter $\mathcal{A}_1 \rhd \mathcal{A}_2 \rhd \mathcal{A}_3$ pour $\mathcal{A}_1 \rhd (\mathcal{A}_2 \rhd \mathcal{A}_3)$ (on reprendra les mêmes conventions pour les autres logiques spatiales). Enfin on adopte la notation $\mathsf{x} \mapsto -$ pour la formule $\exists i.\ \mathsf{x} \mapsto i$.

Exemple 3.1.3 *Reprenons l'axiomatisation de l'affectation citée précédemment*

$$\{\exists i.\mathsf{x} \mapsto i \;*\; (\mathsf{x} \mapsto e \;\mathrel{-\!\!*}\; A)\} \;\; [\mathsf{x}] := e \;\; \{A\}$$

L'instruction $[\mathsf{x}] := e$ s'exécute dans état mémoire (s,h) pour lequel l'adresse pointée par x est déjà alloué, et le transforme en l'état (s,h') dans lequel la valeur à cette adresse a été remplacée par l'évaluation de e. Formellement, pour $s(x) = a$ et $[\![e]\!](s) = \nu$, on a $\mathsf{dom}(h) = \mathsf{dom}(h)'$, $a \in \mathsf{dom}(h)$, $h'(a) = \nu$, et $h'(a') = h(a')$ pour tout $a' \in \mathsf{dom}(h) - \{s(\mathsf{x})\}$. Plus précisément, il existe h'' et μ tels que $h = h'' * a \mapsto \mu$, $h' = h'' * a \mapsto \nu$.

On peut vérifier que la formule de gauche dans le triplet de Hoare précédent définit bien la plus faible précondition. En effet elle est satisfaite exactement par les tas h telles que la restriction de h à son domaine privé de $s(\mathsf{x})$, étendue à nouveau à $s(\mathsf{x})$ avec la nouvelle valeur ν, satisfait $\mathcal{A}$. Or, après cette restriction suivie d'une extension, on a exactement le tas tel qu'il se trouve après l'affectation. Soit en effet (s,h) tel que $s,h \models \mathsf{x} \mapsto - \;*\; (\mathsf{x} \mapsto e \;\mathrel{-\!\!*}\; A)$. Alors la pile h se décompose en deux sous parties $h = h_1 * h_2$, avec d'une part $s,h_1 \models \exists i.\mathsf{x} \mapsto i$ et d'autre part $s,h_2 \models \mathsf{x} \mapsto e \;\mathrel{-\!\!*}\; A$. Donc $h_1 = a \mapsto v_0$ pour un certain v_0, et h_2 vérifie que pour tout $h_3 \perp h_2$ tel que $h_3 \models \mathsf{x} \mapsto e$, $h' = h_2 * h_3$ satisfait $\mathcal{A}$. Or h_3 est nécessairement $a \mapsto v$, autrement dit h' est bien l'état final après affectation, et il vérifie $\mathcal{A}$.

3.2 Logiques spatiales pour les algèbres de processus

En logique séparante, la sémantique des connecteurs spatiaux est basée sur la notion de composition d'espaces mémoire. On peut donc définir de même une logique spatiale pour tout autre famille d'objets pour lesquels on dispose d'une opération de composition spatiale. Dans le cas des algèbres de processus, la bonne notion de composition spatiale est la mise en parallèle de deux systèmes $P|Q$. On notera donc simplement $\mathcal{A}|\mathcal{B}$ la conjonction séparante pour les logiques associées aux algèbres de processus, $\mathcal{A} \triangleright \mathcal{B}$ pour l'implication spatiale, et 0 pour le processus neutre.

3.2.1 Logique spatiale pour CCS

La logique spatiale que l'on considère pour CCS est l'extension spatiale de la logique modale de Hennessy-Milner : en plus des connecteurs classiques, elle comporte une modalité temporelle $\Diamond\mathcal{A}$ qui exprime la possibilité de réduction, et des modalités d'actions. Les formules considérées sont générées par la grammaire :

$$\begin{array}{llr} \mathcal{A} ::= & \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \mathcal{A}|\mathcal{A} \mid \mathcal{A} \triangleright \mathcal{A} \mid 0 \mid \Diamond\mathcal{A} & \text{(logique spatiale dynamique)} \\ & \mid \exists x.\mathcal{A} \mid \langle x\rangle\mathcal{A} \mid \langle\overline{x}\rangle\mathcal{A} & \text{(extension modale)} \end{array}$$

où x est une variable logique prise dans un ensemble infini VLog. On notera $\mathcal{L}_{mod}^{(CCS)}$ l'ensemble de toutes les formules que l'on peut définir ainsi, et $\mathcal{L}_{spat}^{\Diamond}$ le fragment purement spatio-temporel. Comme pour la logique séparante, la sémantique de ces formules est donnée par la relation de satisfaction définie maintenant :

Définition 3.2.1 (Satisfaction en $\mathcal{L}_{mod}^{(CCS)}$) *Soit M un ensemble de processus de CCS. La relation $P, v \models_M \mathcal{A}$ entre un processus $P \in$ CCS, une valuation $v : \mathsf{VLog} \rightarrow \mathsf{Act}$ et une formule $\mathcal{A} \in \mathcal{L}_{mod}^{(CCS)}$ est définie par induction sur $\mathcal{A}$ par*

$$\begin{array}{llll}
P, v & \models_M & \mathcal{A}_1 \wedge \mathcal{A}_2 & si \quad P, v \models_M \mathcal{A}_1 \text{ et } P, v \models_M \mathcal{A}_2 \\
P, v & \models_M & \neg\mathcal{A} & si \quad P, v \not\models_M \mathcal{A} \\
P, v & \models_M & \mathcal{A}_1|\mathcal{A}_2 & si \quad \exists P_1, P_2.\ P \equiv P_1|P_2 \text{ et } P_i, v \models_M \mathcal{A}_i,\ i = 1, 2 \\
P, v & \models_M & \mathcal{A}_1 \triangleright \mathcal{A}_2 & si \quad \forall Q \in M,\ Q, v \models_M \mathcal{A}_1 \text{ implique } P|Q, v \models_M \mathcal{A}_2 \\
P, v & \models_M & 0 & si \quad P \equiv \mathbf{0} \\
P, v & \models_M & \Diamond\mathcal{A} & si \quad \exists P'.\ P \longrightarrow P' \text{ et } P', v \models_M \mathcal{A} \\
P, v & \models_M & \exists x.\mathcal{A} & si \quad \exists \alpha.\ P, (v, x \mapsto \alpha) \models_M \mathcal{A} \\
P, v & \models_M & \langle x\rangle.\mathcal{A} & si \quad \exists P'.\ P \xrightarrow{v(x)} P' \text{ et } P', v \models_M \mathcal{A} \\
P, v & \models_M & \langle\overline{x}\rangle.\mathcal{A} & si \quad \exists P'.\ P \xrightarrow{\overline{v(x)}} P' \text{ et } P', v \models_M \mathcal{A}
\end{array}$$

On remarquera que, par rapport aux logiques extensionnelles proposées au chapitre précédent, cette logique requiert pour sa définition de considérer les termes à congruence structurelle près, et non plus leur graphe de transitions étiquetées.

Remarque : le rôle du paramètre M apparaîtra plus clairement au chapitre 8 et reste spécifique à cette logique spatiale. On va voir qu'en fait on instancie en général M par le calcul considéré tout entier.

3.2.2 Ensemble nominal, quantification canonique

A l'origine, les logiques spatiales introduites pour les algèbres de processus ont porté sur le π-calcul et sur les Ambients [CG00, CC01]. Par rapport à la logique $\mathcal{L}_{mod}^{(CCS)}$, ces logiques spatiales comportent des *connecteurs spécifiques* qui restreignent les modèles que l'on peut considérer pour les formules à des processus de l'un de ces calculs. Une préoccupation naturelle est donc l'unification de ces logiques en une logique spatiale dynamique indépendante du calcul sous-jacent. Cela permet de voir les logiques spécifiques comme des extensions de la logique unifiée, et le cas échéant équivalentes.

On va donc définir une notion générale d'algèbre de processus pour laquelle on définira de façon systématique une logique spatiale. Le problème essentiel est de pouvoir parler de la restriction de nom $(\nu n)P$ dans cette abstraction. On détaille maintenant ce point.

On considère donc un ensemble infini de noms Na, et on appelle *ensemble nominal* [4] tout ensemble E muni d'une action du groupe de permutations $\mathcal{S}(\mathsf{Na})$, notée $t\sigma$, pour $t \in E$ et $\sigma \in \mathcal{S}(\mathsf{Na})$. On note alors $\mathsf{fn}(t)$ l'ensemble des noms n pour lesquels l'ensemble

$$\mathsf{modif}(n,t) \stackrel{\text{def}}{=} \{m \in \mathsf{Na} : \ t(n \leftrightarrow m) \neq t\}$$

est infini. On a ainsi par exemple :

- si $n, m \notin \mathsf{fn}(t)$, alors $t(n \leftrightarrow m) = t$:
  en effet, $\mathsf{Na} - \mathsf{modif}(n,t)$ et $\mathsf{Na} - \mathsf{modif}(m,t)$ sont cofinis, donc il existe n' tel que $P(n \leftrightarrow n') = P(m \leftrightarrow n') = P$. On écrit alors $P(n \leftrightarrow m) = P(n \leftrightarrow n')(m \leftrightarrow n')(n \leftrightarrow n') = P$.
- si $n \in \mathsf{fn}(t)$ et $m \notin \mathsf{fn}(t)$, alors $t(n \leftrightarrow m) \neq t$:
  en effet, il existe $n' \in \mathsf{modif}(n,t) - \mathsf{modif}(m,t)$, et on a alors $t(n \leftrightarrow m) = t(m \leftrightarrow n')(n \leftrightarrow n')(m \leftrightarrow n') = t(m \leftrightarrow n')(n \leftrightarrow n') \neq t(m \leftrightarrow n') = t$.

Exemple 3.2.2 *Soit P un processus du π-calcul, σ une permutation de noms, $P\sigma$ le processus où tout nom a est remplacé par $\sigma(a)$. Alors si a est libre dans P,* $\mathsf{modif}(a,P)$ *contient tous les noms frais pour P, donc est infini. Réciproquement, si a est frais ou lié dans P, tous les noms frais pour P ne sont pas dans* $\mathsf{modif}(a,P)$*, donc* $\mathsf{modif}(a,P)$ *est fini. On a donc bien*

$$n \in \mathsf{fn}(P) \quad \textit{ssi} \quad n \textit{ est libre dans } P$$

Une fonction $f : A \to B$ est dite *équivariante* si pour toute permutation σ et tout élément $P \in A$, $f(P\sigma) = \big(f(P)\big)\sigma$. Les termes t étant appelés à être des objets syntaxiques, on fait l'hypothèse que $\mathsf{fn}(t)$ est fini.

On considère maintenant deux ensembles nominaux $\mathcal{P}$ et $\mathcal{F}$ et une relation équivariante $P \models \mathcal{A}$ définie sur $\mathcal{P} \times \mathcal{F}$. Soit n un nom ; les conditions suivantes sont équivalentes :

1. pour tout $m \in \mathsf{Na} - \mathsf{fn}(P) - \mathsf{fn}(\mathcal{A})$, $P \models \mathcal{A}(n \leftrightarrow m)$
2. il existe $m \in \mathsf{Na} - \mathsf{fn}(P) - \mathsf{fn}(\mathcal{A})$ tel que $P \models \mathcal{A}(n \leftrightarrow m)$

On notera $P \models И n.\mathcal{A}$ une telle situation. $И$ est appelé *quantificateur générique*, au sens où la quantification se fait sur l'ensemble des noms non particularisés.

Précisons le lien qui existe avec les formes de quantification classique. $\mathcal{A}$ étant une formule logique, on définit

$$\begin{array}{lll} P \models \forall n.\mathcal{A} & \text{si} & \text{pour tout } n' \in \mathsf{Na},\ P \models \mathcal{A}\{^n/n'\} \\ P \models \exists n.\mathcal{A} & \text{si} & \text{il existe } n' \in \mathsf{Na}.\ P \models \mathcal{A}\{^n/n'\} \end{array}$$

[4] Selon la terminologie proposée par M.J Gabbay et A.M Pitts [GP99].

où $\mathcal{A}\{^n/n'\}$ est la formule obtenue en remplaçant toutes les occurrences de n dans $\mathcal{A}$ par n'. Précisons que si n' n'est pas un nom libre de $\mathcal{A}$, on aura $\mathcal{A}\{^n/n'\} = \mathcal{A}(n \leftrightarrow n')$.

On a donc les relations suivantes entre les différentes formes de quantification :

$$P \models \forall n.\mathcal{A} \quad \Rightarrow \quad P \models \text{И} n.\mathcal{A} \quad \Rightarrow \quad P \models \exists n.\mathcal{A}$$

De plus, on peut remarquer que le quantificateur générique est son propre dual, autrement dit

$$\text{И} n.\neg\mathcal{A} \quad \dashv\vdash \quad \neg\text{И} n.\mathcal{A}.$$

On vérifie cette relation en utilisant les deux définitions alternatives de $\text{И} n.\mathcal{A}$. Pour de plus amples détails sur la quantification générique on pourra se référer aux travaux de M.J. Gabbay et A.M. Pitts [GP99]. Une autre notion de quantification générique, fondée sur la théorie de la démonstration, est développée parallèlement dans [MT03].

3.2.3 Logique spatiale unifiée, logique spatiale du π-calcul

La quantification générique va nous être utile pour définir une logique spatiale dans laquelle on représentera la notion de nom lié n dans le processus $(\nu n)P$. On donne donc une définition générale des algèbres de processus pour laquelle on définit de façon systématique une logique[5]. On appliquera ensuite cette définition à la fois au cas du π-calcul et au cas des Ambients.

Définition 3.2.3 (Calcul spatial) *Un calcul spatial est un tuple* $\mathcal{M} = (M; |; \mathbf{0}; (\nu.); \equiv ; \rightsquigarrow)$ *tel que*

- M *est un ensemble nominal dont les éléments sont notés* P, Q
- $| : M^2 \to M$ *est un connecteur binaire représentant la composition parallèle,*
- $\mathbf{0} : M$ *est une constante représentant le processus neutre,*
- $(\nu.) : \mathsf{Na} \times M \to M$ *donne pour chaque nom* n *un connecteur unaire noté* $(\nu n)P$*.*
- $\equiv$ *est une relation de congruence équivariante satisfaisant les axiomes suivants :*

$$\begin{array}{rcll}
(\nu n)P & \equiv & (\nu m)\big(P(n \leftrightarrow m)\big) \quad \big(m \notin \mathsf{fn}(P)\big) & (\alpha - equivalence) \\
P|\mathbf{0} & \equiv & P & (neutre) \\
P|\big(Q|R\big) & \equiv & \big(P|Q\big)|R & (associativit\acute{e}) \\
P|Q & \equiv & Q|P & (commutativit\acute{e}) \\
(\nu n)\big(P|Q\big) & \equiv & \big((\nu n)P\big)|Q \quad \big(n \notin \mathsf{fn}(Q)\big) & (extrusion\ de\ nom)
\end{array}$$

- $\rightsquigarrow$ *est une relation binaire équivariante.*

Il est clair que la définition de calcul spatial recouvre à la fois le π-calcul, le calcul des ambients avec noms restreints, les ambients statiques, et de nombreuses variantes de ces calculs. Il est à noter que l'on n'a fait aucune hypothèse particulière pour la relation de réduction , et on pourra considérer pour $\rightsquigarrow$ la réduction en un pas $\longrightarrow$ (*sens fort*) ou sa clôture réflexive-transitive $\Rightarrow$ (*sens faible*).

[5]Une telle approche d'abstraction des algèbres de processus pour avoir une notion de modèle générale pour une logique spatiale a aussi été développée par Luis Caires dans [Cai00].

La logique spatiale unifiée est définie par la grammaire de formules

$$\begin{array}{lll} \mathcal{A} & ::= & \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \mathcal{A}|\mathcal{A} \mid \mathcal{A} \triangleright \mathcal{A} \mid 0 \mid \Diamond\mathcal{A} \quad \text{(logique spatiale dynamique)} \\ & & \mid \text{И}n.\mathcal{A} \mid n®\mathcal{A} \quad \text{(extension nominale)} \end{array}$$

et la notion de satisfaction suivante :

Définition 3.2.4 (Satisfaction en $\mathcal{L}_U$) *Soit $\mathcal{M}$ un calcul spatial.*

La relation $P \models \mathcal{A}$ entre un processus $P \in \mathcal{M}$, et une formule $\mathcal{A} \in \mathcal{L}_U$ est définie par induction sur $\mathcal{A}$ par

$$\begin{array}{llll}
P & \models & \mathcal{A}_1 \wedge \mathcal{A}_2 & \textit{si} \quad P \models \mathcal{A}_1 \textit{ et } P \models \mathcal{A}_2 \\
P & \models & \mathcal{A}_1 | \mathcal{A}_2 & \textit{si} \quad \exists P_1, P_2.\ P \equiv P_1 | P_2 \textit{ et } P_i \models \mathcal{A}_i,\ i = 1, 2 \\
P & \models & \neg\mathcal{A} & \textit{si} \quad P \not\models \mathcal{A} \\
P & \models & \mathcal{A}_1 \triangleright \mathcal{A}_2 & \textit{si} \quad \forall Q \in M,\ Q \models \mathcal{A}_1 \textit{ implique } P|Q \models \mathcal{A}_2 \\
P & \models & 0 & \textit{si} \quad P \equiv \mathbf{0} \\
P & \models & \Diamond\mathcal{A} & \textit{si} \quad \exists P'.\ P \rightsquigarrow P' \textit{ et } P' \models \mathcal{A} \\
P & \models & \text{И}n.\mathcal{A} & \textit{si} \quad \textit{pour tout } m \in \mathsf{Na} - \big(\mathsf{fn}(P) \cup \mathsf{fn}(\mathcal{A})\big),\quad P(n \leftrightarrow m) \models \mathcal{A} \\
P & \models & n®\mathcal{A} & \textit{si} \quad \textit{il existe } P' \in M \textit{ tel que } P \equiv (\nu n)P' \textit{ et } P' \models \mathcal{A}
\end{array}$$

Exemple 3.2.5 *A partir du connecteur ® et du quantificateur générique, on peut définir le quantificateur de révélation considéré initialement par Luis Caires [Cai99] :*

$$Hn.\mathcal{A} \stackrel{\text{def}}{=} \text{И}n.n®\mathcal{A}$$

dont les modèles sont les termes P qui se mettent sous la forme $(\nu n')P$ avec $n' \notin \mathsf{fn}(\mathcal{A})$ et $P \models \mathcal{A}(n \leftrightarrow n')$. On peut alors décrire un processus gardé par une capabilité à l'aide de la formule point fixe :

$$\mathsf{1garde} \stackrel{\text{def}}{=} \nu\mathcal{X}.\ 1 \wedge Hn.\mathcal{X}$$

où 1 est la formule $\neg 0 \ \wedge\ \neg\big((\neg 0) \mid (\neg 0)\big)$ qui exprime que P est insécable.

La logique spatiale du π-calcul, définie par Luis Caires et Luca Cardelli dans [CC01], est une extension de la logique unifiée incluant la quantification au second ordre, la récursion, et surtout pour ce qui nous concernera un connecteur logique supplémentaire qui caractérise le processus message[6]

$$P \models \overline{a}\langle n\rangle \quad \text{si} \quad P \equiv \overline{a}\langle n\rangle.$$

On retrouvera la formule $\overline{a}\langle n\rangle$ au Chapitre 6.

Exemple 3.2.6 (Raisonnement arrière en π-calcul) *On se place dans la logique spatiale du π-calcul et pour la transition*

$$\{?\} \quad P \xrightarrow{an} P' \quad \{\mathcal{A}\}$$

on cherche à exprimer la plus faible précondition sur P afin que P' satisfasse $\mathcal{A}$. On peut déjà noter que cette précondition est précisément exprimée par la formule de la logique modale $\langle an\rangle\mathcal{A}$, mais on cherche à l'exprimer par une formule

[6] Le π-calcul considéré dans [CC01] est asynchrone.

de la logique spatiale. Soit P un tel processus. Alors $P|\overline{a}\langle n\rangle \longrightarrow P'$, autrement dit

$$P \models \overline{a}\langle n\rangle \triangleright \Diamond\mathcal{A}$$

Cette formule est toutefois encore trop imprécise pour exprimer $\langle an\rangle\mathcal{A}$. Par exemple, prenons $\mathcal{A} = \top$; soit P le processus $b(x)|\overline{b}\langle c\rangle$; alors $P\models\overline{a}\langle n\rangle \triangleright \Diamond\top$, puisque $P|\overline{a}\langle n\rangle \longrightarrow \overline{a}\langle n\rangle$, mais $P \overset{an}{\not\longrightarrow}$. On verra au chapitre 6 que l'on peut contourner cette difficulté et exprimer de façon générale la plus faible précondition.

3.2.4 Logiques des Ambients

On a vu comment la logique spatiale pouvait s'appliquer à une algèbre de processus au travers de la logique spatiale unifiée. Toutefois, la logique unifiée ignore l'aspect spatial du processus qu'est le constructeur d'ambient $a[P]$, vu comme une localité ou site d'exécution. Dans la logique des ambients originale de Cardelli et Gordon [CG00], la structure spatiale observée est celle de l'arbre étiqueté non ordonné correspondant à la hiérarchie des localités, ce qui se traduit par l'emploi du constructeur d'ambient dans la logique. On peut ainsi écrire la formule $n[\mathcal{A}]$ garantissant que le processus que l'on considère est un ambient nommé n à l'intérieur duquel $\mathcal{A}$ est vérifiée. D'autre part, de même que la logique spatiale considère le connecteur $\triangleright$ adjoint de $|$, la logique des ambients considère les connecteurs adjoints de $a[.]$ et a®. Les formules de la logique des Ambients sont ainsi celles générées par la grammaire

$$\begin{array}{lcll}
\mathcal{A} & ::= & \mathcal{A}\wedge\mathcal{A} \mid \neg\mathcal{A} \mid \mathcal{A}|\mathcal{A} \mid n[\mathcal{A}] \mid 0 & \text{(logique intensionnelle)} \\
 & \mid & \text{И}n.\mathcal{A} \mid n \circledR \mathcal{A} & \text{(extension nominale)} \\
 & \mid & \mathcal{A}\triangleright\mathcal{A} \mid \mathcal{A}@n \mid \mathcal{A}\oslash n & \text{(adjoints)} \\
 & \mid & \Diamond\mathcal{A} & \text{(logique temporelle)} \\
 & \mid & \exists n.\mathcal{A} & \text{(quantification existentielle)}
\end{array}$$

et dont la sémantique est donnée par la relation de satisfaction $P\models\mathcal{A}$ définie comme suit :

Définition 3.2.7 (Satisfaction pour DAL et SAL)

$$\begin{array}{llll}
P \models \mathcal{A}_1\wedge\mathcal{A}_2 & si & P\models\mathcal{A}_1 \textit{ et } P\models\mathcal{A}_2 \\
P \models \mathcal{A}_1|\mathcal{A}_2 & si & \exists P_1,P_2.\ P\equiv P_1|P_2 \textit{ et } P_i\models\mathcal{A}_i,\ i=1,2 \\
P \models \neg\mathcal{A} & si & P \not\models \mathcal{A} \\
P \models \mathcal{A}_1\triangleright\mathcal{A}_2 & si & \forall Q\in M,\ Q\models\mathcal{A}_1 \textit{ implique } P|Q\models\mathcal{A}_2 \\
P \models 0 & si & P\equiv\mathbf{0} \\
P \models \Diamond\mathcal{A} & si & \exists P'.\ P\Rightarrow P' \textit{ et } P'\models\mathcal{A} \\
P \models \exists n.\mathcal{A} & si & \textit{il existe } m\in\mathsf{Na}.\ P\models\mathcal{A}\{^n/m\} \\
P \models \text{И}n.\mathcal{A} & si & \textit{pour tout } n\in\mathsf{Na}-\big(\mathsf{fn}(P)\cup\mathsf{fn}(\mathcal{A})\big),\quad P(n\leftrightarrow m)\models\mathcal{A} \\
P \models n\circledR\mathcal{A} & si & \textit{il existe } P'\in M \textit{ tel que } P\equiv(\nu n)P' \textit{ et } P'\models\mathcal{A} \\
P \models n[\mathcal{A}] & si & \textit{il existe } P' \textit{ tel que } P \equiv n[P'] \textit{ et } P'\models\mathcal{A} \\
P \models \mathcal{A}@n & si & n[P]\models\mathcal{A} \\
P \models \mathcal{A}\oslash n & si & (\nu n)P\models\mathcal{A}
\end{array}$$

On notera DAL (pour *Dynamic Ambient Logic*) le fragment de la logique des ambients sans l'extension nominale ni le connecteur adjoint $\oslash$. C'est une extension de la logique unifiée dans laquelle la modalité de temps est interprétée au sens faible ($\rightsquigarrow\Longrightarrow$). Pour DAL, les modèles seront pris dans le calcul des ambients publics MA ; cette logique est essentiellement étudiée au chapitre 7.

On notera par ailleurs SAL (pour *static Ambient Logic*) le fragment statique de la logique des ambients, c'est à dire les formules sans modalité de temps $\diamond$, et on ne considèrera pas non plus la quantification existentielle - on notera plutôt $SAL^{\exists}$ le même fragment dans lequel on considère aussi la quantification existentielle. Pour cette logique, on considérera les ambients statiques (SA), c'est à dire le calcul des ambients sans instructions de mouvement. Cette logique est étudiée essentiellement aux chapitres 4 et 8.

On se propose maintenant d'illustrer le pouvoir expressif et l'utilisation de ces logiques pour la spécification de certaines propriétés. Tout d'abord pour les Ambients, on rappelle la propriété dite du *pare-feu parfait* (*perfect firewall equation*) : si l'on considère le processus $(\nu n)n[a[P]]|b[Q]$, où $n \notin \mathsf{fn}(P)$, alors ce processus définit une localité imperméable dans les deux sens (a ne peut pas sortir, et b ne peut pas rentrer). D'un point de vue logique, la formule suivante est valide :

$$a[\top] \to \text{И}n.\ \big(\ b[\top]\ \rhd\ \Box\big(\ b[\top]|\top\ \wedge\ Hm.m[a[\top]|\top]|\top\ \big)\big) \oslash n@n.$$

En effet, elle signifie que pour tout processus de la forme $a[P]$, si l'on place ce processus dans un contexte de la forme $(\nu n)n[a[P]]\ |\ b[Q]$, alors tout réduit R de ce processus enrichi se met sous la forme $(\nu m)\big(m[a[R_1]]|b[R_2]\big)$.

Pour les Ambients statiques, l'application envisagée de ce modèle est la définition d'un langage de requêtes pour des documents XML. Pour en voir un exemple, on peut considérer une base de donnée bibliographique et la recherche d'un ouvrage ayant pour auteur Georges Liqueur. La requête s'écrit alors

$$\exists bookentry\ .\ bookentry[author[\text{Georges Liqueur}]^{\exists}]^{\exists}.$$

Par ailleurs, une interprétation des noms restreints en tant que liens symboliques a été proposée, voir [CG01a].

3.2.5 L'espace par les modalités

Les logiques spatiales présentées jusqu'ici ne sont pas les seules à spécifier des propriétés spatiales ; ce qui les caractérise est plutôt la trilogie de connecteurs spatiaux $0, |, \rhd$, qui permet d'étudier l'espace de façon locale et propose une notion de réutilisation de spécification (la modularité) par le connecteur $\rhd$.

L'espace peut être toutefois décrit de façon plus simple par l'utilisation de modalités[7]. Ainsi, dans [VCHP04], les auteurs définissent une extension du λ-calcul et des types avec une modalité d'espace. La logique ainsi obtenue est exactement la logique modale intuitionniste $IS5$ [Sim94], et la modalité s'interprète par la reconfiguration d'un espace de ressources.

Plus proche du calcul des Ambients, on peut mentionner [MZW03], où les auteurs adaptent les idées de la logique temporelle (sans branchement) pour spécifier des propriétés de reconfigurations d'arbres. Les auteurs définissent ainsi une modalité d'espace $n[\mathcal{A}]$ dont le dual $n\langle\mathcal{A}\rangle$ peut s'interpréter comme la formule $n[\mathcal{A}]|\top$ de la logique des ambients. Les modèles de formules sont des traces

[7]Une modalité d'espace (le "somewhere"), relativement peu étudiée, existe pour la logique des Ambients [CG00].

d'exécution d'arbres reconfigurables, et fournissent en particulier une bonne abstraction du calcul des ambients. Ces modèles ne comportent pas les instructions de mouvement, contrairement aux ambients, ce qui rend la logique beaucoup moins intensionnelle. On peut alors développer une notion de *raffinement* de modèle pour laquelle $\models$ est monotone. Ceci fournit une méthode de spécification incrémentale à mettre en balance, même si elle est relativement différente, avec la spécification modulaire permise par le connecteur $\triangleright$.

On a donc défini les diverses logiques spatiales introduites en amont de cette thèse, et on en a extrait certains fragments auxquels on s'intéressera particulièrement par la suite. On a par ailleurs présenté les motivations de l'introduction des logiques spatiales, les principales étant la recherche d'une notion d'observation spatiale sur les algèbres de processus et un outil de spécification modulaire. Ces logiques présentent toutefois de nombreux aspects inhabituels, et avant d'étudier l'expressivité de chacune de ces logiques, on se propose de donner une meilleure compréhension des connecteurs spatiaux par l'étude de leurs propriétés structurelles.

Chapitre 4

Propriétés structurelles

Il y a au moins deux manières de définir une nouvelle logique. Soit comme on l'a fait pour les logiques spatiales, on associe à une formule une sémantique en terme d'ensembles de modèles qu'elle accepte, ce qui permet de définir la notion de formule *vraie*, soit on se donne un ensemble d'axiomes et de règles de déduction, c'est à dire une notion de démonstration, et on définit alors la notion de formule *prouvable*. De façon schématique, les modèles donnent une *interprétation* du raisonnement, alors que la théorie de la démonstration lui donne sa *structure*.

Il est intéressant de pouvoir combiner les deux définitions afin d'interpréter de diverses façons la logique. Pour les logiques spatiales, qui sont fondées sur une notion de modèle fini, les théories de la démonstration proposées ne sont pas totalement satisfaisantes, la plupart étant incomplètes[1]. On n'est donc pas dans une situation aussi claire que pour les logiques standard, même s'il existe des propositions intéressantes de raisonnement spatial.

C'est donc de façon plus générale à la structure du raisonnement spatial que l'on va s'intéresser ici, et en particulier aux propriétés algébriques vérifiées par les formules, ce qu'on appelle ici les propriétés structurelles. Pour cela, on s'intéresse essentiellement à la logique des Ambients statiques (SAL).

La première notion que l'on considère est celle de dualité. La dualité permet de définir de nombreux connecteurs dérivés ; on utilisera très souvent ces connecteurs dans les autres chapitres, et leur présentation dans ce chapitre permet une bonne familiarisation avec les logiques spatiales. On poursuit alors l'étude de la logique par quelques règles d'inférence proposées dans diverses présentations de théories de la démonstration [CG00, BG03, CC02].

On étudie ensuite la gestion des noms dans les logiques spatiales. On a vu que SAL comporte un quantificateur générique $\text{И}n.\mathcal{A}$ et une modalité de révélation de noms $n®\mathcal{A}$. On s'intéresse d'une part à une définition duale de la paire $(\text{И}, ®)$ et à plusieurs connecteurs dérivés, puis on énonce une propriété de prénexation ; cette propriété sera utilisée pour la preuve d'un résultat de minimalisation au chapitre 8.

On conclut par une théorie de la démonstration pour une logique relativement proche de SAL [BG03].

[1]Voir Chapitre 9.

4.1 Formules dérivées

4.1.1 Dualité, connecteurs dérivés

La première notion de dualité que l'on observe en logique, souvent appelée règle de Morgan, concerne le "et" et le "ou". Plus généralement, on a une notion de dualité entre deux opérateurs lorsqu'ils s'expriment l'un l'autre par le jeu des négations. Du point de vue des quantificateurs, "pour tout" et "il existe" sont ainsi duaux, et du point de vue des modalités "je dois" et "je peux" sont aussi deux modalités duales.

On peut ainsi introduire par dualité les connecteurs dérivés suivants, pour lesquels on introduit une notation particulière tout en précisant leur encodage et leur sémantique :

Notation	Encodage	Interprétation
$\mathcal{A}_1 \vee \mathcal{A}_2$	$\neg(\neg\mathcal{A}_1 \wedge \neg\mathcal{A}_2)$	$P{\models}\mathcal{A}_1$ ou $P{\models}\mathcal{A}_2$
$\forall n.\mathcal{A}$	$\neg\exists n.\neg\mathcal{A}$	pour tout $m \in \mathsf{Na}, P{\models}\mathcal{A}\{^m/n\}$
$\Box\mathcal{A}$	$\neg\Diamond\neg\mathcal{A}$	pour toute évolution $P \rightsquigarrow P',\ P'{\models}\mathcal{A}$
$P{\models}\mathcal{A}_1 \Vert \mathcal{A}_2$	$\neg\big(\neg\mathcal{A}_1 \vert \neg\mathcal{A}_2\big)$	pour toute partition $P \equiv P_1 \vert P_2, P_1{\models}\vert A_1$ ou $P_2{\models}\mathcal{A}_2$
$\mathcal{A}_1 \blacktriangleright \mathcal{A}_2$	$\neg\big(A_1 \triangleright \neg\mathcal{A}_2\big)$	il existe $Q \in M$. $Q{\models}\mathcal{A}_1$ et $P \vert Q{\models}\mathcal{A}_2$

Par ailleurs, les autres connecteurs de la logique classique se définissent à partir des connecteurs primitifs $\neg$ et $\vee$

Notation	Encodage	Interprétation
$\mathcal{A}_1{\rightarrow}\mathcal{A}_2$	$\neg\mathcal{A}_1 \vee A_2$	si $P{\models}\mathcal{A}_1$ alors $P{\models}\mathcal{A}_2$
$\top$	$0 \vee \neg 0$	toujours satisfait
$\bot$	$\neg\top$	impossible

On peut de même définir une certaine forme de quantification spatiale, plus précisément une quantification sur les sous-structures de la structure que l'on considère :

Notation	Encodage	Interprétation
$\mathcal{A}^{\forall}$	$\mathcal{A} \Vert \bot$	$P \models \mathcal{A}^{\forall}$ ssi pour tout Q, R tels que $P \equiv Q \vert R,\ Q{\models}\mathcal{A}$
$\mathcal{A}^{\exists}$	$\mathcal{A} \vert \top$	$P \models \mathcal{A}^{\exists}$ ssi il existe Q, R tels que $P \equiv Q \vert R$ et $Q{\models}\mathcal{A}$

En logique intuitionniste, on distingue une formule de sa double négation. On admet ainsi $\vdash \mathcal{A} \rightarrow \neg\neg\mathcal{A}$, mais pas $\vdash \neg\neg\mathcal{A}{\rightarrow}\mathcal{A}$. La "double négation spatiale" respecte cette particularité, puisque l'on peut vérifier que $P{\models}(\mathcal{A} \triangleright \bot) \triangleright \bot$ ssi il existe Q tel que $Q{\models}\mathcal{A}$, autrement dit ssi $\mathcal{A}$ est satisfiable. Sur cet exemple le sujet P ne joue aucun rôle particulier, la formule $(\mathcal{A} \triangleright \bot) \triangleright \bot$ s'abstrait simplement du sujet qu'on lui fournit pour exprimer une propriété métalogique. C'est une caractéristique du raisonnement par adjoint que de permettre d'exprimer des propriétés qui transcendent le sujet auxquelles elles sont supposées se référer pour exprimer des vérités sur la structure même de la logique. Voici quelques exemples :

Notation	Encodage	Interpretation
$\mathcal{A}^{sat}$	$(\mathcal{A} \triangleright \bot) \ \triangleright \bot$ ou $\mathcal{A} \blacktriangleright \top$	$P \models \mathcal{A}^{sat}$ ssi il existe Q. $Q \models \mathcal{A}$ (cohérence)
$\mathcal{A}^{\vdash}$	$(\neg\mathcal{A}) \triangleright \bot$	$P \models \mathcal{A}^{\vdash}$ ssi pour tout Q, $Q \models \mathcal{A}$ (validité)
$n = m$	$m[\top]@n$	$P \models n = m$ ssi $n = m$

4.1.2 Une logique pour compter

Dans cette section on va se restreindre au fragment de la logique spatiale suivant :

$$\mathcal{A} \ ::= \ \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \mathcal{A}|\mathcal{A} \mid \mathcal{A} \triangleright \mathcal{A} \mid 0$$

La relation $P{\models}\mathcal{A}$ ne dépend alors plus que du nombre d'atomes que possède P. En effet, on peut définir les formules suivantes :

Notation	Encodage	Interpretation
1	$\neg 0 \ \wedge \ 0 \| \| 0$	P est atomique
n	$1\|1\|\ldots\|1$	P contient n atomes

On peut de même associer à chaque formule $\mathcal{A}$ la partie $[\![\mathcal{A}]\!]$ de $\mathbb{N}$ correspondant à l'ensemble des tailles des modèles de $\mathcal{A}$. On a ainsi

$$[\![0]\!] \ = \ \{0\} \qquad [\![\mathcal{A}_1 \wedge \mathcal{A}_2]\!] \ = \ [\![\mathcal{A}_1]\!] \cap [\![\mathcal{A}_2]\!] \qquad [\![\neg\mathcal{A}]\!] \ = \ \mathbb{N}\backslash[\![\mathcal{A}]\!]$$

$$[\![\mathcal{A}_1|\mathcal{A}_2]\!] \ = \ [\![\mathcal{A}_1]\!] + [\![\mathcal{A}_2]\!] \qquad [\![\mathcal{A}_1 \triangleright \mathcal{A}_2]\!] \ = \ [\![\mathcal{A}_1]\!] - [\![\mathcal{A}_2]\!] = \{n - m : n \in [\![\mathcal{A}_1]\!], m \in [\![\mathcal{A}_2]\!]\}$$

On peut ainsi s'intéresser très simplement à l'expressivité de cette logique, autrement dit chercher à caractériser les ensembles de modèles exprimables par des formules. Dans ce cas, on a tout simplement les parties finies ou cofinies de $\mathbb{N}$. On note d'ailleurs que la présence de $\triangleright$ ne joue aucun rôle dans cela, puisque l'on sait exprimer n'importe quelle partie finie au cofinie au moyen des formules $0, 1, 2, \ldots$ et $1|\top, 2|\top, \ldots$ On verra une généralisation de cette propriété au chapitre 8.

D'autre part, Meyssonier et al. [MZL04] ont démontré que si l'on étendait la logique spatiale avec l'étoile de Kleene[2], les contraintes de taille de structures que l'on sait alors exprimer sont exactement celles que l'on peut formuler en arithmétique de Presburger.

On retiendra cet aspect très caractéristique des logiques spatiales, qui servira d'une part à établir l'intensionalité de la logique au chapitre 7, d'autre part à fournir un contre-exemple au chapitre 5.

4.2 Théorèmes admissibles élémentaires

4.2.1 Quelques exemples

Afin d'illustrer les propriétés structurelles de la logique spatiale, on présente maintenant quelques théorèmes admissibles élémentaires. On retrouve les propriétés algébriques usuelles régissant conjonction, implication, et neutre de la

[2]On se donne les formules $\mathcal{A}^*$ avec l'interprétation $\mu X.\ \mathbf{0} \vee (A|X)$

conjonction :

curryfication	$\mathcal{A}_1 \rightarrow \mathcal{A}_2 \rightarrow \mathcal{B}$	$\dashv\vdash$	$(\mathcal{A}_1 \wedge \mathcal{A}_2) \rightarrow \mathcal{B}$	$\mathcal{A}_1 \triangleright \mathcal{A}_2 \triangleright \mathcal{B}$	$\dashv\vdash$	$(\mathcal{A}_1 \vert \mathcal{A}_2) \triangleright \mathcal{B}$
neutre 1	$\mathcal{A} \wedge \top$	$\dashv\vdash$	$\mathcal{A}$	$\mathcal{A} \vert 0$	$\dashv\vdash$	$\mathcal{A}$
neutre 2	$\top \rightarrow \mathcal{A}$	$\dashv\vdash$	$\mathcal{A}$	$0 \triangleright \mathcal{A}$	$\dashv\vdash$	$\mathcal{A}$

Néanmoins, comme en logique linéaire, les règles d'affaiblissement ou de contraction ne sont pas admissibles pour le fragment spatial ; ainsi $\mathcal{A}|\mathcal{B} \not\vdash \mathcal{A}$ et $\mathcal{A} \not\vdash \mathcal{A}|\mathcal{A}$. La conjonction séparante présente toutefois quelques liens avec la conjonction classique dont voici deux exemples :

$$\begin{array}{rcl} \mathcal{A} \mid \mathcal{B} & \vdash & \mathcal{A}^{\exists} \wedge \mathcal{B}^{\exists} \\ 0 \ \wedge \ \mathcal{A} \mid \mathcal{B} & \dashv\vdash & 0 \ \wedge \ \mathcal{A} \ \wedge \ \mathcal{B} \end{array}$$

4.2.2 Formes prénexes du quantificateur générique

Le quantificateur générique propose une quantification intermédiaire entre les quantifications existentielle et universelle ; formellement :

$$\exists n.\mathcal{A} \ \vdash \ \text{И} n.\mathcal{A} \ \vdash \ \forall n.\mathcal{A}.$$

Il est par ailleurs son propre dual, puisque $\neg \text{И} n.\mathcal{A} \dashv\vdash \text{И} n.\neg\mathcal{A}$, ce qui le rend distributif sur tous les connecteurs logiques classiques[3]. Par le jeu de la dualité, on peut aussi toujours échanger l'ordre d'écriture entre quantificateurs et connecteurs logiques classiques. On s'intéresse alors à "remonter" les quantificateurs en tête de formule. En logique classique, on appelle ainsi *forme prénexe* une formule de la forme

$$\mathcal{A} \quad = \quad Q_1 x_1 . Q_2 x_1 \ldots Q_n x_n . \mathcal{A}'$$

où les $Q_i \in \{\forall, \exists\}$ et $\mathcal{A}'$ est une formule sans quantificateur ; en remontant les quantificateurs, on peut donc ramener toute formule logique classique à une forme prénexe. Les formules de la logique classique avec des quantificateurs génériques admettent des formes prénexes, puisque tous les connecteurs (hors quantificateurs) commutent avec И.

Considérons maintenant les logiques spatiales. Tout d'abord, les connecteurs spatiaux intensionnels commutent et se distribuent mal avec les quantificateurs existentiels et universels :

$$\begin{array}{rcl} \exists n.(\mathcal{A}|\mathcal{B}) & \not\dashv\vdash & \exists n.\mathcal{A} \mid \exists n.\mathcal{B} \\ \forall n.(\mathcal{A}|\mathcal{B}) & \dashv\not\vdash & \forall n.\mathcal{A} \mid \forall n.\mathcal{B}. \end{array}$$

On n'aura donc pas de forme prénexe pour ces quantificateurs. En revanche, le quantificateur générique commute avec les connecteurs intensionnels :

$$\begin{array}{rcl} \text{И} n.\ (\mathcal{A}|\mathcal{B}) & \dashv\vdash & \text{И} n.\mathcal{A} \mid \text{И} n.\mathcal{B} \\ \text{И} n.m[\mathcal{A}] & \dashv\vdash & m[\text{И} n.\mathcal{A}] \\ \text{И} n.m \circledR \mathcal{A} & \dashv\vdash & m \circledR \text{И} n.\mathcal{A} \end{array}$$

L'implication spatiale n'admet toutefois pas de règle de commutation similaire. Par exemple, la formule $(\text{И} n.n[\top]) \ \triangleright \ \bot$ est équivalente à $\top$ puisqu'aucun

[3] en revanche, il ne commute pas avec les quantificateurs classiques. Par exemple, $\text{И} n.\exists m.n = m$ est valide, tandis que $\exists m.\text{И} n.n = m$ est absurde

processus ne satisfait $\text{И}n.n[\top]$, mais la formule $\text{И}n.\ \big(n[\top] \triangleright \bot\big)$ est équivalente à $\bot$.

On peut pourtant contourner cette difficulté. Pour cela, on considère un nouveau connecteur dérivé[4] de la logique :

Notation	Encodage	Interpretation
$©n$	$\neg n®\top$ $\big(1 \ \wedge\ n[0] \triangleright (1 \oslash n)\big)^{\exists}$	$P \models ©n$ ssi $n \in \mathsf{fn}(P)$

La formule $©n$ va maintenant permettre de définir la prénexation, puisqu'elle permet de rajouter des conditions de fraîcheur là où elles feraient défaut.

On considère le système de réécriture suivant :

$$
\begin{array}{llcll}
(\wedge) & (\text{И}n.\mathcal{A}_1) \ \wedge\ \mathcal{A}_2 & \rightsquigarrow & \text{И}n.(\mathcal{A}_1 \wedge \mathcal{A}_2) & (n \notin \mathsf{fn}(\mathcal{A}_2)) \\
(\neg) & \neg\text{И}n.\mathcal{A}_1 & \rightsquigarrow & \text{И}n.\neg\mathcal{A}_1 & \\
(|) & (\text{И}n.\mathcal{A}_1) \mid \mathcal{A}_2 & \rightsquigarrow & \text{И}n.(\mathcal{A}_1|\mathcal{A}_2) & (n \notin \mathsf{fn}(\mathcal{A}_2)) \\
(\triangleright L) & (\text{И}n.\mathcal{A}_1) \ \triangleright\ \mathcal{A}_2 & \rightsquigarrow & \text{И}n.\Big(\big(\neg ©n \wedge \mathcal{A}_1\big) \triangleright \mathcal{A}_2\Big) & (n \notin \mathsf{fn}(\mathcal{A}_2)) \\
(\triangleright R) & \mathcal{A}_1 \ \triangleright\ (\text{И}n.\mathcal{A}_2) & \rightsquigarrow & \text{И}n.\Big(\big(\neg ©n \wedge \mathcal{A}_1\big) \triangleright \mathcal{A}_2 \Big) & (n \notin \mathsf{fn}(\mathcal{A}_1)) \\
(Amb) & m[\text{И}n.\mathcal{A}] & \rightsquigarrow & \text{И}n.m[\mathcal{A}] & (m \neq n) \\
(@) & (\text{И}n.\mathcal{A})@m & \rightsquigarrow & \text{И}n.(\mathcal{A}@m) & (m \neq n) \\
(®) & m®\text{И}n.\mathcal{A} & \rightsquigarrow & \text{И}n.m®\mathcal{A} & (m \neq n) \\
(\oslash) & (\text{И}n.\mathcal{A}) \oslash m & \rightsquigarrow & \text{И}n.(\mathcal{A} \oslash m) & (m \neq n)
\end{array}
$$

On dira qu'une formule $\mathcal{A}$ est *bien quantifiée* si pour chaque contexte C, chaque n et chaque $\mathcal{A}'$ tels que $\mathcal{A} = \mathsf{C}[\text{И}n.\mathcal{A}']$, n n'est pas libre dans C. Par α-conversion, il est toujours possible de ramener une formule quelconque à une formule bien quantifiée équivalente.

Lemme 4.2.1 (Terminaison) *Le système de réécriture $\rightsquigarrow$ termine, et les formes normales des formules bien quantifiées sont des formes prénexes.*

Notons qu'on peut aussi établir la confluence du système à condition de considérer les séquences de quantificateurs génériques comme commutatives, ce qui donne une représentation canonique de la forme prénexe d'une formule.

Lemme 4.2.2 (Correction de $\rightsquigarrow$) *Si $\mathcal{A} \ \rightsquigarrow\ \mathcal{A}'$, alors $\mathcal{A} \ \dashv\vdash\ \mathcal{A}'$.*

Preuve: (abrégée) On vérifie la propriété pour chaque règle. Par exemple :
– la règle $\neg\text{И}n.\mathcal{A} \ \rightsquigarrow\ \text{И}n.\neg\mathcal{A}$:

$$
\begin{array}{ll}
 & P \models \neg\text{И}n.\mathcal{A} \\
\Leftrightarrow & P \not\models \text{И}n.\mathcal{A} \\
\Leftrightarrow & \text{non } \Big(\text{pour tout } m \notin \mathsf{fn}(\mathcal{A}) \cup \mathsf{fn}(P).\ P \models \mathcal{A}(m \leftrightarrow n)\Big) \\
\Leftrightarrow & \text{il existe } m \notin \mathsf{fn}(\mathcal{A}) \cup \mathsf{fn}(P).\ P \not\models \mathcal{A}(m \leftrightarrow n) \\
\Leftrightarrow & \text{il existe } m \notin \mathsf{fn}(\mathcal{A}) \cup \mathsf{fn}(P).\ P \models \neg\mathcal{A}(m \leftrightarrow n) \\
\Leftrightarrow & P \models \text{И}n.\neg\mathcal{A}
\end{array}
$$

[4]La formule $©n$ n'est pas atomique dans SAL, mais elle admet plusieurs encodages, dont les deux présentés ici et un troisième présenté au chapitre 6.

– la règle $(\text{И} n.\mathcal{A}_1) \triangleright \mathcal{A}_2$:

$$\begin{array}{ll} & P \models (\text{И} n.\mathcal{A}_1) \triangleright \mathcal{A}_2 \\ \Leftrightarrow & \text{pour tout } Q, \text{pour tout } n' \notin \mathsf{fn}(\mathcal{A}_1) \cup \mathsf{fn}(Q), \\ & Q \models \mathcal{A}_1(n \leftrightarrow n') \ \Rightarrow\ P|Q \models \mathcal{A}_2 \\ \Leftrightarrow & \text{pour tout } Q, \text{pour tout } n' \notin \mathsf{fn}(\mathcal{A}_1 \triangleright \mathcal{A}_2) \cup \mathsf{fn}(P|Q). \\ & Q \models \mathcal{A}_1(n \leftrightarrow n') \ \Rightarrow\ P|Q \models \mathcal{A}_2 \\ \Leftrightarrow & \text{pour tout } Q, \text{pour tout } n' \notin \mathsf{fn}(\mathcal{A}_1 \triangleright \mathcal{A}_2) \cup \mathsf{fn}(P|Q). \\ & Q \models \mathcal{A}_1(n \leftrightarrow n') \ \Rightarrow\ P|Q \models \mathcal{A}_2(n \leftrightarrow n') \\ \Leftrightarrow & \text{pour tout } n' \notin \mathsf{fn}(\mathcal{A}_1 \triangleright \mathcal{A}_2) \cup \mathsf{fn}(P),\ \text{pour tout } Q, \\ & n' \notin \mathsf{fn}(Q) \text{ et } Q \models \mathcal{A}_1(n \leftrightarrow n') \text{ impliquent } P|Q \models \mathcal{A}_2(n \leftrightarrow n') \\ \Leftrightarrow & \text{pour tout } n' \notin \mathsf{fn}(\mathcal{A}_1 \triangleright \mathcal{A}_2) \cup \mathsf{fn}(P), P \models \big(\mathcal{A}_1 \ \wedge\ \neg \copyright n\big) \triangleright \mathcal{A}_2 \\ \Leftrightarrow & P \models \text{И} n.\ \big(\mathcal{A}_1 \ \wedge\ \neg \copyright n\big) \triangleright \mathcal{A}_2. \end{array}$$

□

On a donc établi le résultat suivant :

Théorème 4.2.3 *Pour toute formule $\mathcal{A} \in SAL$, il existe une famille de noms $\tilde{n}$ et une formule sans quantificateurs $\mathcal{A}'$ telle que $\mathcal{A} \dashv\vdash \text{И}\tilde{n}.\mathcal{A}'$.*

Ce résultat donne donc un aperçu des propriétés des quantificateurs génériques. Il a été établi indépendamment dans [Loz03] et [GC04]. Dale Miller et Alwen Tiu donnent par ailleurs un autre résultat de prénexation pour un quantificateur générique légèrement différent [MT03].

Digressions sur © : Notons que la formule © permet d'encoder le quantificateur générique à partir des quantificateurs universels et existentiels :

$$\begin{array}{lll} \text{И} n.\mathcal{A} & \dashv\vdash & \exists n.\ \neg \copyright n \ \wedge\ \bigwedge_{m \in \mathsf{fn}(\mathcal{A}) \setminus \{n\}} n \neq m \ \ \wedge\ \ \mathcal{A} \\ & \dashv\vdash & \forall n.\ \neg \copyright n \rightarrow \bigwedge_{m \in \mathsf{fn}(\mathcal{A}) \setminus \{n\}} n \neq m \ \ \rightarrow\ \ \mathcal{A} \end{array}$$

On peut aussi noter que les paires de connecteurs $(\text{И}, ®)$ et $(H, ©)$ s'encodent l'une l'autre. On a présenté ces derniers connecteurs comme dérivés des premiers par $Hn.\mathcal{A} = \text{И} n.n®\mathcal{A}$ et $©n = \neg n®\top$, mais on peut aussi donner les définitions réciproques suivantes :

$$\text{И} n.\mathcal{A} \ =\ Hn.\big(\neg n© \ \wedge \mathcal{A}\big) \qquad\qquad n®\mathcal{A} \ =\ \neg ©n \ \wedge\ Hn.\mathcal{A}$$

On a jusqu'ici illustré certaines propriétés structurelles des logiques spatiales. Ces propriétés ne fournissent pas une théorie de la démonstration, mais en sont des indicateurs. Elles sont essentiellement tirées des travaux originels sur les logiques spatiales [CG00, CC01, CC02] dans lesquels sont proposées diverses théories de la démonstration. On développe maintenant cette approche.

4.3 Calculs de séquents

Les calculs de séquents pour les logiques spatiales sont difficiles à définir. Un premier problème vient du fait qu'en présence de $\triangleright$ les problèmes de validité et de satisfaction sont équivalents, et que toute théorie de la démonstration doit donc permettre de résoudre les deux simultanément (voir aussi chapitre 9). Une solution pour obtenir un calcul de séquents complet avec $\triangleright$ consiste à intégrer des ensembles test explicites dans le calcul, ces ensembles test fournissant un échantillon représentatif de tous les contextes possiblement introduits par $\triangleright$ [CCG03].

Un second problème est que les logiques spatiales sont très souvent non décidables, et même non semi-décidables, ce qui empêche d'obtenir un calcul de séquents complet, en particulier avec le connecteur $\diamond$ (voir chapitre 9) [5]. Les calculs de séquents proposés sont donc plus des indicateurs de ce qu'est un raisonnement spatial qu'une véritable théorie de la preuve spatiale.

On donne maintenant quelques éléments sur les calculs de séquents proposés pour les logiques des ambients et du π-calcul [CG00, CC01, CC02]. On rappelle enfin l'approche originelle de la logique des implications fagotées (*bunched logic*), et son adaptation à une logique proche de celle des ambients [BG03].

4.3.1 Calculs de séquents pour les logiques des Ambients et du π-calcul

Dans [CG00], les auteurs considèrent des règles de déduction pour des séquents de la forme $\mathcal{A} \vdash \mathcal{B}$. Parmi ces règles, on trouve par exemple la distributivité de $\vee$ pour $|$, ou la règle d'introduction de $\triangleright$:

$$(\mathcal{A}_1 \vee \mathcal{A}_2) \mid \mathcal{B} \quad \dashv\vdash \quad \mathcal{A}_1 \mid \mathcal{B} \ \vee \ \mathcal{A}_2 \mid \mathcal{B} \qquad \frac{\mathcal{A}|\mathcal{A}' \ \vdash \ \mathcal{B}}{\mathcal{A} \ \vdash \ \mathcal{A}' \triangleright \mathcal{B}}$$

Cardelli et Gordon mentionnent aussi le fait qu'en sémantique faible, la modalité de temps $\diamond$ vérifie les axiomes de la logique modale $S4$ (par transitivité de $\Rightarrow$), ainsi que certaines des règles utilisées pour la prénexation. Toutefois, aucune théorie de la démonstration n'est réellement élaborée.

Plus tard, dans [CC02], les auteurs proposent un calcul des séquents pour la logique du π-calcul. Ces séquents sont des jugements de satisfaction de la forme

$$\langle \mathcal{S} \rangle \quad x_1 : \mathcal{A}_1, \ldots, x_n : \mathcal{A}_n \quad \vdash \quad y_1 : \mathcal{B}_1, \ldots, y_m : \mathcal{B}_m$$

où les x_i, y_i sont des *variables de mondes*, instanciables par des processus, et $\mathcal{S}$ est un ensemble de contraintes sur les variables de mondes. On admet les règles de contraction et affaiblissement. Ce système a l'intérêt de pouvoir représenter aussi bien le problème de la validité que celui de la satisfaction. En effet, $\mathcal{A}$ est valide ssi le séquent $\langle\rangle \vdash x : \mathcal{A}$ est valide, et $P \models \mathcal{A}$ ssi le séquent $\langle$"$x = P$"$\rangle \vdash x : \mathcal{A}$ est valide. On cite ci-dessous quelques règles de déduction :

$$\frac{x, y \text{ frais / conclusion} \qquad \langle \mathcal{S}, u = x|y \rangle \ \ \Gamma, \ x : \mathcal{A}, \ y : \mathcal{B} \ \ \vdash \ \ \Delta}{\langle \mathcal{S} \rangle \ \ \Gamma, \ u : \mathcal{A}|B \ \ \vdash \ \ \Delta} \ (|L)$$

$$\frac{u \ =_{\mathcal{S}} \ v|t \qquad \langle \mathcal{S} \rangle \ \ \Gamma, \ v : \mathcal{A}, \ \ \vdash \ \ \Delta \quad \langle \mathcal{S} \rangle \ \ \Gamma, \ t : \mathcal{B}, \ \ \vdash \ \ \Delta}{\langle \mathcal{S} \rangle \ \ \Gamma \ \ \vdash \ \ u : \mathcal{A}|\mathcal{B} \ \Delta} \ (|R)$$

$$\frac{\langle \mathcal{S} \rangle \ \ \Gamma \ \ \vdash \ \ t : \mathcal{A}, \ \Delta \quad \langle \mathcal{S} \rangle \ \ \Gamma, \ t|u : \mathcal{B}, \ \ \vdash \ \ \Delta}{\langle \mathcal{S} \rangle \ \ \Gamma, \ u : \mathcal{A} \triangleright \mathcal{B} \ \ \vdash \ \ \Delta} \ (\triangleright L)$$

$$\frac{v =_{\mathcal{S}} x|u, \ \ x \text{ frais / conclusion} \qquad \langle \mathcal{S} \rangle \ \ \Gamma, \ x : \mathcal{A}, \ \ \vdash \ \ v : \mathcal{B}, \ \Delta}{\langle \mathcal{S} \rangle \ \ \Gamma \ \ \vdash \ \ u : \mathcal{A} \triangleright \mathcal{B}, \ \Delta} \ (\triangleright R)$$

[5] D'un autre point de vue, c'est la constante $\perp$ qui cause l'incomplétude, voir par exemple [POY04].

Les auteurs prouvent la correction et l'élimination des coupures pour leur système. Toutefois, leur système est dédié à la logique du π-calcul et l'ensemble des dérivations correctes est énumérable ; on verra au chapitre 9 que cela montre qu'il est incomplet [6]. Si on ne parvient pas à définir de calcul de séquents qui représente parfaitement et naturellement les logiques spatiales pour la concurrence, c'est peut-être parce qu'elles sont avant tout des langages de spécification. Pour trouver une logique "spatiale" dont on ait une théorie de la démonstration adéquate, il faut se tourner vers la logique des implications fagotées.

4.3.2 Un calcul de séquents complet

La logique des implications fagotées considère des séquents de la forme $\Gamma \vdash \phi$ où la prémice Γ est un assemblage de formules par deux opérateurs : le premier noté , admet contraction et affaiblissement et correspond aux connecteurs $(\rightarrow, \wedge)$, le second noté ; n'admet aucune de ces deux règles et correspond aux connecteurs $(\triangleright, |)$. Les règles d'introduction se traduisent par l'utilisation de ces opérateurs dans la structure des prémices. Les règles sont contextuelles : on note $\Gamma(\Delta)$ pour le contexte de prémice Γ dans lequel le trou est rempli par Δ. On a par exemple

$$\frac{\Delta(\phi, \phi') \vdash \psi}{\Delta(\phi \wedge \phi') \vdash \psi}(\wedge L) \qquad \frac{\Gamma \vdash \phi \quad \Delta(\Delta'; \phi') \vdash \psi}{\Delta(\Delta'; \Gamma; \phi \triangleright \phi') \vdash \psi}(\triangleright\, L)$$

Dans [BG03], les auteurs reprennent une logique proche de la logique des Ambients statiques, sans la révélation. Ils définissent une sémantique en termes de modèle très proche de celle des Ambients, et prouvent la correction et la complétude d'un calcul de séquents directement issu de la logique des implications fagotées, avec la règle suivante pour la prise en compte des locations :

$$\frac{\Gamma \vdash \phi}{n[\Gamma] \vdash n[\phi]}$$

Toutefois les modèles présentent la particularité d'être des arbres potentiellement infinis, ce qui est une situation très différente de celle des algèbres de processus. Ce calcul de séquents fournit néanmoins un exemple intéressant de cadre formel pour le raisonnement spatial. L'originalité des règles contextuelles en fait aussi sa spécificité, qui oblige sans doute à passer par une structure de variables de mondes comme dans [CC02] pour retrouver un calcul des séquents plus habituel.

La structure du raisonnement spatial présente donc des singularités qui la rendent assez difficile à caractériser pour les logiques spatiales de la concurrence ou la logique séparante. Pour cette dernière, la problématique est toutefois un peu différente[7], puisqu'il s'agit avant tout d'utiliser la logique pour dériver des triplets de Hoare sur un langage impératif avec pointeurs. Un système d'inférence pour triplets de Hoare ainsi que des résultats liés à sa complétude ont été énoncés

[6] L'idée des séquents de satisfaction a été reprise par Samuel Mimram [Mim03] en stage de licence pour une logique spatiale statique, et ce système a été prouvé complet

[7] Les systèmes d'inférence pour la validité intéressent toutefois les chercheurs de la communauté de la logique séparante. Récemment, une théorie de la démonstration complète a été proposée pour la validité sur un fragment de la logique séparante, sans $-\!\!*$, mais déjà relativement expressif [BCO04].

(voir [Yan01]). Ce type de résultat est connu pour la logique classique. Dans le prochain chapitre, on se propose justement de comparer la logique séparante à la logique classique du point de vue de leur expressivité.

Deuxième partie

Chapitre 5

Conservatisme en logique séparante

La logique séparante est un langage d'assertion pour la sémantique axiomatique des langages impératifs. Les langages d'assertion généralement utilisés sont fondés sur la logique classique du premier ou du second ordre, mais n'utilisent pas les connecteurs spatiaux. Ces connecteurs facilitent l'axiomatisation et la formulation d'invariants dans les langages à pointeurs. Reynolds [Rey02] présente ainsi le cas d'un court programme de retournement de liste, pour lequel on peut trouver un invariant élémentaire en logique séparante, alors que la logique classique nécessite d'ajouter un certain nombre de conditions de non entrelacement sur les deux listes manipulées pendant le retournement. Pire, si la boucle de retournement n'est qu'une partie d'un programme plus important, il faudra aussi ajouter des conditions de non-entrelacement avec toutes les structures de données qui sont censées être indépendantes de ces deux listes, contrairement au cas de la logique séparante qui permet un raisonnement local indépendant du contexte.

Toutefois, d'un point de vue strictement expressif, cet exemple suggère qu'on peut théoriquement se contenter de la logique classique dans la preuve de petits programmes manipulant des pointeurs. Pour reprendre l'exemple 3.1.3, on peut exprimer la plus faible précondition $\mathcal{W}$ dans le triplet $\{\mathcal{W}\}\ [x] := 3\ \{\mathcal{A}\}$ en substituant dans $\mathcal{A}$ la formule $e \mapsto 3$ à toute sous formule de la forme $e \mapsto \ldots$ telle que x et e s'évaluent de la même façon. Notons E l'ensemble des expressions e qui apparaissent dans $\mathcal{A}$ dans une sous-formule $e \mapsto \ldots$. On ne sait pas a priori si chaque e s'évalue ou non comme x, mais on peut en revanche en faire l'hypothèse et énumérer toutes les configurations possibles, ce qui donnerait une précondition de la forme

$$\begin{array}{lll} \mathcal{W} & = & \bigvee_{(\vec{e_1},\vec{e_2})} \bigwedge_{e\in\vec{e_1}} x = e \ \wedge\ \bigwedge_{e\in\vec{e_2}} x \neq e \ \wedge\ \mathcal{A}\big[e \mapsto 3 \ /\ e \mapsto \ldots, \forall e \in \vec{e_1}, \mathsf{x}\big] \\ & & \text{où } (\vec{e_1},\vec{e_2}) \text{ parcourt les 2-partitions de } E. \end{array}$$

Ce que l'on voit ici, c'est un phénomène de *conservatisme* expressif. En un sens, la logique séparante étend la logique classique tout en restant dans le cadre expressif[1] de celle-ci.

[1]On prend ici une vision très spécifique de la notion d'expressivité, celle des classes de modèles caractérisées par les formules.

Dans ce chapitre, on va s'attacher à donner une formulation précise de cette intuition. On définit tout d'abord un *fragment classique* de la logique séparante, duquel on a exclu les connecteurs spatiaux $*$ et $-\!\!*$. On montre alors que ce fragment a la même expressivité que la logique séparante toute entière. On parlera ainsi de *minimalisation* de la logique séparante. On retrouvera un procédé de minimalisation similaire au chapitre 8. L'outil théorique essentiel que l'on retrouvera régulièrement par la suite est l'équivalence observationnelle spatiale, que l'on appellera par anticipation équivalence intensionnelle (voir chapitre 7).

Pour la définition du fragment classique, on devra tout d'abord reconsidérer l'ensemble des observations primitives sur lequel fonder la logique, ce que l'on appellera la *renormalisation*. Ce phénomène ressemble beaucoup à ce que l'on observera au chapitre 6, où l'on montrera que les logiques spatiales pour la concurrence permettent des observations dérivées qui donneront effectivement la bonne notion d'équivalence intensionnelle.

5.1 Fragment classique de la logique séparante

5.1.1 Renormalisation

On considère la logique séparante $\mathcal{L}_{Sep}$, c'est à dire sans le quantificateur existentiel, et on notera $s, h \models \mathcal{A}$ au lieu de $s, h \models_v \mathcal{A}$ puisqu'on ne considère pas de variables logiques.

On se propose de renormaliser la logique séparante en introduisant quatre nouvelles formules atomiques :

$$\begin{array}{lll} s,h \models & e \hookrightarrow e' & \text{si } [\![e]\!]s \in \mathsf{dom}(h) \text{ et } h([\![e]\!]s) = [\![e']\!]s \\ s,h \models & \mathsf{size} \geq k & \text{si } \sharp\mathsf{dom}(h) \geq k \\ s,h \models & \mathsf{alloc}\ e & \text{si } [\![e]\!] \in \mathsf{dom}(h) \\ s,h \models & e = e' & \text{si } [\![e]\!]s = [\![e']\!]s \end{array}$$

Ces formules s'expriment à partir des autres formules de la logique séparante définie précédemment (Définition 3.1.2). En effet, on peut écrire

$$\begin{array}{rcl} e \hookrightarrow e' & = & e \mapsto e' * \top \\ \mathsf{size} \geq k & = & \neg\mathsf{emp}\ *\ \neg\mathsf{emp} * \cdots * \neg\mathsf{emp} \ \ (k \text{ fois}) \\ \mathsf{alloc}\ e & = & e \mapsto \mathsf{nil}\ -\!\!*\ \bot \\ e = e' & = & \top\ -\!\!*\ \neg\left((\mathsf{alloc}\ e \wedge \neg\mathsf{alloc}\ e')\ *\ \top\right). \end{array}$$

En présence des connecteurs spatiaux, les formules $e \mapsto e'$ et emp permettent donc d'exprimer ces formules. Ce n'est plus le cas sans les connecteurs spatiaux, et on doit donc considérer ces nouvelles formules comme atomiques. Ces formules atomiques du fragment classique disposent toutefois d'une partie de l'expressivité spatiale, et on peut ainsi exprimer les formules atomiques de la logique séparante comme combinaison booléenne des formules atomiques du fragment classique :

$$\begin{array}{rcl} e \mapsto e' & = & \neg\mathsf{size} \geq 2 \ \wedge\ e \hookrightarrow e' \\ \mathsf{emp} & = & \neg\mathsf{size} \geq 1 \end{array}$$

Ces nouvelles formules atomiques sont par ailleurs monotones [2], à savoir que

[2] du point de vue de la théorie de la logique fagotée, elles sont aussi parfois appelées *intuitionnistes*

si l'une est vérifiée par un état mémoire $\sigma = (s, h)$, alors elle est encore vérifiée par tout état mémoire étendu $(s, h * h')$.

On appelle alors *fragment classique* de la logique séparante l'ensemble des formules générées par la grammaire suivante :

$$\mathcal{A} \quad ::= \quad \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \mathsf{size} \geq k \mid e \hookrightarrow e' \mid \mathsf{alloc}\ e \mid e = e'$$

Cette renormalisation effectuée, on va maintenant s'attacher à minimaliser la logique séparante. Le résultat que l'on souhaite établir est le suivant :

Théorème 5.1.1 *Pour toute formule $\mathcal{A}$ de $\mathcal{L}_{Sep}$, il existe une formule classique $\mathcal{A}'$ telle que $\mathcal{A} \dashv\vdash \mathcal{A}'$.*

Dans le même temps, on aura par ailleurs prouvé le résultat suivant : les formules monotones contiennent tout le pouvoir distinctif de la logique séparante.

5.1.2 Equivalence intensionnelle

On reprend ici une notion d'équivalence logique partielle $\approx_{E,w}$, dans la même idée qu'un jeu de Ehrenfreucht-Fraïssé dans lequel on borne le nombre de tours de jeu, ou encore de la stratification de la bisimilarité vue au chapitre 2. Ici, la restriction de l'équivalence porte sur l'ensemble des cases mémoires observables. D'une part, on ne considérera qu'un ensemble fini de variables de programmation pour accéder à ces cases mémoire, d'autre part on ne saura compter que partiellement le nombre de cases allouées. Ces restrictions sont prises à la mesure de la formule $\mathcal{A}$ que l'on souhaite encoder, de sorte que l'on ait la propriété

$$\text{si} \quad s, h \ \approx_{E,w} s', h' \quad \text{et} \quad s, h \models \mathcal{A}, \quad \text{alors} \quad s', h' \models \mathcal{A}.$$

Toujours dans l'esprit des jeux de Ehrenfreucht-Fraïssé on fonde l'équivalence intensionnelle sur les propriétés exprimées par les formules atomiques.

Notons donc tout d'abord quelques propriétés des formules atomiques classiques. La relation “h est une restriction de h'”, que l'on notera $h \leq h'$, est une relation d'ordre sur les tas. Pour cet ordre, les formules de la forme $e = e'$, $e \hookrightarrow e'$ et $\mathsf{alloc}\ e$ sont *stables* : si $\mathcal{A}$ est une telle formule, s une pile, h, h' deux tas, et v une valuation tels que $s, h \models \mathcal{A}$ et $s, h' \models \mathcal{A}$, et si h, h' sont cohérents, autrement dit il existe h_1 tel que $h_1 \geq h$ et $h_1 \geq h'$, alors $s, h \cap h' \models_v \mathcal{A}$. En revanche, les formules $\mathsf{size} \geq k$ ne sont pas stables. Pour E un ensemble d'expressions, on notera Stab_E l'ensemble des formules atomiques stables dont les expressions sont dans E, c'est-à-dire les formules du type $e \hookrightarrow e'$, $\mathsf{alloc}\ e$ et $e = e'$ avec $e, e' \in E$.

Soit Φ un ensemble fini de formules atomiques stables. Soit s, h un modèle de Φ. On peut décomposer $h = h_\Phi * h_\dagger$ où h_Φ est le sous-modèle de Φ minimum au-dessous de h.

On note alors

$$\mathsf{garb}_{s,\Phi}(h) \quad \stackrel{\text{def}}{=} \quad \sharp\mathsf{dom}(h_\dagger) \ = \ \sharp\mathsf{dom}(h) \ - \ \sharp\mathsf{dom}(h_\Phi)$$

la taille de la mémoire en excès dans h.

On cherche à définir une équivalence logique partielle entre deux états s, h et s', h' vis-à-vis d'un ensemble d'expressions E et d'une largeur d'observation w que l'on s'autorise pour distinguer. Par largeur d'observation, on souhaite exprimer que l'on ne compte distinctement le nombre de cellules allouées que jusqu'à w, au-delà la quantité de cellules allouées est un "beaucoup" indistinct. Pour formaliser, on définit la relation d'équivalence $=_w$ sur les entiers par

$$\forall n, m \in \mathbb{N} \qquad n =_w m \quad \text{ssi} \quad \begin{cases} \text{soit } n = m, \\ \text{soit } n \geq w \text{ et } m \geq w \end{cases}$$

On définit maintenant l'équivalence intensionnelle $\approx_{E,w}$ où l'on n'examine que les formules basées sur des expressions prises dans E et un comptage des cellules en excès qui ne dépasse pas w :

Définition 5.1.2 *Soit $E \subset_{fin} \mathsf{Exp}$ un ensemble fini d'expressions, $w \in \mathbb{N}$ un entier, et $\sigma = (s, h)$, $\sigma' = (s', h')$ deux états mémoires. On dit que σ, σ' sont intensionnellement équivalents, ce que l'on note $s, h \approx_{E,w} s', h'$ si*

1. *pour toute formule $\mathcal{A} \in \mathsf{Stab}_E$ $s, h \models \mathcal{A}$ ssi $s', h' \models \mathcal{A}$.*
2. *si $\Phi \subset \mathsf{Stab}_E$ est l'ensemble des formules atomiques stables $\mathcal{A}$ à expressions dans E satisfaites par σ et σ', $\mathsf{garb}_{s,\Phi}(h) =_w \mathsf{garb}_{s',\Phi}(h')$.*

On veut maintenant montrer que cette équivalence admet une formulation logique. Il s'agit essentiellement de préciser comment formuler la condition de comptage. On commence par vérifier que les formules atomiques stables comptent correctement les cellules qu'elles utilisent, puis on en déduit une formulation logique de $\approx_{E,w}$.

Lemme 5.1.3 *Soit $E \subseteq_{fin} \mathsf{Exp}$ et $\Phi \subseteq \mathsf{Stab}_E$, et $\approx$ une relation d'équivalence sur E. Si $s, h \models \Phi$ et $s', h' \models \Phi$ sont deux modèles minimaux de Φ tels que pour tout $e, e' \in E$, $s, h \models e = e'$ ssi $s', h' \models e = e'$ ssi $e \approx e'$, alors*

$$\sharp \mathsf{dom}(h) \;=\; \sharp \mathsf{dom}(h') \;=\; \sharp \;\left\{\, e \in \mathsf{Exp} \;:\; \exists e'. \{e \hookrightarrow e', \mathsf{alloc}\ e\} \cap \Phi \neq \emptyset \,\right\}_{/\approx}.$$

Preuve: Chaque assertion de la forme $e \hookrightarrow e'$ ou $\mathsf{alloc}\ e$ impose que $[\![e]\!]s \in \mathsf{dom}(h)$ (resp. $[\![e]\!]s' \in \mathsf{dom}(h)'$). On a donc $\mathsf{dom}(h) = \{\, [\![e]\!]s : e \in E_1\}$ et $\mathsf{dom}(h)' = \{\, [\![e]\!]s' : e \in E_1\}$ avec

$$E_1 \;=\; \left\{e \in E \;:\; \exists e'. \{e \hookrightarrow e', \mathsf{alloc}\ e\} \cap \Phi \neq \emptyset \,\right\}.$$

□

Proposition 5.1.4 (Formules caractéristique de $\approx_{E,w}$) *Soit $E \subset_{fin} \mathsf{Exp}$ un ensemble fini d'expressions, $w \in \mathbb{N}$ un entier, et $\sigma = (s, h)$ un état mémoire. Alors il existe une formule classique $\mathcal{A}_\sigma^{(E,w)}$ telle que pour tout état $\sigma' = (s', h')$,*

$$s', h' \;\models\; \mathcal{A}_\sigma^{(E,w)} \quad \textit{ssi} \quad s, h \quad \approx_{E,w} s', h'.$$

Preuve: Soit Φ l'ensemble des formules atomiques stables à expressions dans E satisfaites par s, h. C'est un ensemble finie puisque E est fini. Soit $k = \mathsf{garb}_{s,\Phi}(h)$ et $k' = \sharp \mathsf{dom}(h_\Phi)$. Soit enfin $\approx$ la relation d'équivalence $e \approx e'$ ssi $\{e = e'\} \in \Phi$. On pose

$$\mathcal{A}_\sigma^{(E,w)} \;=\; \bigwedge_{\mathcal{A} \in \Phi} \mathcal{A} \;\wedge \bigwedge_{\mathcal{A} \in \mathsf{Stab}_E \setminus \Phi} \neg \mathcal{A} \;\wedge\; \mathsf{SC}$$

où SC est la formule $\mathsf{size} \geq k + k' \quad \wedge \quad \neg \quad \mathsf{size} \geq k + k' + 1$ si $k < w$ et $\mathsf{SC} = \mathsf{size} \geq w + k'$ sinon. Soit s', h' un état mémoire.

$$
\begin{array}{rl}
s', h' \models \mathcal{A} & \Leftrightarrow \quad s', h' \models \mathsf{SC} \\
& \text{et pour tout } \mathcal{A} \in \mathsf{Stab}_E,\ s', h' \models \mathcal{A} \text{ ssi } s, h \models \mathcal{A} \\
& \Leftrightarrow \quad \sharp\mathsf{dom}(h)' \ =_{w+k'} \ k + k' \\
& \text{et pour tout } \mathcal{A} \in \mathsf{Stab}_E,\ s', h' \models \mathcal{A} \text{ ssi } s, h \models \mathcal{A} \\
& \Leftrightarrow \quad \sharp\mathsf{dom}(h)' - k' \ =_{w} \ k \\
& \text{et pour tout } \mathcal{A} \in \mathsf{Stab}_E,\ s', h' \models \mathcal{A} \text{ ssi } s, h \models \mathcal{A} \\
& \Leftrightarrow \quad \mathsf{garb}_{s',\Phi}(h') \ =_{w} \ \mathsf{garb}_{s,\Phi}(h) \\
& \text{et pour tout } \mathcal{A} \in \mathsf{Stab}_E,\ s', h' \models \mathcal{A} \text{ ssi } s, h \models \mathcal{A} \\
& \qquad (k' = \sharp\mathsf{dom}(h'_\Phi) \text{ d'après le Lemme 5.1.3}) \\
& \Leftrightarrow \quad s, h \ \approx_{E,w} \ s', h'
\end{array}
$$

□

5.2 Minimalisation de la logique séparante

On va maintenant utiliser l'équivalence intensionnelle que l'on vient de définir pour établir le Théorème 5.1.1. Il s'agit essentiellement de démontrer que l'équivalence intensionnelle est correcte vis-à-vis de la logique séparante, ce qui requiert d'étudier tout d'abord certaines propriétés structurelles de l'équivalence intensionnelle. Ensuite, en exploitant la caractérisation logique de l'équivalence intensionnelle, on pourra définir l'encodage de la logique séparante dans le fragment classique. L'idée est de remplacer une formule $\mathcal{A}$, pour E, w choisis de manière adéquate en fonction de $\mathcal{A}$, par l'énumération de toutes les classes d'équivalences de $\approx_{E,w}$ sur lesquelles $\mathcal{A}$ est vérifiée, soit

$$[\![\mathcal{A}]\!] \quad = \bigvee_{s,h\ \models \mathcal{A} \ \bmod \approx_{E,w}} \mathcal{A}_{s,h}^{(E,w)}.$$

Cette disjonction est par ailleurs définie dans ce qui suit (Proposition 5.2.3 et Lemme C.3.2). On détaille maintenant ces idées.

5.2.1 Propriétés structurelles de l'équivalence intensionnelle

On a déjà remarqué que les assertions $e \hookrightarrow e'$ et $\mathsf{alloc}\ e$ sont stables pour l'ordre $h \leq h'$ de restriction sur les tas. Du point de vue de l'opérateur de composition $*$, elles sont aussi localisées : si $\mathcal{A}$ est une assertion de ce type, et si $s, h_1 * h_2 \models \mathcal{A}$, alors soit $s, h_1 \models \mathcal{A}$, soit $s, h_2 \models \mathcal{A}$. De même on montre pour l'équivalence intensionnelle le résultat suivant :

Lemme 5.2.1 (Distributivité de $\approx_{E,w}$) *Pour tous s, h_1, h_2, h' tels que $s, h_1 * h_2 \approx_{E,2w} s, h'$, il existe h'_1, h'_2 tels que $h' = h'_1 * h'_2$, $s, h_1 \approx_{E,w} s, h'_1$ et $s, h_2 \approx_{E,w} s, h'_2$.*

On note toutefois que l'on doit aussi distribuer le pouvoir de compter les cellules mémoire en excès.

Preuve: Soit $\Phi \subseteq \mathsf{Stab}_E$ l'ensemble des formules atomiques stables satisfaites par $s, h_1 * h_2$. On a la partition

$$\Phi \quad = \quad \Phi_1 \ \sqcup \ \Phi_2 \ \sqcup \ \Phi_=$$

où Φ_1 est l'ensemble des formules localisées sur h_1, Φ_2 l'ensemble des formules localisées sur h_2, $\Phi_=$ la théorie égalitaire sur les expressions. Soient h'_{Φ_1} et h'_{Φ_2} les sous-modèles minimaux de h' qui satisfont Φ_1 et Φ_2. Alors $h'_{\Phi_1} \perp h'_{\Phi_2}$ puisque $\Phi_1 \cap \Phi_2 = \emptyset$ par distributivité des formules localisées. Il existe donc une décomposition $h' = h'_{\Phi_1} * h'_{\Phi_2} * h'_\dagger$. Soit $\epsilon \in \{1, 2\}$. On note $h_\epsilon = h_{\Phi_\epsilon} * h_{\dagger\epsilon}$ la décomposition sous-modèle minimal - mémoire en excès. Par définition de $\approx_{E,w}$, on a alors $\sharp\mathsf{dom}(h_{\dagger 1}) + \sharp\mathsf{dom}(h_{\dagger 2}) \ =_{2w} \ \sharp\mathsf{dom}(h)'_\dagger$. On utilise alors le résultat suivant : si n_1, n_2, m sont des entiers tels que $n_1 + n_2 =_{2w} m$, alors il existe m_1, m_2 tels que $m = m_1 + m_2$, $n_1 =_w m_1$ et $n_2 =_w m_2$. En effet, si $n_1 + n_2 < 2w$, le résultat est immédiat pour $m_1 = n_1$ et $n_2 = m_2$, sinon par symétrie on peut supposer $n_1 \geq w$, et en posant $m_1 = w$, $m_2 = m - w$, on obtient le résultat. Soient donc m_1, m_2 tels que $m_1 =_w \sharp\mathsf{dom}(h_{\dagger_1})$, $m_2 =_w \sharp\mathsf{dom}(h_{\dagger_2})$, et $m_1 + m_2 = \sharp\mathsf{dom}(h'_\dagger)$. On considère une décomposition $h'_\dagger = h'_{\dagger 1} * h'_{\dagger 2}$ telle que $h'_{\dagger\epsilon} = m_\epsilon$. On pose alors $h'_1 = h'_{\Phi_1} * h'_{\dagger 1}$ et $h'_2 = h'_{\Phi_2} * h'_{\dagger 2}$. □

Cette propriété traduit le fait que $\approx_{E,w}$ est correct vis-à-vis du connecteur $*$. Pour le connecteur $-\!\!*$, on aura besoin d'une sorte de réciproque du lemme 5.2.1. Cette nouvelle propriété s'interprète par ailleurs en termes de congruence :

Lemme 5.2.2 (Congruence de $\approx_{E,w}$) *Pour tout ensemble d'expressions E et tout entier w, pour tout s, h_1, h_2, h'_1, h'_2, si $s, h_1 \approx_{E,w} s, h'_1$ et $s, h_2 \approx_{E,w} s, h'_2$, alors $s, h_1 * h_2 \approx_{E,w} s, h'_1 * h'_2$.*

Preuve: On vérifie que les deux conditions de la définition 5.1.2 sont bien satisfaites :

- $s, h_1 * h_2$ et $s, h'_1 * h'_2$ satisfont les mêmes formules atomiques stables : celles de la forme $e = e'$ ne dépendent que de s, et les autres sont localisées sur l'une des deux tas donc également satisfaites par hypothèse.
- $\mathsf{garb}_{s,\Phi}(h_1 * h_2) = \mathsf{garb}_{s,\Phi}(h_1) + \mathsf{garb}_{s,\Phi}(h_2)$ et $\mathsf{garb}_{s,\Phi}(h'_1 * h'_2) = \mathsf{garb}_{s,\Phi}(h'_1) + \mathsf{garb}_{s,\Phi}(h'_2)$, et par hypothèse $\mathsf{garb}_{s,\Phi}(h_\epsilon) =_w \mathsf{garb}_{s,\Phi}(h'_\epsilon)$, donc $\mathsf{garb}_{s,\Phi}(h_1 * h_2) =_w \mathsf{garb}_{s,\Phi}(h'_1 * h'_2)$.

□

Cette propriété est essentielle. On retrouvera la propriété de congruence pour les autres équivalences intensionnelles que l'on considèrera (cf. chapitres 7, 8).

5.2.2 Minimalisation

Après l'étude des propriétés structurelles de l'équivalence intensionnelle, on va établir la correction de cette équivalence vis-à-vis de la logique séparante, autrement dit que des états intensionnellement équivalents satisfont les mêmes formules spatiales. On en déduit une traduction des formules de $\mathcal{L}_{Sep}$ dans le fragment classique.

Étant donnée une formule $\mathcal{A}$, on définit son degré de séparation comme l'entier $\mathsf{spl}(\mathcal{A}) \in \mathbb{N} \cup \{-\infty\}$ tel que :

$$\begin{array}{rcl}
\mathsf{spl}(e \mapsto e') & = & 0 \\
\mathsf{spl}(e_1 = e_2) & = & -\infty \\
\mathsf{spl}(\perp) & = & -\infty \\
\mathsf{spl}(\mathsf{emp}) & = & 0 \\
\mathsf{spl}(P_1 \Rightarrow P_2) & = & \max\big(\mathsf{spl}(P_1), \mathsf{spl}(P_2)\big) \\
\mathsf{spl}(P_1 \mathrel{-\!\!*} P_2) & = & \max\big(\mathsf{spl}(P_1), \mathsf{spl}(P_2)\big) \\
\mathsf{spl}(P_1 * P_2) & = & \max\big(\mathsf{spl}(P_1), \mathsf{spl}(P_2)\big) \;+1
\end{array}$$

On note $E(\mathcal{A})$ l'ensemble des expressions qui apparaissent dans $\mathcal{A}$.

Proposition 5.2.3 (Correction) *Pour tous $s, h, s', h', \mathcal{A}$, si $s, h \approx_{E,w} s', h'$ et si $\mathcal{A}$ est une formule telle que $E(\mathcal{A}) \subseteq E$, $2^{\mathsf{spl}(\mathcal{A})} \leq w$, alors*

$$s, h \models \mathcal{A} \quad \Leftrightarrow \quad s', h' \models \mathcal{A}\,.$$

Preuve: On montre (cf annexe C.7, Lemme C.7.2) comment se ramener au cas $s = s'$ par une *délocalisation* des tas h et h'. On suppose donc $s = s'$ et on raisonne par induction sur $\mathcal{A}$:

- si $s, h \models (e \mapsto e')$, alors $s, h \models (e \hookrightarrow e')$, et donc $s, h' \models (e \hookrightarrow e')$. De plus $\mathsf{garb}_{s,\{e \hookrightarrow e'\}}(h) = 0 \;=_1\; \mathsf{garb}_{\{e \hookrightarrow e'\},h'}()$, donc $\sharp\mathsf{dom}(h') = 1$, et finalement $s, h' \models (e \mapsto e')$.
- $s, h \models e_1 = e_2$ ssi $s, h' \models e_1 = e_2$ d'après la définition de $\approx_{E,w}$.
- $s, h \models \mathsf{emp}$ ssi $s, h \models \neg(\mathsf{size} \geq 1)$, c'est à dire ssi $s, h' \models \neg(\mathsf{size} \geq 1)$ puisque $w \geq 1$.
- les cas de $\perp$ et $\mathcal{A}_1 \Rightarrow \mathcal{A}_2$ sont immédiats.
- le cas $\mathcal{A}_1 * \mathcal{A}_2$ découle du lemme 5.2.1.
- supposons que $s, h \models \mathcal{A}_1 \mathrel{-\!\!*} \mathcal{A}_2$; soit h_1 tel que $h_1 \perp h'$ et $s, h_1 \models \mathcal{A}_1$; alors $s, h * h_1 \models \mathcal{A}_2$, et d'après le lemme 5.2.2, $s, h * h_1 \approx_{E,w} s, h' * h_1$, donc par hypothèse d'induction $s, h' * h_1 \models \mathcal{A}_2$ aussi.

□

A ce stade, on a montré que le fragment classique est plus distinctif que l'équivalence intensionnelle (Proposition 5.1.4), et qu'elle-même est plus distinctive que la logique séparante (Proposition 5.2.3). Le fragment classique étant exprimé par la logique séparante, celle-ci est naturellement plus distinctive que son fragment, et on a donc montré que les deux logiques ont le même pouvoir distinctif, c'est à dire que l'équivalence logique est la même pour chaque logique, à savoir :

$$s, h =_L s, h' \quad \text{ssi} \quad h = h'$$

On s'intéresse maintenant au pouvoir expressif du fragment classique.

Lemme 5.2.4 (Précompacité) *Pour tout entier w et pour tout $E \subseteq_{fin} \mathsf{Exp}$ fini, $\approx_{E,w}$ admet un nombre fini de classes d'équivalence.*

Par la suite on utilise ce résultat pour E, w vérifiant les conditions de la proposition 5.2.3.

Preuve: Une classe est entièrement représentée par un ensemble de formules $\Phi \subseteq \mathsf{Stab}_E$ et un entier compris entre 0 et w. Comme E est fini, Stab_E est fini et il n'y a donc qu'un choix fini de Φ. □

On peut désormais établir le théorème 5.1.1 en remarquant que toute formule $\mathcal{A}$ est équivalente à l'énumération (finie) des classes d'équivalence intensionnelle pour lesquelles elle est vérifiée : **Preuve:**(du théorème 5.1.1) On pose

$$\mathcal{A}' \quad = \quad \bigvee_{C \in \mathrm{State}_{/\approx_{E,w}}, C \models \mathcal{A}} \mathcal{A}_C^{(E,w)}.$$

$\mathcal{A}'$ est une formule bien définie du fragment classique (lemme C.3.2 et proposition 5.1.4). On veut montrer que $\mathcal{A} \dashv\vdash \mathcal{A}'$. Tout d'abord, si $s, h \models \mathcal{A}$, soit s_0, h_0 le représentant de sa $\approx_{E,w}$ classe d'équivalence. Par la proposition 5.1.4 ($\Leftarrow$), s, h satisfait la formule caractéristique $\mathcal{A}_{s_0,h_0}^{(E,w)}$, donc $\mathcal{A}'$. Réciproquement, si s, h satisfait l'une de ces formules caractéristiques, alors par la proposition 5.1.4 ($\Rightarrow$) $s, h \approx_{E,w} s_0, h_0$, et ce dernier satisfaisant $\mathcal{A}$, s, h aussi (par la proposition 5.2.3). $\square$

5.2.3 Limitations

On peut mentionner deux limitations dans le résultat obtenu au théorème 5.1.1. Tout d'abord, bien qu'on puisse théoriquement trouver une formule classique équivalente à n'importe quelle formule séparante, la preuve que l'on donne ici ne donne pas complètement la méthode qui permet de construire cette formule. En effet, on n'a pas précisé comment choisir les classes d'équivalence de $\approx_{E,w}$ qui vérifient une formule $\mathcal{A}$[3].

Une limitation plus importante concerne la généralisation de ce résultat à la logique $\mathcal{L}_{Sep}^{\exists}$. En effet, comme on le verra par la suite sur d'autres exemples, le quantificateur $\exists$ enrichit considérablement l'expressivité de la logique séparante.

On considère donc la logique $\mathcal{L}_{Sep}^{\exists}$ et on présente succinctement un exemple[4] de propriété pour laquelle il est nécessaire d'avoir recours à l'adjoint. Considérons les formules suivantes :

$$\begin{array}{rcl}
\mathsf{isole}(x) & \stackrel{\text{def}}{=} & \mathsf{alloc}\ x \ \wedge \neg\mathsf{alloc}\ x+1 \wedge \neg\mathsf{alloc}\ x-1 \\
\mathsf{couple}(x) & \stackrel{\text{def}}{=} & \mathsf{alloc}\ x \ \wedge (\mathsf{alloc}\ x+1 \leftrightarrow \neg\mathsf{alloc}\ x-1) \\
\mathcal{A}_1 & \stackrel{\text{def}}{=} & \forall x.\ \mathsf{alloc}\ x \ \rightarrow \mathsf{isole}(x) \\
\mathcal{A}_2 & \stackrel{\text{def}}{=} & \forall x.\ \mathsf{alloc}\ x \ \rightarrow \mathsf{couple}(x) \\
\mathcal{A}_3 & \stackrel{\text{def}}{=} & \quad \forall x.\ \ \mathsf{alloc}\ x \rightarrow \mathsf{isole}(x) \ \vee \ \mathsf{couple}(x) \\
 & & \wedge\ \forall x.\ \ \big(\mathsf{isole}(x) \ \rightarrow \ \exists y.\ \mathsf{couple}(y) \ \wedge \ (y \hookrightarrow x)\big) \\
 & & \wedge\ \forall y.\ \ \big((\mathsf{couple}(y) \ \rightarrow \ \exists x.\ \mathsf{isole}(x) \ \wedge \ (y \hookrightarrow x)\big) \\
 & & \wedge\ \forall x, y_1, y_2.\ (y_1 \hookrightarrow x) \ \wedge \ (y_2 \hookrightarrow x) \rightarrow y_1 = y_2
\end{array}$$

Une adresse de tas est isolée si elle est allouée mais pas les adresses précédente et suivante. Un tas qui satisfait $\mathcal{A}_1$ ne contient que des adresses isolées. Une adresse couplée est une adresse allouée telle que l'adresse précédente ou la suivante, mais pas les deux, est elle aussi allouée. Ainsi un tas qui satisfait $\mathcal{A}_2$ a un domaine de la forme $\biguplus_{c \in C}\{c, c+1\}$ avec $C \cap (2+C) = \emptyset$; en particulier, le nombre d'adresses allouées dans le tas est pair. Un tas qui satisfait $\mathcal{A}_3$ est un tas

[3]Ce problème est lié à la décidabilité de la logique, voir chapitre 9 ; pour la logique séparante, on peut effectivement calculer une formule classique équivalente, mais ce n'est pas forcément toujours le cas.

[4]Cet exemple est inspiré d'une discussion avec Hongseok Yang.

composé d'adresses soit isolées soit couplées, et à chaque adresse couplée correspond bijectivement une adresse isolée. En particulier, il y a autant d'adresses couplées que d'adresses isolées.

On considère maintenant la formule

$$\mathcal{A}_1 \quad \wedge \quad \neg(\mathcal{A}_2 \multimap\!\!\ast \neg\mathcal{A}_3).$$

Un tas qui satisfait cette formule est constitué d'adresses isolées et on peut y rajouter un tas d'adresses couplées et obtenir autant d'adresses couplées que d'adresses isolées. En rajoutant des adresses couplées, les adresses isolées restent isolées (on ne peut pas créer une série de trois cellules $\{a, a+1, a+2\}$ puisque les adresses sont soit isolées soit couplées). Ainsi on en déduit que le tas initial contenait un nombre pair d'adresses isolées. Réciproquement, si s, h ne contient que des adresses isolées et qu'elles sont en nombre pair, alors on peut ajouter à h un tas h' d'adresses couplées qui pointent bijectivement sur les adresses de h, et de ce fait $s, h \models \mathcal{A}_1 \ \wedge \ \neg(\mathcal{A}_2 \multimap\!\!\ast \neg\mathcal{A}_3)$.

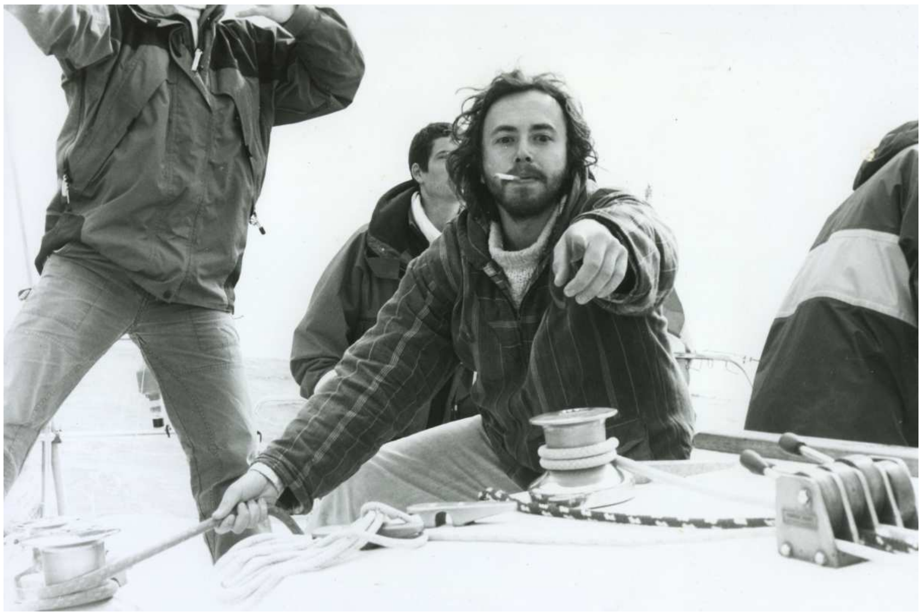
On a donc exprimé par la formule $\mathcal{A}_1 \wedge \neg(\mathcal{A}_2 \multimap\!\!\ast \neg\mathcal{A}_3)$ le fait que le nombre de cellules allouées est pair. En théorie des modèles finis, on montre que la parité de la taille du modèle ne peut pas être exprimée. On pourrait montrer de même que cette propriété n'est pas expressible dans la logique sans $\multimap\!\!\ast$ (voir annexe D.4)). Ainsi, la formule $\mathcal{A}_1 \ \wedge \ \neg(\mathcal{A}_2 \multimap\!\!\ast \neg\mathcal{A}_3)$ constitue un exemple de formule faisant intervenir $\multimap\!\!\ast$ de façon non triviale de telle sorte que cette formule n'admet pas d'équivalent dans la logique sans $\multimap\!\!\ast$.

Pour compléter ce contre-exemple, on aimerait montrer de même que l'élimination de $\ast$ n'est plus possible en présence du quantificateur $\exists$. Toutefois, on ne connaît pas de contre-exemple ou de preuve d'élimination à l'heure actuelle.

5.2.4 Conclusion

L'étude de l'expressivité de la logique séparante montre donc qu'elle correspond à celle d'une logique classique et que les formules atomiques de ces logiques s'encodent mutuellement. Ce résultat ne signifie pas que la logique classique peut remplacer la logique séparante comme langage d'assertion en sémantique dénotationnelle, les travaux de O'Hearn et al. l'illustrent par ailleurs suffisamment bien. Ce résultat est plutôt une sorte de réciproque du discours introductif de [Rey02], où il est dit que la logique séparante exprime de façon compacte et naturelle des propriétés coûteuses à exprimer en logique classique.

Au-delà du résultat proprement dit, cette première étude a permis d'introduire certains des problèmes d'expressivité et des techniques de preuves que l'on rencontrera maintenant dans le cadre des logiques spatiales pour la concurrence. La première étape de l'étude de l'expressivité de la logique séparante a été d'en définir les observations dérivées afin d'en trouver une renormalisation correcte. C'est cette étape que l'on reprend maintenant pour les logiques spatiales pour la concurrence, où la notion d'observation dérivée se traduit par des *jeux contextuels*.



Chapitre 6

Jeux contextuels

Un agent n'exprime pas toutes ses potentialités lorsque le contexte dans lequel il se trouve n'offre pas les interactions nécessaires ; pour les révéler, il faut l'insérer dans une situation nouvelle. C'est sur ce credo de directeur de ressources humaines que se fonde l'étude de l'expressivité contextuelle.

Le seul type d'action auquel correspond un connecteur ($\Diamond$) dans les logiques spatiales que l'on considèrera ici est l'action τ, c'est à dire l'évolution du système par une interaction interne. On ne peut pas en revanche observer les actions qui composent ces interactions. La mise en contexte, qui s'exprime en logique spatiale pour la concurrence par les connecteurs adjoints, et en particulier l'implication spatiale $\triangleright$, permet de faire interagir l'agent étudié avec des agents complémentaires et de révéler ainsi des potentialités d'action de cet agent que son évolution cloisonnée n'aurait pas fait apparaître. De façon un peu surprenante, c'est donc par l'élargissement du système que l'on en fait une observation plus fine.

On parlera par la suite de *jeu contextuel*. Le jeu se déroule de la façon suivante. Un agent testeur extérieur au système est introduit dans le système de l'agent testé, puis le système testeur/testé subit une altération suivant un scénario donné, qu'il réalise correctement à condition que l'agent testé puisse réaliser l'action que l'on cherche à déceler. Sous forme logique, ce jeu s'exprime de la façon suivante :

$$\mathsf{testeur} \ \triangleright\ \Diamond\ \mathsf{scenario}.$$

Une première illustration de jeu contextuel a déjà été vue à l'exemple 3.2.6 où l'on tente de définir un raisonnement arrière pour la transition $P \xrightarrow{an} P'$. C'est aussi sur cette idée de jeu contextuel que se fonde la caractérisation de la bisimulation en terme de congruence à barbe.

Dans ce chapitre on développe différentes applications des jeux contextuels. Essentiellement, on s'attache à exprimer les interactions élémentaires des algèbres de processus que l'on considère, ce qui revient à encoder les logiques modales à l'intérieur des logiques spatiales.

On considère tout d'abord le calcul des Ambients publics pour lequel on sait déjà observer le constructeur d'ambient et obtenir des mises en contexte complexes par les adjoints $\triangleright$ et @. Pour cette logique riche, les propriétés expressibles par des jeux contextuels sont l'observation de modalités de mouvement (et de communication, voir annexe A), mais aussi des propriétés plus intrinsèques du système comme sa finitude (absence de la réplication infinie ! dans le processus), ou encore la présence d'un nom libre (connecteur ©). Ce cas d'étude porte donc seulement sur le calcul des Ambients et une logique spécifique

On étend ensuite ces résultats à une logique non spécifique, la logique unifiée $\mathcal{L}_U$, sur différents calculs (π-calcul synchrone, asynchrone, et Ambients). On considère par ailleurs l'encodage des modalités en logique unifiée tout d'abord fortes, puis faibles. L'utilisation de noms marqueurs pour caractériser le testeur est alors essentielle, ce qui rend l'encodage dans le cas fort relativement simple ; dans le cas faible, cet encodage est plus complexe et passe par des testeurs particuliers appelés *séquentiels* (ou *thread* en anglais).

6.1 Jeux contextuels dans le calcul des Ambients publics

On considère ici le calcul des Ambient publics MA, et la logique des Ambients sans révélation de noms DAL. On commence par montrer que l'on peut représenter les instructions de mouvement dans la logique, c'est à dire donner des formules qui caractérise la présence d'une capabilité donnée en tête de ces modèles. On exhibe ensuite d'autres jeux contextuels exprimant des propriétés intrinsèques du processus telle que sa finitude ou la présence de noms libres.

6.1.1 Modalités faibles de mouvement

La première étape dans la reconnaissance d'une instruction dans un système consiste à reconnaître un processus gardé, c'est à dire de la forme $\mathsf{cap}.P$. Pour cela, il s'agit de dire que le système est atomique, mais aussi que ce n'est pas une localité $n[P]$. Cela s'exprime donc logiquement ainsi :

$$\boxed{\mathsf{1Instr} \stackrel{\text{def}}{=} 1 \quad \wedge \quad \neg\exists m.m[\top].}$$

Sachant qu'on a une instruction, on souhaite exprimer plus précisément quelle est cette instruction. Dans la logique, le connecteur $\Diamond$ permet d'observer une évolution par interaction, qui correspond à l'exécution d'une instruction dans un environnement. Le système que l'on regarde après l'avoir sélectionné avec la formule $\mathsf{1Instr}$ ne comporte pas l'environnement d'exécution de l'instruction. Mais on va pouvoir rajouter cet environnement en utilisant les adjoints. Prenons l'exemple de l'instruction $\mathsf{out}\,n$. Si elle est placée dans une imbrication de deux boîtes $n[a[\mathsf{out}\,n]]$, on peut observer la réduction $n[a[\mathsf{out}\,n]] \longrightarrow n[\mathbf{0}]|a[\mathbf{0}]$. Cette évolution caractérise précisément l'exécution d'une instruction $\mathsf{out}\,n$. On peut faire de même avec toutes les instructions de mouvemement.

$$\boxed{\begin{array}{lcl}
\mathsf{In}\ n & \stackrel{\text{def}}{=} & \mathsf{1Instr} \wedge \text{И}m.\ \big(n[0] \rhd \Diamond n[m[\top]]\big)@m \\
\mathsf{Out}\ n & \stackrel{\text{def}}{=} & \mathsf{1Instr} \wedge \text{И}m.\ \big(\Diamond n[0]|m[\top]\big)@n@m \\
\mathsf{Open}\ n & \stackrel{\text{def}}{=} & \mathsf{1Instr} \wedge \text{И}m.\ n[m[0]] \rhd\ \Diamond m[0]|\top
\end{array}}$$

Détaillons par exemple l'instruction $\mathsf{in}\,n$[1] :

$$\begin{array}{rcll} P \models \mathsf{In}\ n & \Leftrightarrow & P \equiv \mathsf{cap}.P' \text{ et } m[P]|n[\mathbf{0}] \models \Diamond n[m[\top]] & (m \text{ quelconque}) \\ & \Leftrightarrow & P \equiv \mathsf{cap}.P' \text{ et } m[P]|n[\mathbf{0}] \ \longrightarrow\ n[m[P']] & \\ & \Leftrightarrow & P \equiv \mathsf{cap}.P' \text{ et } \mathsf{cap} = \mathsf{in}\,n & \end{array}$$

On aimerait maintenant être plus précis. Si l'on a su caractériser un processus de la forme $\mathsf{in}\,n.P$, on voudrait aussi pouvoir parler de la continuation P comme dans les logiques modales. On peut facilement adapter les formules précédentes pour imposer une condition $\mathcal{A}$ sur le terme gelé P qui est activé après exécution de l'instruction. Cela donne :

$$\boxed{\begin{array}{lcl} \langle\langle \mathsf{in}\,n\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr} \ \wedge\ ИIm.\ \big(n[0]\ \triangleright\ \Diamond\ n[m[\mathcal{A}]]\big)@m \\ \langle\langle \mathsf{out}\,n\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr} \ \wedge\ ИIm.\ \big((\Diamond m[\mathcal{A}]|n[0])@n\big)@m \\ \langle\langle \mathsf{open}\,n\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr} \ \wedge\ ИIm.\ \big(\ n[m[0]]\ \triangleright\ \Diamond\ m[0]|\mathcal{A}\ \big) \end{array}}$$

Pour caractériser ces instructions, on les a fait s'exécuter dans leur environnement. Ainsi pour caractériser $\mathsf{in}\,n$, on place l'instruction dans un ambient m et on met celui-ci en parallèle avec un n, ce qui correspond à la situation $m[\mathsf{in}\,n.P]|n[\mathbf{0}]$, et on demande au processus d'évoluer vers la configuration $n[m[P]]$. Mais on s'est placé dans une logique où la modalité de temps $\Diamond\mathcal{A}$ est faible, ce qui fait qu'on ne peut pas savoir à quel moment l'exécution s'arrête, et on ne peut ainsi pas forcer le processus a s'arrêter dès que l'instruction a été exécutée. De ce fait, ce n'est pas nécessairement le terme gelé P qui satisfait la condition $\mathcal{A}$, mais de façon générale un terme issu de cette exécution. Pour le cas de l'instruction $\mathsf{open}\,n$, les seules réductions qui peuvent apparaître dans le processus $m[\mathbf{0}]|P$ sont les réductions propres à P, puisqu'en général $m \notin \mathsf{fn}(P)$. Mais pour le cas de l'instruction $\mathsf{in}\,n$, le processus $n[m[P]]$ peut évoluer soit par réduction interne de P, soit par sortie de l'ambient m de n. Et par suite, m peut à nouveau rentrer puis sortir de n. Par exemple, les processus $\mathsf{in}\,n.\mathbf{0}$ et $\mathsf{in}\,n.\mathsf{out}\,n.\mathsf{in}\,n.\mathbf{0}$ satisfont tous les deux la formule $\langle\langle \mathsf{in}\,n\rangle\rangle 0$ puisque dans les deux cas $m[P]|n[\mathbf{0}] \ \Rightarrow\ n[m[\mathbf{0}]]$.

Par analogie avec le π-calcul, notons $P \stackrel{\mathsf{cap}}{\to} P'$ si $P \equiv \mathsf{cap}.P_1|P_2$ et $P' \equiv P_1|P_2$, et $P \stackrel{\mathsf{cap}}{\Longrightarrow} P'$ si $P \Rightarrow \mathsf{cap} \Rightarrow P'$. On note alors $P \xRightarrow{(\mathsf{out}\,n,\mathsf{in}\,n)^*} Q$ si il existe une suite de processus $(P_i)_{0\geq i\geq n}$ telle que

$$P \Rightarrow P_0 \stackrel{\mathsf{out}\,n}{\Longrightarrow} P_1 \stackrel{\mathsf{in}\,n}{\Longrightarrow} P_2 \stackrel{\mathsf{out}\,n}{\Longrightarrow} P_3 \ldots \stackrel{\mathsf{in}\,n}{\Longrightarrow} P_n = Q$$

et on parlera de *bégaiement*. De même $P \xRightarrow{(\mathsf{in}\,n,\mathsf{out}\,n)^*} Q$ représente une réduction bégayante qui commence par un $\stackrel{\mathsf{in}\,n}{\Longrightarrow}$.

Dans ce cadre, on peut interpréter les formules introduites précédemment :

Lemme 6.1.1 *Pour tout processus P et toute formule $\mathcal{A}$,*

- *$P \models \langle\langle \mathsf{in}\,n\rangle\rangle\mathcal{A}$ ssi P est de la forme $\mathsf{in}\,n.P'$ et il existe P'' tel que $P' \xRightarrow{(\mathsf{out}\,n,\mathsf{in}\,n)^*} P''$ et $P'' \models \mathcal{A}$;*

[1] On notera qu'a ce point l'interprétation forte ou faible de $\Diamond$ n'a pas d'importance. En sémantique forte pour $\Diamond$, l'encodage de modalité est même beaucoup facilité, tandis qu'en modalité faible on voit apparaître le phénomène de bégaiement (voir plus loin).

– $P \models \langle\langle \mathsf{out}\, n \rangle\rangle \mathcal{A}$ *ssi* P *est de la forme* $\mathsf{out}\, n.P'$ *et il existe* P'' *tel que* $P' \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*} P''$ *et* $P'' \models \mathcal{A}$ *;*
– $P \models \langle\langle \mathsf{open}\, n \rangle\rangle \mathcal{A}$ *ssi* P *est de la forme* $\mathsf{open}\, n.P'$ *et il existe* P'' *tel que* $P' \Rightarrow P''$ *et* $P'' \models \mathcal{A}$ *;*

Notation : On adoptera la notation générique $\langle\langle \mathsf{cap} \rangle\rangle \mathcal{A}$ pour ces modalités, et de même on notera indifféremment $\stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow}$ la relation $\Rightarrow$ pour $\mathsf{cap} = \mathsf{open}\, n$, la relation $\xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*}$ pour $\mathsf{cap} = \mathsf{in}\, n$, et la relation $\xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*}$ pour $\mathsf{cap} = \mathsf{out}\, n$, de telle sorte que le lemme précédent s'énonce : $P \models \langle\langle \mathsf{cap} \rangle\rangle \mathcal{A}$ ssi il existe P' tel que $P \equiv \mathsf{cap}.P'$ et $P' \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} \models \mathcal{A}$.

Ces formules sont à comparer à celles de logiques modales comme la logique de Hennessy-Milner. Il existe une différence majeure entre ces deux logiques : en logique des Ambients, les formules sont basées sur le modèle intensionnel (on définit des modalités de *capabilités*), tandis qu'en logique de Hennessy-Milner on se place au niveau extensionnel par des modalités d'*actions*[2].

Il y a deux raisons à ce choix de l'intensionalité pour les modalités de la logique des Ambients et des autres logiques spatiales. D'une part on cherchera effectivement par la suite à caractériser l'intensionalité de la logique, et donc à caractériser des capabilités et non des actions, mais aussi d'autre part les relations $\xrightarrow{\mathsf{cap}}$ ne suffisent pas à définir la réduction en Ambients et ne sont pas à ce titre un équivalent valable des actions $\xrightarrow{an}$ et $\xrightarrow{\overline{a}b}$ en π-calcul. Enfin on peut mentionner qu'on ne connaît pas d'encodage de la modalité correspondante à la relation $\xrightarrow{\mathsf{open}\, n}$.

Une conséquence indirecte de l'intensionalité de ces modalités est que la formule $\neg\langle\langle \mathsf{cap} \rangle\rangle \neg\mathcal{A}$ n'exprime pas la variante $[\![\mathsf{cap}]\!]\mathcal{A}$, qui force toutes les réductions à satisfaire $\mathcal{A}$ (il suffit de prendre un processus non gardé par une instruction). Pour exprimer la modalité duale $[\![\mathsf{cap}]\!]\mathcal{A}$, on posera la définition

$$[\![\mathsf{cap}]\!]\mathcal{A} \quad \stackrel{\text{def}}{=} \quad \langle\langle \mathsf{cap} \rangle\rangle \top \ \wedge\ \neg\langle\langle \mathsf{cap} \rangle\rangle \neg\mathcal{A}$$

pour laquelle on a

$$P \models [\![\mathsf{in}\, n]\!]\mathcal{A} \quad \text{ssi} \quad \exists.\ P'.\ P \equiv \mathsf{in}\, n.P' \ \wedge\ \forall P''.\ P' \xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*} P'' \text{ et } P'' \models \mathcal{A}$$

et de même pour les autres capabilités.

On n'étudiera pas ici le cas des communications, mais par des formules analogues on saurait caractériser les processus messages et réception (voir annexe A.2).

On notera enfin que la formule

$$\mathcal{F}_{\mathsf{cap}.\mathbf{0}} \stackrel{\text{def}}{=} [\![\mathsf{cap}]\!]0$$

n'admet comme modèle que le processus $\mathsf{cap}.\mathbf{0}$, ce qui est l'une de nos premières formules caractéristiques (on notera par la suite $\mathcal{F}_P$ la formule caractéristique de P).

[2]Les modalité de capabilité $\langle\langle \mathsf{cap} \rangle\rangle$ sont intensionnelles au sens où elles ne sauraient être définies à partir du simple graphe de transitions étiquetées, puisqu'elles imposent au processus d'être gardé.

6.1.2 Formules pour les instructions persistantes

Afin de préparer le terrain à l'étude de l'intensionalité de la logique des Ambients, on veut exprimer l'opérateur de réplication par des formules logiques, puisqu'alors on aura traduit au niveau logique toutes les constructions du calcul.

La réplication s'interprète dynamiquement comme la persistance d'une instruction au-delà de son exécution. On peut donc chercher à caractériser une instruction répliquée par le fait qu'après un nombre quelconque d'exécutions de cette instruction, elle reste toujours disponible. Pour considérer une exécution de longueur arbitrairement longue, il suffit de considérer plus généralement *toutes* les exécutions du processus, d'où l'usage de la modalité $\Box\mathcal{A}$. On place ensuite l'instruction répliquée dans un contexte qui lui fournit un nombre non fixé d'interactions conjuguées (on utilise la même idée pour caractériser un ambient répliqué).

On suppose donc que l'on a déjà trouvé une formule $\mathcal{A}$ qui caractérise les processus de la forme $\mathsf{cap}.P$, et on va construire une formule qui caractérise (à dépliage de ! près) les processus de la forme $!\mathsf{cap}.P$ avec $\mathsf{cap}.P$ qui satisfait $\mathcal{A}$. La formule pour la réplication contient une première partie $\mathcal{A}^\omega$ qui garantit que le modèle contient un certain nombre de copies de modèles de $\mathcal{A}$, mais rien d'autre. La seconde partie exprime la persistance de l'une des copies.

La formule pour la persistance dépend de l'instruction cap : dans chaque cas, l'environnement de test reprend les instructions et agents conjugués avec l'instruction cap. Ainsi pour caractériser un $\mathsf{in}\, n$ répliqué, on va mettre P dans l'environnement $m[P|!\mathsf{out}\, n]|n[\mathbf{0}]$ et vérifier que m peut rentrer une infinité de fois dans n. Pour $\mathsf{out}\, n$, on procède de façon symétrique, pour $\mathsf{open}\, n$, on prend l'environnement de test $P|!n[\mathbf{0}]$ et on vérifie que l'on peut ouvrir des ambients n une infinité de fois, et symétriquement pour l'ambient répliqué. On rappelle que l'on note $\mathcal{A}^\forall$ la formule $\mathcal{A}||\bot$ (voir chapitre 4).

$$
\boxed{
\begin{array}{lcl}
\mathcal{A}^\omega & \stackrel{\text{def}}{=} & \big(1\rightarrow\mathcal{A}\big)^\forall \\
\mathsf{Rep}_{\mathsf{in}\, n}(\mathcal{A}) & \stackrel{\text{def}}{=} & \mathcal{A}^\omega \ \wedge \\
 & & \text{И} m.\ (\mathcal{F}_{\mathsf{out}\, n.0})^\omega \ \triangleright \ \big(\ n[0] \ \triangleright \ \Box\Diamond\ (\ n[m[\mathcal{A} \mid \top]]\ )\big)@m \\
\mathsf{Rep}_{\mathsf{out}\, n}(\mathcal{A}) & \stackrel{\text{def}}{=} & \ \mathcal{A}^\omega \ \wedge \\
 & & \text{И} m.(\mathcal{F}_{\mathsf{in}\, n.0})^\omega \ \triangleright \ \big(\ n[0] \ \triangleright \ \Box\ \Diamond(\ m[\mathcal{A} \mid \top]|n[0]\ )\big)@m \\
\mathsf{Rep}_{\mathsf{open}\, n}(\mathcal{A}) & \stackrel{\text{def}}{=} & \ \mathcal{A}^\omega \ \wedge \ (n[0])^\omega \ \triangleright \ \Box\ (\ \mathcal{A} \mid \top\ ) \\
\mathsf{Rep}_{n[]}(\mathcal{A}) & \stackrel{\text{def}}{=} & \ (n[\mathcal{A}])^\omega \ \wedge \ (\mathcal{F}_{\mathsf{open}\, n.0})^\omega \ \triangleright \ \Box\ (n[\mathcal{A}] \mid \top\ )
\end{array}
}
$$

Détaillons par exemple le cas du $\mathsf{in}\, n$. On fait les hypothèses suivantes sur une formule $\mathcal{A}$:

1. tous les modèles de $\mathcal{A}$ sont de la forme $\mathsf{in}\, n.P$
2. si $\mathsf{in}\, n.P \models \mathcal{A}$ et si $\mathsf{in}\, n.Q$ est un sous-terme de P, alors $\mathsf{in}\, n.Q \not\models \mathcal{A}$

La première hypothèse signifie que $\mathcal{A}$ reconnaît l'instruction $\mathsf{in}\, n$, et on souhaite donc que $\mathsf{Rep}_{\mathsf{in}\, n}(\mathcal{A})$ reconnaisse l'instruction persistante $!\mathsf{in}\, n$. La deuxième hypothèse va nous permettre de vérifier que la réplication apparaît bien sur la première instruction $\mathsf{in}\, n$ rencontrée. Cette dernière condition sera par ailleurs systématiquement satisfaite dans la construction de formules caractéristiques (cf. chapitre 7).

Soit P un processus quelconque ; on a alors :

$$\begin{array}{ll} & P \models \mathsf{Rep}_{\mathsf{in}\, n}(\mathcal{A}) \\ \Leftrightarrow & P \equiv (!)\mathsf{in}\, n.P_1|(!)\mathsf{in}\, n.P_2|\dots|(!)\mathsf{in}\, n.P_k \quad , \quad \mathsf{in}\, n.P_i{\models}\mathcal{A} \\ & \text{et } m[P|(!)\mathsf{out}\, n|\mathsf{out}\, n|\dots|\mathsf{out}\, n]|n[\mathbf{0}] \models \Box\Diamond\big(n[m[\mathcal{A}|\top]]\big) \quad (m \text{ quelconque}) \end{array}$$

et le processus $m[P|(!)\mathsf{out}\, n|\mathsf{out}\, n|\dots|\mathsf{out}\, n]|n[\mathbf{0}]$ constitue l'environnement de test du processus P (on introduit ici la notation $(!)\mathsf{cap}.P$ pour un "motif" de processus qui peut tout aussi bien recouvrir $\mathsf{cap}.P$ que $!\mathsf{cap}.P$).

Si dans P l'un des $\mathsf{in}\, n.P_i$ est répliqué, alors tout P' auquel se réduit

$$m[P|(!)\mathsf{out}\, n|\mathsf{out}\, n|\dots|\mathsf{out}\, n]|n[\mathbf{0}]$$

contient ce $\mathsf{in}\, n.P_i$ répliqué à l'intérieur de m, donc il existe P'' tel que $P' \Rightarrow P''$ tel que m est rentré dans n et dans m il reste toujours un composant qui satisfait $\mathcal{A}$, ie $P''{\models}n[m[\mathcal{A}|\top]]$, donc $P'{\models}\Diamond n[m[\mathcal{A}|\top]]$, et ce pour tout P', donc

$$m[P|(!)\mathsf{out}\, n|\mathsf{out}\, n|\dots|\mathsf{out}\, n] \mid n[\mathbf{0}] \;\models\; \Box\Diamond\, n[m[\mathcal{A}|\top]].$$

Inversement, si aucun des $\mathsf{in}\, n$ n'est répliqué, par des aller-retours de m dans n on peut exécuter tous ces $\mathsf{in}\, n$, et la réduction

$$m[P|\mathsf{out}\, n|\mathsf{out}\, n|\dots|\mathsf{out}\, n] \mid n[\mathbf{0}] \quad\Rightarrow\quad m[P_1|\dots|P_k] \mid n[\mathbf{0}]$$

permet d'épuiser toutes les capabilités $\mathsf{in}\, n$ présentes au premier niveau. On souhaite conclure que m ne peut pas rentrer dans n, mais ce n'est pas vrai a priori, puisque les P_i peuvent eux aussi contenir des $\mathsf{in}\, n$, voire des $\mathsf{in}\, n$ répliqués. Mais si malgré tout $m[P_1|\dots|P_k] \mid n[\mathbf{0}]$ sait se réduire à un processus $n[m[P']]$, P' contient des sous-termes des $\mathsf{in}\, n.P_i$, qui, par l'hypothèse 2 ci-dessus, ne peuvent pas satisfaire $\mathcal{A}$, donc $n[m[P']] \not\models n[m[\mathcal{A}|\top]]$, et ce pour tout P'. On en déduit que $m[P_1|\dots|P_k] \mid n[\mathbf{0}] \not\models \Diamond n[m[\mathcal{A}|\top]]$, et finalement

$$m[P|\mathsf{out}\, n|\mathsf{out}\, n|\dots|\mathsf{out}\, n] \mid n[\mathbf{0}] \quad\not\models\; \Box\Diamond n[m[\mathcal{A}|\top]].$$

Le même raisonnement s'applique aux autres instructions (voir annexe A.2). Pour la suite, on retiendra le résultat suivant, où l'on a encore renforcé[3] l'hypothèse 2 faite sur $\mathcal{A}$ en se basant sur les notions de degré de séquentialité et de profondeur introduites au Chapitre 1.

Lemme 6.1.2 *Soit $\mathcal{A}$ une formule dont tous les modèles ont le même degré de séquentialité, et soit* cap *une capabilité donnée. On suppose de plus que tous les modèles de $\mathcal{A}$ sont de la forme* $\mathsf{cap}.P$. *Alors $P{\models}\mathsf{Rep}_{\mathsf{cap}}(\mathcal{A})$ ssi P est de la forme*

$$!\mathsf{cap}.P_1|(!)\mathsf{cap}.P_2|\dots|(!)\mathsf{cap}.P_k$$

avec $\mathsf{cap}.P_i{\models}\mathcal{A}$ *pour tout i (les ! entre parenthèses sont facultatifs).*

Lemme 6.1.3 *Soit $\mathcal{A}$ une formule dont tous les modèles ont le même degré de profondeur. Alors $P{\models}\mathsf{Rep}_{n[]}(\mathcal{A})$ ssi P est de la forme*

$$!n[P_1]|(!)n[P_2]|\dots|(!)n[P_k]$$

avec $P_i{\models}\mathcal{A}$ pour tout i.

[3]En effet, si $\mathsf{in}\, n.Q$ est un sous-terme strict de $\mathsf{in}\, n.P$, alors $\mathsf{ds}(\mathsf{in}\, n.Q) < \mathsf{ds}(\mathsf{in}\, n.P)$ et donc $\mathsf{in}\, n.Q$ n'est pas un modèle de $\mathcal{A}$.

Ces résultats montrent que dans une certaine mesure on peut exprimer dans la logique l'opérateur de réplication $!-$ du calcul des Ambients. Avec les opérateurs $\mathbf{0}$, $n[-]$ et $-|-$ qui sont primitifs à la logique, et les capabilités $\mathsf{cap}.-$, on a donc vu comment exprimer l'intégralité du calcul sous l'angle logique. Ceci permettra par la suite de dériver des formules caractéristiques des processus. Mais pour l'instant, on souhaite présenter deux autres résultats qui illustrent l'utilisation des jeux contextuels dans la caractérisation de propriétés syntaxiques des processus.

6.1.3 Autres jeux contextuels

On va maintenant montrer comment par une mise en contexte assez simple on peut faire jouer toutes les capabilités d'un processus de façon à en ronger la substance peu à peu. Ce procédé a deux applications immédiates. D'une part, il permet de caractériser les termes finis par rapport à ceux qui contiennent des éléments persistants ; en effet les termes finis sont ceux que l'on peut ronger jusqu'au $\mathbf{0}$, les termes infinis gardant leurs éléments persistants. D'autre part, il permet de ramener en surface n'importe quel sous-terme par usure des couches supérieures. On peut alors quantifier sur l'ensemble des sous-termes du processus et vérifier l'existence d'un sous-terme qui exhibe un nom donné n en surface. Un tel sous-terme existe si et seulement si le terme initial contient n à une profondeur arbitraire, autrement dit si n est un nom présent. On peut alors exprimer l'occurrence d'un nom par une seule formule (que l'on a notée ©n au Chapitre 4).

Commençons par exprimer la finitude. On a vu précédemment que par des mises en contextes on peut permettre au processus d'exercer ses capabilités. La mise en contexte dépend toutefois de la capabilité que l'on considère. Ici on cherche une mise en contexte qui recouvre toutes les situations requises. Pour les mouvements $\mathsf{in}\, n$ et $\mathsf{out}\, n$, il faut que l'on ait mis le processus à l'intérieur d'un ambient m et qu'en parallèle ou autour se trouve un ambient n. Pour les $\mathsf{open}\, n$ ou les $n[.]$, on doit plus simplement placer en parallèle l'élément conjugué (ce qu'explique l'exemple ci-dessous). On représente ces deux situations par la mise en contexte de P

$$m\big[P|P_{inout}|P_{openamb}\big] \mid P'_{inout}$$

où

- P_{inout} contient les capabilités *in* et *out* de mouvement inverse de celles de P,
- P'_{inout} contient des ambients objets de ces mêmes capabilités *in* et *out*,
- $P_{openamb}$ contient les ambients et *open* conjugués de ceux de P.

Par exemple, pour le terme $P = \mathsf{in}\, n.a[\mathbf{0}]$, on aura la mise en contexte

$$m[P \mid \mathsf{out}\, n.\mathbf{0}|\mathsf{open}\, a.\mathbf{0}] \mid n[\mathbf{0}].$$

Si l'on exécute P dans un tel contexte, on pourra user tous ses éléments et se réduire à $m[\mathbf{0}]|n[\mathbf{0}]$. En rajoutant un processus P_{garb} qui contient tous les *open* conjugués de ces ambients, on aura donc

$$m\big[P|P_{inout}|P_{openamb}\big] \mid P'_{inout}|P_{garb} \quad \Rightarrow \quad \mathbf{0}$$

pour un P fini, alors que pour un P infini qui contient un sous-terme $!P'$ avec $P' \not\equiv \mathbf{0}$, $!P'$ restera toujours un sous-terme de tout réduit de P dans un contexte quelconque[4]. On en déduit le lemme suivant :

Lemme 6.1.4 *Un processus P est fini ssi il existe P_1, P_2 et un nom $m \notin \mathsf{fn}(P)$ tel que $m[P|P_1]|P_2 \;\Rightarrow\; \mathbf{0}$.*

Cette propriété particulièrement simple s'exprime par la logique de la façon suivante :

Corollaire 6.1.5 *Un processus P satisfait la formule*

$$\boxed{\text{И}m.\top \;\blacktriangleright\; \left(\top \blacktriangleright \Diamond\mathbf{0}\right)@m}$$

ssi P est fini.

Voyons maintenant comment exprimer l'occurrence de nom. Le procédé de rognage de terme fait intervenir les termes $P_{inout}, P'_{inout}, P_{openamb}$ qui sont *plats*, autrement dit dans ces termes tout préfixe, capabilité ou ambient, est immédiatement suivi de $\mathbf{0}$. Ce type de processus se caractérise facilement par la formule logique

$$\mathsf{plat} \;\stackrel{\text{def}}{=}\; \left(\exists m.\ \mathcal{F}_{\mathsf{in}\,m.0} \;\vee\; \mathcal{F}_{\mathsf{out}\,m.0} \;\vee\; \mathcal{F}_{\mathsf{open}\,m.0} \;\vee m[0]\right)^{\omega}$$

Pour les processus plats, déceler la présence d'un nom libre revient à examiner la surface du processus. Cette opération est réalisable dans la logique à partir des formules pour les capabilités définies précédemment. En effet, la formule

$$©^1 n \;\stackrel{\text{def}}{=}\; \left(\langle\langle \mathsf{in}\, n\rangle\rangle.\top \;\vee\; \langle\langle \mathsf{out}\, n\rangle\rangle.\top \;\vee\; \langle\langle \mathsf{open}\, n\rangle\rangle.\top \;\vee\; n[\top]\ \right) \;\Big|\; \top$$

caractérise les processus dans lesquels le nom n est libre et apparaît en surface.

Pour avoir n libre dans P, il suffit donc de mettre P dans un contexte de rognage dans lequel on s'est assuré qu'il n'y a pas n (on ne rogne que les portions de P qui enterrent l'occurrence de n), puis de tester l'occurrence de n en surface. D'un point de vue logique, cela donne :

$$©n \;\stackrel{\text{def}}{=}\; \text{И}m.\left(\mathsf{plat} \wedge \neg ©^1 n\right) \;\blacktriangleright\; \Big(\left(\mathsf{plat} \wedge \neg ©^1 n\right) \;\blacktriangleright\; \Diamond\ m[©^1 n]\ \Big)@m$$

Le lemme suivant garantit l'interprétation de cette formule.

Lemme 6.1.6 *Soit P un processus et n un nom. Alors $n \in \mathsf{fn}(P)$ ssi il existe $m \notin \mathsf{fn}(P)$, P_1, P_2, P' tels que*
- *P_1, P_2 sont plats et $n \notin \mathsf{fn}(P_1) \cup \mathsf{fn}(P_2)$;*
- *$m[P|P_1]|P_2 \;\Rightarrow\; m[P']$*
- *n apparaît en surface dans P'*

Corollaire 6.1.7 *$P \models ©\ n$ ssi $n \in \mathsf{fn}(P)$*

On notera toutefois que par rapport aux encodages précédents de la formule pour l'occurrence de nom, le jeu contextuel présenté ici reste relativement compliqué.

[4]Ceci pourrait éventuellement être mis en défaut si le calcul contenait un opérateur de choix $P + Q$. Dans ce cas, ce résultat serait certainement à revoir.

6.2 Encodage de modalités en logique unifiée

On considère maintenant la logique unifiée vue au Chapitre 3. Bien que cette logique soit beaucoup moins riche que la logique des Ambients, on va montrer qu'elle est suffisante pour exprimer les instructions du modèle auquel on l'applique, que ce soit le π-calcul synchrone ou asynchrone, ou dans une certaine mesure les Mobile Ambients.

6.2.1 Modalités fortes pour le π-calcul et les Ambients

On se place donc dans le cadre de $\mathcal{L}_U$ pour la suite de cette section. On commence par interpréter la modalité $\Diamond\mathcal{A}$ au sens fort, autrement dit $P \models \Diamond\mathcal{A}$ si il existe P' tel que $P \longrightarrow P'$ et $P' \models \mathcal{A}$. Cette interprétation permet de dériver des formules qui expriment les capacités avec peu de complications techniques. La démarche est toujours la même : on cherche tout d'abord à définir une notion d'atome, puis on s'en sert comme testeurs dans un jeu contextuel, et enfin on reconnaît les capabilités du calcul.

Regardons tout d'abord le cas du π-calcul synchrone. On considère les formules suivantes :

$$\boxed{\begin{array}{rcl}
\mathsf{1Instr} & \stackrel{\text{def}}{=} & Ии n.\ \neg n \circledR \copyright n \ \wedge\ 1 \\
\mathsf{test}(n,m) & \stackrel{\text{def}}{=} & \mathsf{1Instr}\ \wedge\ \copyright n\ \wedge\ \copyright m\ \wedge\ \big(\mathsf{1Instr} \blacktriangleright \Diamond 0\big) \\
\langle \mathsf{input}\big(m(n)\big)\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr}\ \wedge\ Ии n.\ \big(\mathsf{test}(n,m) \blacktriangleright \Diamond\mathcal{A}\big) \\
m \multimap m' & \stackrel{\text{def}}{=} & \langle \mathsf{input}\big(m(a)\big)\rangle \mathsf{test}(m',a) \\
\langle \mathsf{output}\big(\overline{m}n\big)\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr}\ \wedge \\
 & & Ии m'.\ (m \multimap m')\ \blacktriangleright\ \Diamond\ (\mathsf{test}(m',n) | \mathcal{A})
\end{array}}$$

- La formule $\mathsf{1Instr}$ caractérise les processus de la forme $a(x).P$ ou $\overline{a}\langle b\rangle.P$: ce sont les atomes du calcul ; dans le cas où la restriction de nom est présente dans le calcul, la formule 1 ne suffit pas à vérifier que le terme est bien gardé par une capabilité, puisque par exemple $(\nu n)\big(\overline{a}\langle n\rangle | \overline{n}\langle a\rangle\big)$ est un modèle de 1.
- La formule $\mathsf{test}(n,m)$ caractérise la paire de processus $\{\overline{n}\langle m\rangle, \overline{m}\langle n\rangle\}$. Les modèles de $\mathsf{test}(n,m)$ sont en effet des processus qui s'achèvent après une interaction, donc de la forme $a(x).\mathbf{0}$ ou $\overline{a}\langle n\rangle.\mathbf{0}$, mais comme ils ont de plus deux noms libres, on élimine le cas de $a(x).\mathbf{0}$ (cette formule n'a donc de sens que pour $m \neq n$).
- La formule $\langle \mathsf{input}\big(m(n)\big)\rangle\mathcal{A}$ caractérise les processus de la forme $m(n').P$ tels que $P \models \mathcal{A}(n' \leftrightarrow n)$. L'ambiguïté sur le rôle de m et n est levée par le fait que l'on prend n frais pour P et qu'on s'assure que l'interaction avec P est possible.
- La formule[5] $m \multimap m'$ caractérise la paire de processus $\{m(x).\overline{m'}\langle x\rangle.\mathbf{0},\ m(x).\overline{x}\langle m'\rangle\}$.
- La formule $\langle \mathsf{output}(\overline{m}\langle n\rangle)\rangle\mathcal{A}$ caractérise les processus $\overline{m}\langle n\rangle.P$ tels que $P \models \mathcal{A}$. On se sert du processus $m \multimap m'$ pour lire le nom qui a été émis par P.

On a donc pu définir des formules pour toutes les capabilités du calcul. Par rapport au cas des Ambients vu précédemment, les modalités sont fortes au sens

[5] la notation est empruntée aux *linear forwarders* de Laneve [GLW03]

où l'on observe le terme directement gardé par la capabilité et non une de ses évolutions. Par contre, on est toujours sur des modalités intensionnelles, et dans ce cas la modalité $\langle \mathsf{output}(\overline{m}\langle n\rangle)\rangle\mathcal{A}$ et la modalité $\big[\mathsf{output}(\overline{m}\langle n\rangle)\big]\mathcal{A}$ définie par dualité sont équivalentes.

Dans une certaine mesure, on peut exprimer les modalités d'action à la Hennessy-Milner. En effet, les formules suivantes

$$\begin{array}{|rcl|}\hline \langle ab\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \reflectbox{N}m.\ \big(\langle \mathsf{output}(\overline{a}\langle b\rangle)\rangle\langle \mathsf{output}(\overline{m}\langle m\rangle)\rangle 0\big) \ \blacktriangleright\ \Diamond\ \big((\langle \mathsf{output}(\overline{m}\langle m\rangle)\rangle 0) \mid \mathcal{A}\big) \\ \langle \overline{a}b\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \reflectbox{N}m.\ a \multimap m\ \blacktriangleright\ \Diamond\ \big(\mathsf{test}(b,m)|\mathcal{A}\big) \\ \langle \overline{a}(b)\rangle\mathcal{A} & \stackrel{\text{def}}{=} & b\circledR\langle \overline{a}b\rangle\mathcal{A} \\ \hline\end{array}$$

correspondent précisément aux actions :

- $P \models \langle ab\rangle\mathcal{A}$ ssi il existe P' tel que $P \xrightarrow{ab} P'$ et $P'{\models}\mathcal{A}$;
- $P \models \langle \overline{a}b\rangle\mathcal{A}$ ssi il existe P' tel que $P \xrightarrow{\overline{a}b} P'$ et $P'{\models}\mathcal{A}$;
- $P \models \langle \overline{a}(b)\rangle\mathcal{A}$ ssi il existe P' tel que $P \xrightarrow{\overline{a}(b)} P'$ et $P'{\models}\mathcal{A}$.

Théorème 6.2.1 *La logique unifiée $\mathcal{L}^{\Diamond}_{spat}$ est plus expressive que la logique de Hennessy-Milner pour le π-calcul synchrone.*

π-calcul asynchrone. Les formules vues plus haut ne gardent pas a priori la même interprétation dans le calcul asynchrone, et surtout pas forcément l'interprétation restreinte au calcul asynchrone, puisque l'ensemble des processus introduits par $\rhd$ n'est plus le même. Pourtant, dans ce cas particulier, les formules des capabilités produites précédemment sont particulièrement stables et prennent dans le calcul asynchrone l'interprétation synchrone restreinte au calcul asynchrone. Ainsi la formule $\langle \mathsf{output}(m,n)\rangle\mathcal{A}$ a pour seul modèle $\overline{m}\langle n\rangle$, à condition que $\mathbf{0}{\models}\mathcal{A}$. En revanche, on ne peut pas définir les modalités d'action $\xrightarrow{ab}$ à cause de l'absence d'un testeur fiable (qui serait de la forme $\overline{a}\langle b\rangle.P'$). Ce fait a déjà été remarqué dans l'étude de la bisimilarité de processus asynchrones (voir [SW01], p.198), et conduit à ne considérer comme observable, au sens de la bisimulation à barbe asynchrone, que les seules barbes d'émission $\downarrow_{\overline{x}}$.

Les Ambients. On peut changer plus radicalement de calcul et considérer le calcul des ambients avec noms restreints. La formule $\mathsf{1Instr}$ caractérise alors les processus de la forme $\mathsf{cap}.P$, ou $n[P]$ avec P sans nom restreint externe (à cause de la règle $n[(\nu m)P] \equiv (\nu m)n[P]$). On ne pourra pas différencier $\mathsf{in}\,n$ de $\mathsf{out}\,n$ parce qu'il faudrait une mise en contexte (non disponible en logique unifiée) à l'aide de $(-)@n$ pour que ces capabilités puissent être exprimées. On peut regrouper les deux cas $n[\mathbf{0}]$ et $\mathsf{open}\,n.\mathbf{0}$ en un seul sous la formule

$$\mathsf{1Atom} \stackrel{\text{def}}{=} \mathsf{1Instr} \wedge \big(\mathsf{1Instr}\ \blacktriangleright\ \Diamond\ 0\big)$$

qui n'admet de modèles que sous l'une de ces deux formes. Puis pour distinguer la construction $n[-]$ de $\mathsf{open}\,n.-$, on utilise le fait que l'une est bloquante et l'autre non. Ainsi, la formule

$$\mathsf{TestAmb} \stackrel{\text{def}}{=} \mathsf{1Instr} \wedge \big(\mathsf{1Atom}\ \blacktriangleright\ \Diamond\ 1Atom\big)$$

admet pour seuls modèles les processus de la forme $n[\mathsf{open}\,m.\mathbf{0}|m[\mathbf{0}]]$. Enfin, on peut utiliser le processus $\mathsf{TestAmb}$ et le test d'occurrence de nom $\copyright n$ pour dériver les formules pour la capabilité $\mathsf{open}\,n$ et la construction $n[-]$.

$$\langle \mathsf{open}\, n\rangle\mathcal{A} \stackrel{\text{def}}{=} \mathsf{1Instr} \wedge \text{И}m.\ \big((\mathsf{TestAmb} \wedge ©n \ \wedge\ ©m) \ \blacktriangleright\ \Diamond\ \big((\mathsf{1Atom} \wedge ©m)^2|\mathcal{A}\big)\big)$$
$$\langle n[-]\rangle\mathcal{A} \stackrel{\text{def}}{=} \mathsf{1Instr} \wedge \text{И}m.\ \Big(\langle \mathsf{open}\, n\rangle.(\mathsf{1Atom} \wedge ©m) \ \blacktriangleright\ \Diamond\ \big((\mathsf{1Atom} \wedge ©m)|\mathcal{A}\big)\Big)$$

Pour résumer, on peut établir le résultat suivant :

Lemme 6.2.2 *Il existe des formules $\langle \mathsf{input}(a,n)\rangle\mathcal{A}$, $\langle \mathsf{output}(a,b)\rangle\mathcal{A}$, $\langle \mathsf{open}\, n\rangle\mathcal{A}$, et $\langle n[-]\rangle\mathcal{A}$ de la logique unifiée telles que :*
- *dans le π-calcul synchrone (resp. asynchrone), $P \models \langle \mathsf{input}(a,n)\rangle\mathcal{A}$ ssi $P \equiv a(n').P'$ avec $P' \models \mathcal{A}(n \leftrightarrow n')$.*
- *dans le π-calcul synchrone (resp. asynchrone), $P \models \langle \mathsf{output}(a,b)\rangle\mathcal{A}$ ssi $P \equiv \overline{a}\langle b\rangle.P'$ avec $P' \models \mathcal{A}$ (resp. $P \models \langle \mathsf{output}(a,b)\rangle 0$ ssi $P \equiv \overline{a}\langle b\rangle$)*
- *dans le calcul des ambients, $P \models \langle \mathsf{open}\, n\rangle\mathcal{A}$ ssi $P \equiv \mathsf{open}\, n.P'$ avec $P' \models \mathcal{A}$.*
- *dans le calcul des ambients, $P \models \langle n[-]\rangle\mathcal{A}$ ssi $P \equiv n[P']$, $P' \models \mathcal{A}$ et P' ne contient pas de noms cachés externes (ie pour tout P'', m tel que $P' \equiv (\nu m)P''$, $m \notin \mathsf{fn}(P'')$).*

On voit donc que l'on sait exprimer de nombreuses capabilités du calcul sous-jacent à l'aide de la logique unifiée. Toutefois, de nombreux constructeurs manquent. D'une part, la réplication, qu'il ne semble pas possible d'exprimer dans l'interprétation forte de la modalité $\Diamond\mathcal{A}$. D'autre part les capacités $\mathsf{in}\, n$ et $\mathsf{out}\, n$. Enfin la modalité d'ambient $\langle n[-]\rangle\mathcal{A}$ n'est pas tout à fait équivalente à la construction $n[\mathcal{A}]$ de la logique des ambients.

Malgré ces limitations, les formules développées ici tendent à montrer que l'approche des logiques spatiales est fondamentalement intensionnelle, au sens où même sans constructeurs logiques relatifs au calcul sur lequel elle porte (la formule $\overline{a}\langle n\rangle$ en logique spatiale du π-calcul, les constructeurs $n[-]$ et $(-)@n$ en logique des ambients), une logique spatiale dépouillée parvient à caractériser pour une bonne part la syntaxe du calcul.

6.2.2 Modalités faibles pour le π-calcul

On souhaite renforcer le résultat précédent en montrant qu'il s'applique aussi bien au cas de l'interprétation forte de $\Diamond\mathcal{A}$ qu'à l'interprétation faible

$$P \models \Diamond\mathcal{A} \quad \text{si} \quad \exists P'.\ P \Rightarrow P' \text{ et } P' \models \mathcal{A}$$

que l'on adopte dans cette section. Par ailleurs, on n'étudie ici, par souci de concision, que le cas du π-calcul, mais les mêmes idées s'appliquent aussi au calcul des ambients.

Le plan général de la caractérisation des capabilités est toujours le même : on cherche à caractériser les "atomes" du langage, puis on en utilise des suffisamment particuliers pour jouer le rôle de testeur, et enfin on encode les capabilités à l'aide du testeur. Dans le cas fort, les atomes étaient les processus $a(x).\mathbf{0}$, $\overline{a}\langle n\rangle.\mathbf{0}$, $n[\mathbf{0}]$, etc. Dans le cas faible, il est difficile d'avoir accès à ce type de processus à cause des nombreuses évolutions possibles du sous-terme gardé P dans la construction $\alpha.P$. Mais on peut en revanche caractériser des processus qui restent "petits" au cours de leur réduction. Plus particulièrement, on va s'intéresser au processus de la forme

$$\alpha_1.\alpha_2 \ldots \alpha_n.\mathbf{0}$$

où les α sont des gardes (sans restriction ν). Un tel processus est purement séquentiel, au sens où son système de transition étiqueté est une chaîne. On appellera par la suite *séquentiel* un tel processus.

On cherche donc à caractériser les séquentiels. On peut remarquer que tout séquentiel P admet au moins un séquentiel conjugué $\overline{P}$, et que $P|\overline{P}$ se réduit à $\mathbf{0}$. De plus, au cours de la réduction, on observe en permanence deux séquentiels en parallèle, et toute autre forme de processus est écartée. On appellera une telle forme de réduction un *dialogue*, ce que l'on peut exprimer par la formule $\Diamond 0 \ \wedge\ \Box(2 \vee 0)$. On cherche donc à caractériser les processus séquentiels par la formule

$$\mathsf{1Instr}\ \wedge\ \big(\mathsf{Instr}\ \blacktriangleright\ \big((\Diamond\ 0)\ \wedge\ \Box(\mathsf{2Instr} \vee 0)\big)\big).$$

où l'on a noté $\mathsf{2Instr}$ la formule $\mathsf{1Instr}|\mathsf{1Instr}$. Cette formule est toutefois imparfaite ; en effet, si l'on prend le processus $P \equiv a(x).\big(b(y).\mathbf{0}|\overline{b}\langle c\rangle\big)$, ce n'est pas un séquentiel et pourtant il vérifie la formule précédente (lorsqu'on le met en présence de $\overline{a}\langle b\rangle$). On utilise alors une propriété du mode d'échange de noms en π-calcul monadique. A chaque étape de réduction d'un dialogue, un des processus émet un nom et l'autre le reçoit, donc en particulier un agent ne peut augmenter sa connaissance que d'au plus un nom en une étape de réduction[6]. Plus précisément, on se place dans la situation suivante :

- P, Q, R_1, R_2 sont des processus gardés (ils satisfont $\mathsf{1Instr}$).
- P, R_1 contiennent les noms libres a et b
- Q, R_2 ne contiennent pas les noms libres a et b.

Alors R_1 est nécessairement un sous-terme de P ; plus précisément, on est soit dans le cas $(P,Q) = (\alpha.R_1, \overline{\alpha}.R_2)$, soit dans le cas $(P,Q) = \big(\alpha.(R_1|R_2), \overline{\alpha}.\mathbf{0}\big)$. Dans tous les cas, Q ne "branche" pas après son instruction de tête, ce qui est justement la propriété que l'on cherche à tester pour caractériser les séquentiels. On en déduit le résultat suivant :

Lemme 6.2.3 *Soit* Thread *la formule*

$$\begin{aligned}\mathsf{Thread} \overset{\text{def}}{=}\ & \mathsf{1Instr} \wedge \\ & Иn,m.\Big(\mathsf{testeur}(n,m)\ \blacktriangleright\ \big(\ \ \Diamond\mathsf{testeur}(n,m)\ \wedge \\ & \qquad\qquad \Box\big(\mathsf{testeur}(n,m)|(\mathsf{teste}(n,m) \vee 0)\big)\big)\Big)\end{aligned}$$

avec $\mathsf{testeur}(n,m) \overset{\text{def}}{=} \mathsf{1Instr} \wedge ©n \wedge ©m$ *et* $\mathsf{teste}(n,m) \overset{\text{def}}{=} \mathsf{1Instr} \wedge \neg©n \wedge \neg©m$. *Alors dans le π-calcul synchrone,* $P \models \mathsf{Thread}$ *ssi P est un séquentiel.*

Un séquentiel sera dit *bavard* si il ne contient que des émissions. Un séquentiel qui discute avec un séquentiel bavard et qui a quelque chose de neuf à dire ne l'aura toujours pas dit à la fin de la discussion puisque le bavard ne l'aura pas écouté. Cette propriété permet de caractériser facilement les séquentiels bavards.

On note $\mathsf{Thread}(n_1, \ldots, n_k, \neg m_1, \ldots \neg m_l)$ pour la formule $\mathsf{Thread} \wedge ©n_1 \wedge \cdots \wedge ©n_k \wedge \neg©m_1 \wedge \cdots \wedge \neg©m_l$. On considère alors la formule

$$\begin{aligned}\mathsf{Bavard} \overset{\text{def}}{=}\ & \mathsf{Thread} \wedge \\ & Иn,m\ \big(\mathsf{Thread}(n,m) \triangleright \Box\ \big((\mathsf{Thread}(\neg n, \neg m) \vee 0)|\mathsf{Thread}(n,m)\big)\end{aligned}$$

[6]A ce point, on utilise de façon cruciale le fait que le π-calcul considéré est monadique. Le résultat de cette section serait peut-être à revoir dans le cas polyadique.

Soit P un séquentiel bavard, et Q un séquentiel qui contient deux noms frais n et m pour P. Soit R tel que $P|Q \Rightarrow R$. Deux cas sont possibles : soit $R \equiv P'|Q'$ avec P qui se réduit à P' par des émissions et Q qui se réduit à Q' par des réceptions, et dans ce cas P' n'a pu apprendre aucun nouveau nom et Q n'a pu oublier que des noms communs avec P, donc $P'|Q' \models \mathsf{Thread}(\neg n, \neg m)|\mathsf{Thread}(n, m)$, soit $R \equiv Q'$ un sous-thread de Q qui contient toujours les noms frais, donc non nul. Si P n'est pas un séquentiel bavard, il est de la forme $\tilde{\alpha}.a(x).P'$ où $\tilde{\alpha}$ est une suite de messages et P' un séquentiel éventuellement nul. Dans ce cas on pose $Q \equiv \tilde{\beta}.\overline{a}\langle m\rangle.\overline{a}\langle n\rangle$ avec m, n frais pour P et $\tilde{\beta}$ la suite de réceptions conjuguées de $\tilde{\alpha}$. Alors $P|Q \Rightarrow P'\{^m/_x\}|\overline{a}\langle n\rangle$, et $P'\{^m/_x\}|\overline{a}\langle n\rangle \not\models (\mathsf{Thread}(\neg n, \neg m) \vee 0)|\mathsf{Thread}(n, m)$. Donc la formule Bavard caractérise précisément les séquentiels bavards.

On va maintenant caractériser quelques séquentiels courts, qui fourniront les testeurs essentiels pour l'encodage des capabilités.

$$\begin{array}{rcl}
\mathsf{1Ecoute} & \stackrel{\text{def}}{=} & \mathsf{Thread} \\
 & & \wedge\ \mathsf{Bavard} \blacktriangleright \Diamond 0 \\
 & & \wedge\ \mathsf{Bavard} \triangleright \Box(\mathsf{Thread} \rightarrow \mathsf{Bavard}) \\
m(n).\mathbf{0} & \stackrel{\text{def}}{=} & \mathsf{1Ecoute}\ \wedge\ \copyright m \\
\overline{m}\langle n\rangle.\mathbf{0} & \stackrel{\text{def}}{=} & \mathsf{Thread}\ \wedge\ \big(m(n).\mathbf{0} \triangleright \Diamond 0\big) \\
m(n).\overline{p}\langle n\rangle & \stackrel{\text{def}}{=} & \mathsf{Thread}\ \wedge\ \text{И}n.\ \big(\overline{m}\langle n\rangle.\mathbf{0} \blacktriangleright \Diamond \overline{p}\langle n\rangle\big) \\
\overline{m}\langle n\rangle.\overline{p}\langle q\rangle & \stackrel{\text{def}}{=} & \mathsf{Thread}\ \wedge\ \big(m(n).\mathbf{0} \blacktriangleright \Diamond \overline{p}\langle q\rangle.\mathbf{0}\big)
\end{array}$$

La formule $\mathsf{1Ecoute}$ caractérise les séquentiels de la forme $m(n).\mathbf{0}$: en effet, le conjugué de ces séquentiels est bavard, et à la fin de toute discussion avec un bavard, si il reste un des séquentiels c'est nécessairement le bavard. Les autres formules caractérisent les processus qui correspondent à leur notation.

On définit enfin les formules pour les actions à la Hennessy-Milner, puis pour les capabilités :

$$\boxed{\begin{array}{rcl}
\langle\langle an\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \text{И}p, q.\ \big(\overline{a}\langle n\rangle.\overline{p}\langle q\rangle \triangleright \Diamond(\overline{p}\langle q\rangle|\mathcal{A})\big) \\
\langle\langle \mathsf{In}(a, n)\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr}\ \wedge\ \text{И}n.\ \langle\langle an\rangle\rangle\mathcal{A} \\
\langle\langle \overline{a}b\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \text{И}p, q.\ \big(a(n).\overline{p}\langle n\rangle \triangleright \Diamond(\overline{p}\langle b\rangle|\mathcal{A})\big) \\
\langle\langle \mathsf{Out}(a, b)\rangle\rangle\mathcal{A} & \stackrel{\text{def}}{=} & \mathsf{1Instr}\ \wedge\ \langle\langle \overline{a}b\rangle\rangle\mathcal{A}
\end{array}}$$

Lemme 6.2.4 *Dans le π-calcul synchrone,*
- $P \models \langle\langle \mathsf{In}(a, n)\rangle\rangle\mathcal{A}$ *ssi il existe n', P', P'' tels que $P \equiv a(n').P'$, $P' \Rightarrow P''$, et $P'' \models \mathcal{A}(n \leftrightarrow n')$*
- $P \models \langle\langle \mathsf{Out}(a, b)\rangle\rangle\mathcal{A}$ *ssi il existe P', P'' tels que $P \equiv \overline{a}\langle b\rangle.P'$, $P' \Rightarrow P''$, et $P'' \models \mathcal{A}$;*
- $P \models \langle\langle ab\rangle\rangle\mathcal{A}$ *ssi il existe P' tel que $P \stackrel{ab}{\Longrightarrow} P'$ et $P' \models \mathcal{A}$.*
- $P \models \langle\langle \overline{a}b\rangle\rangle\mathcal{A}$ *ssi il existe P' tel que $P \stackrel{\overline{a}b}{\Longrightarrow} P'$ et $P' \models \mathcal{A}$.*

Le cas du π-calcul asynchrone. Les séquentiels asynchrones ne comportent qu'au plus un message. Donc un dialogue de deux séquentiels ne dure que le temps d'une communication. Comme dans le scénario de la formule Thread, on demande au testeur de connaître deux noms frais pour le testé, c'est qu'il les contient sous sa première capabilité, qui est donc une réception, et par conséquent le testé est nécessairement un message. On en déduit que la formule Thread caractérise exactement les processus messages $\overline{n}\langle p\rangle$. A partir de là, on encode facilement la modalité $\langle\langle \mathsf{In}(a,n)\rangle\rangle\mathcal{A}$ (mais pas la modalité $\langle\langle ab\rangle\rangle\mathcal{A}$, comme précédemment en sémantique forte).

Il est intéressant de noter que la stabilité des formules par changement de fragment que l'on avait observée pour les modalités fortes n'est plus du tout respectée en sémantique faible : pour avoir cette propriété de stabilité, il faudrait que Thread caractérise tous les séquentiels synchrones, et pas seulement les messages.

6.3 Conclusions sur les jeux contextuels

Les jeux contextuels traduisent l'idée, en apparence paradoxale, qu'un système peut être mieux observé lorsqu'on y rajoute des éléments. Les propriétés que l'on observe ce faisant sont très intensionnelles : constructeurs du calcul, finitude, noms libres... et on verra au chapitre suivant que l'équivalence logique est elle aussi très proche de la congruence structurelle, ce qui est radicalement différent de la logique extensionnelle de Hennessy-Milner.

L'altération du système testeur/testé, traduite par le connecteur $\Diamond$, est autant essentielle que la mise en contexte à l'aide de l'implication spatiale $\triangleright$. On peut comparer la requête logique à une mesure et le processus à un système qu'on met à l'étude. En sciences expérimentales, bien que fortement souhaitables, les mesures non intrusives sont rares, on est le plus souvent amené à introduire les appareils de mesure au sein du système observé. Ce faisant, le système se trouve altéré par ces appareils de mesure.

En logique spatiale, sans introduire de testeurs dans le système que l'on étudie, on ne sait donner qu'une observation grossière. De même on verra au chapitre 8 que sans la possibilité d'altérer le système, la mise en contexte perd tout à fait son intérêt, et donc qu'en quelque sorte les logiques spatiales sont soumises à la même contrainte que les sciences expérimentales : altérer pour mesurer.

Les logiques spatiales, comme on l'a déjà annoncé, sont fortement intensionnelles. Le raisonnement contextuel, toutefois, ne participe pourtant apparemment en rien à l'observation interne du système, et les véritables connecteurs intensionnels sont plutôt $|, n[-], n®-$.

Un résultat de Daniel Hirschkoff [Hir04] montre qu'effectivement, dans le cadre du π-calcul, une logique spatiale qui ne considère que les adjoints, et non les connecteurs intensionnels, est complètement extensionnelle. Ce résultat se démontre en suivant certaines techniques développées ici, avec de fortes contraintes néanmoins du fait de l'absence de caractérisation des processus gardés par un instruction (formule 1Instr). Cette équivalence entre extensionnalité et jeu contextuel (spécifique au cadre du π-calcul), n'est pas sans rappeler l'équivalence entre la bisimulation et la congruence à barbe, même si les techniques mises en oeuvres pour établir ces équivalences sont relativement différentes.


PIGS
IN
SPACE

Chapitre 7

Intensionalité

On a vu que les logiques spatiales permettent d'exprimer de nombreux connecteurs du calcul qu'elles examinent. Cette situation est très différente de celle observée dans le cadre des logiques modales comme la logique de Hennessy-Milner pour le π-calcul. Par exemple, une logique spatiale fera la différence entre les processus $a.a$ et $a|a$, alors qu'ils sont bisimilaires, donc indiscernables par les logiques modales.

On a initialement défini la logique spatiale au-dessus de la classe de congruence structurelle du processus examiné, et on voit sur cet exemple que l'on ne pourrait pas étendre cette définition à sa classe de bisimilarité, ce qui fait que le système de transition étiqueté associé au processus ne correspond pas au niveau d'observation adopté par la logique. La question qui se pose alors de façon naturelle est celle du niveau d'observation induit par la logique dans l'examen d'un processus. En d'autres termes, on cherche à caractériser l'équivalence logique $P{=_L}Q$ entre processus.

Prenons l'exemple du π-calcul synchrone sous la logique unifiée avec modalité de temps $\Diamond\mathcal{A}$ forte. Pour tout processus (fini) P, on sait donner une formule $\mathcal{F}_P$ qui reprend exactement la syntaxe de P[1]. Pour tout processus Q, on a alors $Q{\models}\mathcal{F}_P$ ssi $Q \equiv P$. On en déduit que $P{=_L}Q$ entraîne $P \equiv Q$, et comme par définition de $\models$, $P \equiv Q$ entraîne $P{=_L}Q$, on a montré que

$$=_L \quad = \quad \equiv,$$

c'est à dire que la logique unifiée est strictement intensionnelle sur le π-calcul synchrone.

L'hypothèse que $\Diamond\mathcal{A}$ est interprété au sens fort est essentielle. Sans cela, les formules pour caractériser les capabilités $\langle\langle \mathsf{cap} \rangle\rangle\mathcal{A}$ dans le cas faible ne font pas porter la postcondition $\mathcal{A}$ sur le sous-terme P de leur modèle $\mathsf{cap}.P$, mais sur un des réduits P' de P ($P \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} P'$). La construction par induction de $\mathcal{F}_P$ en suivant strictement la syntaxe de P n'a donc plus de sens.

Dans ce chapitre, on va se concentrer sur la caractérisation de l'équivalence logique sur le calcul des Ambients *publics*, dans le cadre de la logique des Am-

[1] on utilise pour cela les connecteurs intensionnels Ⓡ et $|$ ainsi que les formules dérivées $\langle \mathsf{In}(n,m)\rangle\mathcal{A}$ et $\langle \mathsf{Out}(n,m)\rangle\mathcal{A}$ vues précédemment ; par exemple, pour $P \equiv (\nu a)a(n).\mathbf{0}$, on pose $\mathcal{F}_P = n$Ⓡ$\mathsf{input}(a,n).\mathbf{0}$.

bients. La modalité de temps s'interprétant au sens faible[2], l'équivalence logique est difficile à caractériser. On procède par étapes progressives :

- On donne une reformulation opérationnelle de l'équivalence logique sous forme d'une bisimilarité dite intensionnelle, notée $\approx_{int}$. Pour cela, on reprend les observations induites par les connecteurs logiques, ainsi que les observations dérivées correspondant aux formules pour les capabilités vues au chapitre précédent.
- On montre la correction de $\approx_{int}$, c'est-à-dire que la bisimilarité intensionnelle induit l'équivalence logique. Ce résultat découle essentiellement de la congruence de $\approx_{int}$.
- On montre la complétude de $\approx_{int}$, c'est à dire que deux processus logiquement équivalents sont bisimilaires. Ce résultat est difficile à établir dans le cas le plus général. On l'établit tout d'abord pour un fragment représentatif du calcul des ambients que l'on note MA_{IF}. La preuve passe par un résultat plus fort que la simple complétude pour MA_{IF} : on établit l'existence d'une formule caractéristique $\mathcal{F}_P$ pour la classe de bisimilarité intensionnelle de P. Autrement dit,

  $$Q \models \mathcal{F}_P \quad \text{ssi} \quad Q \ \approx_{int} \ P,$$

  ce qui entraîne immédiatement $=_L \subseteq \approx_{int}$.
- On établit ensuite le résultat de complétude sur le calcul tout entier. La preuve de ce résultat utilise un découpage des processus entre une partie "active" et une partie "gelée". Par une particularité du calcul des ambients, la partie gelée d'un processus P' réduit de P est prévisible à partir de la partie gelée de P, ce qui restreint l'examen des images de P à un ensemble fini sans faire l'hypothèse standard de finitude que l'on aurait faite pour le π-calcul.

Une fois acquise la caractérisation opérationnelle de $=_L$, on cherche à l'interpréter au regard des autres équivalences entre processus $\equiv$ et $\approx$ (la congruence à barbe vue au chapitre 2) :

- on donne tout d'abord une série d'axiomes valides pour $\approx$ et non pour $=_L$;
- on montre ensuite que pour un fragment significatif du calcul des Mobile Ambients (noté MA_{IF}^{syn}), l'équivalence logique correspond à la congruence structurelle, autrement dit la logique est strictement intensionnelle. On discute aussi la prise en compte des communications dans ce résultat.

7.1 Bisimilarité intensionnelle

Dans cette section, on s'intéresse spécifiquement à la caractérisation opérationnelle de $=_L$ en termes de bisimilarité intensionnelle. On établit la correction de $\approx_{int}$ sur le calcul complet, puis sa complétude sur le calcul MA_{IF} par la construction de formules caractéristiques.

---

[2]C'est la définition originale de Cardelli et Gordon. En interprétation forte, l'équivalence logique est bien plus facile à caractériser comme $\equiv$.

7.1.1 Définition

On a vu que la logique permet de caractériser directement les constructeurs $n[-]$, $-|-$ et $\mathbf{0}$, et indirectement les constructeurs $!-$ et $\mathsf{cap}.-$. On ne s'intéresse pas pour le moment à la réplication, parce qu'en un sens elle est déjà prise en compte par la composition parallèle. Par contre, toutes les autres observations logiques, directes ou indirectes, fournissent des clauses de tests pour vérifier que deux processus sont logiquement équivalents. On laisse de côté pour le moment les jeux de tests directement proposés par les connecteurs adjoints[3] $- \triangleright -$ et $(-)@n$, mais on prend néanmoins en compte le connecteur $\Diamond$. On définit alors la bisimilarité intensionnelle comme suit

Définition 7.1.1 *$\approx_{int}$ est la plus grande relation symétrique sur les processus de MA telle que si $P \approx_{int} Q$, les processus P, Q vérifient les conditions suivantes :*
- *si $P \equiv \mathbf{0}$, alors $Q \equiv \mathbf{0}$;*
- *si il existe P_1, P_2 tels que $P \equiv P_1|P_2$, alors il existe Q_1, Q_2 tels que $Q \equiv Q_1|Q_2$, $P_1 \approx_{int} Q_1$, et $P_2 \approx_{int} Q_2$;*
- *si il existe n, P' tel que $P \equiv n[P']$, alors il existe Q' tel que $Q \equiv n[Q']$ et $P' \approx_{int} Q'$;*
- *si il existe cap, P' tel que $P \overset{\mathsf{cap}}{\to} P'$, alors il existe Q' tel que $Q \overset{\mathsf{cap}}{\to}\overset{\langle\mathsf{cap}\rangle}{\Rightarrow} Q'$ et $P' \approx_{int} Q'$.*
- *si il existe P' tel que $P \longrightarrow P'$, alors il existe Q' tel que $Q \Rightarrow Q'$ et $P' \approx_{int} Q'$.*

On rappelle que la notation $\overset{\langle\mathsf{cap}\rangle}{\Rightarrow}$ représente $\Rightarrow$ lorsque $\mathsf{cap} = \mathsf{open}\, n$, $\xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*}$ lorsque $\mathsf{cap} = \mathsf{out}\, n$, et $\xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*}$ lorsque $\mathsf{cap} = \mathsf{in}\, n$. Ainsi la bisimilarité est confrontée au phénomène de bégaiement. Les processus

$$P \;\equiv\; !\mathsf{in}\, n.\mathsf{out}\, n.\mathsf{in}\, n.\mathsf{out}\, n \quad \text{et} \quad Q \equiv\; P|\mathsf{in}\, n.\mathsf{out}\, n$$

sont sur une boucle de réduction par bégaiement, au sens où $P \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*} Q$ et $Q \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*} P$. P et Q savent donc se simuler l'un l'autre, ce qui entraîne que $\mathsf{out}\, n.P \;\approx_{int}\; \mathsf{out}\, n.Q$. Comme on va le voir, ces deux processus sont aussi logiquement équivalents.

On établit maintenant quelques propriétés de $\approx_{int}$:

Lemme 7.1.2 *Soit $\approx_{int}$ la relation symétrique définie précédemment. Alors*
- *$\approx_{int}$ est une relation d'équivalence.*
- *Si $P \approx_{int} Q$, alors $\mathsf{fn}(P) = \mathsf{fn}(Q)$.*

Preuve: $\approx_{int}$ est trivialement réflexive. La transitivité se prouve par examen des clauses de jeu : la seule clause non trivialement transitive est celle des capabilités, on montre la transitivité en utilisant en même temps la clause des capabilités et celle correspondant à $\Diamond$. L'égalité des noms libres se prouve par induction sur la profondeur de l'occurrence d'un nom libre donné. □

[3] En fait, on en a exprimé l'essentiel par les observations sur les capabilités. Cette approche diffère de l'approche naturelle par des jeux de Ehrenfreucht-Fraïssé comme définis dans [DGG], où l'on définirait des clauses pour $\triangleright$ et $(-)@n$ sans se soucier des observations dérivées sur les capabilités, donc nettement moins opérationnelle et moins facile à interpréter.

7.1.2 Congruence et correction

La propriété que l'on aimerait prouver est la congruence de $\approx_{int}$. On n'est pas assuré d'avoir effectivement une équivalence logique qui soit une congruence. Par exemple, pour la logique de Hennessy-Milner, l'équivalence logique (qui est la bisimilarité) n'est pas une congruence dans le cas général. Ceci vaut aussi pour les logiques spatiales : dans une logique sans référence aux noms, comme c'est le cas dans la logique de CCS présentée au Chapitre 3, les permutations de noms sont indiscernables, et on a $a.\mathbf{0}{=_L}b.\mathbf{0}$ pour tout a, b. Dans ce cas, $=_L$ n'est pas non plus une congruence puisqu' alors $a.\mathbf{0}|\overline{a}.\mathbf{0} \not=_L b.\mathbf{0}|\overline{a}.\mathbf{0}$ pour $a \neq b$.

Toutefois, dans notre cas, on a bien une équivalence logique qui est une congruence, ce qui va faciliter la preuve de correction de $\approx_{int}$. La congruence de $\approx_{int}$ n'est pas immédiate à cause de la clause de $\diamond$. Pour établir ce résultat, on passe par une variante syntaxique de la bisimilarité intensionnelle, dans laquelle en particulier il n'y a pas de clause pour $\diamond$. Cette relation passe clairement au contexte, mais n'est pas a priori transitive, et la preuve de congruence de $\approx_{int}$ se ramène alors à prouver que les deux bisimilarités coïncident.

Définition 7.1.3 *$\approx_{syn}$ est la plus grande relation symétrique telle que si $P \approx_{syn} Q$ on puisse vérifier :*
- *si $P \equiv \mathbf{0}$, alors $Q \equiv \mathbf{0}$*
- *si il existe P_1, P_2 tels que $P \equiv P_1|P_2$, alors il existe Q_1, Q_2 tels que $Q \equiv Q_1|Q_2$, $P_1 \approx_{syn} Q_1$, et $P_2 \approx_{syn} Q_2$;*
- *si il existe n, P' tel que $P \equiv n[P']$, alors il existe Q' tel que $Q \equiv n[Q']$ et $P' \approx_{syn} Q'$;*
- *si il existe cap, P' tel que $P \equiv \mathsf{cap}.P'$, alors il existe Q', Q'' tels que $Q \equiv \mathsf{cap}.Q'$, $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} Q''$ et $P' \approx_{syn} Q''$.*

Lemme 7.1.4 *Si $P \approx_{syn} Q$ et si C est un contexte de processus de MA, alors $\mathsf{C}[P] \approx_{syn} \mathsf{C}[Q]$.*

Lemme 7.1.5 *Si $P \approx_{syn} Q$ et si il existe cap, P' tels que $P \stackrel{\mathsf{cap}}{\to} P'$, alors il existe Q' tel que $Q \stackrel{\mathsf{cap}}{\to} \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} Q'$ et $P' \approx_{syn} Q'$.*

Preuve: $P \stackrel{\mathsf{cap}}{\to} P'$ signifie que $P \equiv \mathsf{cap}.P_1|P_2$ et $P' \equiv P_1|P_2$. On applique alors la clause de $|$, puis la clause de $\mathsf{cap}.-$, puis le lemme 7.1.4. □

Lemme 7.1.6 *Si $P \approx_{syn} Q$ et si il existe P' tel que $P \longrightarrow P'$ alors il existe Q' tel que $Q \Rightarrow Q'$ et $P' \approx_{syn} Q'$.*

Preuve: En allant chercher la réduction où elle a lieu, on écrit $P \equiv \mathsf{C}[P_1|P_2]$, $P' \equiv \mathsf{C}[P'']$ et $P_1|P_2 \longrightarrow P''$ où P_1, P_2 sont des capabilités ou des ambients, et la réduction $P_1|P_2 \longrightarrow P''$ est une interaction de deux agents/capabilités en surface. Par les règles de $|$ et $n[-]$, $Q \equiv \mathsf{C}'[Q_1|Q_2]$ avec C bisimilaire à C', et $P_i \approx_{syn} Q_i$. On vérifie alors à partir du lemme 7.1.5 et de la clause $n[]$ que $Q_1|Q_2$ peut réaliser une interaction de même nature (in, out, open) que P_1 et P_2, et de plus par le Lemme 7.1.4 que le résultat de cette interaction est bisimilaire à P'', d'où le résultat. □

Corollaire 7.1.7 *$\approx_{syn}$ est transitive et $\approx_{syn} \subseteq \approx_{int}$*

Preuve: Les Lemmes 7.1.5 et 7.1.6 assurent que $\approx_{syn}$ est une $\approx_{int}$-bisimulation. De plus, ils permettent d'établir la transitivité de la clause cap de $\approx_{syn}$, qui était la seule qui rendait cette propriété non triviale. □

Lemme 7.1.8 $\approx_{int} \subseteq \approx_{syn}$

Preuve: Il s'agit de vérifier que $\approx_{int}$ induit bien la clause de cap pour $\approx_{syn}$. On remarque que si $\mathsf{cap}.P \approx_{int} Q$, par application des clauses $|$ et $\mathbf{0}$ Q doit être insécable. On conclut alors par application de la clause cap de $\approx_{int}$. □

Corollaire 7.1.9 *$\approx_{int}$ et $\approx_{syn}$ définissent la même relation, qui est une congruence.*

Cette propriété essentielle donne une preuve élémentaire de la correction de $\approx_{int}$. On a représenté dans $\approx_{int}$ tous les connecteurs *intensionnels* de la logique. Pour ces connecteurs, $\approx_{int}$ est donc naturellement correcte. En revanche, on n'a pas exprimé de clause de jeu correspondant aux adjoints $\triangleright$ et $(-)@n$, mais la congruence de $\approx_{int}$ suffit à montrer que $\approx_{int}$ est aussi correcte vis-à-vis de ces connecteurs :

Théorème 7.1.10 *Si $P \approx_{int} Q$, alors $P{=}_L Q$, autrement dit pour toute formule $\mathcal{A}$,*

$$P \models \mathcal{A} \quad \textit{ssi} \quad Q \models \mathcal{A}.$$

Preuve: Par induction sur $\mathcal{A}$. Si $\mathcal{A}$ commence par un connecteur intensionnel, on applique la clause correspondante de $\approx_{int}$. Sinon $\mathcal{A}$ commence par un connecteur adjoint. Supposons $\mathcal{A} = \mathcal{A}_1 \triangleright \mathcal{A}_2$, et $P{\models}\mathcal{A}$. Soit R tel que $R{\models}\mathcal{A}_1$. Alors $P|R{\models}\mathcal{A}_2$, par le Lemme C.2.3, $P|R \approx_{int} Q|R$, et par induction $Q|R{\models}\mathcal{A}_2$, et ceci pour tout R, donc $Q{\models}\mathcal{A}_1 \triangleright \mathcal{A}_2$. Supposons enfin que $\mathcal{A} = \mathcal{A}'@n$ et que $P{\models}\mathcal{A}$. Alors $n[P]{\models}\mathcal{A}'$, $n[P] \approx_{int} n[Q]$ par le Lemme C.2.3, donc par induction $n[Q]{\models}\mathcal{A}'$, et finalement $Q{\models}\mathcal{A}$. □

Avant de clore cette section, on mentionne une dernière propriété de la bisimilarité intensionnelle qui nous sera utile par la suite :

Lemme 7.1.11 *Si $P \approx_{syn} Q$, alors $\mathsf{ds}(P) = \mathsf{ds}(Q)$.*

Preuve:(abrégée) On raisonne par induction sur le degré de séquentialité de P. Le cas intéressant est $P \equiv \mathsf{cap}.P'$. Alors la clause de cap impose que $Q \equiv \mathsf{cap}.Q'$ et $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} Q'' \approx_{syn} P'$. Alors $\mathsf{ds}(Q') \geq \mathsf{ds}(Q'')$ (lemme 1.4.2), par induction $\mathsf{ds}(P') = \mathsf{ds}(Q'')$, donc finalement $\mathsf{ds}(P) \leq \mathsf{ds}(Q)$, et par symétrie, on a l'égalité. □

7.1.3 Formules caractéristiques sur MA_{IF}

On vient d'établir la correction de $\approx_{int}$ par rapport à $=_L$. Pour en établir la complétude, on a vu qu'en logique de Hennessy-Milner on doit prendre une hypothèse de finitude sur l'ensemble de processus atteignables par les actions faibles $\stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow}$. On raisonne ici par analogie, quoique l'on donnera ensuite une preuve de la complétude sans hypothèse de finitude.

La condition de finitude qui correspond à la *image-finiteness* pour le π-calcul est la suivante dans notre cas :

Définition 7.1.12 *On dit qu'un processus P vérifie la condition de finitude sémantique si pour tout sous-terme* $\mathsf{cap}.P'$ *de P, l'ensemble quotient*

$$\mathcal{E}^{\mathsf{cap}}_{P'} \stackrel{\text{def}}{=} \{P'' : P' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P''\}_{/\approx_{int}}$$

est fini. On note MA_{IF} *l'ensemble des processus qui vérifient cette condition.*

Ce sous-calcul est stable par $\approx_{int}$. Il comporte la plupart des processus envisageables - on verra par la suite une condition de finitude plus restrictive pour laquelle le sous-calcul est déjà Turing complet - et n'interdit pas en particulier l'utilisation de la réplication tant qu'elle reste contrôlée.

Exemple 7.1.13 *Le processus* $!n[!\mathsf{in}\, n.\mathbf{0}]$ *est dans* MA_{IF}*, malgré l'infinité de ces réduits, puisque le seul sous-terme à inspecter est* $\mathsf{in}\, n.\mathbf{0}$*, et que* $\mathbf{0}$ *n'a qu'un nombre fini d'images. De même, le processus* $!n[!\mathsf{in}\, n.!\mathsf{out}\, n.\mathbf{0}]$ *est dans* MA_{IF}*. En revanche, le processus* $\mathsf{out}\, n.!\mathsf{in}\, n.\mathsf{out}\, n.n[\mathbf{0}]$ *n'est pas dans* MA_{IF}*.*

On va maintenant construire pour tout processus $P \in \mathsf{MA}_{IF}$ une formule caractéristique $\mathcal{F}_P$ pour la classe de bisimilarité de P. La construction procède par induction sur le degré de séquentialité de P et par induction structurelle sur P. Comme cette induction n'est pas tout à fait standard, on la formalise ici.

Définition 7.1.14 *On appelle* ordre sous-terme dynamique *la relation $P < Q$ définie par $P < Q$ si*
- *soit P est un sous-terme strict de Q*
- *soit* $\mathsf{ds}(P) < \mathsf{ds}(Q)$

Lemme 7.1.15 $<$ *définit un ordre bien fondé.*

En effet, c'est l'union de deux ordres bien fondés, et si P est un sous-terme de Q, alors $\mathsf{ds}(P) \leq \mathsf{ds}(Q)$.

On établit maintenant un résultat d'inversion de $\approx_{int}$ qui permet de déterminer précisément quels sont les processus bisimilaires à un processus donné. Ce résultat d'inversion peut être aussi vu comme une caractérisation inductive de $\approx_{int}$ pour l'ordre sous-terme dynamique.

Lemme 7.1.16 *Soit P, P_1, P_2, Q des processus, et* cap *une capabilité. Alors :*
- $\mathbf{0} \approx_{int} Q$ *ssi* $Q \equiv \mathbf{0}$
- $n[P] \approx_{int} Q$ *ssi il existe Q' tel que $Q \equiv n[Q']$ et $P \approx_{int} Q$.*
- $P_1|P_2 \approx_{int} Q$ *ssi il existe Q_1, Q_2 tels que $Q \equiv Q_1|Q_2$, $P_1 \approx_{int} Q_1$, et $P_2 \approx_{int} Q_2$*
- $\mathsf{cap}.P \approx_{int} Q$ *ssi il existe Q' tels que $Q \equiv \mathsf{cap}.Q'$, $P' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow}\approx_{int} Q'$ et $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow}\approx_{int} P'$.*
- $!P \approx_{int} Q$ *ssi il existe une famille non vide $(Q_i)_{i=1\ldots k}$ telle que $Q \equiv\ !Q_1|(!)Q_2|(!)Q_3|\ldots|(!)Q_k$ et $P \approx_{int} Q_i$ pour tout $i = 1\ldots k$.*

Preuve: A part pour la réplication, toutes les assertions découlent essentiellement de la définition de $\approx_{int}$ et de sa congruence. Pour la réplication, le sens "seulement si" découle lui aussi de la congruence. Pour le sens "si", on remarque qu'il suffit d'établir le résultat pour un processus P insécable. Alors par application des clauses $|$ et $\mathbf{0}$, on vérifie que Q ne contient que des processus insécables bisimilaires à P, et qu'il en contient une infinité. $\square$

On définit maintenant par induction selon $<$ les formules caractéristiques des processus de MA_{IF}. A congruence structurelle près, on se ramène à construire les formules caractéristiques de processus pour lesquels tout sous-terme $!P$ est tel que P est insécable.

$$
\begin{array}{|rcl rcl|}
\hline
\mathcal{F}_{\mathbf{0}} & \stackrel{\text{def}}{=} & 0 & \mathcal{F}_{n[P]} & \stackrel{\text{def}}{=} & n[\mathcal{F}_P] \\
\mathcal{F}_{P_1|P_2} & \stackrel{\text{def}}{=} & \mathcal{F}_{P_1}|\mathcal{F}_{P_2} & \mathcal{F}_{\mathsf{cap}.P} & \stackrel{\text{def}}{=} & \langle\langle \mathsf{cap}\rangle\rangle \mathcal{F}_P \wedge \\
 & & & & & [\![\mathsf{cap}]\!] \bigvee_{P' \in \mathcal{E}_P^{\mathsf{cap}}} \mathcal{F}_{P'} \\
\mathcal{F}_{!n[P]} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{n[-]}(n[\mathcal{F}_P]) & \mathcal{F}_{!\mathsf{cap}.P} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{\mathsf{cap}}(\mathcal{F}_{\mathsf{cap}.P}) \\
\hline
\end{array}
$$

On a besoin de l'ordre $<$ plutôt que de l'ordre sous-terme dans le cas de la formule $\mathcal{F}_{\mathsf{cap}.P}$. On admet en effet que l'on a déjà construit les formules caractéristiques pour tout P' dans $\mathcal{E}_P^{\mathsf{cap}} = \{P' : P \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow} P'\}_{/\approx_{int}}$, ce qui est justifié par $P' < \mathsf{cap}.P$ d'après le Lemme 1.4.2. L'hypothèse de finitude sémantique est elle aussi essentielle puisqu'elle garantit que la disjonction indexée par $\mathcal{E}_P^{\mathsf{cap}}$ est finie.

Ces formules s'interprètent à l'aide des résultats du chapitre précédent (les hypothèses de sélectivité sur les degrés de séquentialité et de profondeur, lemmes 6.1.2 et 6.1.3, sont vérifiées). De plus, elle traduisent précisément les conditions nécessaires et suffisantes du Lemme 7.1.16. On en déduit :

Théorème 7.1.17 *Pour tout processus P appartenant à MA_{IF}, il existe une formule caractéristique $\mathcal{F}_P$, c'est à dire pour tout $Q \in MA$,*

$$Q \models \mathcal{F}_P \quad \textit{ssi} \quad Q \approx_{int} P.$$

Corollaire 7.1.18 *Si P est un processus de MA_{IF} et $P{=_L}Q$, alors $P \approx_{int} Q$.*

Si l'on ajoute à cela le résultat du Théorème 7.1.10, on a donc

Corollaire 7.1.19 *$=_L$ et $\approx_{int}$ coïncident sur MA_{IF}, et $\mathcal{F}_P$ caractérise les processus logiquement équivalents à P.*

7.2 Complétude

Dans la construction des formules caractéristiques, l'hypothèse de finitude sémantique est essentielle. La clause des capabilités dans la bisimilarité intensionnelle, caractéristique des bisimulations faibles, requiert d'aller examiner, parmi tous les réduits d'un terme Q donné, si l'un d'eux est bisimilaire à un autre terme P donné. Avec l'hypothèse de finitude sur ces réduits, on peut par une formule logique les examiner tous un par un.

On va maintenant établir la complétude dans le cas général, c'est-à-dire sans hypothèse de finitude. On ne peut donc plus examiner les réduits un par un en une formule finie, mais on va décomposer l'examen des réduits en deux étapes : dans un premier temps on se ramène a un ensemble de termes finis, et dans la seconde étape on examine tous les cas restants. On explique maintenant l'idée générale de ces deux étapes.

D'un terme, on sait exprimer clairement la partie active à l'aide des connecteurs logiques primitifs et des formules pour les capabilités. Par partie active, on entend ici l'arbre des localités et les capabilités portées sur chaque noeud de localité, sans se soucier de ce dont elles sont suivies. C'est précisément cette partie qui détermine quelles sont les réductions immédiates que peut réaliser le processus. En revanche, les sous-termes gardés par des capabilités sont gelés, et ne jouent éventuellement un rôle qu'après après avoir été dégelés par une réduction. Par la suite, on considérera cette décomposition d'un processus P en un *contexte actif* C et une famille de *termes gelés* $\tilde{P}$, de sorte que $P \equiv \mathsf{C}[\tilde{P}]$.

Une propriété intéressante du calcul est qu'il ne peut que dégeler des sous-termes, de sorte que si Q' est un réduit de Q, les termes gelés de Q' étaient déjà présents dans Q. De ce fait, les réduits Q' de Q qui ont pour contexte actif un C fixé sont en nombre fini, puisque les termes gelés qui peuvent compléter ce contexte sont à prendre parmi l'ensemble fini des termes gelés de Q. Cette propriété permet donc de ramener l'examen des réduits de Q à un examen fini en testant tout d'abord le contexte actif, puis en exprimant une énumération finie sur les termes gelés.

7.2.1 Contextes actifs et termes gelés

Définition 7.2.1 *Un contexte actif est un contexte* C *avec des trous numérotés* $[]_i$*, respectant la grammaire suivante :*

$$\mathsf{C} \ ::= \ \mathbf{0} \mid \mathsf{C}|\mathsf{C}' \mid n[\mathsf{C}] \mid !n[\mathsf{C}] \mid \mathsf{cap}.[]_i \mid !\mathsf{cap}.[]_i \ .$$

P' est un processus gelé de P si il existe une capabilité cap *telle que* $\mathsf{cap}.P'$ *est un sous-terme de P. On note* $\mathsf{geles}(P)$ *l'ensemble de ces processus.*

Exemple 7.2.2 *Soit P le processus* $\mathsf{in}\, n.!n[\mathsf{out}\, n.\mathsf{open}\, n.\mathbf{0}] \mid n[!\mathsf{out}\, n.\mathbf{0}]$*. Un contexte actif pour ce terme est* $\mathsf{C} = \mathsf{in}\, n.[]_1 \mid n[!\mathsf{out}\, n.[]_2]$*, et on écrit alors* $P \equiv \mathsf{C}[!n[\mathsf{out}\, n.\mathsf{open}\, n.\mathbf{0}], \mathbf{0}]$*. Enfin l'ensemble des termes gelés de P est* $\mathsf{geles}(P) = \{!n[\mathsf{out}\, n.\mathsf{open}\, n.\mathbf{0}], \mathsf{open}\, n.\mathbf{0}, \mathbf{0}\}$*. De façon générale, pour tout processus P il existe un unique contexte actif* C *et un ensemble de processus* $\vec{P} \subset \mathsf{geles}(P)$ *tels que* $P = \mathsf{C}[\vec{P}]$ *(mais cette décomposition n'est pas la même pour deux processus congruents).*

On établit tout d'abord une observation évidente mais qui est essentielle par la suite.

Lemme 7.2.3 *Pour tout P,* $\mathsf{geles}(P)$ *est fini.*

La propriété suivante exprime le fait que les réductions peuvent dégeler des termes gelés, mais ne peuvent pas en créer de nouveaux :

Lemme 7.2.4 *Si* $P \stackrel{\mathsf{cap}}{\to} Q$*,* $P \longrightarrow Q$*,* $P \Rightarrow Q$ *ou* $P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} Q$*, alors* $\mathsf{geles}(Q) \subseteq \mathsf{geles}(P)$.

On veut maintenant exprimer le fait que l'on sait exprimer le contexte actif par une sorte de formule caractéristique. La réplication complique un peu les choses, puisqu'on a vu que l'on sait caractériser la réplication modulo la présence d'autres copies bisimilaires mais éventuellement non structurellement équivalentes au processus répliqué. On fait donc intervenir la notion de dépliage suivante ; soient C, C' deux contextes actifs, et σ une fonction des indices de trous de C' vers ceux de C. On notera $\mathsf{C} \lhd_\sigma \mathsf{C}'$ si $\mathsf{C}'\sigma \equiv \mathsf{C}$, où $\mathsf{C}'\sigma$ est le contexte actif dans lequel on renumérote les indices de trou par σ.

Exemple 7.2.5 $!\mathsf{in}\, n.[]_1 \lhd_\sigma !\mathsf{in}\, n.[]_2|\mathsf{in}\, n.[]_3$ *pour* $\sigma = 2, 3 \mapsto 1$.

Lemme 7.2.6 *Pour tout contexte actif* C*, il existe un contexte de formule* $\mathcal{F}_\mathsf{C}$ *dans lequel à chaque trou* $\mathsf{cap}_i.[]_i$ *correspondent deux trous* $[]_i^\diamond$ *et* $[]_i^\square$*, tel que pour tout processus* P *et toutes familles de formules* $(\mathcal{F}_i^\diamond), (\mathcal{F}_i^\square)$ *, si les formules*

$$\mathcal{F}_i \stackrel{\text{def}}{=} \langle\langle \mathsf{cap}_i \rangle\rangle \mathcal{F}_i^\diamond \ \wedge \ [\![\mathsf{cap}_i]\!] \mathcal{F}_i^\square$$

ont des modèles de degré de séquentialité fixé, alors les deux propriétés suivantes sont équivalentes :

1. $P \models \mathcal{F}_\mathsf{C}[\tilde{\mathcal{F}^\diamond}, \tilde{\mathcal{F}^\square}]$
2. *il existe* $\mathsf{C}', \sigma, \tilde{P}$ *tels que* $P \equiv \mathsf{C}'[\tilde{P}]$, $\mathsf{C} \lhd_\sigma \mathsf{C}'$*, et pour tout indice* i *de* C', $\mathsf{cap}_i.P_i \models \mathcal{F}_{\sigma(i)}$.

La condition sur le degré de séquentialité est nécessaire pour pouvoir appliquer le lemme 6.1.2 ; par ailleurs, les $\mathcal{F}_i$ sont appelées à être des formules caractéristiques et vérifieront donc automatiquement cette propriété.

Preuve: Par induction sur C. On utilise les mêmes constructions que pour le cas des formules caractéristiques. □

En comparaison, les contextes actifs sont caractérisés de façon identique par la bisimilarité.

Lemme 7.2.7 *Soit* C *un contexte actif et* $\tilde{P}$ *une famille de processus. Alors pour tout processus* Q*, les propriétés suivantes sont équivalentes :*

1. $\mathsf{C}[\tilde{P}] \approx_{int} Q$
2. *il existe* $\mathsf{C}', \sigma, \tilde{Q}$ *tels que* $Q \equiv \mathsf{C}'[\tilde{Q}]$, $\mathsf{C} \lhd_\sigma \mathsf{C}'$*, et pour tout indice* i *de trou de* C'*, on a* $\mathsf{cap}_i.Q_i \ \approx_{int} \ \mathsf{cap}_{\sigma(i)}.P_{\sigma(i)}$.

Preuve: Par induction sur C (on applique le Lemme 7.1.16). □

La clause faible des capabilités induit que pour vérifier que $\mathsf{cap}.P \approx_{int} \mathsf{cap}.Q$, on doit chercher parmi tous les réduits de Q si l'un d'entre eux est bisimilaire à P. Les formules caractéristiques de contexte actif permettent de restreindre cette recherche à l'ensemble de tous les réduits de Q qui commencent par le contexte actif de P. Cet ensemble est alors fini d'après le Lemme 7.2.3. On va maintenant montrer comment développer cette idée pour établir la complétude.

7.2.2 Formules $\mathcal{E}$-caractéristiques

Un résultat de complétude de $\approx_{int}$ pour $=_L$ peut se voir comme la preuve de l'existence de formules caractéristiques en un sens restreint. On dira que $\mathcal{F}$ est $\mathcal{E}$-caractéristique pour P sur un ensemble de processus $\mathcal{E}$ si $P \models \mathcal{F}$ et si pour tout processus Q de $\mathcal{E}$, $Q \models \mathcal{F}$ est équivalent à $P \approx_{int} Q$. La complétude peut donc s'exprimer comme l'existence pour tout couple (P, Q) d'une formule $\mathcal{E}$-caractéristique pour P sur le singleton $\{Q\}$. Pour établir la complétude, on va établir l'existence d'une formule $\mathcal{E}_Q$-caractéristique pour P sur l'ensemble $\mathcal{E}_Q = \{Q' : \exists \mathsf{cap}, Q \overset{\langle\mathsf{cap}\rangle}{\Rightarrow} Q'\}$, ce qui est un résultat plus fort mais fournit l'hypothèse d'induction adaptée.

Lemme 7.2.8 *Pour tout P, il existe une formule $\mathcal{F}_{\geq \mathsf{ds}(P)}$ telle que*

1. $P {\models} \mathcal{F}_{\geq \mathsf{ds}(P)}$.
2. *pour tout Q, si $Q {\models} \mathcal{F}_{\geq \mathsf{ds}(P)}$, alors $\mathsf{ds}(Q) \geq \mathsf{ds}(P)$.*

Preuve: On utilise le même procédé que pour les formules caractéristiques, sauf pour les capabilités, pour lesquelles on définit la formule $\mathcal{F}_{\geq \mathsf{cap}.P}$ comme $\langle\langle \mathsf{cap} \rangle\rangle \mathcal{F}_{\geq \mathsf{ds}(P)}$. □

Théorème 7.2.9 *Pour tout couple (P, Q), il existe une formule $\mathcal{E}_Q$-caractéristique $\mathcal{F}_{P,Q}$ pour P.*

Preuve: On procède par induction sur le degré de séquentialité de P. Si $\mathsf{ds}(P) = 0$, c'est un arbre et P admet une formule caractéristique. Si $\mathsf{ds}(P) > \mathsf{ds}(Q)$, alors $\mathcal{F}_{\geq \mathsf{ds}(P)}$ convient (d'après le Lemme 7.1.11). On suppose donc $\mathsf{ds}(Q) \geq \mathsf{ds}(P)$. Une utilisation répétée de l'hypothèse d'induction donne le résultat suivant :

$$\begin{array}{ll} \forall P' \in \mathsf{geles}(P), \forall Q' \in \mathsf{geles}(Q), & \exists \mathcal{F}_{P',Q'} \ \mathcal{E}_{Q'}\text{-caractéristique pour } P' \text{ et} \\ & \exists \mathcal{F}_{Q',P'} \ \mathcal{E}_{P'}\text{-caractéristique pour } Q' \end{array}$$

Pour les $\mathcal{F}_{P',Q'}$, les formules viennent de l'hypothèse d'induction, puisque $\mathsf{ds}(P') < \mathsf{ds}(P)$. Pour les $\mathcal{F}_{Q',P'}$, deux cas sont possibles : si $\mathsf{ds}(Q') \leq \mathsf{ds}(P')$, on peut aussi appliquer l'hypothèse d'induction ; sinon $\mathsf{ds}(Q') > \mathsf{ds}(P')$, et dans ce cas on prend pour $\mathcal{F}_{Q',P'}$ la formule $\mathcal{F}_{\geq \mathsf{ds}(Q')}$ (ce qui est justifié par le Lemme 7.1.11). On construit maintenant une formule $\mathcal{F}_i$ qui caractérise précisément $\mathsf{cap}_i.P_i$ parmi tous les termes de la forme $\mathsf{cap}.R$ où $R \in \mathcal{E} = \mathsf{geles}(P) \cup \mathsf{geles}(Q)$. Pour cela, on pose

$$\mathcal{F}_i^{\diamond} \overset{\text{def}}{=} \bigwedge_{R \in \mathcal{E}} \mathcal{F}_{P_i,R} \qquad \text{et} \qquad \mathcal{F}_i^{\square} \overset{\text{def}}{=} \neg \bigvee_{R \in \mathcal{E} \setminus \mathcal{E}_{P_i}} \mathcal{F}_{R,P_i} \,.$$

et

$$\mathcal{F}_i \overset{\text{def}}{=} \langle\langle \mathsf{cap}_i \rangle\rangle \mathcal{F}_i^{\diamond} \ \wedge \ [\![\mathsf{cap}_i]\!] \mathcal{F}_i^{\square}$$

On vérifie alors que

$$\mathsf{cap}.R \models \mathcal{F}_i \quad \text{ssi} \quad \mathsf{cap}.R \ \approx_{int} \ \mathsf{cap}_i.P_i$$

d'après la clause de cap dans le Lemme 7.1.16.

On note alors $P = \mathsf{C}[\tilde{P}]$ la décomposition de P avec C un contexte actif. Alors $\mathcal{F}_{\mathsf{C}}[(\mathcal{F}_i^{\diamond}), (\mathcal{F}_i^{\square})]$ est la formule semi-caractéristique $\mathcal{F}_{P,Q}$ cherchée. Soit en effet Q' un réduit de Q. D'après le Lemme 7.2.4, les sous-termes gelés de Q' sont dans l'ensemble $\mathcal{E}$ sur lequel les formules $\mathcal{F}_i$ sont justement expressives (en particulier leurs modèles ont un degré de séquentialité fixé). On est donc dans les conditions du lemme 7.2.6 pour Q', donc $Q' \equiv \mathsf{C}'[\vec{Q'}]$ avec $\mathsf{C} \lhd_\sigma \mathsf{C}'$ et $\mathsf{cap}_i.Q'_i {\models} \mathcal{F}_{\sigma(i)}$, c'est à dire par construction $\mathsf{cap}_i.Q'_i \approx_{int} \mathsf{cap}_{\sigma(i)}.P_{\sigma(i)}$. On est donc dans les conditions du lemme 7.2.7 et on en déduit que $P \approx_{int} Q'$. De même $P {\models} \mathcal{F}_{\mathsf{C}}[(F_i^{\diamond}), (F_i^{\square})]$, ce qui achève de montrer que cette formule est $\mathcal{E}_Q$-caractéristique pour P. □

Du fait qu'une formule semi-caractéristique de P sur $\mathcal{E}_Q$ est aussi une formule semi-caractéristique sur $\{Q\}$, on a par ailleurs démontré le résultat suivant :

Corollaire 7.2.10 (Complétude) $=_L \ \subseteq \approx_{int}$.

En conséquence, et d'après le Théorème 7.1.10, on a donc démontré le résultat suivant :

Théorème 7.2.11 *Deux processus sont logiquement équivalents ssi ils sont intensionnellement bisimilaires.*

On a donc finalement donné une caractérisation opérationnelle, quasi syntaxique, de l'équivalence logique. Le fait d'avoir une bisimilarité faible ne permet pas a priori de donner une caractérisation plus informative de l'équivalence logique dans le cas général. Toutefois, sous certaines hypothèses, on va pouvoir établir à partir de la bisimilarité intensionnelle une caractérisation plus précise de l'équivalence logique. On peut par ailleurs établir une caractérisation similaire dans un calcul des Ambients avec communications (voir annexes A.2, D.6, B.1 et B.3), ou avec des noms restreints dans le calcul et le connecteur de révélation $n®-$ dans la logique. Toutefois, on ne saurait pas établir un lemme équivalent au lemme 7.2.4 si le calcul comporte à la fois des noms restreints et des communications, comme dans le cas du π-calcul[4] ; de même on ne serait étendre cette méthode à un calcul avec récursion au lieu de la réplication.

7.3 Axiomatisations

La bisimilarité intensionnelle donne une caractérisation exacte de l'équivalence logique. On va maintenant chercher à situer cette équivalence logique vis-à-vis des autres équivalences entre processus. On peut s'intéresser d'une part à la congruence structurelle, qui est l'équivalence la plus discriminante entre processus, d'autre part à l'équivalence observationnelle, définie comme une congruence à barbe sur le calcul des Mobile Ambients. C'est par cette dernière que l'on commence.

7.3.1 Axiomatisation extensionnelle

Il n'est pas clair que l'équivalence logique, disons l'équivalence observationnelle spatiale, et l'équivalence comportementale en terme de congruence à barbe, soient des équivalences comparables, essentiellement parce qu'il n'est pas clair sur la définition de la relation $\models$ que l'équivalence logique soit une congruence. On a toutefois déjà établi ce résultat, et l'observation d'une barbe $\Downarrow n$ correspond au test par la formule $\Diamond(n[\top]|\top)$. On a donc facilement le résultat suivant :

Théorème 7.3.1 *L'équivalence spatiale est strictement plus fine que la congruence à barbe, autrement dit* $=_L \ \subsetneq \approx$.

[4] En effet, les termes gelés doivent être clos par toutes substitutions possibles de noms issus du contexte (voir annexe B.2.3) ; si le contexte comporte un générateur de noms infinis (ex : $!(\nu n)\overline{a}\langle n\rangle$), cela engendre un ensemble de terme infinis. Ce point est mieux expliqué en annexe B.2.3.

Preuve: L'inclusion est une conséquence de la possibilité d'observer les barbes par la logique et de la congruence de $=_L$ (Lemme C.2.3). Pour établir l'inclusion stricte, on considère l'égalité suivante vraie pour $\approx$ et non pour $=_L$:

$$\mathsf{cap}.P \quad \approx \quad \mathsf{cap}|P \qquad \text{pour } P \equiv \mathsf{cap}.\mathsf{cap}\ldots\mathsf{cap}$$

□

On peut de plus mentionner d'autres processus équivalents pour la congruence à barbe, mais non pour $=_L$, lorsque l'on considère des communications [San01] :

$$\begin{array}{rcll} (x)P & \approx & (x)\mathbf{0}|P & \qquad \text{pour } P \equiv (x)(x)\ldots(x)\mathbf{0} \\ (x)\langle x\rangle & \approx & \mathbf{0} & \end{array}$$

Le dernier axiome est typique des calculs avec communications asynchrones, l'autre est un équivalent de celui pour les capabilités. Une question ouverte par Davide Sangiorgi est de donner une axiomatisation complète des différences entre $=_L$ et $\approx$.

7.3.2 Intensionalité sur MA_{IF}^{syn}

On veut maintenant comparer $=_L$ à la congruence structurelle. Par définition de $\models$, l'inclusion

$$\equiv \ \subsetneq \ =_L$$

est immédiatement vérifiée. En revanche, malgré le caractère très intensionnel des observations spatiales, l'inclusion réciproque n'est pas vérifiée.

Exemple 7.3.2 *On reprend les processus*

$$P \ \equiv\ !\mathsf{in}\,n.\mathsf{out}\,n.\mathsf{in}\,n.\mathsf{out}\,n.\mathbf{0} \qquad \text{et} \qquad Q \ \equiv\ P|\mathsf{in}\,n.\mathsf{out}\,n.\mathbf{0}$$

alors les processus $\mathsf{out}\,n.P$ *et* $\mathsf{out}\,n.Q$ *sont bisimilaires, donc logiquement équivalents, et pourtant* $\mathsf{out}\,n.P \not\equiv \mathsf{out}\,n.Q$*. De plus, ces processus appartiennent au calcul restreint* MA_{IF}*.*

C'est l'existence de boucles entre processus qui fait qu'une caractérisation plus fine de l'équivalence logique n'est pas possible dans le cas général. Pire, on verra au Chapitre 9 que du fait de ces boucles de réduction l'équivalence logique est indécidable sur le calcul complet.

Pour progresser, on fait donc l'hypothèse que de telles boucles de réduction n'existent pas sur les processus que l'on considèrera. Plutôt que de faire simplement cette hypothèse, on donne une condition syntaxique sur les processus qui induira l'absence de boucle. On rappelle qu'un processus P est fini s'il est structurellement équivalent à un processus sans réplication.

Définition 7.3.3 *On dit qu'un processus vérifie l'hypothèse de finitude syntaxique si tous ses sous-termes* $\mathsf{cap}.P$ *sont tels que* P *est fini. On note* MA_{IF}^{syn} *l'ensemble des processus qui vérifient cette hypothèse.*

Cette condition est plus restrictive que l'hypothèse de finitude sémantique, et on a la classification suivante

$$\text{Ambients finis} \ \subsetneq\ \mathsf{MA}_{IF}^{syn} \ \subsetneq\ \mathsf{MA}_{IF} \ \subsetneq\ \mathsf{MA}$$

où chacun des sous-calculs est stable pour $\approx_{int}$.

Le processus $\mathsf{out}\, n.P$ de l'exemple 7.3.2 est en effet dans MA_{IF} et pas dans MA_{IF}^{syn}, alors que le processus $!n[!\mathsf{in}\, n.\mathsf{out}\, n.\mathbf{0}]$ est dans MA_{IF}^{syn} et pas dans les Ambients finis.

Pour les Ambients finis, il n'existe pas de boucles de réduction, c'est à dire

Lemme 7.3.4 *Si P, Q sont deux processus finis et si $P \longrightarrow Q$, alors $Q \not\Rightarrow P$.*

Preuve: On peut compter le nombre de capabilités $\mathsf{cap}(P)$ d'un processus à $\equiv$ près sur le calcul fini. Or $\mathsf{cap}(P)$ diminue strictement à chaque étape de réduction. □

Lemme 7.3.5 *Soient P, Q deux processus finis. Si $P \approx_{int} Q$, alors $\mathsf{cap}(P) = \mathsf{cap}(Q)$.*

Preuve: Par induction sur $\mathsf{cap}(P)$ □

Théorème 7.3.6 *L'équivalence logique coïncide avec la congruence structurelle sur MA_{IF}^{syn}, autrement dit si $P \in \mathsf{MA}_{IF}^{syn}$, alors*

$$P =_L Q \quad ssi \quad P \equiv Q.$$

Preuve: D'après la caractérisation inductive du Lemme 7.1.16, il suffit de prouver le résultat pour les processus de la forme $\mathsf{cap}.P'$. La clause des capabilités donne

$$\mathsf{cap}.P' \;\approx_{int}\; Q \quad \text{ssi} \quad Q \equiv \mathsf{cap}.Q',\ Q' \Rightarrow\approx_{int} P' \text{ et } P' \Rightarrow\approx_{int} Q'.$$

Par le lemme 7.3.5, $\mathsf{cap}(P') = \mathsf{cap}(Q')$, donc d'après le lemme 7.3.4, on peut les $\Rightarrow$ sont des $\equiv$ dans l'équivalence ci-dessus, soit

$$\mathsf{cap}.P \;\approx_{int}\; Q \quad \text{ssi} \quad Q \equiv \mathsf{cap}.Q' \text{ et } P \approx_{int} Q'$$

ce qui fournit le résultat. □

Ambients avec communication

Comme mentionné précédemment, la caractérisation de l'équivalence logique sur le calcul des ambients avec communications suit exactement le même schéma que précédemment.

Dans le cas de communications synchrones, on rajoute à la définition de $\approx_{int}$ deux clauses pour les émissions et réceptions, identiques aux clauses faibles pour les capabilités :

– si $P \xrightarrow{?n} P'$, alors il existe Q' tel que $Q \xrightarrow{?n} \Rightarrow Q'$ et $P' \approx_{int} Q'$.

– si $P \xrightarrow{!n} P'$, alors il existe Q' tel que $Q \xrightarrow{!n} \Rightarrow Q'$ et $P' \approx_{int} Q'$.

Dans le calcul MA_{IF}^{syn}, on obtient donc le même résultat que sans communication : équivalence logique et congruence structurelle coïncident.

En revanche, dans le cas de communications asynchrones, comme on l'a déjà observé au chapitre précédent sur le cas du π-calcul, on ne sait pas encoder exactement la modalité de réception, et on a donc les clauses faibles suivantes :

– si $P \equiv \langle n \rangle$, alors $Q \equiv \langle n \rangle$.

– si $P \xrightarrow{?n} P'$, alors il existe Q' tel que $Q|\langle n\rangle \Rightarrow Q'$ et $P' \approx_{int} Q'$.

Du fait de l'affaiblissement de la clause de réception, deux processus bisimilaires n'ont donc pas forcément le même nombre de préfixes de réception. Ainsi, par exemple, les processus suivants sont bisimilaires :

$$(x)P \quad \approx_{int} \quad (y)\big(\langle y\rangle|(x)P\big)$$

ce qui est une nouvelle forme de bégaiement.

On définit alors la congruence structurelle modulo eta équivalence, notée $\equiv_E$, en rajoutant l'axiome $(x)P \ \equiv_E \ (y)\big(\langle y\rangle|(x)P\big)$. Sur le calcul fini, on peut alors identifier $\equiv_E$ et $=_L$.

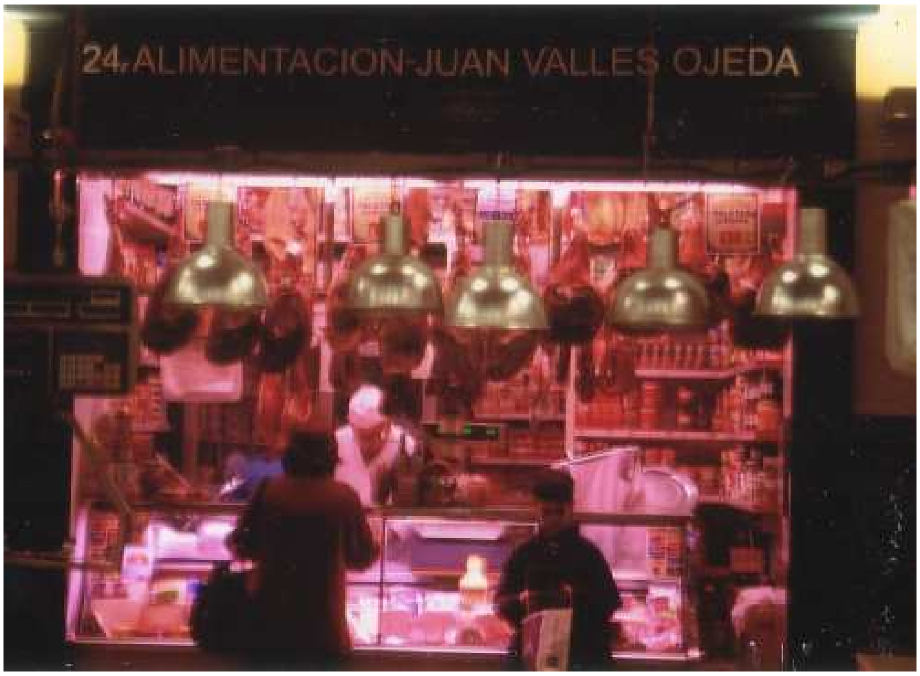
Pour établir l'équivalence, on reprend le schéma de preuve du cas synchrone, mais l'équivalent du lemme 7.3.5 suppose que P, Q sont sous forme eta réduite. Ce point est détaillé en annexe B.3.

7.4 Conclusion

On a développé pour le calcul des Mobile Ambients une caractérisation opérationnelle d'une notion de bisimilarité, puis une axiomatisation, de l'équivalence entre processus induite par la logique. Cette étude révèle le caractère fortement intensionnel des logiques spatiales sur les algèbres de processus. Sous des hypothèses raisonnables, on peut en effet exprimer le fait que l'équivalence logique correspond à la congruence structurelle. Une autre propriété marquante de l'intensionalité des logique spatiales est que la bisimilarité qui leur est associée est extrêmement syntaxique, ce qui permet pour le calcul des Mobile Ambients d'en établir la complétude dans le cas faible sans hypothèse de finitude, la logique ayant une expressivité suffisante pour pouvoir représenter l'opérateur infinitaire de réplication.

Si l'étude de l'intensionalité de la logique est complexe sur les structures dynamiques des algèbres de processus, elle est en revanche immédiate sur des structures statiques, dont la logique reflète directement toutes les constructions. Les connecteurs adjoints ne jouent donc aucun rôle dans le pouvoir *distinctif* de la logique.

Au delà du pouvoir distinctif de la logique se pose le problème plus général de son pouvoir *expressif*. Comme on va le voir maintenant pour les structures statiques, les connecteurs adjoints sont parfois non seulement sans influence sur le pouvoir distinctif, mais aussi sur le pouvoir expressif tout entier.


24.ALIMENTACION-JUAN VALLES OJEDA

Chapitre 8

Minimalité

Les deux derniers chapitres ont donné des indications sur l'expressivité des logiques spatiales sur les algèbres de processus, soit à un niveau extensionnel par l'encodage de modalités d'actions comme en logique de Hennessy-Milner, soit au niveau intensionnel par la définition de formules caractéristiques des processus modulo congruence structurelle. On s'intéresse maintenant à l'expressivité des logiques spatiales *vis-à-vis d'elles-mêmes*, indépendamment des modèles auxquels on les applique [1].

On cherche donc à comprendre quels connecteurs logiques peuvent être exprimés par une logique spatiale, ou plus précisément les dépendances qui peuvent exister entre les multiples connecteurs introduits. On a par exemple déjà noté pour le prédicat d'occurrence de nom (©n) qu'il existe deux encodages statiques (chapitre 4) et un encodage par jeu contextuel (chapitre 6), ce qui montre que ce connecteur est fortement dépendant des autres. Il est important d'avoir une *logique minimale*, c'est à dire une logique dans laquelle les connecteurs sont indépendants. Une telle logique synthétise les observations propres à la logique spatiale dont elles est issue, et permet de ramener à un objet mathématique plus simple les problèmes soulevés par ses extensions pléthoriques.

On peut définir divers niveaux de dépendance entre les connecteurs. Pour le connecteur ©, la dépendance avec les autres connecteurs est forte : il est définissable par une périphrase dans le langage des autres connecteurs. On peut néanmoins envisager une contrainte de dépendance moins forte : un connecteur sera dit *éliminable* si toute formule qui l'utilise admet une formulation équivalente qui ne l'utilise pas. Ceci vaut par exemple pour les connecteurs $*$ et $-\!\!*$ de la logique séparante (voir chapitre 5), qui n'ajoutent pas d'expressivité à logique classique bien qu'il n'admette pas une traduction constante dans celle-ci ; dit autrement, un connecteur définissable par périphrase est éliminable localement, ou *dérivable* (c'est du "sucre syntaxique"), alors que $*$ et $-\!\!*$ sont éliminables globalement, ou *admissibles.*

On s'intéresse ici à la minimalisation de deux logiques spatiales. La première

[1]Sur ce point toutefois, on est limité par le fait qu'en l'absence de système d'inférence générique, on doit nécessairement choisir une algèbre de processus jouant le rôle de modèles pour les logiques considérées ; on peut toutefois espérer, dans une certaine mesure, s'abstraire de ces modèles, d'une part parce qu'ils n'apparaissent pas vraiment dans les énoncés des résultats obtenus, d'autre part parce que des résultats très semblables sont dérivables avec d'autres choix de modèles, comme on le mentionne par la suite. On ne sait pas vraiment comment faire cette abstraction, malheureusement.

est la logique des Ambients statiques (SAL), mais la comparaison de l'approche développée ici et de celle du chapitre 5 montrent assez que le modèle considéré importe peu, et on retiendra essentiellement qu'on considère ici une logique spatiale *statique*. Dans ce contexte, on montre que les connecteurs adjoints sont éliminables au sens où ils sont admissibles. On montre par ailleurs que la présence de quantification sur les noms de ressources est un obstacle à l'élimination des adjoints sur $SAL^{\exists}$, et que les adjoints sont les seuls connecteurs éliminables.

La deuxième minimalisation concerne la logique $\mathcal{L}_{mod}^{(CCS)}$. A nouveau, le calcul CCS est pris comme paradigme des algèbres de processus, mais le résultat se généralise à d'autres calculs (voir annexe D.6) et on se propose de considérer plus généralement le cadre des logiques spatiales dynamiques. On montre que l'on peut éliminer les quantificateurs existentiels sur les noms de façon locale. L'élimination des quantificateurs est toutefois plus complexe à mettre en place que pour ©, et sujette à quelques restrictions, même si l'idée initiale reste assez proche de celle des jeux contextuels.

8.1 Élimination des adjoints dans les logiques spatiales statiques

Dans cette section, on s'intéresse à l'expressivité des logiques spatiales sur des structures statiques. La même technique de minimalisation que l'on a mise en place pour la logique séparante s'applique aux Ambients statiques, autrement dit le calcul des Ambients sans capabilités ni communications.

On note tout de même quelques nuances entre logique des Ambients statique et logique séparante. D'une part, le connecteur logique $-|-$ exprime non seulement la séparation de deux ressources, mais permet aussi dans certains cas de compter le nombre de copies d'une même ressource, ce qui n'a pas de sens en logique séparante. Ainsi, la formule $n[0]|n[0]$, caractérise les modèles qui contiennent deux copies de l'ambient n, alors que la formule $x \mapsto 1 * x \mapsto 1$ n'a pas de modèle en logique séparante. On ne pourra donc pas éliminer le connecteur $|$ sans perte d'expressivité. De même les autres connecteurs intensionnels ($n®-, n[-]$) expriment des propriétés qu'il ne paraît pas naturel d'exprimer autrement.

Pour les connecteurs adjoints, la situation est assez différente. On a bien vu que dans le cas dynamique, ces connecteurs permettent de définir des jeux contextuels qui testent la présence de constructeurs dynamiques du calcul. Dans le cas statique, d'une part tous les constructeurs du modèle sont déjà reflétés dans la logique par les connecteurs intensionnels, d'autre part on ne peut plus parler de jeu contextuel sans la modalité de temps. Le rôle des adjoints est donc fondamentalement différent.

On va étudier précisément la contribution du raisonnement contextuel dans le cas statique. On établit tout d'abord que pour la logique des ambients statiques (SAL, cf Chap. 3), cette contribution est nulle, du moins d'un point de vue modèle-théorique[2]. Toute formule logique admet ainsi une reformulation

[2]Mais peut-être pas vraiment d'un point de vue effectif, comme le montrent les résultats d'indécidabilité du chapitre suivant

équivalente sans connecteurs adjoints[3]. On établit ensuite que ces connecteurs sont les seuls redondants, et que la perte de tout autre connecteur logique induit une perte d'expressivité. Ainsi, la logique sans adjoints SAL_{int} est minimale. On montre enfin que les connecteurs adjoints peuvent apporter une certaine expressivité dans le cas où la logique prend en compte une quantification classique ($SAL^{\exists}$).

8.1.1 Procédé de minimalisation

On présente ici un procédé de minimalisation basé sur l'équivalence entre modèles. L'équivalence entre modèles est héritée des jeux de Ehrenfreucht-Fraïssé, et ce cadre théorique permettrait sans doute de donner une formulation générale de ce type de procédé de minimalisation pour de nombreuses logiques sous-structurelles[4]

Ici, on choisit toutefois d'adhérer à la vision héritée des algèbres de processus d'une équivalence entre modèles en tant que bisimilarité intensionnelle.

On considère une stratification[5] de la bisimilarité intensionnelle comme une famille $(\approx_{i,N})_{i\geq 0, N\subset \mathsf{Na}}$ telle que $\approx_{int} = \bigcap_{i,N} \approx_{i,N}$. On démontre la correction de $\approx_{i,N}$ par une propriété de congruence, comme on l'a déjà fait pour $\approx_{int}$ au Chap. 7. On établit ensuite l'existence *dans la logique sans adjoints* de formules caractéristiques $\mathcal{F}_C$ pour C une classe de $\approx_{i,N}$, et on montre enfin la finitude du nombre de classes de $\approx_{i,N}$, une propriété que l'on appelle précompacité. On obtient alors un encodage d'une observation $\mathcal{A}$ dans SAL en une formule de SAL_{int} en énumérant les classes sur lesquelles $\mathcal{A}$ est vérifiée :

$$\mathcal{A} \quad \rightsquigarrow \quad \bigvee_{C \models \mathcal{A}} \mathcal{F}_C.$$

On formalise maintenant ces idées.

Définition 8.1.1 *Étant fixé un ensemble de noms $N \subseteq \mathsf{Na}$, $\approx_{i,N}$ est la relation symétrique définie par induction sur i de sorte que $P \approx_{0,N} Q$ pour tout P, Q, et pour tout $i \geq$ et tous processus P, Q, $P \approx_{i+1,N} Q$ ssi les conditions suivantes sont vérifiées :*
- *si $P \equiv \mathbf{0}$, alors $Q \equiv \mathbf{0}$*
- *si il existe P_1, P_2 tels que $P \equiv P_1|P_2$, alors il existe Q_1, Q_2 tels que $Q \equiv Q_1|Q_2$, $P_1 \approx_{i,N} Q_1$ et $P_2 \approx_{i,N} Q_2$.*
- *pour tout $n \in N$, si il existe P' tel que $P \equiv n[P']$, alors il existe Q' tel que $Q \equiv n[Q']$ et $P' \approx_{i,N} Q'$.*
- *pour tout $n \in N$, si il existe P' tel que $P \equiv (\nu n)P'$, alors il existe Q' tel que $Q \equiv (\nu n)Q'$ et $P' \approx_{i,N} Q'$.*

$\approx_{i,N}$ définit clairement une relation d'équivalence, et on a la propriété de monotonie suivante : si $i_1 \geq i_2$ et $N_1 \supseteq N_2$, alors $\approx_{i_1,N_1} \subseteq \approx_{i_2,N_2}$. On établit maintenant les propriétés de congruence suivantes :

[3]Ce résultat est à comparer à l'élimination de $\multimap$ dans [Dam88] : $\phi \multimap \psi \;=\; \bigvee\{\gamma :\vdash \gamma \circ \phi {\rightarrow} \psi\}$.

[4]Voir [DGG] pour une présentation plus récente de l'élimination des adjoints en terme de jeux de Ehrenfreucht-Fraïssé.

[5]Cette technique est très classique. Elle est utilisée pour démontrer la complétude de la bisimilarité intensionnelle sur les Mobile Ambients finis, cf. annexe B.1.4. On l'a par ailleurs rencontrée au chapitre 2.

Lemme 8.1.2 *Soit P, Q tels que $P \approx_{i,N} Q$. Alors :*
- *pour tout R, $P|R \approx_{i,N} Q|R$;*
- *pour tout $n \in \mathsf{Na}$, $n[P] \approx_{i,N} n[Q]$;*
- *pour tout $n \in N$, $(\nu n)P \approx_{i,N} (\nu n)Q$.*

Preuve: Par induction sur i. $\square$

On ne peut toutefois pas généraliser la dernière propriété de congruence à tout nom. En effet, soient n, a, b trois noms deux à deux distincts, et $N = \{n\}$. Alors $a[\mathbf{0}] \approx_{i,N} b[\mathbf{0}]$, mais $(\nu a)a[\mathbf{0}] \not\approx_{i,N} (\nu a)b[\mathbf{0}]$. Cette notion de congruence partielle est néanmoins suffisante pour dériver le résultat de correction suivant :

Lemme 8.1.3 *Soient i, N donnés et $\mathcal{A}$ une formule de SAL* sans quantificateurs И, *comportant au plus i connecteurs et telle que* $\mathsf{fn}(A) \subseteq N$. *Alors pour tous processus P, Q tels que $P \approx_{i,N} Q$,*

$$P \models \mathcal{A} \quad ssi \quad Q \models \mathcal{A}$$

Preuve: Par induction sur $\mathcal{A}$. Pour chaque connecteur intensionnel $(|, n®, n[])$, on utilise la clause de la définition de $\approx_{i,N}$ correspondante. Pour les connecteurs adjoints $(\triangleright, \oslash n, @n)$, on applique le lemme précédent. $\square$

On établit la complétude de $\approx_{i,N}$ par le biais de formules caractéristiques. On commence par établir le résultat suivant :

Lemme 8.1.4 *Soient i, N donnés tels que N est fini. Alors $\approx_{i,N}$ admet un nombre fini de classes d'équivalence.*

Preuve: Par induction sur i. Si $\approx_{i,N}$ admet au plus k classes, alors on peut borner le nombre de classes de $\approx_{i+1,N}$ par le nombre de tests induits par les clauses : si ce nombre de tests est fini, on n'a donc qu'un nombre fini de configurations possibles par rapport à ces tests. Pour la clause de $\mathbf{0}$, on n'a qu'un seul test ; pour $-|-$, k^2 tests distincts sont possibles ; pour $-[-]$, on en a $k\sharp N$, et autant pour $(\nu-)-$. D'où le résultat. $\square$

Lemme 8.1.5 *Soient i, N donnés tels que N est fini. Alors pour toute classe d'équivalence C de $\approx_{i,N}$, il existe une formule $\mathcal{F}_C$ dans SAL_{int} telle que*

$$\forall P \qquad P \models \mathcal{F}_C \quad ssi \quad P \in C.$$

Preuve:(abrégée) Par induction sur i, voir proposition C.3.3. Chaque test intensionnel s'exprime dans SAL_{int}, par exemple si les processus dans C admettent une partition $C_1|C_2$, on l'exprime par la formule $\mathcal{F}_{C_1}|\mathcal{F}_{C_2}$. La formule $\mathcal{F}_C$ est alors la conjonction de l'ensemble (fini) de ces tests exprimés par des formules intensionnelles. $\square$

Lemme 8.1.6 *Pour toute formule $\mathcal{A} \in SAL$* sans quantificateur И, *il existe une formule $[\![\mathcal{A}]\!]$ dans SAL_{int} équivalente, c'est à dire*

$$\forall P \qquad P \models \mathcal{A} \quad ssi \quad P \models [\![\mathcal{A}]\!].$$

Preuve: Soit $\mathcal{A}$ sans quantificateur. On prend i, N suffisamment grands pour pouvoir appliquer le Lemme 8.1.3. Alors $[\![\mathcal{A}]\!] = \bigvee_{C \approx_{i,N}-\text{classe}, C \models \mathcal{A}} \mathcal{F}_C$. $\square$

Théorème 8.1.7 *Pour toute formule $\mathcal{A} \in SAL$ il existe une formule équivalente dans SAL_{int}.*

Preuve: D'après le Théorème 4.2.3, il existe $\tilde{n}$ et $\mathcal{A}'$ sans quantificateurs telle que $\mathcal{A} \dashv\vdash И\tilde{n}.\mathcal{A}'$. La formule équivalente à $\mathcal{A}$ est alors $И\tilde{n}.[\![\mathcal{A}']\!]$. □

Ce dernier résultat signifie en particulier que SAL n'est pas une logique minimale.

Exemple 8.1.8 *On montre sur un exemple comment appliquer les idées de la preuve d'élimination. Soit*

$$\mathcal{A} \quad ::= \quad \Big(Hm'.m'[\top] \ \blacktriangleright \ \big(Hn_1.n_1[0] \mid Hn_2.n_2[Hn_3.n_3[0]]\big)\Big) \oslash m@m$$

La forme prénexe de $\mathcal{A}$ est

$$Иm', n_1, n_2, n_3. \ \Big((m'®\top\wedge.m'®m'[\top]) \ \blacktriangleright \ \big(n_1®n_1[0] \mid n_2®n_2[n_3®.n_3[0]]\big)\Big)\oslash m@m$$

Alors $P \models \mathcal{A}$ ssi il existe Q tel que

$$(\nu m)\, m[P] \mid (\nu m')\, m'[Q] \quad \equiv \quad (\nu n_1)(\nu n_2)(\nu n_3)\big(n_1[\mathbf{0}]|n_2[n_3[\mathbf{0}]]\big)$$

Les seules solutions de cette équation sont $P \equiv \mathbf{0}$ ou $P \equiv (\nu n_3)n_3[\mathbf{0}]$. Autrement dit, $\mathcal{A}$ est équivalente à la formule sans adjoint $\mathcal{B} = 0 \vee Hn_3.n_3[0]$.

8.1.2 Minimalité de SAL_{int}

On vient de voir que SAL n'est pas minimale. On va maintenant établir que SAL_{int} est minimale. On peut considérer deux versions de la minimalité :

- version forte : tout fragment est moins expressif, i.e. pour tout fragment il existe des formules qui ne s'expriment pas dans ce fragment.
- version faible : tout fragment est moins distinctif, i.e. pour tout fragment il existe des processus distinguables dans la logique complète mais pas dans le fragment.

On va établir que SAL_{int} est minimale au sens fort, mais au sens faible le fragment minimal est encore plus petit : on peut en effet enlever les connecteurs $\wedge, \neg, И, 0$ et différencier deux processus non congruents quelconques. Ces connecteurs seront dits expressifs, tandis que les connecteurs $|, n[-], n®-$ seront dit séparatifs. Pour établir la minimalité, on doit prouver que pour chaque connecteur κ, le fragment $SAL_{int} - \{\kappa\}$ est moins expressif que SAL_{int}. Pour les connecteurs distinctifs, il suffit en fait de trouver deux processus non congruents indissociables sans κ. Pour les connecteurs expressifs, la preuve est plus complexe, puisque de tels processus n'existent pas. En revanche, on doit donner une propriété qui s'exprime dans SAL_{int} et ne s'expriment pas sans κ.

Théorème 8.1.9 *SAL_{int} est minimale.*

Preuve: La preuve complète comporte de nombreux lemmes techniques (voir annexe C.6). On donne ici seulement les idées essentielles :

- $\kappa = \wedge$: on ne peut pas exprimer la disjonction $n_1[n_2[0]] \vee n_2[n_1[0]]$.
- $\kappa = \neg$: on ne peut pas exprimer $©n = \neg n®\top$. Pour établir ce résultat, on remarque que les formules sans négation ne détectent que les noms libres présents avant une certaine profondeur.

- $\kappa = И$: on ne peut pas exprimer $Иn.n®\neg n®\top$ (présence d'un nom restreint). Pour $N = \{n_1, \ldots n_r\}$ on pose $P_N = n[n_1[\mathbf{0}]|\ldots|n_r[\mathbf{0}]]$ avec $n \notin N$. Alors pour toute formule $\mathcal{A}$ sans quantificateur générique avec $\mathsf{fn}(\mathcal{A}) \subseteq N$, $P \models \mathcal{A}$ iff $(\nu n)P \models \mathcal{A}$.
- $\kappa = 0$: ici on suppose que dans le fragment restreint on prend $\top$ à la place de 0 comme formule élémentaire. On ne peut pas exprimer 0. En effet, pour toute formule $\mathcal{A}$ sans 0, si $n \notin \mathsf{fn}(\mathcal{A})$, alors $\mathbf{0} \models \mathcal{A}$ ssi $n[\mathbf{0}] \models \mathcal{A}$.
- $\kappa = |$: on ne peut pas distinguer $n[\mathbf{0}]|n[\mathbf{0}]$ de $n[\mathbf{0}]|n[\mathbf{0}]|n[\mathbf{0}]$.
- $\kappa = n[.]$: on ne peut pas distinguer $n_1[n_2[\mathbf{0}]]$ de $n_2[n_1[\mathbf{0}]]$.
- $\kappa = n®.$: on ne peut pas distinguer $(\nu n)n[\mathbf{0}]$ de $(\nu n)n[n[\mathbf{0}]]$.

□

On n'a toutefois pas démontré que SAL_{int} est le seul fragment minimal de SAL. C'est probablement vrai, mais les preuves sont nettement plus complexes du fait de la prise en considération des adjoints. A titre d'exemple, on peut exprimer certaines disjonctions sans $\wedge$.

Exemple 8.1.10 *La formule*

$$\neg Иn.n®\neg n® \left(Иm_1.m_1®Иm_2.m_2®m_1[m_2[0]]\right) \oslash n_1 \oslash n_2$$

est en fait équivalente à $n_1[n_2[0]] \vee n_2[n_1[0]]$, *qui est justement la propriété choisie pour la preuve de l'expressivité de* $\wedge$.

8.1.3 Cas de non élimination des adjoints

On se place maintenant dans la logique intensionnelle des Ambients statiques avec une quantification existentielle sur les noms, notée $SAL_{int}^{\exists}$ (cf Chap. 3). Dans cette logique, les adjoints prennent une expressivité qui dépassent le fragment intensionnel. On a déjà vu, au chapitre 5, que l'implication spatiale n'est pas éliminable[6]. Pour montrer la non élimination d'un quelconque connecteur adjoint, on exhibe maintenant une autre propriété, liée à l'élimination de l'égalité de noms.

Théorème 8.1.11 *Pour tout opérateur adjoint* $\kappa \in \{\triangleright, @, \oslash\}$, *l'extension* $SAL_{int}^{\exists} \cup \{\kappa\}$ *est strictement plus expressive que* $SAL_{int}^{\exists}$

Ce résultat découle des deux résultats suivants :

Lemme 8.1.12 *Pour tout opérateur adjoint* $\kappa \in \{\triangleright, @, \oslash\}$, *il existe une formule* $\mathcal{A} \in SAL_{int}^{\exists} \cup \{\kappa\}$ *telle que*

$$(FN1) \qquad \forall\, P \quad P \models \mathcal{A} \quad ssi \quad \sharp\mathsf{fn}(P) \;\leq\; 1$$

Preuve: Pour $\kappa = \oslash$, on prend $\mathcal{A} = \exists n.\big(\forall n.\neg ©n\big) \oslash n$. Pour $\kappa = \triangleright, \oslash$, on remarque d'abord que ces connecteurs expriment l'égalité de noms :

$$n = m \quad \dashv\vdash \quad (n[\top])@m \quad \dashv\vdash \quad n[0] \wedge m[0] \blacktriangleright \top.$$

On prend alors $\mathcal{A} = \exists n.\forall m.(m \neq n \rightarrow \neg ©m)$. □

[6]Voir [DGG, CL04] pour les adaptations du contre-exemple du chapitre 5 à la logique des Ambients et à $\mathcal{L}_{spat}^{\diamond}$.

Lemme 8.1.13 *Il n'existe pas de formule* $\mathcal{A} \in SAL^{\exists}_{int}$ *qui vérifie* $(FN1)$.

Preuve: (abrégée) Ce résultat est basé sur deux lemmes techniques d'invariance par substitution dans $SAL^{\exists}_{int}$ (voir annexe C.5)). De façon informelle : pour toute formule $\mathcal{A}$, si on sature suffisamment un terme en noms, et que ce terme contient à une profondeur suffisante un sous terme $n[\mathbf{0}]|m[\mathbf{0}]$, alors on peut le remplacer par $n[\mathbf{0}]|n[\mathbf{0}]$ sans que $\mathcal{A}$ ne s'en aperçoive. □

L'expressivité des connecteurs adjoints dans les structures statiques dépend donc de la forme de quantification adoptée par la logique : en quantification générique, les adjoints ne permettent pas d'exprimer de nouvelles propriétés, tandis qu'en quantification existentielle, l'ajout d'un quelconque adjoint permet d'exprimer la majoration du nombre de noms libres.

Dans les structures dynamiques, l'élimination directe des adjoints entraîne aussi une perte d'expressivité : comme on l'a vu, les adjoints jouent un rôle essentiel pour caractériser les capabilités du calcul. Dans le cas de la logique séparante, l'adjoint exprime des observations dérivées comme le fait qu'une cellule est allouée (formule $\mathsf{alloc}\, x$). La prise en compte de ces observations dérivées, ce que l'on a appelé la renormalisation de la logique, permet alors d'éliminer les connecteurs $*$ et $-\!\!*$. Semblablement, pour les logiques spatiales dynamiques (sans quantification existentielle), on pourrait donc chercher une renormalisation de la logique, avec par exemple des modalités d'action pour exprimer les observations dérivées, et de cette façon retrouver la propriété d'élimination des adjoints. Par l'étude de la dérivabilité des quantificateur, on va maintenant répondre indirectement à cette question (Théorème 8.2.4).

8.2 Élimination des quantificateurs dans les logiques spatiales dynamiques

On considère maintenant la logique spatiale $\mathcal{L}^{\diamond}_{spat}$. C'est la première logique spatiale dynamique à considérer, au sens où elle inclut les connecteurs spatiaux $|$, $\triangleright$ et 0, et où elle prend en compte le temps par la modalité (ici, forte) $\diamond\mathcal{A}$. Sans rien de plus, c'est une logique minimale, mais étonnament elle recèle une grande richesse d'expressivité. Elle permet de mettre en place des jeux contextuels (chapitre 6), et on pourra donc exprimer, d'une part les capabilités du calcul sous-jacent, d'autre part des propriétés de finitude ou de degré de séquentialité.

Sans référence aux noms, on ne peut toutefois pas mener le même programme de caractérisation par une bisimilarité intensionnelle, parce qu'une telle relation serait difficile à étudier. En effet, on perd la propriété de congruence par rapport à $|$ comme on l'a déjà dit en introduction de la section 7.1.2, à cause de l'invariance de $\models$ par permutation de noms. La logique n'est toutefois pas invariante par substitution en général, puisque par exemple $a|\overline{b} \not\models \diamond\top$ mais $a|\overline{a} \models \diamond\top$. Pour mieux comprendre la sensibilité de la logique aux noms, on considère l'extension avec modalités d'action et quantificateurs $\mathcal{L}^{(CCS)}_{mod}$. Pour cette logique, on a clairement

$$P =_L Q \quad \text{ssi} \quad \text{il existe une permutation } \sigma \text{ t.q. } P \equiv Q\sigma$$

puisque l'on sait donner des formules caractéristiques. On montre ici que les deux logiques sont en réalité aussi expressives, et que les quantificateurs sont, dans un certain sens, dérivables dans $\mathcal{L}_{spat}^{\diamond}$.

8.2.1 Equivalence entre $\mathcal{L}_{spat}^{\diamond}$ et $\mathcal{L}_{mod}^{(CCS)}$

On veut donc donner un encodage de $\mathcal{L}_{mod}^{(CCS)}$ dans $\mathcal{L}_{spat}^{\diamond}$. La première difficulté technique revient à donner une représentation de la relation $P, v \models \mathcal{A}$ sur $\mathcal{L}_{mod}^{(CCS)}$ à partir de la satisfaction sans valuation dans $\mathcal{L}_{spat}^{\diamond}$. On choisit donc d'encoder la valuation v sous la forme d'un processus $\mathbf{val}(v)$ à placer en parallèle avec le véritable processus à tester P. Afin de distinguer P de $\mathbf{val}(v)$, on fait l'hypothèse que P est de degré de séquentialité inférieur à K, et on construit $\mathbf{val}(v)$ de degré de séquentialité supérieur à K, si bien qu'un test de taille (expliqué ci-dessous) suffit à différencier P de $\mathbf{val}(v)$. Notons $\mathsf{ds}(\mathcal{A})$ l'imbrication maximale de modalité $\diamond$ dans $\mathcal{A}$. On va maintenant établir le résultat suivant :

Théorème 8.2.1 *Pour toute formule close $\mathcal{A} \in \mathcal{L}_{mod}^{(CCS)}$, et pour tout entier $K > \mathsf{ds}(\mathcal{A})$, on peut construire une formule $[\![\mathcal{A}]\!]_K \in \mathcal{L}_{spat}^{\diamond}$ telle que pour tout processus P,*

$$P \models [\![\mathcal{A}]\!]_K \quad ssi \quad \begin{cases} P \models \mathcal{A} \ et \\ \mathsf{ds}(P) \leq K \end{cases}$$

On ne montre donc pas une équivalence d'expressivité stricte entre les deux logiques, mais sous une hypothèse de taille sur les processus. Toutefois, d'une part cela suffit pour établir que les deux logiques induisent la même équivalence entre modèles, d'autre part la prise en compte de termes de taille arbitraire n'est pas problématique. Notons M_k l'ensemble des processus de degré de séquentialité inférieur ou égal à k, et M_∞ l'ensemble des processus complet. On a le résultat suivant :

Proposition 8.2.2 *Pour toute formule $\mathcal{A} \in \mathcal{L}_{mod}^{(CCS)}$, pour tout $k > \mathsf{ds}(\mathcal{A})$, pour tout processus P, et pour toute valuation v,*

$$P, v \models_{M_\infty} \mathcal{A} \quad ssi \quad \pi_k(P), v \models_{M_\infty} \mathcal{A} \quad ssi \quad \pi_k(P), v \models_{M_k} \mathcal{A}.$$

où $\pi_k(P)$ est la troncature de P à profondeur k.

Preuve: Par induction sur $\mathcal{A}$. □

On peut résumer quelques conséquences immédiates de ces deux derniers résultats de la façon suivante :

Théorème 8.2.3 *Pour tout $P \in \mathsf{CCS}$, il existe une formule $\mathcal{F}_P$ dans $\mathcal{L}_{spat}^{\diamond}$ telle que pour tout Q, les conditions suivantes soient équivalentes :*

1. $Q \models \mathcal{F}_P$
2. $P =_L Q$ *dans* $\mathcal{L}_{spat}^{\diamond}$
3. $P =_L Q$ *dans* $\mathcal{L}_{mod}^{(CCS)}$
4. *il existe une permutation d'actions σ telle que $P \equiv Q\sigma$*

Preuve: On a (4) $\Rightarrow$ (3) par absence de nom libres dans la logique, (3) $\Rightarrow$ (2) car $\mathcal{L}^{\diamond}_{spat} \subset \mathcal{L}^{(CCS)}_{mod}$, et (2) $\Rightarrow$ (1) car $\mathcal{F}_P \in \mathcal{L}^{\diamond}_{spat}$. On pose $\mathcal{F}_P = [\![\chi_P]\!]_{\mathsf{ds}(()P)+1}$ où χ_P est la formule caractéristique de P dans $\mathcal{L}^{(CCS)}_{mod}$ (obtenue en suivant la syntaxe de P), de sorte que $Q \models \chi_P$ ssi (4) est vérifiée. Alors $Q \models \mathcal{F}_P$ entraîne $Q \models \chi_P$, donc (1) $\Rightarrow$ (4). $\Box$

Enfin on peut mentionner le résultat suivant qui répond à la question initiale de l'élimination des adjoints dans les logiques pour la concurrence :

Théorème 8.2.4 *Les logiques spatiales pour la concurrence (CCS, π-calcul, Ambients) n'admettent pas l'élimination des adjoints, même dans une renormalisation prenant en compte des modalités d'action.*

Preuve: Voir annexe D.4. $\Box$

Ce résultat est une conséquence indirecte de l'élimination des quantificateurs, dans la mesure où les quantificateurs font obstacle à l'élimination des adjoints (voir section 5.2.3).

La preuve du théorème 8.2.1 nécessite un certain travail, d'une part dans la représentation du processus $\mathbf{val}(v)$, d'autre part dans sa caractérisation logique. Par la suite, on présente les idées essentielles de la preuve.

8.2.2 Le processus valuation

Pour représenter v, on associe une taille à chaque variable par une *numérotation* $e : \mathsf{VLog} \rightharpoonup_{fin} \mathbb{N}$ (injective), puis un nom à chaque taille par un *nommage* $\nu : \mathbb{N} \rightharpoonup_{fin} \mathsf{Act}$, de sorte que
$$v \;=\; \nu \circ e.$$

Ceci permet de représenter chaque assignation $x \mapsto \alpha$ de la valuation v par un processus séquentiel de taille correspondante, soit le processus

$$\mathbf{row}(n, \alpha) \quad \stackrel{\text{def}}{=} \quad \underbrace{\alpha.\alpha \ldots \alpha.\mathbf{0}}_{n \text{ fois}}$$

où n est la taille associée à x. On note $|e|$ la dernière taille introduite, soit la valeur maximale de e, qui est aussi l'entier tel que $\mathsf{dom}(\nu) = \{1, \ldots, |e|\}$. Le processus valuation est donc la composition parallèle de processus séquentiels de tailles distinctintes, chacun représentant l'une des affectations $x \mapsto \alpha$. On prendrait donc le processus valuation :

$$\prod_{i=1\ldots|e|} \mathbf{row}(K + i, \nu_i)$$

Cependant, la règle $|$ force à découper le processus $\mathbf{val}(v)$. Pour que chaque composante parallèle dispose du processus valuation, il doit donc contenir plusieurs copies des séquentiels $\mathbf{row}(n, \alpha)$. Pour cela, on compte par avance sur une formule $\mathcal{A}$ le nombre maximal de $|$ imbriqués, noté $w(\mathcal{A})$, et on recopie le processus précédent 2^w fois. Cela donne finalement :

$$\mathbf{val}(e, \nu, w)_K \;\stackrel{\text{def}}{=}\; \prod_{i=1\ldots|e|} \mathbf{row}(K + i, \nu_i)^{2^w}$$

Il est intéressant de noter qu'avec cette définition de l'environnement on peut contrôler simplement les interactions avec le processus testé P. En effet, si $P|\mathbf{val}(e,\nu,w)_K \longrightarrow R$ et si R est de la forme $R \equiv P'|\mathbf{val}(e,\nu,w)_K$, alors la réduction a nécessairement lieu dans P, puisque tous les séquentiels de $\mathbf{val}(e,\nu,w)_K$ ont gardé leur taille. De même, si l'on découpe pour le distribuer le processus valuation $\mathbf{val}(e,\nu,w)_K \equiv \mathbf{val}(e,\nu',w-1)_K|\mathbf{val}(e,\nu'',w-1)_K$, on est assuré que $\nu = \nu' = \nu''$ puisque les séquentiels de même taille doivent représenter la même action.

Pour caractériser logiquement cette construction, on utilise des jeux contextuels simples. Toutes les formules sont données dans la figure ci-dessous. On a besoin d'une part de caractériser les processus $\mathbf{row}(n,\alpha)$, ce que fait la formule $\mathsf{Row}(n)$; puis on doit tester l'égalité de noms pour assurer que les différentes copies de séquentiels de même taille sont identiques, ce que fait la formule $\mathsf{Equals}(k)$; enfin on doit mesurer le degré de séquentialité d'un processus, ce que fait la formule $\mathcal{M}_k$.

formule	implémentation	interprétation
$\mathsf{Thread}(1)$	$1 \wedge (1 \blacktriangleright \Diamond 0)$	$\exists\alpha.\ P \equiv \alpha.\mathbf{0}$
$?.\mathcal{A}$	$1 \wedge (\mathsf{Thread}(1) \blacktriangleright \Diamond\mathcal{A})$	$\exists\alpha, P'.\ P \equiv \alpha.P',\ P' \models \mathcal{A}$
$\mathsf{Thread}(k+1)$	$?.\mathsf{Thread}(k)$	$\exists\alpha_1,\ldots,\alpha_{k+1}$ $P \equiv \alpha_1\ldots\alpha_{k+1}$
$\mathcal{M}_0$	0	
$\mathcal{M}_{k+1}$	$(1 \rightarrow ?.\mathcal{M}_k)^{\forall}$	$P \in M_{k+1}$
$\mathsf{Equals}(k)$	$\mathcal{M}_k \wedge (\mathsf{Thread}(k+1) \blacktriangleright$ $\big((\mathsf{Thread}(k+1)\|1) \rightarrow \Diamond\top\big)^{\forall})$	$\exists\alpha, \tilde{P} \in M_{k-1}$ $P \equiv \alpha.P_1\|\ldots\|\alpha.P_r$
$\mathsf{RowCol}(0)$	0	
$\mathsf{RowCol}(k+1)$	$\big(\mathsf{Thread}(n+1)\|\mathsf{Equals}(1)\big)$ $\wedge\Diamond\mathsf{RowCol}(n)$	$\exists\alpha.\ P \equiv$ $\mathbf{row}(k+1,\alpha)\|\overline{\alpha}\|\overline{\alpha}\|\ldots\|\overline{\alpha}$
$\mathsf{Row}(k)$	$\mathsf{Thread}(n) \wedge (\top \blacktriangleright \mathsf{RowCol}(n))$	$\exists\alpha.\ P \equiv \mathbf{row}(k,\alpha)$

Ces formules permettent de caractériser un processus valuation et un processus testé. On pose

$$
\begin{array}{rcl}
\mathsf{Val}(e,w)_K & \stackrel{\text{def}}{=} & \prod_{i=1\ldots|e|} \big(\mathsf{Row}(K+i)^{2^w} \wedge \mathsf{Equals}(K+i)\big) \\
\mathsf{ProcVal}(e,w)_K & \stackrel{\text{def}}{=} & \mathcal{M}_K|\mathsf{Val}(e,w)_K
\end{array}
$$

Lemme 8.2.5 *Pour tout processus P, pour toute enumération e et pour tous entiers $K, w \geq 1$,*

$$
\begin{array}{llll}
P \models & \mathsf{Val}(e,w)_K & \textit{ssi} & \exists\nu.\ P \equiv \mathbf{val}(e,\nu,w)_K \\
P \models & \mathsf{ProcVal}(e,w)_K & \textit{ssi} & \exists Q \in M_K, \exists\nu.\ P \equiv Q|\mathbf{val}(e,\nu,w)_K
\end{array}
$$

On vient donc de voir que l'on peut représenter une valuation comme un processus placé en parallèle avec le processus réellement testé, et que l'on sait dissocier ces deux parties et contrôler leurs interactions. Il reste maintenant à voir comment utiliser ce processus valuation pour encoder la quantification et les modalités d'action de $\mathcal{L}_{mod}^{(CCS)}$.

8.2.3 Encodage des quantificateurs et des modalités

Le processus valuation représente chaque assignation $x \mapsto \alpha$ par un jeu de copies d'un même séquentiel de taille $e(x) + K$, que l'on peut caractériser par la formule :

$$\mathsf{EnvX}(x, e, w)_K \stackrel{\text{def}}{=} \mathsf{Equals}(K + |e|) \ \wedge \ (\mathbf{row}(e(x)+_K,)\ 2^w$$

Le quantificateur $\exists x.\mathcal{A}$ quant à lui peut être vu comme l'action d'introduire dans la valuation courante une nouvelle association $x \mapsto \alpha$. Du point de vue du processus valuation, il s'agit donc d'une extension par un nouveau jeu de copies en parallèle, de taille $|e| + 1$. Cette opération est traduite à l'aide du connecteur $\blacktriangleright$:

$$[\![\exists x.\mathcal{A}]\!]_{(e,w)} \stackrel{\text{def}}{=} \mathsf{ProcVal}(e, w)_K \ \wedge \big(\mathsf{EnvX}(x, e', w)_K \ \blacktriangleright \ [\![\mathcal{A}]\!]_{(e',w)}\big)$$

avec $e' = e\{^x/|e| + 1\}$.

Représenter les modalités est légèrement plus technique. On adopte toujours la même idée de forcer une interaction entre le processus testé et un processus testeur qu'on lui rajoute. Ici, on construit le processus testeur pour la modalité $\langle x \rangle\mathcal{A}$ comme un complémentaire du séquentiel de taille $e(x)$ présent dans le processus valuation, tandis que pour $\langle \overline{x} \rangle$ on utilise directement l'un des séquentiels de taille $e(x)$. Dans les deux cas, on "nettoie" le processus après l'interaction pour le retrouver de la forme $P|\mathbf{val}(e, \nu, w)_K$. Ces tests et nettoyages se font au moyen des formules suivantes :

$$\begin{array}{rcl}
\mathsf{XRow}(x, e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(K + e(x)) \\
\mathsf{UsedXRow}(x, e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(K + e(x) - 1) \\
\\
\mathsf{Test}(e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(|e| + K + 2) \\
\mathsf{UsedTest}(e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(|e| + K + 1) \\
\mathsf{TestMatchesX}(x, e, w)_K & \stackrel{\text{def}}{=} & (\mathsf{Test}(e)_K|\mathsf{EnvX}(x, e, w)_K) \ \wedge \ \Diamond\top
\end{array}$$

Quant aux autres connecteurs de $\mathcal{L}_{mod}^{(CCS)}$, ils sont déjà présents dans la logique $\mathcal{L}_{spat}^{\Diamond}$ et se représentent donc naturellement, en tenant compte toutefois du processus valuation et du nombre de copies dont on dispose pour chaque séquentiel. On définit finalement l'encodage de $\mathcal{L}_{mod}^{(CCS)}$ dans $\mathcal{L}_{spat}^{\Diamond}$ de la façon suivante :

$$
\begin{array}{lcl}
[\![\mathcal{A} \wedge \mathcal{B}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge [\![\mathcal{A}]\!]_{(e,w)} \ \wedge \ [\![\mathcal{B}]\!]_{(e,w)} \\
[\![\neg\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \neg [\![\mathcal{A}]\!]_{(e,w)} \\
[\![0]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \mathsf{Val}(e,w)_K \\
[\![\mathcal{A}|\mathcal{B}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge ([\![\mathcal{A}]\!]_{(e,w-1)} \mid [\![\mathcal{B}]\!]_{(e,w-1)}) \\
[\![\mathcal{A} \triangleright \mathcal{B}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \\
& & \quad \big([\![\mathcal{A}]\!]_{(e,w)} \ \triangleright \ \big(\mathsf{ProcVal}(e,w+1)_K \rightarrow [\![\mathcal{B}]\!]_{(e,w+1)}\big)\big) \\
[\![\Diamond\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \Diamond [\![\mathcal{A}]\!]_{(e,w)} \\
[\![\exists x.\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \big(\mathsf{EnvX}(x,e',w)_K \ \blacktriangleright \ [\![\mathcal{A}]\!]_{(e',w)}\big) \\
& & \quad \text{avec } e' = e\{^x/|e|+1\} \\
[\![\langle x\rangle\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \\
& & \quad \mathsf{Test}(e)_K \ \blacktriangleright \ \big((\mathsf{TestMatchesX}(x,e,w)_K|\top) \ \wedge \\
& & \qquad\qquad \Diamond(\mathsf{UsedTest}(e)_K|[\![\mathcal{A}]\!]_{(e,w)})\big) \\
[\![\langle\overline{x}\rangle\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \\
& & \quad \Diamond\big(\mathsf{UsedXRow}(x,e)_K \mid (\mathsf{XRow}(x,e)_K \ \blacktriangleright \ [\![\mathcal{A}]\!]_{(e,w)})\big)
\end{array}
$$

Cet encodage est validé par le résultat suivant :

Lemme 8.2.6 *Pour toute formule $\mathcal{A} \in \mathcal{L}_{mod}^{(CCS)}$, pour toute numérotation e telle que $\mathsf{fn}(\mathcal{A}) \subseteq \mathsf{dom}(e)$, pour tout entier $w > w(\mathcal{A})$, pour tout $K > 0$, et pour tout processus P,*

$$P \ \models_{M_\infty} \ [\![\mathcal{A}]\!]_{(e,w)} \qquad ssi \qquad \exists Q \in M_K, \exists \nu. \begin{cases} P \equiv Q|\mathbf{val}(e,\nu,w)_K \\ Q, \nu \circ e \models_{M_K} \mathcal{A} \end{cases}$$

De plus, si $\mathcal{A}$ est une formule close, $\mathbf{val}(e,\nu,w)_K = \mathbf{0}$, et on a donc

$$P \models_{M_\infty} \ [\![\mathcal{A}]\!]_{(e,w)} \ \ ssi \ \ P \in M_K \ et \ P \models_{M_K} \mathcal{A}.$$

Le théorème 8.2.1 se déduit alors du lemme D.2.6 pour le cas où $e = \emptyset$, ce qui donne en effet $\mathbf{val}(e,\nu,w)_K = \mathbf{0}$ et le résultat souhaité.

Un fait important est que ce résultat n'est pas spécifique au calcul CCS mais se généralise dans une certaine mesure au π-calcul et aux Mobile Ambients (voir annexe D.6)).

On peut aussi noter que $\mathcal{L}_{mod}^{(CCS)}$ sait exprimer le quantificateur générique si l'on fait l'hypothèse que le processus que l'on considère est de degré de séquentialité borné.

Cette dernière hypothèse peut sembler un peu décevante, et on pourrait attendre un résultat d'équivalence expressive stricte entre $\mathcal{L}_{spat}^{\Diamond}$ et $\mathcal{L}_{mod}^{(CCS)}$. On ne sait pas comment dériver un tel résultat dans l'état actuel. En revanche, il est sûrement possible d'obtenir une équivalence expressive stricte si l'on autorise la quantification générique et les modalités d'action dans $\mathcal{L}_{spat}^{\Diamond}$. En effet, la difficulté essentielle est de savoir distinguer le processus réellement testé du processus valuation, et si l'on peut marquer le processus $\mathbf{val}(e,\nu,w)_K$ par un nom extérieur au processus réellement testé, alors on peut remplacer le test de taille par le test de ce nom frais et ne faire aucune hypothèse sur le processus testé.

8.3 Conclusion

On a donné deux illustrations de minimalisation de logiques spatiales.

D'une part, pour une logique spatiale statique, on a montré que l'on pouvait, de façon non constructive, remplacer toute formule par une formule équivalente sans adjoints, ce qui montre que contrairement au cas dynamique, les adjoints n'apportent pas d'expressivité supplémentaire à la logique. On a de plus démontré que ce sont les seuls connecteurs superflus au sens où la logique sans adjoints est minimale. On a enfin montré que cette minimalisation repose de façon déterminante sur l'absence de quantification existentielle.

D'autre part, pour une logique spatiale dynamique, on a montré que l'on pouvait exprimer, dans un fragment minimal très dépouillé, à la fois des modalités d'actions et la quantification existentielle sur les noms d'actions. La minimalisation est dans ce cas constructive, ce qui, on va le voir maintenant, a certaines conséquences sur la décidabilité[7] des logiques spatiales pour la concurrence.

[7] En logique classique, l'élimination des quantificateurs s'interprète aussi en terme de décidabilité et de complexité pour le modèle de calcul associé [BPR94]. Toutefois, on n'a pas ici de véritable modèle de calcul attaché à la logique et il est difficile de faire un lien entre le cas classique et le cas spatial.

Chapitre 9

Problèmes de décision

Les chapitres précédents ont tenté de caractériser l'expressivité des logiques spatiales et des algèbres de processus d'un point de vue relativement théorique, par des comparaisons avec d'autres objets mathématiques préexistants. Pour conclure cette thèse, on adopte le point de vue de l'informatique et de la possibilité de représenter les objets que l'on a définis et d'en automatiser la manipulation. Ce point de vue fournit lui aussi une évaluation de l'expressivité de ces objets. Historiquement, les travaux de Turing sur la calculabilité ou de Gödel sur la logique classique ont montré qu'une notion de calcul ou qu'un cadre formel de raisonnement qui sont suffisamment expressifs pour se représenter eux-mêmes ne sont pas mécanisables.

C'est donc ce dernier critère d'expressivité que l'on étudie ici[1].

On s'intéresse tout d'abord au calcul des Ambients, pour lequel peu de résultats de décidabilité étaient connus au début de la thèse. On montre que, de même que pour le π-calcul, l'évolution, et particulièrement la terminaison ou la cyclicité des processus, sont en général imprévisibles. Plus précisément, on s'intéresse au problème de l'axiomatisation de l'équivalence logique sur ce calcul dans le cas général (on a donné une axiomatisation significative pour le fragment MA_{IF}^{syn} au chapitre 7, mais seulement une caractérisation en terme de bisimilarité intensionnelle pour le calcul complet), et on montre qu'aucune axiomatisation mécanisable n'existe. La preuve de ce résultat pour le calcul des Ambients est très différente de celle pour le π-calcul, qui définit un encodage du λ-calcul, et pour laquelle les notions de communication et de noms restreints sont essentielles. Ici, on se base uniquement sur la mobilité des localités pour représenter non pas le λ-calcul, mais des machines de Turing.

On s'intéresse ensuite à la question de la décidabilité des logiques spatiales. Cette question a été longuement étudiée [CTZ+04, CZG+01, CYO01, CCG03] et de nombreux résultats concernant leur décidabilité ont été obtenus. On en rappelle ici les principaux. Il est à noter que la décidabilité de la logique $\mathcal{L}_{spat}^{\diamond}$ est restée assez longtemps sans réponse. On verra dans cette section que l'étude de son expressivité et la minimalisation de $\mathcal{L}_{mod}^{(CCS)}$ permettent de répondre à cette question.

[1]Dans certains cas, on mentionnera par ailleurs brièvement la complexité calculatoire de ces objets, qui elle aussi donne une mesure pertinente de leur expressivité.

9.1 Décidabilité de l'équivalence logique

Dans cette section, on étudie la décidabilité de l'équivalence logique sur le calcul des Ambients. On a vu que l'équivalence logique coïncide plus ou moins exactement avec la congruence structurelle. En dehors des cas d'intensionalité stricte, on a rencontré la congruence structurelle modulo eta conversion (calcul MA_{IF}^{syn}, chapitre 7) ou encore modulo permutation d'actions (CCS, chapitre 8). Le seul cas où l'on n'a pas du donner une interprétation précise de $=_L$ par rapport à $\equiv$ est le cas des Mobile Ambients sans hypothèse de finitude.

On montre maintenant que $=_L$ est indécidable dans ce cas précis, alors que justement dans tous les autres cas, on peut décider $=_L$. L'exercice est intéressant en ce qu'il permet de jouer avec le calcul des Ambients et d'en illustrer le pouvoir calculatoire.

Afin de montrer l'indécidabilité de $=_L$, on développe un encodage des machines de Turing à l'intérieur du calcul. On a vu que les processus $\mathsf{open}\, n.P$ et $\mathsf{open}\, n.Q$ sont logiquement équivalents ssi il existe une boucle de réduction commune à P et Q. La décidabilité de $=_L$ dépend donc de la décidabilité de la relation “être sur la même boucle de réduction”. De là, on construit justement des processus P, Q qui sont sur la même boucle de réduction à condition qu'une machine de Turing donnée s'arrête. On prend de plus ces processus P, Q dans le calcul restreint MA_{IF}^{syn} ce qui montre sa Turing complétude.

9.1.1 Rubans

Digits et mots On associe aux booléens vrai et faux deux noms distincts tt et ff ; un digit d est l'un de ces deux noms, et un mot w est une concaténation $w^1 \dots w^k$ de digits. Dans une machine de Turing, les mots sont inscrits sur des rubans.

Un ruban est une imbrication de cellules, chaque cellule contenant un digit. Au départ, un ruban est gelé, puis après activation il entame une phase de croissance, jusqu'à une taille arbitraire, ce qui permet de simuler un ruban infini. Cette croissance est assurée par un agent extenseur qui crée une nouvelle cellule dans la cellule la plus imbriquée du ruban. Après sa croissance le ruban sert à l'exécution d'une machine de Turing, et quand cette exécution s'achève, la machine de Turing le désactive et il devient un vieux ruban, qui ne joue plus aucun rôle.

On définit maintenant des contextes de processus pour définir l'encodage. Pour ne pas confondre avec la notation $n[P]$ du calcul des Ambients, on adopte la notation $[\,]$ pour les trous de contexte et $\mathsf{C}\{\!|P|\!\}$ pour le remplissage du contexte C par le processus P.

Cellules et Rubans		
$\mathtt{cell}(d)[\,]$	$:=$	$cell[d[\mathbf{0}] \mid !\mathsf{open}\, wo \mid [\,]\,]$
$\mathtt{word}(w)[\,]$	$:=$	$\mathtt{cell}(w^1)\{\!\vert \mathtt{cell}(w^2)\{\!\vert \ldots \mathtt{cell}(w^r)[\,] \ldots \vert\!\}\vert\!\}$ $\qquad (w = w^1w^2\ldots w^r)$
Extenseur de ruban		
$\mathtt{deadextcode}$	$:=$	$!\mathsf{open}\, coin.\mathsf{open}\, newcell.\mathsf{in}\, cell.coin[\mathbf{0}] \mid !newcell[\mathtt{cell}(\mathit{ff})\{\!\vert \mathsf{out}\, ext \vert\!\}]$
$\mathtt{sendstart}$	$:=$	$msg[\mathsf{out}\, ext.!\mathsf{out}\, cell \mid \mathsf{out}\, ribbon\_left.start[\mathsf{in}\, TM]]$
$\mathtt{ExtensorFrozen}$	$:=$	$ext[\mathtt{deadextcode} \mid \mathsf{open}\, coin.\mathtt{sendstart}]$
$\mathtt{ExtensorAlive}$	$:=$	$ext[coin[\mathbf{0}] \mid \mathtt{deadextcode} \mid \mathsf{open}\, coin.\mathtt{sendstart}]$
$\mathtt{ExtensorDead}$	$:=$	$ext[\mathtt{deadextcode}]$
Rubans		
$\mathtt{cleaninst}$	$:=$	$\mathsf{open}\, cleaner.\mathsf{open}\, runclean \mid runclean[\mathtt{deadcleancode}]$
$\mathtt{deadcleancode}$	$:=$	$!\mathsf{open}\, \mathit{ff} \mid !\mathsf{open}\, tt \mid !\mathsf{open}\, cell \mid !\mathsf{open}\, wo$
$\mathtt{FrozenRibbon}(w)$	$:=$	$ribbon\_left[\mathtt{cleaninst} \mid \mathtt{word}(w)\{\!\vert \mathtt{ExtensorFrozen} \vert\!\}]$
$\mathtt{GrowingRibbon}(w)$	$:=$	$ribbon\_left[\mathtt{cleaninst} \mid \mathtt{word}(w)\{\!\vert \mathtt{ExtensorAlive} \vert\!\}]$
$\mathtt{WorkRibbon}(w_1, w_2)[\,]$	$:=$	$ribbon\_left[\mathtt{cleaninst}] \mid \mathtt{word}(w_1)\{\!\vert [\,] \mid \mathtt{word}(w_2)\{\!\vert \mathtt{ExtensorDead} \vert\!\}\vert\!\}$
$\mathtt{OldRibbon}$	$:=$	$ribbon\_left[\mathtt{deadcleancode} \mid \mathtt{ExtensorDead}]$

Fait 9.1.1 *Pour tout mot w et pour tout $n \in \mathbb{N}$, soit $P_n = \mathtt{GrowingRibbon}(w.\mathit{ff}^n)$. Alors :*

- $P_n \;\Rightarrow\; P_{n+1}$;
- $P_n \;\Rightarrow\; R$ *avec* $R = \mathtt{WorkRibbon}(\epsilon, w.\mathit{ff}^n)\{\!\vert msg[!\mathsf{out}\, cell \mid \mathsf{out}\, ribbon\_left.start[\mathsf{in}\, TM]]\vert\!\}$;
- *pour tout terme Q étape intermédiaire dans la réduction $P_n \;\Rightarrow\; P_{n+1}$ ou $P_n \;\Rightarrow\; R$, il existe Q' tel que $Q \equiv ribbon\_left[Q']$.*

De plus, pour tout mot w,

$$\mathtt{WorkRibbon}(w, \epsilon)\{\!\vert \mathbf{0} \vert\!\} \mid cleaner[\mathsf{in}\, ribbon\_left] \quad\Rightarrow\quad \mathtt{OldRibbon}.$$

9.1.2 Machine de Turing

Définition 9.1.2 (Machine de Turing) *On introduit trois symboles $\leftarrow$, $\downarrow$ et $\rightarrow$ pour chaque mouvement de la tête de lecture.*

Une Machine de Turing est un quadruplet $(\mathcal{Q}, q_{start}, q_A, \delta)$ où $\mathcal{Q}$ est l'ensemble des états, q_{start} est l'état initial, q_A est l'état d'acceptation, et $\delta : \mathcal{Q} \times \{\mathit{ff}, tt\} \to \mathcal{Q} \times \{\mathit{ff}, tt\} \times \{\leftarrow, \downarrow, \rightarrow\}$ est la fonction de transition.

Notation : on écrit

$$(w_1, q, w_2) \rightarrowtail\!\!\!\twoheadrightarrow (w'_1, q', w'_2)$$

lorsque la machine de Turing dans l'état q, avec la tête de lecture sur la dernière lettre de w_1 et avant la première lettre de w_2, évolue en un pas de calcul vers l'état q' avec la tête de lecture entre w'_1 et w'_2.

L'encodage de la machine de Turing consiste à définir un agent "tête de lecture" qui contient l'état dans lequel se trouve la machine ainsi que la liste des états et des actions associées à chaque digit lu. Un programme spécial s'exécute lorsque l'état d'acceptation est atteint : la tête de lecture ressort du ruban, envoie un agent qui nettoie le ruban et le transforme en vieux ruban, dégèle un ruban gelé et commence une nouvelle exécution lorsque celui-ci a fini sa phase de croissance. Cette dernière phase s'explique par le fait que l'on veut construire une boucle de réduction.

Fonction de transition

$$
\begin{array}{lcl}
\mathtt{clear}(d); P & := & wo[\ \mathsf{out}\, head.\mathsf{open}\, d.cl\_ack[\mathsf{in}\, head]\ ] \\
 & & \quad \mid\ \mathsf{open}\, cl\_ack.P \\
\mathtt{write}(d); P & := & wo[\ \mathsf{out}\, head.d[\mathbf{0}] \mid wr\_ack[\mathsf{in}\, head]\ ] \ \mid\ \mathsf{open}\, wr\_ack.P \\
\mathtt{become}(mo); P & := & mo[\mathsf{out}\, head.\mathsf{open}\, head.P] \mid \mathsf{in}\, mo \\
\mathtt{domove}(mv); P & := & \begin{cases} \mathsf{in}\, cell.P & \text{if } mv = \leftarrow \\ P & \text{if } mv = \downarrow \\ \mathsf{out}\, cell.P & \text{if } mv = \rightarrow \end{cases} \\
\mathtt{transcode}(d_r, q_w, d_w, mv) & := & \mathtt{clear}(d_r); \mathtt{write}(d_w); \mathsf{become}(mo); \mathsf{in}\, TM.\mathsf{domove}(mv); \mathsf{open}\, q_w
\end{array}
$$

Etat

$$
\begin{array}{lcl}
ff{\rightarrow}P\ +\ tt{\rightarrow}Q & := & coin[\mathsf{in}\, ff.\mathsf{out}\, ff.P] \mid coin[\mathsf{in}\, tt.\mathsf{out}\, tt.Q] \mid \mathsf{open}\, coin \\
\mathsf{code}(q) & := & !q[head[\ \mathsf{out}\, TM.(\ ff{\rightarrow}\mathtt{transcode}(ff, d_{ff}, q_{ff}, mv_{ff}) \\
 & & \qquad\qquad + \ tt{\rightarrow}\mathtt{transcode}(tt, d_{tt}, q_{tt}, mv_{tt}))\ ]] \\
 & & \mid\ !coin[\mathsf{in}\, ff.\mathsf{out}\, ff.\mathtt{transcode}(ff, d_{ff}, q_{ff}, mv_{ff})] \\
 & & \mid\ !coin[\mathsf{in}\, tt.\mathsf{out}\, tt.\mathtt{transcode}(tt, d_{tt}, q_{tt}, mv_{tt})] \\
\mathsf{code}(q_A) & := & !q_A[\,get\_out[\mathbf{0}]\,]
\end{array}
$$

Fin d'exécution de la machine

$$
\begin{array}{l}
\mathtt{getout} := \\
\quad !\mathsf{open}\, get\_out.\mathsf{out}\, cell.get\_out[\mathbf{0}] \\
\mid\ !\mathsf{open}\, get\_out.\mathsf{out}\, ribbon\_left.(\quad cleaner[\mathsf{out}\, TM.\mathsf{in}\, ribbon\_left] \\
\qquad\qquad \mid\ coin[\mathsf{out}\, TM.\mathsf{in}\, ribbon\_left.\mathsf{in}\, cell^{\mathsf{length}(w)}.\mathsf{in}\, ext] \\
\qquad\qquad \mid\ \mathsf{open}\, start.\mathsf{in}\, ribbon\_left.\mathsf{in}\, cell.\mathsf{open}\, q_{start}\ )
\end{array}
$$

Définition 9.1.3 *L'encodage de la machine de Turing Machine est basé sur un ambient TM qui contient le codage de la fonction de transition*

$$\mathtt{tmsoup} \ := \ \mathsf{code}(q_0)|\ldots|\mathsf{code}(q_n) \mid \mathtt{getout} \mid !\mathsf{open}\, mo\,.$$

On considère deux configurations particulières de la machine. Avant d'être active, la machine est dans l'état de départ, défini par

$$\mathtt{TMStart} \ := \ TM[\ \mathsf{open}\, start.\mathsf{in}\, ribbon\_left.\mathsf{in}\, cell.\mathsf{open}\, q_{start} \mid \mathtt{tmsoup}\ ]\,.$$

Une fois que le calcul est lancé, la machine dans l'état q est représentée par le processus

$$\mathtt{TM}(q) \ := \ TM[\,\mathsf{open}\, q \mid \mathtt{tmsoup}\,]\,.$$

Fait 9.1.4 *Les processus considérés dans l'encodage appartiennent à MA_{IF}^{syn}.*

On établit maintenant un résultat de simulation pour la machine. On commence par définir une évolution déterministe :

Définition 9.1.5 *On dira qu'un processus P* évolue déterministiquement *vers Q, $P \rightsquigarrow Q$, lorsque $P{\rightarrow}Q$ et pour tout autre Q' tel que $P \rightarrow Q'$, soit $Q' \not\rightarrow$ soit $Q \equiv Q'$.*

Lemme 9.1.6 *Soit $\mathcal{M}$ une machine de Turing, q un état différent de q_A, et w_1, w_2 deux mots, avec $w_2 \neq \epsilon$. Si $(w_1, q, w_2) \rightarrowtail\!\!\!\rightarrow (w_1', q', w_2')$, alors*

$$\mathtt{WorkRibbon}(w_1, w_2)\{\!|\mathtt{TM}(q)|\!\} \ \rightsquigarrow^* \ \mathtt{WorkRibbon}(w_1', w_2')\{\!|\mathtt{TM}(q')|\!\}\,.$$

Preuve: Voir annexe B.4. □

Corollaire 9.1.7 *Soit $\mathcal{M}$ une machine de Turing, w un mot et $n \in \mathbb{N}$. La machine de Turing $\mathcal{M}$ reconnaît le mot w sur le ruban $w.\mathit{ff}^n$ ssi il existe deux mots w_1, w_2 tels que*

$$\texttt{WorkRibbon}(\epsilon, w.\mathit{ff}^n)\{\!|\texttt{TM}(q_{start})|\!\} \rightsquigarrow^* \texttt{WorkRibbon}(w_1, w_2)\{\!|\texttt{TM}(q_A)|\!\}\,.$$

On décrit maintenant la phase postacceptation de l'exécution :

Fait 9.1.8 (Acceptation) *Soient w_1, w_2 deux mots. Alors*

$$\begin{aligned}&\texttt{WorkRibbon}(w_1, w_2)\{\!|\texttt{TM}(q_A)|\!\}\\ &\qquad \Rightarrow \texttt{OldRibbon} \mid \texttt{TMStart} \mid coin[\mathsf{in}\, ribbon\_left.\mathsf{in}\, cell^{\mathsf{length}(w)}.\mathsf{in}\, ext]\end{aligned}$$

où w est le mot initial de la machine.

Cette dernière phase permet de faire redémarrer l'exécution à son état initial, de sorte que tous les processus qui représentent des états intermédiaires de la simulation de la machine sont sur une même boucle de réduction. Toutefois, si l'état d'acceptation n'est pas atteint, la phase postacceptation n'a pas lieu et une telle boucle n'existe pas :

Lemme 9.1.9 *Soit $\mathcal{M}$ une machine de Turing et w un mot. On considère les processus suivants :*

$$\begin{aligned} Q &:= \;!\texttt{FrozenRibbon}(w) \mid\, !\texttt{OldRibbon} \mid\, !\mathsf{open}\, msg \mid\, !\mathsf{out}\, cell \mid \texttt{TMStart}\,,\\ P_0 &:= \; Q \mid \texttt{GrowingRibbon}(w) \quad and \quad P_1 \;:=\; Q \mid \texttt{GrowingRibbon}(w.\mathit{ff})\,.\end{aligned}$$

Alors $P_0 \Rightarrow P_1$. De plus $P_1 \Rightarrow P_0$ si et seulement si le mot w est reconnu par la machine $\mathcal{M}$.

Théorème 9.1.10 *$=_L$ est indécidable sur* MA.

Preuve: D'après l'équivalence de $=_L$ avec $\approx_{int}$ et la clause des capabilités au lemme 7.1.16 décider si $\mathsf{open}\, n.P_0 =_L \mathsf{open}\, n.P_1$ revient à décider si $P_0 \Rightarrow\approx_{int} P_1 \Rightarrow\approx_{int} P_0$. D'après le théorème 7.3.6, ceci est équivalent à $P_0 \Rightarrow P_1 \Rightarrow P_0$, ce qui revient encore d'après le lemme précédent à décider de l'arrêt de la machine de Turing. □

Par contre, l'égalité logique sur MA^{syn}_{IF}, qui correspond à $\equiv$ (Théorème 7.3.6), est décidable [Zil00].

9.1.3 Problèmes de décision sur le calcul des ambients

On peut considérer divers problèmes de décision qui concernent purement les algèbres de processus. On ne mentionne ici que ceux liés aux Mobile Ambients, mais on peut trouver quelques résultats de décidabilité pour le π-calcul dans[SW01]. Les problèmes envisagés sont

- *terminaison* : étant donné un processus P, admet-il une réduction infinie ?
- *atteignabilité* : étant donnés deux processus P, Q, a-t-on $P \Rightarrow Q$?
- *barbe* : étant donné P, a-t-on $P \Downarrow n$?

On pourrait aussi mentionner la congruence structurelle, qui est généralement trivialement décidable [Zil00].

Cardelli et Gordon donnent les premiers un encodage des machines de Turing dans leur présentation du calcul des Ambients [CG98], mais cet encodage utilise la restriction de nom, élément clé de la Turing complétude du π-calcul. En revanche, la preuve d'indécidabilité que l'on vient d'établir montre qu'elle n'est pas nécessaire dans le calcul des Mobile Ambients :

Théorème 9.1.11 *Sur* MA_{IF}^{syn}[2]*, les problèmes de terminaison, atteignabilité et exhibition de barbe sont indécidables.*

Preuve: On reprend les processus du lemme 9.1.9. $P_1 \Rightarrow P_0$ ssi la machine de Turing s'arrête montre que l'atteignabilité est indécidable. En modifiant légèrement l'encodage (ce serait même plus simple), on peut faire arrêter le processus après terminaison de la machine (au lieu d'en relancer une autre), et dans ce cas le processus termine ssi la machine de Turing termine, ce qui montre l'indécidabilité de la terminaison. De même on peut déclencher l'exhibition d'une barbe en fin d'exécution de la machine et prouver ainsi l'indécidabilité de l'exhibition de barbe. □

Ce résultat a connu de nombreux raffinements, en particulier quant aux capabilités et forme de persistance nécessaires. On en cite ici quelques uns.

Théorème 9.1.12 ([BT03]) *Sur le fragment des ambients publics sans capabilité* open*, les problèmes de terminaison, atteignabilité et exhibition de barbe sont indécidables. En revanche, si l'on remplace la règle de congruence structurelle* $!P \equiv !P|P$ *par une règle de réduction* $!P \longrightarrow !P|P$*, le problème d'atteignabilité est décidable par réduction à l'atteignabilité sur un réseau de Petri, le problème de terminaison est décidable, et le problème d'exhibition de barbe reste indécidable.*

Théorème 9.1.13 ([BZ04]) *Sur le fragment des ambients avec restriction de nom et sans capabilités* in *et* out*, le problème de terminaison est*
- *décidable sur le calcul avec réplication,*
- *indécidable sur le calcul avec récursion.*

La terminaison est en revanche décidable sur le calcul public sans capabilités in *et* out*, à la fois pour la réplication et la récursion.*

9.2 Vérification en logiques statiques

Dans cette section, on s'intéresse aux différents problèmes de décisions attachés aux logiques spatiales sur les logiques statiques. On s'intéresse essentiellement aux deux problèmes suivants :
- satisfaction : étant donnés un processus P et une formule $\mathcal{A}$, est-ce que $P \models \mathcal{A}$?
- validité : étant donnée une formule $\mathcal{A}$, a-t-on $\vdash \mathcal{A}$, ou de façon équivalente, peut-on trouver un contre-modèle de $\mathcal{A}$?

[2] Ce résultat est donc tout aussi vrai sur MA_{IF} et MA.

Dans de nombreux cas, on a observé que l'on pouvait construire effectivement des formules caractéristiques des modèles. Ceci permet de ramener le problème de la satisfaction à celui de la validité :

$$P \models \mathcal{A} \quad \rightsquigarrow \quad \vdash \mathcal{F}_P \rightarrow \mathcal{A}$$

De même, dès que la logique comporte le connecteur $\triangleright$, on a déjà vu que l'on pouvait ramener le problème de la validité à celui de la satisfaction (voir section 4.1.1) :

$$\vdash \mathcal{A} \quad \rightsquigarrow \quad \mathbf{0} \models (\mathcal{A} \triangleright \bot) \triangleright \bot.$$

Dans ces conditions, les deux problèmes sont équivalents.

On rappelle maintenant quelques résultats connus sur la décidabilité des logiques spatiales :

Théorème 9.2.1 ([CT01]) *Dans le fragment*

$$\mathcal{A} \ ::= \ 0 \mid \mathcal{A} \wedge \mathcal{A} \mid \neg \mathcal{A} \mid \mathcal{A}|\mathcal{A} \mid \eta[\mathcal{A}] \mid \exists x.\mathcal{A}$$

de la logique des ambients, le problème de validité est indécidable.

La preuve de se résultat passe par un encodage de la logique du premier ordre sur les modèles finis, ce qui d'après le théorème de Trakhtenbrot [Tra50], en montre l'indécidabilité (voir Annexe D.5).

Comme conséquence, sur le même fragment étendu avec $\triangleright$, le problème de satisfaction est lui aussi indécidable. En revanche, sur le fragment sans $\triangleright$ la satisfaction est facilement décidable, au moins tant que le calcul n'a pas d'opérateur de réplication : pour décider $P \models \mathcal{A}|\mathcal{B}$, on essaie toutes les partitions de P en deux et on s'appelle récursivement.

La perte de $\triangleright$ dans la logique est toutefois assez peu satisfaisante. Un autre résultat montre que l'on peut décider des logiques spatiales avec $\triangleright$.

Théorème 9.2.2 ([CYO01], [CCG03]) *Dans le fragment*

$$\mathcal{A} \ ::= \ \mathsf{emp} \mid x \mapsto e \mid \mathcal{A} \wedge \mathcal{A} \mid \neg \mathcal{A} \mid \mathcal{A} * \mathcal{A} \mid \mathcal{A} -\!\!* \mathcal{A}$$

de la logique séparante, les problèmes de validité et de satisfaction sont décidables. De même dans le fragment

$$\mathcal{A} \ ::= \ 0 \mid \mathcal{A} \wedge \mathcal{A} \mid \neg \mathcal{A} \mid \mathcal{A}|\mathcal{A} \mid \mathcal{A} \triangleright \mathcal{A} \mid n[\mathcal{A}]$$

de la logique des ambients, les deux problèmes sont décidables.

L'idée originale de Peter O'Hearn est de remarquer que dans la vérification de $P \models \mathcal{A} \triangleright \mathcal{B}$, qui est la seule étape problématique pour la satisfaction, on peut simplement considérer un ensemble représentatif, fini et calculable, de processus Q sur lesquels tester $Q \models \mathcal{A}$ et $P|Q \models \mathcal{B}$. On montre en effet qu'en tronquant correctement les autres processus, on ne change pas les observations logiques et on se ramène à cet ensemble de tests.

Cette idée est très proche de celle que l'on a utilisée pour la preuve d'élimination des adjoints. En réalité, elle demande plus de travail, car on doit effectivement savoir construire l'ensemble test.

Ces deux résultats suggèrent que l'indécidabilité obtenue au théorème 9.2.1 est essentiellement due au fait que la logique comporte une forme de quantification existentielle. Il n'est pas clair que la décidabilité s'étende à la quantification générique, qui est a peine plus faible que la quantification existentielle, mais en revanche on aurait pu s'attendre à pouvoir étendre la décidabilité à la logique avec révélation. Ghelli et Conforti ont toutefois établi le résultat suivant :

Théorème 9.2.3 ([GC04]) *Le fragment de SAL sans révélation (mais avec* И*) est décidable. En revanche, le fragment de SAL avec révélation (et même sans* И*) est indécidable.*

L'idée de la preuve du résultat est que la révélation $n \circledR \mathcal{A}$ sur le processus

$$(\nu n_1)P_1|(\nu n_2)P_2|\dots|(\nu n_k)P_k$$

laisse un grand choix sur le nom à révéler, dans la même mesure que le connecteur $\exists x.\mathcal{A}$ laisse le choix sur le nom à associer à x parmi les noms libres (les autres noms étant interchangeables), et par le jeu de l'alpha conversion induite par la révélation, il est à nouveau possible d'encoder la logique du premier ordre. SAL et SAL_{int} sont donc deux logiques équivalentes en terme d'expressivité, mais la décision de $\models$ sur SAL est indécidable alors que celle sur SAL_{int} est décidable. Cet état de fait assez surprenant est dû au fait que la démonstration de l'élimination des adjoints que l'on a donnée est non constructive, et on peut même préciser maintenant :

Corollaire 9.2.4 *Il n'existe pas d'algorithme qui à une formule de SAL associe une formule équivalente dans SAL_{int}.*

Ce résultat montre bien l'écart qu'il y a entre la construction d'ensembles de test finis dans la preuve du théorème 9.2.2, et celle de précompacité pour la preuve d'élimination des adjoints, la seconde étant purement modèle-théorique.

La situation sur la décidabilité des logiques spatiales statiques peut donc se résumer ainsi : elles sont décidables tant que la logique ne comporte pas un mécanisme de quantification existentielle sur les noms, soit directement par le quantificateur $\exists x.\mathcal{A}$, soit de manière détournée par la révélation de noms $n \circledR \mathcal{A}$. En revanche, le quantificateur générique est particulièrement simple à décider.

On va maintenant étudier la question de la décidabilité sur les logiques spatiales dynamiques.

9.3 Vérification en logiques dynamiques

La question de la décidabilité des logiques spatiales dynamiques avec adjoint est restée longtemps en suspens. Dans les premières présentations de la logique des Ambients [CG00, CG01b] ou de la logique spatiale du π-calcul [CC01, CC02], les auteurs proposent des ensembles de règles de déduction logique, voire des systèmes d'inférence, mais n'établissent jamais leur complétude. On ne sait pas bien en fait comment étendre les résultats sur les ensembles tests finis lorsque vient s'ajouter la modalité de temps, parce que les méthodes de troncature échouent. Le résultat d'expressivité obtenu au chapitre précédent explique cet échec.

Théorème 9.3.1 *Sur les algèbres de processus de* CCS*, des Mobile Ambients, ou du* π*-calcul, la logique* $\mathcal{L}^{\diamond}_{spat}$ *est indécidable.*

Preuve:(abrégée) Notons que la fonction $[\![.]\!]_K : \mathcal{A} \to [\![\mathcal{A}]\!]_K$ est récursive[3]. Donc le problème de décision $P \models [\![\mathcal{A}]\!]_K$ dans $\mathcal{L}^{\diamond}_{spat}$ est équivalent à $P \models_{M_K} \mathcal{A}$ dans $\mathcal{L}^{(CCS)}_{mod}$. On adapte alors la preuve de [CT01] pour montrer que ce dernier problème est indécidable pour un K assez grand (voir annexe, section D.5). □

Il est intéressant de noter que cette indécidabilité signifie de plus l'impossibilité de mettre en place un système d'inférence mécaniquement vérifiable. En effet, par énumérabilité des inférences, il impliquerait au moins la semi-décidabilité du problème de décision qu'il représente. Or,

- si $\models$ était semi-décidable, la satisfiabilité d'une formule serait elle aussi semi-décidable, ainsi que la validité par l'encodage $\mathbf{0} \models \mathcal{A} \rhd \bot \rhd \bot$, et donc tous ces problèmes seraient complètement décidables.
- si $\vdash$ était semi-décidable, $\models$ serait lui aussi semi-décidable par l'encodage $\vdash \mathcal{F}_P \to \mathcal{A}$, et à nouveau tous les problèmes seraient décidables.

On a donc le corollaire suivant :

Corollaire 9.3.2 *Tout système d'inférence mécanique correct pour un problème de décision sur* $\mathcal{L}^{\diamond}_{spat}$ *(satisfaction ou validité) est incomplet.*

Les logiques spatiales dynamiques avec raisonnement contextuel par $\rhd$ sont donc difficiles à mettre en place en pratique[4].

Malgré ce résultat négatif, la décision de $\models$ sur des logiques sans $\rhd$ est possible. Dans le cas d'une sémantique forte pour $\diamond$, et sans réplication dans le calcul (mais éventuellement de la récursion), $\models$ est facilement décidable : pour décider $P \models \diamond \mathcal{A}$, on essaie tous les réduits en un pas de P, qui sont en nombre fini, et pour $P \models \mathcal{A}_1 | \mathcal{A}_2$, on essaie toutes les partitions en deux de P, qui sont elles aussi en nombre fini (sans !). $P \models \diamond \mathcal{A}$ est en revanche indécidable lorsque l'on se place en sémantique faible et que le calcul est Turing complet, ce qui est le cas en général. Une façon efficace de développer un algorithme de décision consiste à restreindre le calcul par une condition de contrôle qui garantit que le processus n'engendre pas de réduction infinie [CGT02, Cai04]. Les travaux de Talbot et al. montrent qu'alors la complexité de la vérification de $\models$ est généralement polynômiale en espace (PSPACE) [CZG+01].

[3] C'est toute la différence avec l'encodage de SAL dans SAL_{int}.

[4] Du moins tant qu'on le prend dans son interprétation stricte, mais l'on pourrait imaginer une forme de raisonnement contextuel plus restreinte, qui reste décidable et permet néanmoins de donner des spécifications intéressantes. Un candidat que l'on n'a pas eu le temps de considérer est le fragment positif de la logique des Ambients avec $\diamond$ fort, qui semble exprimer des conditions liées au contrôle des ressources.


عز لمولانا
السلطان ابي
عبد الله

Conclusion

Les logiques spatiales montrent des propriétés d'expressivité très particulières qui les différencient nettement des autres logiques.

Comme première particularité, on a montré qu'elles sont fortement intensionnelles : d'une part des processus. comportementalement équivalents peuvent être distingués par des observations spatiales, ce qui montre que la logique spatiale n'est pas extensionnelle contrairement à la logique modale de Hennessy-Milner, d'autre part pour être indistinguables par la logique spatiale, les processus doivent être structurellement congruents. Les connecteurs responsables de l'intensionalité sont $|$, $\mathbf{0}$ et $n[.]$ pour la logique des Ambients, alors que les connecteurs $\triangleright$ et $\Diamond$ restent à un niveau extensionnel [Hir04]. Ce fait était par ailleurs déjà mentionné dans [Dam88].

Cette situation est tout à fait originale dans la spécification de systèmes concurrents. Habituellement, les logiques proposées pour la vérification de systèmes concurrents [HM85, Dam88] sont extensionnelles. Cette particularité s'explique par le fait que les logiques spatiales sont spécifiquement des logiques pour la mobilité, et essaient de rendre les notions de localité et de mouvement par une description de la distribution des divers agents d'un système dans l'espace. Ceci montre en particulier que l'on ne souhaite pas considérer que les processus $a.a$ et $a|a$ sont équivalents pour des observations spatiales. On peut néanmoins se demander si la congruence structurelle est la seule équivalence spatiale possible. On pourrait notamment considérer les processus $a.a$ et $(\nu s)\big(a.s|\overline{s}.a\big)$ comme spatialement équivalents pour une notion d'observation spatiale qui ne révèle pas de noms restreints. Une caractérisation de l'équivalence spatiale en terme de bisimilarité est peut-être à chercher dans les travaux de Ilaria Castellani sur la bisimulation distribuée [CH89].

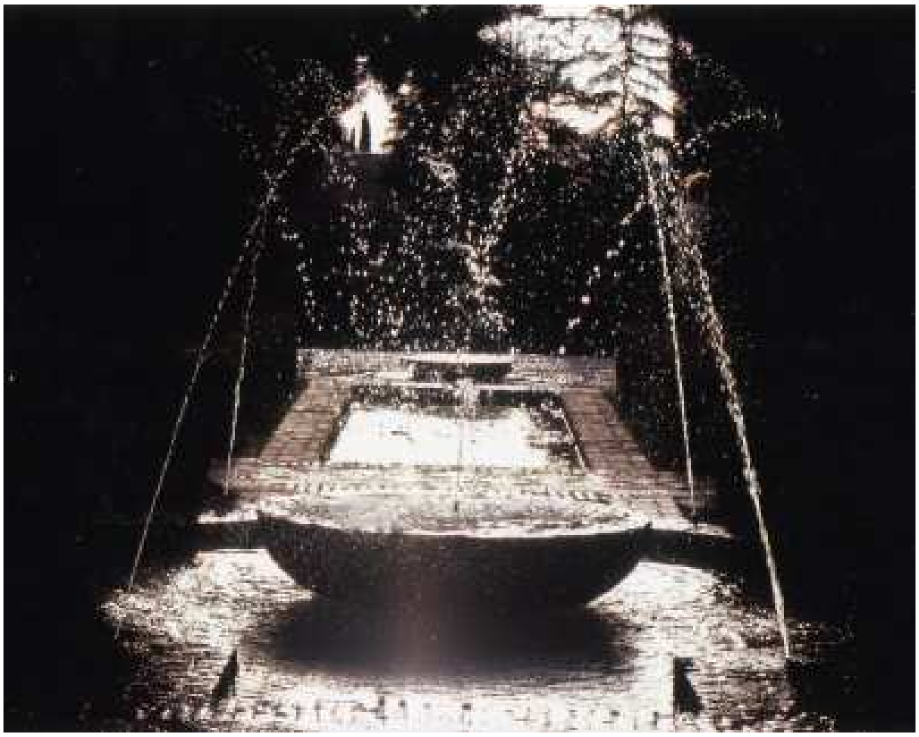
Une deuxième particularité des logiques spatiales est leur richesse en terme de connecteurs proposés. Ces connecteurs ne sont pas complètement indépendants, et on a montré hors des cas connus (encodage de Ⓒ par Ⓡ ou ⃠) des encodages non triviaux. Dans ce souci de minimalité, on a en particulier donné deux techniques de minimalisation pour des logiques spatiales statiques ou dynamiques. Les encodages proposés sont toutefois assez coûteux à mettre en place, voire non constructifs pour l'élimination de $\triangleright$ en présence de la révélation.

On a pu par ailleurs comparer les logiques spatiales avec les logiques dont elles se proposent d'être des alternatives. Ainsi la logique séparante, qui remplace la logique classique dans le rôle du langage d'assertion pour la vérification de programmes par des triplets de Hoare, apparaît dans certains fragments aussi expressive que la logique classique. Ce résultat ne signifie pas que la logique classique serait aussi bien adaptée à la spécification de langages à pointeurs, puisque d'une part le fragment pour lequel on obtient l'équivalence ne suffit pas à établir la complétude, d'autre part on ne saurait pas formuler la plus faible précondition de nombreuses instructions sans les connecteurs spatiaux. Ce résultat est plutôt à voir comme une réciproque du discours introductif de Reynolds dans [Rey02] : ce qui est coûteux à écrire en logique classique s'écrit souvent beaucoup mieux en logique séparante, et réciproquement ce qui s'écrit en logique séparante peut s'écrire, de façon nettement plus coûteuse, en logique classique.

Pour les structures mobiles, on a montré que la logique spatiale du π-calcul permet d'encoder la logique de Hennessy-Milner. On n'a toutefois pas étendu complètement ce résultat au cas des Ambients pour la notion d'action que l'on avait proposée. Cette absence d'encodage de modalités pour les actions $\stackrel{\mathsf{cap}}{\to}$ est toutefois assez peu significative, puisque ces actions ne correspondent pas à celles utilisées pour la définition de graphes de transitions étiquetées dans ce calcul [MN03]. Pour établir ces encodages, on a fait appel au principe des "jeux contextuels", qui avaient déjà été utilisé sous une forme assez différente pour établir l'équivalence entre congruence à barbe et bisimilarité.

Tous ces résultats d'expressivité ont permis d'établir quelques conséquences quant à la décidabilité des logiques spatiales. La plus importante est l'indécidabilité des logiques spatiales pour la mobilité lorsqu'elles incluent simultanément les connecteurs $\triangleright$ et $\Diamond$. En revanche de multiples fragments des logiques spatiales sont décidables, incluant parfois $\triangleright$ dans le cas statique. Pour la concurrence, il reste toutefois à trouver une forme affaiblie de raisonnement contextuel qui remplace l'actuel $\triangleright$ tout en garantissant la décidabilité de la logique.

L'apport essentiel des logiques spatiales est peut-être celui d'avoir changé la vision que l'on avait des algèbres de processus. De part leur origine, on s'était jusqu'à présent attaché pour les calculs concurrents à une notion d'observation liée à un graphe de transitions étiquetées. Les logiques spatiales considèrent en revanche les calculs mobiles pour ce qu'ils sont, des structures spatiales évolutives. Ce point de vue est tout à fait nouveau et prometteur, et commence par ailleurs à percer dans certaines applications (l'automatisation de la vérification de processus [Cai04, AK04], la justification de traductions d'un calcul mobile dans un autre [Gua04], et la formalisation de ces logiques dans des assistants de preuve [Aff04, Com04, SM02]). Un grand travail reste sans doute à poursuivre dans cette perspective, à la fois en ce qui concerne les logiques et les calculs spatiaux.



Bibliographie

[ACS98] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195 :291–324, 1998.

[Aff04] R. Affeld. A coq library for verification of concurrent programs. In *Proc. of 4th International Workshop on Logical Frameworks and Meta-languages (LFM'04)*, 2004.

[AK04] R. Affeld and N. Kobayashi. Partial order reduction for verification of spatial properties of pi-calculus processes. In *Proc. of EXPRESS'04*, september 2004. London.

[BB90] G. Berry and G. Boudol. The chemical abstract machine. In *Proc. of 17h Symposium on Principles of Programming Languages*, 1990.

[BCO04] J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. To appear in Proc. of FSTTCS'04, december 2004, Chennai, 2004.

[BG03] N. Biri and D. Galmiche. A separation logic for resource distribution. In *Proc. of FSTTCS'03*, Mumbai, december 2003.

[Bou92] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA, 1992.

[BPR94] S. Basu, R. Pollack, and M-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. In *IEEE Symposium on Foundations of Computer Science*, 1994.

[BT03] I. Boneva and J-M. Talbot. When ambients cannot be opened. In *Proc. of FOSSACS'03*, volume LNCS 2620, pages 169 – 184, 2003.

[BZ04] N. Busi and G. Zavattaro. On the expressive power of movement and restriction in pure mobile ambients. To appear in TCS, Elsevier, 2004.

[Cai99] L. Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility.* PhD thesis, Universidade Nova de Lisboa, 1999.

[Cai00] Luis Caires. A specification logic for mobility. Technical Report 4/2000, DI/FCT/UNL, 2000.

[Cai04] Luis Caires. Behavioral and spatial observations in a logic for the pi-calculus. In *Proc. of FOSSACS'04*, 2004.

[CC01] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). In *Proc. of TACS'01*, LNCS. Springer Verlag, 2001.

[CC02] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). In *Proc.of CONCUR'02*, 2002.

[CCG03] C.Calcagno, L. Cardelli, and A. Gordon. Deciding Validity in a Spatial Logic for Trees. In *Proc. of TLDI'03*, pages 62–73. ACM, 2003.

[CG98] L. Cardelli and A. Gordon. Mobile Ambients. In *Proc. of FOSSACS'98*, volume 1378 of *LNCS*, pages 140–155. Springer Verlag, 1998.

[CG99] L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proc. 26th POPL*, pages 79–92. ACM Press, 1999.

[CG00] L. Cardelli and A. Gordon. Anytime, Anywhere, Modal Logics for Mobile Ambients. In *Proc. of POPL'00*, pages 365–377. ACM Press, 2000.

[CG01a] L. Cardelli and G. Ghelli. A Query Language Based on the Ambient Logic. In *Proc. of ESOP'01*, volume 2028 of *LNCS*, pages 1–22. Springer Verlag, 2001. invited paper.

[CG01b] L. Cardelli and A. Gordon. Logical Properties of Name Restriction. In *Proc. of TLCA'01*, volume 2044 of *LNCS*. Springer Verlag, 2001.

[CGT02] W. Charatonik, A. D. Gordon, and J-M. Talbot. Finite-control mobile ambients. In SPRINGER, editor, *Proc. of ESOP'02*, volume LNCS 2305, pages 295 – 313, 2002.

[CH89] I. Castellani and M. Hennessy. Distributed bisimulation. *Journal of the ACM*, 1989.

[CL04] L. Caires and E. Lozes. Elimination of quantifiers and undecidability in spatial logics for concurrency. In *Proc. of CONCUR'04*, 2004.

[Com04] Céline Coma. Logique séparante en coq. Master's thesis, ENS Lyon - Université Claude Bernard Lyon 1, 2004. stage de Maîtrise.

[CT01] W. Charatonik and J-M. Talbot. The Decidability of Model Checking Mobile Ambients. In *Proc. of CSL'01*, LNCS. Springer LNCS, 2001.

[CTZ+04] W. Charatonik, J-M. Talbot, Sivano Dal Zilio, Andrew D. Gordon, and Supratik Mukhopadhyay. Model Checking Mobile Ambients. In *Theoretical Computer Science*, 2004.

[CYO01] C. Calcagno, H. Yang, and P. O'Hearn. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *Proceedings of FSTTCS '01*, volume 2245 of *LNCS*. Springer Verlag, 2001.

[CZG+01] W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, and J-M. Talbot. The complexity of model checking mobile ambients. In *Proc. of FOSSACS'01*, volume LNCS 2030, pages 152 – 167, 2001.

[Dam88] M. Dam. Relevance logic and concurrent composition. In *Proc. LICS'88*, pages 178–185, 1988.

[DGG] A. Dawar, P. Gardner, and G. Ghelli. Adjunct elimination through games. to appear in Proc of FSTTCS'04, december 2004, Chennai.

[DL03] V. Danos and C. Laneve. Graphs for core molecular biology. In *Proc. of CMSB'03*, 2003.

[DP04] V. Danos and S. Pradalier. Projective brane calculus. In *Proc. of CMSB'04*, 2004.

[GC04] G. Ghelli and G. Conforti. Decidability of freshness, undecidability of revelation. In *Proc. of FOSSACS'04*, march 2004.

[Gir87] J-Y. Girard. Linear logic. *Theoretical Computer Science*, 50 :1–101, 1987.

[GLW03] P. Gardner, C. Laneve, and L. Wishick. Linear forwarders. In *Proc. of CONCUR'03*, LNCS 2761, Marseille, august 2003.

[GP99] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, Washington, 1999.

[GS86] S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of CCS. *Information and Control*, 68 :125–145, 1986.

[Gua04] Xudong Guan. Name-passing in an ambient-like calculus and its proof using spatial logic. In *Proc. of EXPRESS'04*, september 2004. London.

[Hir97] D. Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions. In *Proceedings of TPHOL's 97, LNCS 1275, © Springer Verlag*, volume LNCS 1275. Springer Verlag, 1997.

[Hir04] D. Hirschkoff. An extensional spatial logic for mobile processes. In Springer-Verlag, editor, *Proc. of CONCUR 2004 (15th International Conference), Lecture Notes in Computer Science.*, 2004.

[HLS02] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, Expressiveness and Decidability in the Ambients Logic. In *Proc. of 17th IEEE Symposium on Logic in Computer Science*, pages 423–432. IEEE Computer Society, 2002.

[HLS03] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Minimality results for spatial logics. In *Proc. of FSTTCS'03*, Mumbai, december 2003.

[HM85] M.C. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *Journal of ACM*, 32(1) :137–161, 1985.

[How96] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2) :103–112, 1996.

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communications. In *Proc. of ECOOP'91*, volume L.N.C.S 512. Springer-Verlag, 1991.

[Loz03] E. Lozes. Adjuncts elimination in the static ambient logic. In *Proc. of Express'03*, Marseille, september 2003.

[Loz04a] E. Lozes. Elimination of spatial connectives in static spatial logics. *Theoretical Computer Science*, 2004. To appear.

[Loz04b] E. Lozes. Separation logic preserves the expressiveness of classical logic. In *Proc. of Space'04*, Venezia, january 2004.

[LS00a] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proc. POPL'00*, 2000.

[LS00b] F. Levi and D. Sangiorgi. Controlling interference in ambients. Short version appeared in *Proc. 27th POPL*, ACM Press, 2000.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mim03] Samuel Mimram. Un calcul de séquent complet pour la logique spatiale. Master's thesis, ENS Lyon, 2003. stage de Licence.

[MN03] M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. In LNCS, editor, *Proc. of ICALP'03*, volume 2719, 2003.

[MT03] Dale Miller and Alwen Tiu. A proof theory for generic judgments. In Phokion Kolaitis, editor, *Proc. of LICS'03*, pages 118–127, July 2003.

[MZL04] C. Meyssonier, S. Dal Zilio, and D. Lugiez. A logic you can count on. In *Proc. of POPL - 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 135–146. ACM Press, 2004.

[MZW03] Stephan Merz, Júlia Zappe, and Martin Wirsing. A spatio-temporal logic for the specification and refinement of mobile systems. In Mauro Pezzè, editor, *Fundamental Approaches to Software Engineering (FASE 2003)*, volume 2621 of *Lecture Notes in Computer Science*, pages 87–101, Warsaw, Poland, April 2003. Springer-Verlag.

[OP92] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4 :497–543, 1992.

[Pal97] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In *Proc. of Symposium on Principles of Programming Languages*. ACM Press, 1997.

[POY04] David Pym, Peter O'Hearn, and Hongseok Yang. Possible worlds and resources : The semantics of bi. *Theoretical Computer Science 315(1)*, pages 257–305, 2004.

[Rey02] J. Reynolds. Separation logic : a logic for shared mutable data structures. In *Proc of LICS'02 (invited paper)*, 2002.

[San01] D. Sangiorgi. Extensionality and Intensionality of the Ambient Logic. In *Proc. of 28th POPL*, pages 4–17. ACM Press, 2001.

[SI94] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1) :149–163, 1994.

[Sim94] Alex K. Simpson. *The proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edimburgh, 1994.

[SM02] Ivan Scagnetto and Marino Miculan. The ambient calculus and its logic in the calculus of inductive construction. In *Proceedings of LFM'02*, 2002.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus : a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tra50] B.A. Trakhtenbrot. *The impossibility of an algorithm for the decision problem for finite models.*, pages 70 :569–572. Doklady Akademii Nauk SSR, 1950.

[TZH02] D. Teller, P. Zimmer, and D. Hirschkoff. Using Ambients to Control Resources. In *Proc. of CONCUR'02*, volume 2421 of *LNCS*. Springer Verlag, 2002.

[Urq72] A. Urquhart. Semantics for relevants logics. *Journal of Symbolic Logic*, 37 :159–169, 1972.

[VCHP04] Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. Technical Report CMU-CS-04-105, Carnegie Mellon University, 2004.

[VM94] B. Victor and F. Moller. The mobility workbench : A tool for the π-calculus. LFCS Report ECS-LFCS-94-285, february 1994.

[Win85] G. Winskel. A complete proof system for sccs with modal assertions. *Lecture Notes in Computer Science*, 206 :392–410, 1985.

[Yan01] H. Yang. *Local Reasoning for Stateful Programs.* PhD thesis, University of Illinois at Urbana-Champaign, july 2001.

[Zil00] S. Dal Zilio. Structural Congruence for Ambients is Decidable. In *Proc. of ASIAN'00*, volume 1961 of *LNCS*. Springer Verlag, 2000.

Annexes

Appendix A

On the expressiveness of the Ambient Logic

This appendix develops the results on the expressiveness of the Ambient Logic sketched in chapter 6, section 6.1; they are also presented in [HLS02]. The calculus we consider is however richer than the one presented in the french part, namely it includes asynchronous, names communications. We moreover extend our results to other forms of communications (synchronous, with capabilities).

We first restate some usefull definitions about the calculus, the Ambient Logic we consider, and some usefull notions in section A.1. Then we illustrate the use of *contextual plays* both to derive modal formulas for capabilities and communications (section A.2), and to express other non modal properties such as persistency, finiteness, or name occurence (section A.3). We then establish the theorem 7.1.17 on the existence of characteristic formulas (section A.4, and we conclude with some discussions on how to extend this results to other forms of communication (section A.5).

A.1 Background

This section collects the necessary background for this paper. It includes the MA calculus [CG98] semantic and syntax, and the Ambient Logic [CG00].

A.1.1 Syntax of Mobile Ambients

We recall here the syntax of MA, from [CG98] — we shall sometimes call this calculus the *Ambient calculus*. In the calculus we study, only names, not capabilities, can be communicated (this allows us to work in an untyped calculus). We analyse extensions of the calculus in Section A.5.

As in [CG00], the calculus has no restriction operator for creating new names. The restriction-free calculus has a more direct correspondence with edge-labelled trees and semistructured data.

Table A.1 shows the syntax. Both the set of names and that of variables are infinite. Letters n, m, h range over names, x, y, z over variables; η ranges over names and variables. The expressions $\mathsf{in}\,\eta$, $\mathsf{out}\,\eta$, and $\mathsf{open}\,\eta$ are the *capabilities*, and are ranged over using cap. Messages and abstractions are the *input/output*

$h, k, \ldots n, m$			*Names*
η			*Names* $\cup$ *Variables*
			Expressions
M, N	$::=$	cap	capability
	$\mid$ $\ldots$		cf. Sect. A.5
			Capabilities
cap	$::=$	$\mathsf{in}\,\eta$	enter
	$\mid$	$\mathsf{out}\,\eta$	exit
	$\mid$	$\mathsf{open}\,\eta$	open
			Processes
P, Q, R	$::=$	$\mathbf{0}$	nil
	$\mid$	$P \mid Q$	parallel
	$\mid$	$!P$	replication
	$\mid$	$M.P$	prefixing
	$\mid$	$\eta[P]$	ambient
	$\mid$	$\langle \eta \rangle$	message
	$\mid$	$(x)\,P$	abstraction

Table A.1: The syntax of finite MA

(I/O) primitives. A *closed* process has no free variables. We ignore syntactic differences due to alpha conversion, and we write $P\{^n\!/\!_x\}$ for the result of substituting x with n in P.

Processes having the same internal structure are identified. This is expressed by means of the *structural congruence relation*, $\equiv$, the smallest congruence such that:

$$P \mid \mathbf{0} \;\equiv\; P \qquad P \mid Q \;\equiv\; Q \mid P \qquad P \mid (Q \mid R) \;\equiv\; (P \mid Q) \mid R$$

$$!P \;\equiv\; !P \mid P \qquad !\mathbf{0} \;\equiv\; \mathbf{0} \qquad !(P \mid Q) \;\equiv\; !P \mid !Q \qquad !!P \;\equiv\; !P$$

As a consequence of the results presented in [Zil00], that studies a richer calculus than the one we study, we have:

Theorem A.1.1 *Relation $\equiv$ is decidable.*

The two following syntactical notions will be useful below.

Definition A.1.2 (Finite and single processes)

- *A process P is* finite *iff there exists a process P' with no occurrence of the replication operator such that $P \equiv P'$.*
- *A process P is* single *if there exists P' such that either $P \equiv \mathsf{cap}.P'$ for some* cap *or $P \equiv n[P']$ for some n.*

$$\frac{}{\mathsf{open}\, n.\, P \mid n[Q] \longrightarrow P \mid Q}\ \texttt{Red-Open}$$

$$\frac{}{n[\mathsf{in}\, m.\, P_1 \mid P_2] \mid m[Q] \longrightarrow m[n[P_1 \mid P_2] \mid Q]}\ \texttt{Red-In}$$

$$\frac{}{m[n[\mathsf{out}\, m.\, P_1 \mid P_2] \mid Q] \longrightarrow n[P_1 \mid P_2] \mid m[Q]}\ \texttt{Red-Out}$$

$$\frac{}{\langle M\rangle \mid (x)\, P \longrightarrow P\{^{M}\!/\!_{x}\}}\ \texttt{Red-Com}$$

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ \texttt{Red-Par}$$

$$\frac{P \longrightarrow P'}{n[P] \longrightarrow n[P']}\ \texttt{Red-Amb}$$

$$\frac{P \equiv P' \quad P' \longrightarrow P'' \quad P'' \equiv P'''}{P \longrightarrow P'''}\ \texttt{Red-Str}$$

Table A.2: The rules for reduction

A.1.2 Operational Semantics

The operational semantics of the calculus is given by a reduction relation $\longrightarrow$, defined by the rules given in Table A.1.2. The reflexive transitive closure of $\longrightarrow$ is written $\Longrightarrow$.

Lemma A.1.3 *If $P \longrightarrow Q$ then there is a derivation of the reduction in which* `Red-Struct` *is applied, if at all, only as the last rule.*

Lemma A.1.3 shows that every reduction $P \longrightarrow P'$ has a normalised derivation proof. As a consequence, we have:

Lemma A.1.4 *If $P \longrightarrow Q$ then either*

1. $P \equiv R_1 \mid m[n[\mathsf{out}\, m.\, P_1 \mid P_2] \mid P_3] \mid R_2$ *and* $Q \equiv R_1 \mid n[P_1 \mid P_2] \mid m[P_3] \mid R_2$*, or*
2. $P \equiv R_1 \mid n[\mathsf{in}\, m.\, P_1 \mid P_2] \mid m[P_3] \mid R_2$ *and* $Q \equiv R_1 \mid m[n[P_1 \mid P_2] \mid P_3] \mid R_2$*, or*
3. $P \equiv R_1 \mid \mathsf{open}\, n.\, P_1 \mid n[P_2] \mid R_2$ *and* $Q \equiv R_1 \mid P_1 \mid P_2 \mid R_2$*, or*
4. $P \equiv R_1 \mid \langle n\rangle \mid (x)\, P_1 \mid R_2$ *and* $Q \equiv R_1 \mid P_1\{^{n}\!/\!_{x}\} \mid R_2$*, or*
5. $P \equiv R \mid n[P_1]$*,* $Q \equiv R \mid n[Q_1]$ *and* $P_1 \longrightarrow Q_1$*.*

A consequence of this lemma is the following property, which we state here only informally: if two reductions $P \longrightarrow P'$ and $P \longrightarrow P''$ consume the same subterms of P, then $P' \equiv P''$. We shall frequently use this property implicitely in the remainder of the paper.

We now introduce some forms of labelled transitions that we will use to give the interpretation of some of our logical constructions.

$\mathcal{A}$	$::=$	$\top$	true	classical logic
	$\mid$	$\neg\mathcal{A}$	negation	
	$\mid$	$\mathcal{A} \vee \mathcal{B}$	disjunction	
	$\mid$	$\forall x.\, \mathcal{A}$	universal quantification over names	
	$\mid$	$\Diamond\, \mathcal{A}$	sometime	temporal and spatial connectives
	$\mid$	0	void	
	$\mid$	$\eta[\mathcal{A}]$	edge	
	$\mid$	$\mathcal{A} \mid \mathcal{B}$	composition	
	$\mid$	$@\mathcal{A}\eta$	localisation	logical adjuncts
	$\mid$	$\mathcal{A} \,\triangleright\, \mathcal{B}$	guarantee	

Table A.3: The syntax of logical formulas

Definition A.1.5 (Labelled transitions) *Let P be a closed process. We write:*

- $P \xrightarrow{\mathsf{cap}} P'$, *where* cap *is a capability, if* $P \equiv \mathsf{cap}.\, P_1 \mid P_2$ *and* $P' = P_1 \mid P_2$.
- $P \xrightarrow{!n} P'$ *if* $P \equiv \langle n \rangle \mid P'$.
- $P \xrightarrow{?n} P'$ *if* $P \equiv (x)\, P_1 \mid P_2$ *and* $P' \equiv P_1\{n/x\} \mid P_2$.
- $P \stackrel{\mu}{\Longrightarrow} P'$, *where* μ *is one of the above labels, if* $P \Longrightarrow \xrightarrow{\mu} \Longrightarrow P'$ *(where* $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$ *is relation composition).*
- **(stuttering)** $P \xRightarrow{(M_1,M_2)^*} P'$ *if there is* $i \geq 1$ *and processes* $P_1, \ldots, P_i$ *with* $P = P_1$ *and* $P' = P_i$ *such that* $P_r \stackrel{M_1}{\Longrightarrow} \stackrel{M_2}{\Longrightarrow} P_{r+1}$ *for all* $1 \leq r < i$.
- *Finally,* $\stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow}$ *is a convenient notation for compacting statements involving capability transitions.* $\stackrel{\langle \mathsf{in\ n} \rangle}{\Rightarrow}$ *is* $\xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*}$; *similarly* $\stackrel{\langle \mathsf{out\ n} \rangle}{\Rightarrow}$ *is* $\xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*}$; *and* $\stackrel{\langle \mathsf{open\ n} \rangle}{\Rightarrow}$ *is* $\Rightarrow$.

A.1.3 The Ambient Logic

To define the set of formulas of the Ambient Logic, given on Table A.3, we introduce an infinite set of *variables*, ranged over with x, y, z; η ranges over names and variables. $\mathcal{A}\{n/x\}$ is the formula obtained from $\mathcal{A}$ by substituting variable x by name n. A formula without free variables is *closed*.

The logic has the propositional connectives, $\top, \neg\mathcal{A}, \mathcal{A} \vee \mathcal{B}$, and universal quantification on names, $\forall x.\ \mathcal{A}$, with the standard logical interpretation. The temporal connective, $\Diamond\mathcal{A}$ has been discussed in the Introduction. The spatial connectives, 0, $\mathcal{A} \mid \mathcal{B}$, and $\eta[\mathcal{A}]$, are the logical counterpart of the corresponding constructions on processes. $\mathcal{A} \triangleright \mathcal{B}$ and $@\mathcal{A}\eta$ are the logical adjuncts of $\mathcal{A} \mid \mathcal{B}$ and $\eta[\mathcal{A}]$ respectively, in the sense of being, roughly, their 'contextual inverse', as expressed in Definition A.1.6 below.

The logic in [CG00] has also a *somewhere* connective, that holds of a process containing, at some arbitrary level of nesting of ambients, an ambient whose content satisfies $\mathcal{A}$. The addition of this connective would not change the results in the paper.

Definition A.1.6 (Satisfaction) *The satisfaction relation between processes and closed formulas, written $P \models \mathcal{A}$, is defined as follows:*

$$
\begin{array}{lcl}
P \models \top & \stackrel{\text{def}}{=} & \textit{always true} \\
P \models \forall x.\,\mathcal{A} & \stackrel{\text{def}}{=} & \textit{for any } n,\ P \models \mathcal{A}\{n/x\} \\
P \models \neg\mathcal{A} & \stackrel{\text{def}}{=} & \textit{not } P \models \mathcal{A} \\
P \models \mathcal{A}_1 \mid \mathcal{A}_2 & \stackrel{\text{def}}{=} & \exists P_1, P_2 \textit{ s.t. } P \equiv P_1 \mid P_2 \\
 & & \quad \textit{and } P_i \models \mathcal{A}_i,\ i = 1,2 \\
P \models \mathcal{A} \vee \mathcal{B} & \stackrel{\text{def}}{=} & P \models \mathcal{A} \textit{ or } P \models \mathcal{B} \\
P \models n[\mathcal{A}] & \stackrel{\text{def}}{=} & \exists P' \textit{ s.t. } P \equiv n[P'] \textit{ and } P' \models \mathcal{A} \\
P \models 0 & \stackrel{\text{def}}{=} & P \equiv \mathbf{0} \\
P \models \Diamond\mathcal{A} & \stackrel{\text{def}}{=} & \exists P' \textit{ s.t. } P \Rightarrow P' \textit{ and } P' \models \mathcal{A} \\
P \models @\mathcal{A}n & \stackrel{\text{def}}{=} & n[P] \models \mathcal{A} \\
P \models \mathcal{A} \rhd \mathcal{B} & \stackrel{\text{def}}{=} & \forall R,\ R{\models}\mathcal{A} \textit{ implies } P \mid R \models \mathcal{B}
\end{array}
$$

By definition, satisfaction is closed by structural congruence:

Lemma A.1.7 ([CG00]) *If $P \equiv Q$ and $P \models \mathcal{A}$, then also $Q \models \mathcal{A}$.*

We give $\vee$ and $\wedge$ the least syntactic precedence, thus $\mathcal{A}_1 \rhd \mathcal{A}_2 \wedge \mathcal{A}_3$ reads $(\mathcal{A}_1 \rhd \mathcal{A}_2) \wedge \mathcal{A}_3$, and $\mathcal{A}_1 \rhd (\Diamond\mathcal{A}_2 \wedge \Diamond\mathcal{A}_3)$ reads $\mathcal{A}_1 \rhd ((\Diamond\mathcal{A}_2) \wedge (\Diamond\mathcal{A}_3))$. We shall use the dual of some connectives, namely the duals of linear implication $(\mathcal{A} \blacktriangleright \mathcal{B})$, of the sometime modality $(\Box\mathcal{A})$, of the parallel operator $(\|)$, and the standard duals of universal quantification $(\exists x.\,\mathcal{A})$ and disjunction $(\mathcal{A} \wedge \mathcal{B})$; we also define (classical) implication $(\mathcal{A} \to \mathcal{B})$:

$$
\mathcal{A} \wedge B \stackrel{\text{def}}{=} \neg(\neg\mathcal{A} \vee \neg\mathcal{B}) \qquad \Box\mathcal{A} \stackrel{\text{def}}{=} \neg\Diamond\neg\mathcal{A} \qquad \mathcal{A} \to \mathcal{B} \stackrel{\text{def}}{=} \neg\mathcal{A} \vee \mathcal{B}
$$

$$
\mathcal{A}\|\mathcal{B} \stackrel{\text{def}}{=} \neg(\neg\mathcal{A} \mid \neg\mathcal{B}) \qquad \exists x.\,\mathcal{A} \stackrel{\text{def}}{=} \neg\forall x.\,\neg\mathcal{A} \qquad \mathcal{A} \blacktriangleright \mathcal{B} \stackrel{\text{def}}{=} \neg(\mathcal{A} \rhd \neg\mathcal{B})
$$

$$
\bot \stackrel{\text{def}}{=} \neg\top
$$

Thus $P \models \mathcal{A} \blacktriangleright \mathcal{B}$ iff there exists Q with $Q \models \mathcal{A}$ and $P \mid Q \models \mathcal{B}$, and $P \models \Box\mathcal{A}$ iff $P' \models \mathcal{A}$ for all P' such that $P \Longrightarrow P'$.

We now define the induced equivalence between processes induced by the logic:

Definition A.1.8 (Logical equivalence) *For processes P and Q, we write $P{=_L}Q$ if for any closed formula $\mathcal{A}$ it holds that $P \models \mathcal{A}$ iff $Q \models \mathcal{A}$.*

A.2 Formulas for capabilities and communications

In this section, we show that we can capture at a logical level prefixes of the language, both for movement and for communication.

A.2.1 Preliminary formulas: counting components and comparing names

We start by recalling some formulas from [CG00] that will be useful for some constructions presented below.

The Ambient Logic allows one to count the number of parallel components of a process. The formula below is true of a process that has exactly one parallel component that is different from **0**.

$$\boxed{\texttt{1comp} \stackrel{\text{def}}{=} \neg(\neg 0 \mid \neg 0) \wedge \neg 0}$$

Lemma A.2.1 *It holds that $P \models \texttt{1comp}$ iff P is single.*

Proof: Easy. □

Similarly we define

$$\boxed{\texttt{2comp} \stackrel{\text{def}}{=} \texttt{1comp} \mid \texttt{1comp}}$$

We may impose a given formula $\mathcal{A}$ to be satisfied by all single parallel components of a process, using the following definitions:

$$\boxed{\begin{array}{rcl} \mathcal{A}^{\forall} & \stackrel{\text{def}}{=} & \mathcal{A} \mid\mid \neg\top \\ \mathcal{A}^{\omega} & \stackrel{\text{def}}{=} & (\texttt{1comp} \rightarrow \mathcal{A})^{\forall} \end{array}}$$

Lemma A.2.2

- *$P \models \mathcal{A}^{\forall}$ iff for any Q, R such that $P \equiv Q \mid R$, it holds that $Q \models \mathcal{A}$.*
- *$P \models \mathcal{A}^{\omega}$ iff all single parallel components of P satisfy $\mathcal{A}$.*

Proof: Easy. □

Finally, the following formula expresses equality between names, independently from the tested process:

$$\boxed{m = n \stackrel{\text{def}}{=} (n[\top])@m}$$

Lemma A.2.3 *For any P, $P \models m = n$ iff names m and n are equal.*

A.2.2 Formulas for capabilities

The two formulas below are true of a process that is (structurally congruent to) an ambient and (to) an empty ambient, respectively.

$$\boxed{\begin{array}{rcl} \texttt{1amb} & \stackrel{\text{def}}{=} & \exists x.\, x[\top] \\ \texttt{1amb0} & \stackrel{\text{def}}{=} & \exists x.\, x[0] \end{array}}$$

Lemma A.2.4

- $P \models \texttt{1amb}$ *iff* $P \equiv n[Q]$, *for some* n *and* Q.
- $P \models \texttt{1amb0}$ *iff* $P \equiv n[\mathbf{0}]$, *for some* n.

Proof: Easy. $\square$

To help understanding the definitions of the capability formulas, we first discuss some simpler formulas, which do not talk about the process underneath the prefix. We define, for names $n \neq h$:

$$\begin{array}{rcl} \langle \mathsf{open}\ n \rangle & \stackrel{\text{def}}{=} & n[h[0]] \triangleright \Diamond\ (h[0] \mid \top) \\ & & \wedge\ \texttt{1comp} \\ & & \wedge\ \neg\texttt{1amb} \\ \langle \mathsf{out}\ n \rangle & \stackrel{\text{def}}{=} & @@(\Diamond\ (h[\top] \mid n[\mathbf{0}]))nh \\ & & \wedge\ \texttt{1comp} \\ & & \wedge\ \neg\texttt{1amb} \end{array}$$

It holds that $P \models \langle \mathsf{open}\ n \rangle$ iff $P \equiv \mathsf{open}\, n.\, P'$ for some P'. We sketch the proof. The subformula $\texttt{1comp} \wedge \neg\texttt{1amb}$ says that P is single and is not an ambient. Thus, modulo $\equiv$, process P can only be $\mathbf{0}$, $\mathsf{open}\, m.\, P'$ $\mathsf{in}\, m.\, P'$, $\mathsf{out}\, m.\, P'$, $(x)\ P'$, or $\langle m \rangle$. The subformula $n[h[0]] \triangleright \Diamond\ (h[0] \mid \top)$ says that $P \mid n[h[\mathbf{0}]]$ can reduce to a process with an empty ambient h at the outermost level. From these requirements, we conclude that $P \equiv \mathsf{open}\, n.\, P'$, for some P'.

Similarly we prove that $P \models \langle \mathsf{out}\ n \rangle$ iff $P \equiv \mathsf{out}\, n.\, P'$, for some P'. By the subformula $\texttt{1comp} \wedge \neg\texttt{1amb}$, process P is single and is not an ambient. By the subformula $@@(\Diamond\ (h[\top] \mid n[\mathbf{0}]))nh$,

$$n[h[P]] \models \Diamond\ (h[\top] \mid n[0])$$

hence $P \equiv \mathsf{out}\, n.\, P'$, for some P', otherwise $h[P]$ could not exit n.

To obtain the full capability formulas we add some quantification on names. Formula $\langle \mathsf{open}\ n \rangle.\, \mathcal{A}$ is thus defined as follows:

$$\boxed{\begin{array}{rcl} \langle \mathsf{open}\ n \rangle.\, \mathcal{A} & \stackrel{\text{def}}{=} & \forall x.\, n[x[0]] \triangleright \Diamond\ (x[0] \mid \mathcal{A}) \\ & & \wedge\ \texttt{1comp} \\ & & \wedge\ \neg\texttt{1amb} \\ \texttt{1open} & \stackrel{\text{def}}{=} & \exists x.\, \langle \mathsf{open}\ x \rangle.\, \top \end{array}}$$

Lemma A.2.5 *$P \models \langle \mathsf{open}\ n \rangle.\, \mathcal{A}$ iff $P \equiv \mathsf{open}\, n.\, P'$, for some P' such that $P' \Longrightarrow P''$ and $P'' \models A$.*

Proof: The implication from right to left is easy.

For the reverse implication, we set

$$G \stackrel{\text{def}}{=} n[h[0]] \triangleright \Diamond\ (h[0] \mid \mathcal{A})$$

where $h \notin \mathrm{n}(P)$. Since $P \models \texttt{1comp}$, we have $P \equiv Q$, for some Q that is not a parallel composition. Since also $P \models \neg\texttt{1amb}$, we infer that Q is not an ambient. Finally since $P \models G$, process Q cannot be of the form $\mathbf{0}$, $\mathsf{in}\, n.\, Q'$, $\mathsf{out}\, n.\, Q'$, $(x)Q'$, $\langle p \rangle$. For the same reason, Q cannot be a prefix $\mathsf{open}\, m.\, Q'$ with $m \neq n$. The only possibility left is $Q = \mathsf{open}\, n.\, Q'$, for some Q'.

Moreover, we have

$$n[h[\mathbf{0}]] \mid \mathsf{open}\, n.\, Q' \Longrightarrow R \text{ and } \models R h[\mathbf{0}] \mid \mathcal{A}$$

for some R. The first step of this reduction must be

$$n[h[\mathbf{0}]] \mid \mathsf{open}\, n.\, Q' \longrightarrow h[\mathbf{0}] \mid Q'$$

(up to $\equiv$). Since h is fresh, $h[\mathbf{0}]$ cannot interact with Q'. Hence

$$R \equiv h[\mathbf{0}] \mid Q''$$

for some Q'' such that $Q' \Longrightarrow Q''$. □

Corollary A.2.6 *$P \models$ `1open` iff $P \equiv \mathsf{open}\, n.\, P'$, for some n and P'.*

Along the lines of our construction for the open prefix, we can define characteristic formulas for the in and out prefixes.

$$\begin{array}{rcl}
\langle \mathsf{out}\ n\rangle.\, \mathcal{A} & \stackrel{\text{def}}{=} & \forall x.\, \Big(@@(\Diamond\ (x[\mathcal{A}] \mid n[\mathbf{0}]))nx\Big) \\
& & \wedge\ \mathtt{1comp} \\
& & \wedge\ \neg\mathtt{1amb} \\
\mathtt{1out} & \stackrel{\text{def}}{=} & \exists x.\, \langle \mathsf{out}\ x\rangle \\
\\
\langle \mathsf{in}\ n\rangle.\, \mathcal{A} & \stackrel{\text{def}}{=} & \forall x.\, \Big(@(n[0] \rhd \Diamond\ n[x[\mathcal{A}]])x\Big) \\
& & \wedge\ \mathtt{1comp} \\
& & \wedge\ \neg\mathtt{1amb} \\
\mathtt{1in} & \stackrel{\text{def}}{=} & \exists x.\, \langle \mathsf{in}\ x\rangle
\end{array}$$

Lemma A.2.7 *$P \models \langle \mathsf{out}\ n\rangle.\, \mathcal{A}$ iff $P \equiv \mathsf{out}\, n.\, P'$, for some P' such that $P' \xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*} P''$ and $P'' \models \mathcal{A}$.*

Proof: Similar to the proof for the open prefix. The formula `1comp` $\wedge$ $\neg$`1amb` forces P to be single and not an ambient. Therefore $P \equiv Q$, for some Q whose outermost operator is not a parallel composition or an ambient. Then we should have

$$n[h[Q]] \models \Diamond\ (h[A] \mid n[0])$$

This can only happen if Q is of the form $\mathsf{out}\, n.\, Q'$, for some Q' such that $Q' \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*} n Q''$ and $Q'' \models \mathcal{A}$. □

Lemma A.2.8 *$P \models \langle \mathsf{in}\ n\rangle.\, \mathcal{A}$ iff $P \equiv \mathsf{in}\, n.\, P'$, for some P' such that $P' \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*} n P''$ and $P'' \models \mathcal{A}$.*

Proof: Similar to the previous proofs. The formula `1comp` $\wedge$ $\neg$`1amb` forces P to be single and not an ambient. Therefore $P \equiv Q$, for some Q whose outermost operator is not a parallel composition or an ambient. Then we should have

$$h[Q] \mid n[\mathbf{0}] \models \Diamond\ n[h[\mathcal{A}]]$$

where h is fresh.

As by previous arguments, this can only happen if Q is of the form $\mathsf{in}\, n.\, Q'$. Then we have:

$$n[h[P]] \models \Diamond\, n[h[\mathcal{A}]]$$

As by previous arguments, we derive the conclusion. $\Box$

Necessity formulas. Given a capability cap, we may define the *'necessity'* version of the *'possibility'* formulas we have just introduced as follows:

$$\boxed{[\![\mathsf{cap}]\!].\, \mathcal{A} \stackrel{\text{def}}{=} \langle\langle \mathsf{cap} \rangle\rangle.\, \top \ \wedge\ \neg\langle\langle \mathsf{cap} \rangle\rangle.\, \neg\mathcal{A}}$$

Lemma A.2.9 *For any capability* cap*, formula* $\mathcal{A}$ *and term* P*,* $P \models [\![\mathsf{cap}]\!].\, \mathcal{A}$ *iff there is* P' *such that* $P \equiv \mathsf{cap}.\, P'$*, and, for any* P'' *such that* $P' \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} P''$*,* $P'' \models \mathcal{A}$*.*

Proof: Easy. $\Box$

Note that necessity formulas are not the dual of the possibility formulas, as in standard modal logics, because of the spatial aspects of AL. For instance, $[\![\mathsf{in}\, n]\!]\top$ does not have the same interpretation as $\neg\langle\langle \mathsf{in}\, n \rangle\rangle.\, \neg\top$, the latter being actually equivalent to $\top$.

Remark: We could think of deriving formulas for *modalities* $\stackrel{\mathsf{cap}}{\Longrightarrow}$, as in standard modal logics for concurrency [HM85], instead of capturing the syntactical prefixes corresponding to a capability cap. It turns out that spatial logics are more intensional, and make actions more difficult to express than connectives. In particular, we do not know how to express directly a modality corresponding to action $\stackrel{\mathsf{open}\, n}{\Longrightarrow}$.

A.2.3 Formulas for communication

The first step to characterize I/O processes (i.e., messages or abstractions) is to get rid of other possible constructs for single terms, as follows:

$$\boxed{\begin{array}{lcl} \mathtt{1comm} & \stackrel{\text{def}}{=} & \mathtt{1comp} \wedge \neg\mathtt{1amb} \wedge \neg\mathtt{1open} \wedge \neg\mathtt{1out} \wedge \neg\mathtt{1in} \\ \mathtt{2comm} & \stackrel{\text{def}}{=} & \mathtt{1comm} \mid \mathtt{1comm} \end{array}}$$

Lemma A.2.10 $P \models \mathtt{1comm}$ *iff* ($P \equiv \langle p \rangle$ *or* $P \equiv (x)\, P'$)*, for some* p *and* P'*.*

The difficult part, however, is the definition of the I/O formulas for separating messages from abstractions, and also, within the messages and the abstractions, messages with different contents and abstractions with different behaviours.

The capability formulas are easier to define than the I/O formulas because capabilities act on ambients, and the logic has a connective, $n[A]$, for talking about ambients. By contrast, the I/O primitives act on themselves. To define the I/O formulas, we proceed as follows:

1. We define a formula, `TestComm`, that characterises the special abstraction $(x)\, x[\mathbf{0}]$.

2. We use `TestComm` to define the formula for messages:

$$\mathcal{F}_{\langle n\rangle} \stackrel{\text{def}}{=} \mathtt{1comm} \wedge (\mathtt{TestComm} \rhd \Diamond\, n[0])$$

3. We then use $\mathcal{F}_{\langle n\rangle}$ to define the formulas for abstractions:

$$\langle ?n\rangle.\, \mathcal{A} \stackrel{\text{def}}{=} \mathtt{1comm} \wedge (\neg \exists x.\, \mathcal{F}_{\langle x\rangle}) \wedge (\mathcal{F}_{\langle n\rangle} \rhd \Diamond\, \mathcal{A})$$

Lemma A.2.11 *Given $(x)\, R$, suppose there are q, p such that*

$$\langle p\rangle \mid (x)\, R \models \Diamond\, \mathtt{1amb0}$$
$$\langle q\rangle \mid (x)\, R \models \Box(\mathtt{2comm} \vee \mathtt{1amb0})$$

and R contains no abstractions. Then $R \equiv \eta[0]$, for some η.

Proof: Immediate. $\Box$

We call *ambient abstraction* any closed abstraction described by the following grammar:

$$P ::= (x)\, \eta[\mathbf{0}] \;\mid\; (x)\, (\langle \eta\rangle \mid P)$$

The following lemma shows how to characterise ambient abstractions using formulas.

Lemma A.2.12 *Given an abstraction $(x) R$, suppose there are $q, p, \tilde{k}, \tilde{k'}, \tilde{z}$ such that*

$$\langle p\rangle \mid (x)\, (R\{\tilde{k}/\tilde{z}\}) \models \Box(\mathtt{2comm} \vee \mathtt{1amb0}) \tag{A.1}$$

and

$$\langle q\rangle \mid (x)\, (R\{\tilde{k'}/\tilde{z}\}) \models \Diamond\, \mathtt{1amb0}. \tag{A.2}$$

Then $(x)\, R$ is an ambient abstraction.

Proof: By induction on the number of nested abstractions in R. If this number is 0 then by Lemma A.2.11 we derive $R \equiv \eta[0]$.

Suppose the number is greater than 0. From (A.1) and

$$\langle p\rangle \mid (x)\, (R\{\tilde{k}/\tilde{z}\}) \longrightarrow R\{\tilde{k}/\tilde{z}\}\{p/x\}$$

we derive

$$R\{\tilde{k}/\tilde{z}\}\{p/x\} \models \mathtt{2comm} \vee \mathtt{1amb0}$$

Since R should contain an abstraction, the formula `1amb0` is not satisfied, hence

$$R\{\tilde{k}/\tilde{z}\}\{p/x\} \models \mathtt{2comm}$$

Using this, the fact that R should contain an abstraction, and (A.2) we infer that

$$R \equiv \langle \eta\rangle \mid (y)\, Q$$

for some η, y, Q. This information on R and the judgements (A.1) and (A.2) imply:

$$\langle \eta\{\tilde{k}/\tilde{z}\}\{p/x\}\rangle \mid (x)\,(Q\{\tilde{k}/\tilde{z}\}\{p/x\}) \models \Box(\texttt{2comm} \vee \texttt{1amb0})$$

and

$$\langle \eta\{\tilde{k}/\tilde{z}\}\{q/x\}\rangle \mid (x)\,(Q\{\tilde{k}/\tilde{z}\}\{q/x\}) \models \Diamond\ \texttt{1amb0}.$$

We can now conclude, using the inductive hypothesis on Q. $\Box$

We say that an ambient abstraction P is *simple* if $P =_\beta (x)\,x[0]$, where $=_\beta$ is the least congruence that is closed under the rule

$$\langle M\rangle \mid (x)\,P = P\{M/x\}\,.$$

Lemma A.2.13 *Suppose $(x)\,Q$ is an ambient abstraction, and*

$$(x)\,Q \models \texttt{1comm} \blacktriangleright \Diamond\ n[0]$$
$$(x)\,Q \models \texttt{1comm} \blacktriangleright \Diamond\ m[0]$$

with $m \neq n$. Then $(x)\,Q$ is simple.

Proof: From the hypothesis, there are p and q such that

$$\langle p\rangle \mid (x)\,Q \models \Diamond\ n[0] \quad \text{and}$$
$$\langle q\rangle \mid (x)\,Q \models \Diamond\ m[0]\,.$$

If $(x)\,Q$ were not simple, then the name of the ambient to which it reduces to would not depend on the argument x. (Note that any ambient abstraction is $=_\beta$ to an abstraction of the form $(x)\,\eta[0]$, for some x, η. The hypothesis of the lemma implies that $\eta = x$.) $\Box$

As hinted above, the key step is the definition of the formula below, which is the characteristic formula of simple ambient abstractions.

$$\begin{aligned}
\texttt{TestComm} \stackrel{\text{def}}{=}\ & \texttt{1comm} \\
& \wedge\ \texttt{1comm} \rhd \Box(\texttt{2comm} \vee \texttt{1amb0}) && \text{(A.3)} \\
& \wedge\ \texttt{1comm} \blacktriangleright \Diamond\ n[0] && \text{(A.4)} \\
& \wedge\ \texttt{1comm} \blacktriangleright \Diamond\ m[0] && \text{(A.5)}
\end{aligned}$$

where n, m are different names.

Lemma A.2.14 *$P \models \texttt{TestComm}$ iff P is a simple ambient abstraction and is closed.*

Proof: The implication from right to left is easy. We consider the opposite.

Process P must be an I/O, since $P \models \texttt{1comm}$. Also, P cannot be a message, otherwise it would not satisfy the formula

$$\texttt{1comm} \rhd \Box(\texttt{2comm} \vee \texttt{1amb0})$$

since a message in parallel with $(x)\,\mathbf{0}$ can reduce to $\mathbf{0}$, which does not satisfy $\texttt{2comm} \vee \texttt{1amb0}$.

We conclude that P should be an abstraction, say $(x)\,Q$. Now, from (A.3) and (A.4), we get that there are messages p, q such that

$$\langle p\rangle \mid (x)\,Q \models \Box(\texttt{2comm} \vee \texttt{1amb0})$$
$$\langle q\rangle \mid (x)\,Q \models \Diamond\, n[0]$$

From Lemma A.2.12 we infer that $(x)\,Q$ is an ambient abstraction. Moreover, by (A.4), (A.5) and Lemma A.2.13, $(x)\,Q$ must be simple. $\Box$

Now we are finally in the position of defining the characteristic formula for a message $\langle n\rangle$, where n is a name:

$$\boxed{\begin{array}{lll}\mathcal{F}_{\langle n\rangle} & \stackrel{\text{def}}{=} & \texttt{TestComm} \rhd \Diamond\, n[0] \\ & & \wedge \texttt{1comm}\end{array}}$$

and, then, the characteristic formula for a message is

$$\boxed{\texttt{1mess} \stackrel{\text{def}}{=} \exists x.\, \mathcal{F}_{\langle x\rangle}}$$

Nota: strictly speaking, in writing formula $\exists x.\, \mathcal{F}_{\langle x\rangle}$, we make an abuse of notation, since we should not be able to use x in subformula $\mathcal{F}_{\langle x\rangle}$. The formula above may actually be seen as an abbreviation for $\exists x.\,(\mathcal{F}_{\langle n_x\rangle}\{{}^{x}\!/\!{}_{n_x}\})$. We shall use the same notational mechanism below implicitely.

Lemma A.2.15 *$P \models \mathcal{F}_{\langle n\rangle}$ iff $P \equiv \langle n\rangle$.*

Proof: The right to left direction is easy. For the converse, we observe that P must be an I/O, and that P cannot be an abstraction (otherwise, when adding a process satisfying `TestComm`, we could not obtain an ambient). Hence $P \equiv \langle m\rangle$, for some m.

Conversely, if Q is a simple ambient abstraction, then

$$Q \mid \langle m\rangle \models \Diamond\, n[\mathbf{0}] \quad \text{iff} \quad m = n\,.$$

$\Box$

We may now define two modalities for the input connective:

$$\boxed{\begin{array}{lll}\langle ?n\rangle.\,\mathcal{A} & \stackrel{\text{def}}{=} & \texttt{1comm} \ \wedge\ \mathcal{F}_{\langle n\rangle} \rhd \Diamond\mathcal{A} \\ [?n].\,\mathcal{A} & \stackrel{\text{def}}{=} & \langle ?n\rangle.\,\top \ \wedge\ \neg\langle ?n\rangle.\,\neg\mathcal{A} \\ \texttt{1input} & \stackrel{\text{def}}{=} & \exists x.\ \langle ?x\rangle.\,\top\end{array}}$$

Lemma A.2.16

- *$P \models \langle ?n\rangle.\,\mathcal{A}$ iff there are P', P'' such that $P \equiv (x)P'$, $(x)P' \mid \langle n\rangle \Rightarrow P''$, and $P'' \models \mathcal{A}$.*
- *$P \models \langle ?n\rangle.\,\mathcal{A}$ iff there is P' with $P \equiv (x)P'$, and for all P'' such that $(x)P' \mid \langle n\rangle \Rightarrow P''$, $P'' \models \mathcal{A}$.*

The constructions we have presented in this section capture the observations that will be needed in the definition of intensional bisimilarity in Section A.4.1. Before introducing this relation and studying its properties, we further study the expressive power of the logic.

A.3 Other intensional properties

As we have just seen, the Ambient Logic can capture several syntactical constructions of the calculus, even though not all of them are primitive in the logic. We now further explore AL's expressiveness, and go beyond the results we have established about capabilities and communication.

We first define a formula ϕ_{fin} that characterises finite terms, using a form of *contextual reasoning*. The same method is applied to derive a formula ©n that characterises the terms containing n as a free name. We then introduce formulas that characterise in a restricted sense *persistent* single terms of the calculus. These formulas will be used in Annex B to characterise the logical equivalence through characteristic formulas.

A.3.1 Capturing finiteness

We now present a formula that is satisfied by all and only finite processes. Detecting replication seems *a priori* unfeasable in the present version of the Ambient Logic, as it does not provide a recursion operator. We may actually capture the 'finite' character of a term, using the fact that a replicated process is *persistent*, i.e., it is always present along the reductions of a term.

The characterisation of finiteness relies on the existence of a scenario which guarantees reachability of $\mathbf{0}$, as expressed by the two following lemmas:

Lemma A.3.1 *Let P, Q be two terms such that $P \Rightarrow Q$. Then P is finite iff Q is finite.*

Proof: By induction over the length of the $\Rightarrow$ derivation, then induction over the structure of the proof of the $\rightarrow$ transition. □

Lemma A.3.2 *$P \in \mathsf{MA}$ is finite iff there are Q, R, n such that $n[P \mid Q] \mid R \Rightarrow \mathbf{0}$.*

Proof:

- Let us assume that P is finite. We prove by induction on P that there exist Q and R such that for any P',

$$n[P \mid P' \mid Q] \mid R \;\Rightarrow\; n[P']$$

The left to right implication can then be obtained using this property with $P' = \mathbf{0}$ and adding $\mathsf{open}\, n$ in parallel with R.

1 For $P = \mathbf{0}$, take $Q = R = \mathbf{0}$.

2 For $P \equiv m[P_1]$, we have by induction Q_1, R_1 such that $n[P_1 \mid P' \mid Q_1] \mid R_1 \Rightarrow n[P']$ for any P'. Now we set $Q = \mathsf{open}\, m \mid Q_1$ and $R = R_1$. Then it is clear that $n[m[P_1] \mid P' \mid Q] \mid R \Rightarrow n[P']$ for any P'.

3 For $P \equiv P_1 \mid \ldots \mid P_r$ (with no replicated component), we use the induction hypothesis to obtain Q_i and R_i, and then set $Q = Q_1 \mid \ldots \mid Q_r$, $R = R_1 \mid \ldots \mid R_r$ such that for any P',

$$\begin{aligned} n[P \mid P' \mid Q] \mid R \;&\Rightarrow\; n[P_2 \mid \ldots \mid P_r \mid P' \mid Q_2 \mid \ldots \mid Q_r] \mid R_2 \mid \ldots \mid R_r \\ &\Rightarrow \cdots \Rightarrow\; n[P'] \end{aligned}$$

reasoning inductively on r.

4 For $P \equiv \mathsf{cap}.\, P_1$, we use the induction hypothesis to get Q_1 and R_1, and we define Q and R according to the shape of cap as follows:

  * $\mathsf{cap} = \mathsf{in}\, m$. Then we set $Q = Q_1$ and $R = m[\mathbf{0}] \mid \mathsf{open}\, m \mid R_1$ Then for any P':

$$\begin{aligned} n[P \mid P' \mid Q] \mid R &\rightarrow m[n[P_1 \mid P' \mid Q]] \mid \mathsf{open}\, m \mid R_1 \\ &\rightarrow n[P_1 \mid P' \mid Q_1] \mid R_1 \\ &\Rightarrow n[P'] \end{aligned}$$

  * $\mathsf{cap} = \mathsf{out}\, m$. Then we set $Q = \mathsf{in}\, m \mid Q_1$ and $R = m[\mathbf{0}] \mid \mathsf{open}\, m \mid R_1$, so that we can conclude.
  * $\mathsf{cap} = \mathsf{open}\, m$. Then we set $Q = m[\mathbf{0}] \mid Q_1$ and $R = R_1$.

– For $P \equiv \langle n \rangle$: set $Q = (x)\mathbf{0}$, and $R = \mathbf{0}$.

– For $P \equiv (x)P_1$: by induction hypothesis applied to $P_1\{n/x\}$, we get Q_1 and R_1; then we set $Q = \langle n \rangle \mid Q_1$ and $R = R_1$.

The first implication is thus established.

- Let us assume P is not finite. Then for any n, Q, R, $n[P \mid Q] \mid R$ is also infinite, and by the previous lemma, it is also the case for any of its reducts, and hence it cannot reduce to $\mathbf{0}$.

The lemma is proved. □

We can now define:

$$\boxed{\mathsf{finite} \stackrel{\text{def}}{=} \exists x. \big(\top \blacktriangleright (\top \blacktriangleright \Diamond 0)@x\big)}$$

Proposition A.3.3 *For any $P \in \mathsf{MA}$, $P \models \mathsf{finite}$ iff P is finite.*

Proof: Direct consequence of the previous lemma. □

A.3.2 Formula for name occurence

Our aim is now to define a formula corresponding to the connective ©n, defined by:

$$P \models \copyright n \quad \text{iff} \quad n \in \mathrm{fn}(P)\,.$$

In presence of name restriction in the calculus, the logical connective of *name revelation* [CG01b] allows one to derive ©n. In our setting, we exploit the ability, using the formulas for capabilities, to detect unguarded occurrences of names, together with Lemma A.3.4.

We say that a process P is *flat* if the only process underneath all prefixes and inside all ambients of P is $\mathbf{0}$. We say that a process P has an occurrence of name n at top level if $P \equiv \mathsf{cap}.\, P'$ with $\mathsf{cap} = \mathsf{in}\, n$, $\mathsf{out}\, n$ or $\mathsf{open}\, n$, $P \equiv m[\mathbf{0}]$ or $P \equiv \langle n \rangle$.

Lemma A.3.4 *For all P, n, $n \in \mathrm{fn}(P)$ iff for any name m, there exist some flat processes Q, R, in which n does not occur free, and a process S with an occurrence of n at top level such that $m[P \mid Q] \mid R \Rightarrow m[S]$.*

Proof: We first introduce an auxiliary notion which will be useful for the proof.

The *occurrence depth* of a name n in a term is given by a function $\mathsf{deep}_n : \mathcal{P}{\to}\mathbb{N}\cup\{\infty\}$, inductively defined as follows:

- $\mathsf{deep}_n(\mathbf{0}) = \infty$.
- $\mathsf{deep}_n(n[P_1]) = 0$, and for $n \neq \eta$, $\mathsf{deep}_n(\eta[P_1]) = \mathsf{deep}_n P_1 + 1$.
- $\mathsf{deep}_n((!)P_1 \mid \ldots \mid (!)P_r) = \min_{1\leq i\leq r} \mathsf{deep}_n(P_i)$ (here $(!)Q$ stands for Q or $!Q$).
- $\mathsf{deep}_n(\mathsf{cap}.\, P) = 0$ for $\mathsf{cap} \in \{\mathsf{in}\, n, \mathsf{out}\, n, \mathsf{open}\, n\}$, and $\mathsf{deep}_n(\mathsf{cap}.\, P) = \mathsf{deep}_n(P) + 1$ otherwise.
- $\mathsf{deep}_n((x)P) = \mathsf{deep}_n(P) + 1$, $\mathsf{deep}_n(\langle n\rangle)) = 0$ and $\mathsf{deep}_n(\langle\eta\rangle) = \infty$ for $\eta \neq n$.

Note that the property of S having an occurrence of n at top level is equivalent to $\mathsf{deep}_n(S) = 0$. We are now ready to prove the lemma:

- first implication: Let us assume that $\mathsf{deep}_n(P) < \infty$. We consider a name m, and prove by induction on $\mathsf{deep}_n(P)$ that there exist Q, R, S satisfying the conditions of the lemma.
  - if $\mathsf{deep}_n(P) = 0$, we take $Q = R = \mathbf{0}$ and $S = P$.
  - if $\mathsf{deep}_n(P) = i{+}1$, let us assume for example that $P \equiv \mathsf{in}\, m_1.\, P_1 \mid P_2$ with $\mathsf{deep}_n(P_1) = i$. By induction hypothesis, there are Q_1, R_1, S_1 and m satisfying the conditions of the lemma P_1. But $Q_1, R_1, S_1 \mid P_2$ also satisfy these conditions for $P_1 \mid P_2$. So if we set $Q = Q_1$, $R = m_1[\mathbf{0}] \mid \mathsf{open}\, m_1 \mid R_1$ and $S = S_1 \mid P_2$, then Q, R, S can be chosen for P. For the other cases, we proceed as in the proof of Lemma A.3.2.

  The first implication is proved.
- second implication: Let us assume that $n \notin \mathrm{fn}(P)$. We consider $m \neq n$, and some Q, R as in the statement of the lemma. Then $n \notin \mathrm{fn}(m[P \mid Q] \mid R)$, so that for any T such that $m[P \mid Q] \mid R \;\Rightarrow\; T$, $n \notin \mathrm{fn}(T)$. The second implication thus follows.

□

We can now define the formula to capture the set of free names of a process; it is given in Table A.4.

$$
\begin{array}{lcl}
\mathsf{flat} & \stackrel{\text{def}}{=} & (\exists x.\ [\![\mathsf{in}\, x]\!].\, 0 \ \vee\ [\![\mathsf{out}\, x]\!].\, 0 \ \vee\ [\![\mathsf{open}\, x]\!].\, 0 \ \vee x[0] \ \vee\ \mathcal{F}_{\langle x\rangle})^{\omega} \\
©^1 n & \stackrel{\text{def}}{=} & \big(\langle\langle \mathsf{in}\, n\rangle\rangle.\top \ \vee\ \langle\langle \mathsf{out}\, n\rangle\rangle.\top \ \vee\ \langle\langle \mathsf{open}\, n\rangle\rangle.\top \ \vee\ n[\top] \ \vee\ \mathcal{F}_{\langle n\rangle}\ \big) \ \mid\ \top \\
©n & \stackrel{\text{def}}{=} & \forall x.\ (\mathsf{flat} \wedge \neg ©^1 n) \ \blacktriangleright\ \big(\ (\mathsf{flat} \wedge \neg ©^1 n) \ \blacktriangleright\ \Diamond\ x[©^1 n]\ \big)@x
\end{array}
$$

Table A.4: Formulas for free names

Formula $©n$ detects whether name n occurs in a process, while $©^1 n$ detects whether n occurs at top level (i.e. P satisfies this formula iff $\mathsf{deep}_n(P) = 0$).

$\mathsf{Rep}_{\mathsf{in}\,n}(\mathcal{A})$	$\stackrel{\text{def}}{=}$	$\mathcal{A}^\omega \;\wedge\; \forall m.\ (\neg © m) \;\rightarrow$ $(\llbracket \mathsf{out}\, n \rrbracket.\, 0)^\omega \;\triangleright\; \big(\, n[0] \;\triangleright\; \Box\Diamond\; (\; n[m[\mathcal{A} \;\mid\; \top]]\;)\big)@m$
$\mathsf{Rep}_{\mathsf{out}\,n}(\mathcal{A})$	$\stackrel{\text{def}}{=}$	$\mathcal{A}^\omega \;\wedge\; \forall m.\ (\neg © m) \;\rightarrow$ $(\llbracket \mathsf{in}\, n \rrbracket.\, 0)^\omega \;\triangleright\; \big(\, n[0] \;\triangleright\; \Box\, \Diamond(\; m[\mathcal{A} \;\mid\; \top] \mid n[0]\;)\big)@m$
$\mathsf{Rep}_{\mathsf{open}\,n}(\mathcal{A})$	$\stackrel{\text{def}}{=}$	$\mathcal{A}^\omega \;\wedge\; (n[0])^\omega \;\triangleright\; \Box\; (\; \mathcal{A} \;\mid\; \top\;)$
$\mathsf{Rep}_{n[]}(\mathcal{A})$	$\stackrel{\text{def}}{=}$	$(n[\mathcal{A}])^\omega \;\wedge\; (\llbracket \mathsf{open}\, n \rrbracket.\, 0)^\omega \;\triangleright\; \Box\; (\; n[\mathcal{A}] \;\mid\; \top\;)$
$\mathcal{F}_{!\langle n\rangle}$	$\stackrel{\text{def}}{=}$	${\mathcal{F}_{\langle n\rangle}}^\omega \;\wedge\; \mathtt{TestComm}^\omega \;\triangleright\; \Box\; (\; \mathcal{F}_{\langle n\rangle} \;\mid\; \top\;)$
$\mathsf{Rep}_{input}(\mathcal{A})$	$\stackrel{\text{def}}{=}$	$\mathcal{A}^\omega \;\wedge\; \mathtt{1mess}^\omega \;\triangleright\; \Box\; (\; \mathcal{A} \;\mid\; \top\;)$

Table A.5: Formulas for persistent single terms

Proposition A.3.5 (Name occurrence) $P \models © n$ *iff* $n \in \mathrm{fn}(P)$.

Proof: Consequence of the previous Lemma. □

A.3.3 Formulas for persistence

We now move to the definition of formulas that characterise persistence. A formula $!\mathcal{A}$ cannot be defined in the general case, for any $\mathcal{A}$ but only, modulo some hypotheses, for a formula $\mathcal{A}$ that captures single terms. In this case, the definition of $!\mathcal{A}$ has two parts. The first part says that if $P {\models} !\mathcal{A}$ then all parallel components in P that are single and at top level satisfy $\mathcal{A}$. This is expressed by the formula $\mathcal{A}^\omega$. The second part of the definition of $!\mathcal{A}$ addresses persistence, by saying that there are infinitely many processes at top level that satisfy $\mathcal{A}$ in the sense that we may not consume all copies by some finite sequence of reduction. Definitions are given in Table A.5: there is one formula for each possible topmost constructor (recall that we are considering a single process).

Formula $\mathcal{F}_{!\langle n\rangle}$ is actually a characteristic formula, since it is satisfied only by the process $!\langle n\rangle$. For this reason, we anticipate the notation $\mathcal{F}_P$ of the characteristic formula of P (see Sect A.4). For the other formulas, we express the replication of a process satisfying $\mathcal{A}$; the interpretation of these formulas hence relies on the actual meaning of $\mathcal{A}$.

To illustrate the point, consider formula $\mathsf{Rep}_{\mathsf{open}\,n}(\langle \mathsf{open}\, n\rangle.\top)$. This formula only specifies that any number of capabilities $\mathsf{open}\, n$ should be present at toplevel, and thus holds for process $!\mathsf{open}\, n.\, \mathbf{0}$, but also for $\mathsf{open}\, n.\, !\mathsf{open}\, n.\, \mathbf{0}$. On the other hand, $\langle \mathsf{open}\, n\rangle.\top$ can be replaced by the more discriminating formula $\big[\mathsf{open}\, n\big].\, 0$: then we obtain a formula that only accepts process $!\mathsf{open}\, n.\, \mathbf{0}$.

In light of these observations, we define the following measures on terms:

Definition A.3.6 (Sequentiality degree, $\mathsf{ds}()$) *The sequentiality degree of a term is defined as follows:*

- $\mathsf{ds}(()\mathbf{0}) = 0$, $\mathsf{ds}(()P \mid Q) = \max\big(\mathsf{ds}(()P), \mathsf{ds}(()Q)\big)$;
- $\mathsf{ds}(()\eta[P]) = \mathsf{ds}(()!P) = \mathsf{ds}(()P)$;
- $\mathsf{ds}(()\mathsf{cap}.\, P) = \mathsf{ds}(()P)$.

- $\mathsf{ds}(()\langle\eta\rangle) = 1$ *and* $\mathsf{ds}(()(x)P) = \mathsf{ds}(()P') + 1$ *if* $(x)P'$ *is the eta-normalised form of* $(x)P$.

Definition A.3.7 (Depth degree) *The depth degree of a process is given by a function* deep *from* MA *processes to natural numbers, inductively defined by:*

- $\mathsf{deep}(\mathbf{0}) \ = \ 0$, $\mathsf{deep}(\mathsf{cap}.\, P) = \mathsf{deep}((x)P) = \mathsf{deep}(\langle\eta\rangle) \ = \ 0$*;*
- $\mathsf{deep}(\eta[P]) \ = \ \mathsf{deep}(P) + 1$*;*
- $\mathsf{deep}((!)P_1 \mid \ldots \mid (!)P_r) \ = \ \max_{1 \le i \le r} \mathsf{deep}(P_i)$.

Lemma A.3.8 *For any processes* P *and* Q, $P \equiv Q$ *implies* $\mathsf{ds}(()P) = \mathsf{ds}(()Q)$ *and* $\mathsf{deep}(P) = \mathsf{deep}(Q)$.

Note that the possibility to define quantities $\mathsf{ds}(()P)$ and $\mathsf{deep}(P)$ up to structural congruence relies on the usage of the replication operator (instead of recursion) to represent infinite behaviours (or, more accurately, some form of spatial deployment).

Definition A.3.9 (Selective and expressive formulas) *We say that a formula is* sequentially *(resp.* depth*)* selective *if all processes satisfying it have the same sequentiality (resp. depth) degree.*

For any capability cap *(resp. name* n*) and formula* $\mathcal{A}$, $\mathcal{A}$ *is* cap-expressive *(resp.* n-expressive, input-expressive*) if all terms satisfying it are of the form* $\mathsf{cap}.\, P$ *(resp.* $n[P]$,$(x)P$*).*

Example A.3.10 $\langle \mathsf{in}\, n\rangle.\, n[0]$ *is* $\mathsf{in}\, n$*-expressive but not sequentially selective: it admits both* $\mathsf{in}\, n.\, n[\mathbf{0}]$ *and* $\mathsf{in}\, n.\, (n[\mathbf{0}] \mid \mathsf{open}\, n.\, n[\mathbf{0}])$ *as models. On the other hand,* $\left[\mathsf{in}\, n\right].\, n[0]$ *is both sequentially selective and* $\mathsf{in}\, n$*-expressive. As we will see below, the combination of* $\langle\mathsf{cap}\rangle$ *and* $\left[\mathsf{cap}\right]$ *modalities allows us to define sequentially selective formulas.*

These two forms of selectivity are useful for the characterisation of persistence. Indeed, the sequentiality (resp. depth) degree of a single prefixed (resp. ambient) term is strictly decreasing when consuming the prefix (resp. opening the ambient). This property is needed in order to detect the presence of replication at toplevel in a process, and interpret the formulas introduced above.

Lemma A.3.11 *Let* P, Q *be two terms of* MA*. If* $P \to Q$ *or* $P \xrightarrow{\mu} Q$ *for some* μ*, then* $\mathsf{ds}(()P) \geq \mathsf{ds}(()Q)$.

Proof: The property for $\xrightarrow{\mu}$ follows from the definition of $\mathsf{ds}(()P)$. For $P \to Q$, one reasons by induction and case analysis (using Lemma A.1.4). $\square$

Corollary A.3.12 *For all* cap*, if* $P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} Q$*, then* $\mathsf{ds}(()P) \geq \mathsf{ds}(()Q)$.

Lemma A.3.13 (Characterisation of replication of single processes)

1. *Given a capability* cap*, and a sequentially selective and* cap*-expressive formula* $\mathcal{A}$*, define* $!\mathcal{A} \stackrel{\text{def}}{=} \mathit{Rep}_{\mathsf{cap}}(\mathcal{A})$*. Then* $P \models !\mathcal{A}$ *iff there are* $r \geq 1, s \geq r, P_i$ *(*$1 \leq i \leq s$*) such that* $P \equiv \Pi_{1 \leq i \leq r} !\mathsf{cap}.\, P_i \ \mid \ \Pi_{r+1 \leq i \leq s} \mathsf{cap}.\, P_i$*, and* $\mathsf{cap}.\, P_i \models \mathcal{A}$ *for all* $1 \leq i \leq s$*, where* $\Pi_{1 \leq i \leq t} Q_i$ *abbreviates* $Q_1 \mid \ldots \mid Q_t$.

2. *For any name n and depth selective and n-expressive formula $\mathcal{A}$, define* $!\mathcal{A} \stackrel{\text{def}}{=} \mathsf{Rep}_{n[]}(\mathcal{A})$. *Then $P \models !\mathcal{A}$ iff there are $r \geq 1, s \geq r, P_i$ $(1 \leq i \leq s)$ such that $P \equiv \Pi_{1 \leq i \leq r} !n[P_i] \mid \Pi_{r+1 \leq i \leq s} n[P_i]$, and $n[P_i] \models \mathcal{A}$ for all $1 \leq i \leq s$.*

3. *For any formula $\mathcal{A}$ that is sequentially selective and input expressive, define* $!\mathcal{A} \stackrel{\text{def}}{=} \mathsf{Rep}_{input}(\mathcal{A})$. *Then $P \models !\mathcal{A}$ iff there are $r \geq 1, s \geq r, P_i$ $(1 \leq i \leq s)$ such that $P \equiv \Pi_{1 \leq i \leq r} !n[P_i] \mid \Pi_{r+1 \leq i \leq s} n[P_i]$, and $n[P_i] \models \mathcal{A}$ for all $1 \leq i \leq s$.*

Proof:
Case 1, $\mathsf{cap} = \mathsf{in}\, n$**.** Assume there exist some terms $P_1, \ldots, P_s$ satisfying the condition expressed in 1. Then the first part of $!\mathcal{A}$ is satisfied, i.e. $P \models \mathcal{A}^\omega$.

To establish the second part, we have to show that for any $Q \equiv \mathsf{out}\, n^\omega$ (where $\omega \in \mathbb{N}^* \cup \{\infty\}$), any fresh name m, and any term R such that $m[P \mid Q] \mid n[\mathbf{0}] \Rightarrow R$, there is a further reduction $R \Rightarrow n[m[R_1 \mid R_2]]$ for some R_1, R_2 such that $R_1 \models \mathcal{A}$, which entails in particular $R_1 \equiv \mathsf{in}\, n. R_1'$. Since ambient n does not contain any active process, and since there is no active process at toplevel in $m[P \mid Q] \mid n[\mathbf{0}]$, ambient n remains at toplevel in all evolutions of this term. Moreover, we have that m is fresh for P and Q; therefore, no ambient may get out of m, so for any reduct R, there exists R' such that either (i) $R \equiv m[R'] \mid n[\mathbf{0}]$, and $P \mid Q \xRightarrow{(\mathsf{in}\, n, \mathsf{out}\, n)^*} R'$, or (ii) $R \equiv n[m[R']]$, and $P \mid Q \xrightarrow{\mathsf{in}\, n} \xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*} R'$. In the first case, because of the shape of P, we may perform one more step of reduction to reach a situation like (ii), and then, since $P \mid Q \xrightarrow{\mathsf{in}\, n} \xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*} R'$, there exists R'' such that $R' \equiv !\mathsf{in}\, n. P_1 \mid R''$. The first implication is thus proved.

Conversely, let us now assume that $P \models \mathsf{Rep}_{\mathsf{in}\, n}(\mathcal{A})$. Then according to the first part of the formula, there exist some P_is satisfying $P \equiv (!)\mathsf{in}\, n. P_1 \mid \ldots \mid (!)\mathsf{in}\, n. P_r$ and $\mathsf{in}\, n. P_i \models \mathcal{A}$. Suppose now by absurd that no component is replicated. We exploit the sequential selectivity hypothesis to obtain a contradiction. Indeed, we have the reduction $m[P \mid (\mathsf{out}\, n)^r] \mid n[\mathbf{0}] \Rightarrow R = n[m[P_1 \mid \ldots \mid P_r]]$ and R is a term whose sequentiality degree is stricly smaller than $\mathsf{ds}(()P)$. Then it is also the case for any of its reducts, and therefore the same reasoning holds for any R_1, R_2 such that $R \Rightarrow n[m[\mathsf{in}\, n. R_1 \mid R_2]]$, $\mathsf{in}\, n. R_1$ has a sequentiality degree too small to satisfy $\mathcal{A}$ because of sequential selectivity. Thus, P cannot satisfy $\mathsf{Rep}_{\mathsf{in}\, n}(\mathcal{A})$, and we obtain a contradiction. Hence, at least one of the P_is is replicated, and the reverse implication is proved.
The proofs for **Case 1**, other capabilities, and **Case 3** follow from similar arguments.

Case 2. Let us now treat case 2, and assume that $P \equiv !n[P_1] \mid \ldots \mid (!)n[P_r]$, with the P_is such that $P_i \models \mathcal{A}$. Then P satisfies $\mathsf{Rep}_{n[]}(\mathcal{A})$ iff for any $Q \equiv \mathsf{open}\, n^\omega$, and any R such that $P \mid Q \Rightarrow R$, there are R_is such that $R \equiv n[R_1] \mid R_2$ with $n[R_1] \models \mathcal{A}$. Since for any R, $R \equiv !n[P_1] \mid R'$, the first implication is established.

Conversely, suppose P satisfies $\mathsf{Rep}_{n[]}(\mathcal{A})$. Then $P \equiv (!)n[P_1] \mid \ldots \mid (!)n[P_r]$. Moreover, if no P_i is replicated, $P \mid !\mathsf{open}\, n \Rightarrow P_1 \mid \ldots \mid P_r \mid !\mathsf{open}\, n$, and if in some P_i there are $P_{i,j}$ $(j = 1, 2)$ such that $P_i \equiv n[P_{i,1}] \mid P_{i,2}$, then the depth degree of $P_{i,1}$ is too small for $n[P_{i,1}]$ to satisfy $\mathcal{A}$, which gives us the second implication. $\square$

The formulas for persistence, together with the constructions of Section A.2, will be used to derive characteristic formulas w.r.t. $=_L$ for a subcalculus of MA in Section A.4.

A.4 Characteristic formulas

In this section we will establish the existence of characteristic formulas for a very general class of processes. Given a process P, a characteristic formula for P is a formula $\mathcal{F}_P$ such that:

$$\forall Q. \quad Q \models \mathcal{F}_P \quad \text{iff} \quad Q =_L P\,.$$

The definability of characteristic formulas is an interesting property since these allow one to relate the model-checking and validity problems. Indeed, we have that

$$P \models \mathcal{A} \quad \text{iff} \quad \vdash \mathcal{F}_P \to \mathcal{A}\,.$$

In spatial logics, moreoever, we have the following property, that follows from the interpretation of operator $\triangleright$:

$$\vdash \mathcal{A} \quad \text{iff} \quad \mathbf{0} \models (\neg\mathcal{A}) \triangleright \bot\,.$$

To be able to carry out our programme, we have first to understand what $=_L$ represents. For this, we look for a co-inductive characterisation of $=_L$, as a form of labelled bisimilarity. Then, making an intensive use of the formulas for the connectives of the calculus we defined above, we will derive characteristic formulas for a subcalculus of MA.

A.4.1 Intensional bisimilarity

We now use the labelled transitions (Definition A.1.5) to define a notion of intensional bisimilarity in order to capture $=_L$.

Definition A.4.1 Intensional bisimilarity *is the largest symmetric relation $\approx_{int}$ on processes such that $P \approx_{int} Q$ implies:*

1. *If $P \equiv P_1 \mid P_2$ then there are Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ and $P_i \approx_{int} Q_i$, for $i = 1, 2$.*
2. *If $P \equiv \mathbf{0}$ then $Q \equiv \mathbf{0}$.*
3. *If $P \longrightarrow P'$ then there is Q' such that $Q \Longrightarrow Q'$ and $P' \approx_{int} Q'$.*
4. *If $P \xrightarrow{\mathsf{in}\,n} P'$ then there is Q' such that $Q \overset{\mathsf{in}\,n}{\Longrightarrow} \xRightarrow{(\mathsf{out}\,n, \mathsf{in}\,n)^*} Q'$ and $P' \approx_{int} Q'$.*
5. *If $P \xrightarrow{\mathsf{out}\,n} P'$ then there is Q' such that $Q \overset{\mathsf{out}\,n}{\Longrightarrow} \xRightarrow{(\mathsf{in}\,n, \mathsf{out}\,n)^*} Q'$ and $P' \approx_{int} Q'$.*
6. *If $P \xrightarrow{\mathsf{open}\,n} P'$ then there is Q' such that $Q \overset{\mathsf{open}\,n}{\Longrightarrow} Q'$ and $P' \approx_{int} Q'$.*

7. *If* $P \xrightarrow{!n} P'$ *then there is* Q' *such that* $Q \stackrel{!n}{\Longrightarrow} Q'$ *and* $P' \approx_{int} Q'$.

8. *If* $P \xrightarrow{?n} P'$ *then there is* Q' *such that* $Q \mid \langle n \rangle \Longrightarrow Q'$ *and* $P' \approx_{int} Q'$.

9. *If* $P \equiv n[P']$ *then there is* Q' *such that* $Q \equiv n[Q']$ *and* $P' \approx_{int} Q'$.

The definition of $\approx_{int}$ has (at least) two intensional clauses, namely (1) and (2), which allow us to observe parallel compositions and the terminated process. These clauses correspond to the intensional connectives '$|$' and '0' of the logic. The clause (8) for abstraction is similar to the input clause of bisimilarity in asynchronous message-passing calculi [ACS98]. This is the case because communication in MA is asynchronous. Another consequence of this is that the logic is insensitive to the following rewrite rule (modulo associativity-commutativity of $|$):

$$(x)\big(\langle x \rangle \mid (x)P\big) \ \rightarrow_\eta \ (x)P\,.$$

This rule induces a notion of normal form of processes, that we shall call the *eta-normalised form.*

Definition A.4.2 (Eta-equivalence) *We will note* $P \equiv_E Q$ *if the normal forms of* P *and* Q *for* $\rightarrow_\eta$ *are related by* $\equiv$.

Lemma A.4.3 (cf. Appendix B) *In* MA, $P \equiv_E Q$ *implies* $P \approx_{int} Q$.

By Proposition C.2.4 below, this result says that the logic is insensitive to $\rightarrow_\eta$. We shall thus reason using normalised processes w.r.t. $\rightarrow_\eta$, which will make proofs simpler.

The most peculiar aspect of the definition of $\approx_{int}$ is the use of the stuttering relations. Although they could be avoided when restricting on finite processes, they cannot in the full calculus.

Example A.4.4 (Equivalent processes modulo stuttering) *Consider the processes*

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & !\mathsf{open}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}] \mid n[\mathbf{0}] \\ Q & \stackrel{\text{def}}{=} & !\mathsf{open}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}] \mid \mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}]\,. \end{array}$$

It holds that $P \not\approx_{int} Q$. *However, since* $P \xRightarrow{(\mathsf{in}\,n,\mathsf{out}\,n)^*} Q \xRightarrow{(\mathsf{in}\,n,\mathsf{out}\,n)^*} P$, *we have*

$$\mathsf{out}\,n.\,P \ \approx_{int} \ \mathsf{out}\,n.\,Q\,.$$

To see why the extra capabilities of Q *do not affect its behaviour, consider the following reduction of* $\mathsf{out}\,n.\,P$, *put in some context:*

$$n[m[\mathsf{out}\,n.\,P \mid R]] \ \longrightarrow \ n[\mathbf{0}] \mid m[P \mid R]$$

In the same context, process $\mathsf{out}\,n.\,Q$ *can match this transition using three reductions:*

$$\begin{array}{rcl} n[m[Q \mid R]] & \longrightarrow & n[\mathbf{0}] \mid m[\mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}] \mid Q' \mid R] \\ & \longrightarrow & n[m[\mathsf{out}\,n.\,n[\mathbf{0}] \mid Q' \mid R]]S \\ & \longrightarrow & n[\mathbf{0}] \mid m[P \mid R] \end{array}$$

where $Q' = !\mathsf{open}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}]$.

Conversely, we have

$$n[m[\mathsf{out}\, n.\, Q \mid R]] \longrightarrow n[\mathbf{0}] \mid m[Q \mid R]\,,$$

and $\mathsf{out}\, n.\, P$ *can mimic this reduction as follows:*

$$\begin{array}{rcl} n[m[\mathsf{out}\, n.\, P \mid R]] & \longrightarrow & n[\mathbf{0}] \mid m[n[\mathbf{0}] \mid Q' \mid R] \\ & \longrightarrow & n[\mathbf{0}] \mid m[Q' \mid \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}] \mid R] \\ & \longrightarrow & n[m[Q' \mid \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}] \mid R]] \\ & \longrightarrow & n[\mathbf{0}] \mid m[Q \mid R] \end{array}$$

By contrast, stuttering does not show up in Safe Ambients [LS00b], where movements are achieved by means of synchronisations involving a capability and a *co-capability*.

We now state some results about $\approx_{int}$ that are proved in annex B.

Proposition A.4.5 (cf. Appendix B) $\approx_{int}$ *is a congruence relation.*

Proposition A.4.6 (cf. Appendix B) *For any* P, Q, $P \approx_{int} Q$ *implies* $P {=_L} Q$.

The latter result establishes *correction* of $\approx_{int}$ w.r.t. $=_L$. Given a process P, we try and characterise the equivalence class of P w.r.t. $\approx_{int}$ with a formula $\mathcal{F}_P$. The definability of such a formula will actually entail that $=_L \;\subseteq\; \approx_{int}$ (*completeness*), and hence that $\mathcal{F}_P$ actually characterises the $=_L$-equivalence class of P.

We now mention a useful induction principle that allows us to reason *'almost inductively'* on the structure of a process when checking relation $\approx_{int}$. This principle is given by the following inductive order:

Definition A.4.7 *We write* $P > Q$ *if either* $\mathsf{ds}(P) > \mathsf{ds}(Q)$ *or* Q *is a subterm of* P.

This order allows us, using the following result, to derive an inductive characterisation of $\approx_{int}$ (see appendix B).

Proposition A.4.8 (cf. Appendix B) *Let* P, P_1, P_2, Q *be processes of* MA. *Then*

1. $\mathbf{0} \;\approx_{int}\; Q$ *iff* $Q \equiv \mathbf{0}$.
2. $n[P] \;\approx_{int}\; Q$ *iff there exists* Q' *such that* $Q \;\equiv\; n[Q']$ *and* $P \;\approx_{int}\; Q'$.
3. $P_1 \mid P_2 \;\approx_{int}\; Q$ *iff there exist* Q_1, Q_2 *such that* $Q \;\equiv\; Q_1 \mid Q_2$ *and* $P_i \;\approx_{int}\; Q_i$ *for* $i = 1, 2$.
4. $!P \;\approx_{int}\; Q$ *iff there exist* $r \geq 1, s \geq r, Q_i$ $(1 \leq i \leq s)$ *such that* $Q \equiv \Pi_{1\leq i\leq r}!Q_i \mid \Pi_{r+1\leq i\leq s}Q_i$, *and* $P \;\approx_{int}\; Q_i$ *for* $i = 1 \ldots s$.
5. $\mathsf{cap}.\, P \approx_{int} Q$ *iff there exists* Q' *such that* $Q \;\equiv\; \mathsf{cap}.\, Q'$ *with* $P \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow}\approx_{int}\; Q'$ *and* $Q' \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow}\approx_{int}\; P$.
6. $\langle n\rangle \;\approx_{int}\; Q$ *iff* $Q \equiv \langle n\rangle$.
7. $(x)P \;\approx_{int}\; Q$ *iff there exists* P', Q', Q'' *and* $n \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q)$ *such that* $Q \equiv (x)Q'$, $Q \mid \langle n\rangle \Rightarrow Q''$, $P\{{}^{n}\!/\!{}_{x}\} \approx_{int} Q''$, $(x)P \mid \langle n\rangle \Rightarrow P'$ *and* $P' \approx_{int} Q'\{{}^{n}\!/\!{}_{x}\}$.

A.4.2 The subcalculus MA_{IF}

As we mentioned above, characteristic formulas and completeness for an algebraic characterisation of logical equivalence are two related problems. In fact, the existence of characteristic formulas is a stronger result than completeness of $\approx_{int}$ w.r.t. $=_L$: while we establish completeness in appendix B on the whole calculus, we are only able to derive characteristic formulas on a subcalculus of MA. To introduce the necessity of restricting the class of processes we consider, and to illustrate the basic ideas behind the construction of characteristic formulas, we examine some examples.

Example A.4.9 *We consider processes* $P_1 = !\mathsf{open}\, n.\, n[\mathbf{0}]$, $P_2 = \mathsf{open}\, n \mid n[\mathbf{0}]$ *and* $P_3 = !\mathsf{open}\, n.\, P_2$. *A characteristic formula for* P_1 *is easy to define since the continuation term* $n[\mathbf{0}]$ *has no reducts. Hence the formula* $[\![\mathsf{open}\, n]\!] n[0]$, *using a formula for* necessity, *satisfies the conditions of Lemma A.3.13, and a characteristic formula for* P_1 *is*

$$\mathcal{F}_1 \stackrel{\text{def}}{=} ![\![\mathsf{open}\, n]\!] n[0]\,.$$

In order to define a characteristic formula for P_3*, we first look for a characteristic formula for* P_2*. We can set*

$$\mathcal{F}_2 \stackrel{\text{def}}{=} [\![\mathsf{open}\, n]\!].\, 0 \;\mid\; n[0]\,.$$

However, in P_3 *the continuation process (*P_2*) is not static, as it may reduce to* $\mathbf{0}$*. Hence* $[\![\mathsf{open}\, n]\!].\mathcal{F}_2$ *does not satisfy the conditions of Lemma A.3.13 (and this formula is not a charateristic formula for* $\mathsf{open}\, n.\, P_2$*, although* $\mathcal{F}_2$ *is for* P_2*), so that we need to add the possibility to reduce to* $\mathbf{0}$*, yielding to the formula* $[\![\mathsf{open}\, n]\!].\, (\mathcal{F}_2 \;\vee\; 0)$*. But then we also accept the term* $\mathsf{open}\, n.\, \mathbf{0}$*, which shows why we are led to add a* possibility *condition in the formula.*

A characteristic formula for P_3 *is finally the following:*

$$\mathcal{F}_3 \stackrel{\text{def}}{=} \mathit{Rep}_{\mathsf{open}\, n}(\langle\langle \mathsf{open}\, n\rangle\rangle.\, \mathcal{F}_2 \;\wedge\; [\![\mathsf{open}\, n]\!].\, (\mathcal{F}_2 \;\vee\; 0))\,.$$

We see on this example that characterising the continuation of a process starting with a capability or an input requires to enumerate also all the possible reducts after consuming the topmost constructor. This is not always effective, and we are thus led to suppose that it is, which leads us to the definition of the following subclass of MA processes:

Definition A.4.10 (Subcalculus MA_{IF}) *We call* image-finite processes *processes* P *such that in any subterm of the form* $\mathsf{cap}.\, P'$ *(resp.* $(x)P'$*), the process* P' *admit only finitely many distinguishable reducts under* $\stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow}$ *(resp.* $P'\{^n\!/\!_x\}$ *admit only finitely many distinguishable reducts under* $\Rightarrow$ *for some* $n \notin \mathrm{fn}(P)$*).*

MA_{IF} *is the set of image-finite MA processes.*

Using the notations of this definition, image-finiteness amounts to finiteness of the following sets of processes (that are quotiented w.r.t. $\equiv$):

$$\{P'' \;:\; P' \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow} P''\}_{/\approx_{int}} \quad \text{and} \quad \{P'' \;:\; P'\{^n\!/\!_x\} \;\Rightarrow\; P''\}_{/\approx_{int}}\,.$$

For example, process $\mathsf{in}\, n.\,!(n[\mathbf{0}] \mid \mathsf{open}\, n.\,\mathbf{0})$ is in MA_{IF}, but not $\mathsf{in}\, n.\,!(n[\mathbf{0}] \mid \mathsf{open}\, n.\, a[\mathbf{0}])$

To construct a characteristic formula $\mathcal{F}_P$ of a closed MA_{IF} process P, we can suppose (up to $\equiv_E$) that replication only appears above single terms and that P is eta normalised. We then define the characteristic formula $\mathcal{F}_P$ of P by induction using the order of Definition A.4.7. The defining formulas are given in Table A.6 (this defines a valid induction by Lemma B.2.2).

$$
\begin{array}{lll}
\mathcal{F}_{\mathbf{0}} & \stackrel{\text{def}}{=} & 0 \\
\mathcal{F}_{P|Q} & \stackrel{\text{def}}{=} & \mathcal{F}_P \;\mid\; \mathcal{F}_Q \\
\mathcal{F}_{n[P]} & \stackrel{\text{def}}{=} & n[\mathcal{F}_P] \\
\mathcal{F}_{\mathsf{cap}.P} & \stackrel{\text{def}}{=} & \langle\langle \mathsf{cap} \rangle\rangle.\,\mathcal{F}_P \;\wedge\; [\![\mathsf{cap}]\!].\bigvee_{\{P',\; P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P'\}_{/\approx_{int}}} \mathcal{F}_{P'} \\
\mathcal{F}_{!n[P]} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{n[]}(\mathcal{F}_P) \\
\mathcal{F}_{!\mathsf{cap}.P} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{\mathsf{cap}}(\mathcal{F}_{\mathsf{cap}.P}) \\
\mathcal{F}_{\langle n \rangle} & \stackrel{\text{def}}{=} & \text{cf. Lemma A.2.15} \\
\mathcal{F}_{(x)P} & \stackrel{\text{def}}{=} & \exists x.\; \neg \copyright x \;\wedge\; \langle ?x \rangle.\,\mathcal{F}_P \;\wedge \qquad\qquad (n_x \notin \mathrm{fn}(P)) \\
 & & [?x]\Big((\mathcal{F}_{\langle x \rangle} \mid \mathtt{1input}) \;\vee\; \bigvee_{\{P':\; P\{n_x/x\} \Rightarrow P'\}_{/\approx_{int}}} \mathcal{F}_{P'\{x/n_x\}} \Big) \\
\mathcal{F}_{!\langle n \rangle} & \stackrel{\text{def}}{=} & \text{cf. Table A.5} \\
\mathcal{F}_{!(x)P} & \stackrel{\text{def}}{=} & \mathsf{Rep}_{input}(\mathcal{F}_{(x)P})
\end{array}
$$

Table A.6: Characteristic formulas in MA_{IF}

Theorem A.4.11 (Characteristic formulas for MA_{IF}) *For any term P, define $\mathcal{F}_P$ according to Table A.6. Then*

$$Q \models \mathcal{F}_P \qquad \textit{iff} \qquad P \;\approx_{int}\; Q.$$

Do note that in formula $\mathcal{F}_{(x)P}$ of Table A.6, we reason like if we were defining characteristic formulas for processes that have free variables. This should be understood as involving an implicit replacement of these free variables by appropriately chosen fresh names (see also the discussion just before Lemma A.2.15), so that we only manipulate closed processes.

Proof:of Theorem A.4.11 The proof is by induction, using the order of Definition A.4.7.

- $\mathcal{F}_{\mathbf{0}}$ characterises $\mathbf{0}$: this holds by Prop. A.4.8.
- $\mathcal{F}_{\langle n \rangle}$ characterises $\langle n \rangle$ and $\mathcal{F}_{!\langle n \rangle}$ characterises $!\langle n \rangle$: by Lemma A.2.15, Lemma A.3.13 and Prop. A.4.8.
- if $\mathcal{F}_P$ characterises P, then $\mathcal{F}_{n[P]}$ characterises $n[P]$: by Prop. A.4.8.
- if $\mathcal{F}_{P_1}$ characterises P_1 and $\mathcal{F}_{P_2}$ characterises P_2, then $\mathcal{F}_{P_1|P_2}$ characterises $P_1 \mid P_2$: by Prop. A.4.8.
- Suppose now that for every P' such that $\mathsf{ds}(()P') \leq \mathsf{ds}(()P)$, $\mathcal{F}_{P'}$ is a characteristic formula for P'. We then have:

– $\mathcal{F}_{\mathsf{cap}.P}$ characterises $\mathsf{cap}.\,P$.

  By Lemma A.2.9, $\mathsf{ds}(()P') \leq \mathsf{ds}(()P)$ for any P' such that $P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P'$, so $\mathcal{F}_{P'}$ is a characteristic formula for such processes. We examine each of the two implications. In one direction, $P \models \langle\mathsf{cap}\rangle.\,\mathcal{F}_P$, and by Lemma A.2.9, $P \models [\mathsf{cap}].\,\bigvee_{\{P',\ P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P'\}_{/\approx_{int}}} \mathcal{F}_{P'}$, so $\mathsf{cap}.\,P \models \mathcal{F}_{\mathsf{cap}.P}$. Conversely, if $Q \models \mathcal{F}_P$, then from $Q \models \langle\mathsf{cap}\rangle.\,\mathcal{F}_P$ we deduce the existence of Q', Q'' such that $Q \equiv \mathsf{cap}.\,Q'$, $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} Q''$, and $Q'' \models \mathcal{F}_P$. Moreover, from $Q \models [\mathsf{cap}].\,\bigvee_{\{P',\ P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P'\}_{/\approx_{int}}} \mathcal{F}_{P'}$, we deduce that there is P' such that $P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P'$ and $Q' \models F_{P'}$, so $Q' \approx_{int} P'$, and by Prop. A.4.8, $Q \approx_{int} \mathsf{cap}.\,P$.

– $\mathcal{F}_{(x)P}$ characterises $(x)P$.

  We may assume without loss of generality that $(x)P$ is eta normalised, since logical equivalence is up to $\equiv_E$ A.4.3. We pick n_0 fresh for P. We can apply the induction hypothesis for $P\{n_0/x\}$ and for all of its reducts P'. The right to left direction follows from Lemma A.2.16.

  For the other direction, let Q be such that $Q \models \mathcal{F}_{(x)P}$. Up to $\equiv_E$, we may assume that Q is eta normalised. Let n_0 be a name that is used to satisfy formula $\mathcal{F}_{(x)P}$. Then $n_0 \notin \mathrm{fn}(Q)$, and there are Q', Q'' such that $Q \equiv (x)Q'$, $\langle n_0\rangle \mid (x)Q' \Rightarrow Q''$, and $Q'' \models \mathcal{F}_{P\{n_0/x\}}$, that is, by hypothesis, $Q'' \approx_{int} P\{n_0/x\}$. Moreover, since Q is eta normalised, $Q'\{n_0/x\}$ is not of the form $\langle n_0\rangle \mid (x)R$, and hence does not satisfy $(\mathcal{F}_{\langle x\rangle} \mid \mathtt{1input})$, so there is some P' such that $P\{n_0/x\} \Rightarrow P'$ and $Q'\{n_0/x\} \models \mathcal{F}_{P'}$, that is, by induction, $Q'\{n_0/x\} \approx_{int} P'$. Using Proposition A.4.8, we deduce $Q \approx_{int} (x)P$.

– $\mathcal{F}_{!\mathsf{cap}.P}$ characterises $!\mathsf{cap}.\,P$ and $\mathcal{F}_{!(x)P}$ characterises $!(x)P$: these results follow from the replication case in Prop. A.4.8 and from Lemma A.3.13. In particular, the requirements in terms of sequential (or depth) selectiveness, and cap (or n, input) expressiveness are satisfied because the formulas we are using in our constructions are characteristic formulas, which, by induction, satisfy such requirements.

$\square$

Corollary A.4.12 *On the subcalculus MA_{IF}, $\approx_{int} = =_L$, and in particular,*
$$Q \models \mathcal{F}_P \quad \textit{iff} \quad P =_L Q.$$

This strong expressiveness result says that, rather remarkably, we can express the whole descriptive power of the logic in a single formula.

A.5 Extensions of the calculus

In this section, we study extensions of MA with different forms of communication: we first examine the possibility to emit capabilities (and not only names) in messages, and then consider synchronous communication. We only show how to capture the modifications brought to the language, without porting all the constructions seen in the previous sections. We however believe that our approach would go through without any major modification.

We start by remarking that Lemmas A.2.5,A.2.8,A.2.7 about the interpretation of formulas $\langle \mathsf{cap} \rangle. \mathcal{A}$ hold in the extensions we consider, since their proofs are insensitive to the presence of communication in the calculus.

A.5.1 Capabilities in messages

In the original MA calculus [CG98], messages can also carry *paths of capabilities*. To accommodate this in the grammar of Table A.1, all occurrences of η are replaced by M, and the path productions

$$M ::= \mathsf{cap} \;\mid\; M_1. M_2 \;\mid\; \epsilon\,,$$

are added to those for expressions, where ϵ stands for the empty path. Thus a capability can be a path, such as $\mathsf{open}\, n.\, \mathsf{in}\, m.\, \mathsf{open}\, h$. Also, the rules

$$\epsilon. P \;\equiv\; P \qquad\qquad (M_1. M_2). P \;\equiv\; M_1. M_2. P$$

are added to those of $\equiv$. Since messages can now carry names or capabilities, a type system is introduced [CG99] to avoid run-time errors. We shall assume that all processes are well-typed (according to the basic Ambient types), which means in particular that in the interpretation of a formula of the form $\mathcal{A} \triangleright \mathcal{B}$, processes that are added in parallel are of the right type.

Our main focus will be on the characterisation of these new forms of messages. For this, we need a formula `TestCap`, the analogous of the formula `TestComm` of Section A.2.3, satisfied by all abstractions that are eta-congruent to $(x)\, m[x. \mathbf{0}]$, where m is some fixed name.

We also need a formula $(\!| M |\!)$, for any closed capability M, that identifies those processes that are structurally congruent to $M. \mathbf{0}$. We first discuss an example, namely the formula $(\!| \mathsf{in}\, n. \mathsf{open}\, m |\!)$. For this, $\langle \mathsf{in}\ n \rangle. \langle \mathsf{open}\ m \rangle. 0$ is not enough: this formula is satisfied by $\mathsf{in}\, n. \mathsf{open}\, m. \mathbf{0}$ but also, for instance, by processes such as $\mathsf{in}\, n. (\langle M \rangle \mid (x)\ \mathsf{open}\, m)$, which has some additional I/O, or $\mathsf{in}\, n. \mathsf{out}\, n. \mathsf{in}\, n. \mathsf{open}\, m. \mathbf{0}$, which stutters. A formula $\mathcal{F}$ for $(\!| \mathsf{in}\, n. \mathsf{open}\, m |\!)$ could thus be (the actual definition of $(\!| \mathsf{in}\, n. \mathsf{open}\, m |\!)$ will be different; the formula below is easier to read and semantically equivalent):

$$\begin{aligned} \mathcal{F} \;\stackrel{\text{def}}{=}\; & \langle \mathsf{in}\ n \rangle. \langle \mathsf{open}\ m \rangle. 0 \\ & \wedge \neg \langle \mathsf{in}\ n \rangle. \neg \mathtt{1comp} \\ & \wedge \neg \langle \mathsf{in}\, n \rangle. \langle \mathsf{out}\, n \rangle. \top \\ & \wedge \neg \langle \mathsf{in}\ n \rangle. \langle \mathsf{open}\ m \rangle. \neg 0 \end{aligned}$$

In the definition of $\mathcal{F}$, the second, third and fourth $\wedge$-components take care of the problems with I/O and stuttering mentioned above.

Here is the complete definition of $(\!|M|\!)$ for any path M:

$$
\begin{array}{rcl}
(\!|\mathsf{open}\, n.\, M|\!) & \stackrel{\text{def}}{=} & \langle \mathsf{open}\ n \rangle.\, (\!|M|\!) \\
 & & \wedge\ \neg\langle \mathsf{open}\ n \rangle.\, (\neg\mathtt{1comp} \vee \mathtt{1amb}) \\
(\!|\mathsf{out}\, n.\, M|\!) & \stackrel{\text{def}}{=} & \langle \mathsf{out}\ n \rangle.\, (\!|M|\!) \\
 & & \wedge\ \neg\langle \mathsf{out}\ n \rangle.\, (\neg\mathtt{1comp} \vee \mathtt{1amb}) \\
 & & \wedge\ \neg\langle \mathsf{out}\ n \rangle.\, \langle \mathsf{in}\ n \rangle.\, \langle \mathsf{out}\ n \rangle.\, (\!|M|\!) \\
(\!|\mathsf{in}\, n.\, M|\!) & \stackrel{\text{def}}{=} & \langle \mathsf{in}\ n \rangle.\, (\!|M|\!) \\
 & & \wedge\ \neg\langle \mathsf{in}\ n \rangle.\, (\neg\mathtt{1comp} \vee \mathtt{1amb}) \\
 & & \wedge\ \neg\langle \mathsf{in}\ n \rangle.\, \langle \mathsf{out}\ n \rangle.\, \langle \mathsf{in}\ n \rangle.\, (\!|M|\!) \\
(\!|\epsilon.\, M|\!) & \stackrel{\text{def}}{=} & (\!|M|\!) \\
(\!|\mathbf{0}|\!) & \stackrel{\text{def}}{=} & 0
\end{array}
$$

In the definition of $(\!|M|\!)$, subformula $\neg\mathtt{1comp}\vee\mathtt{1amb}$ is used to control process reductions, see Lemma A.5.1.

Lemma A.5.1 *Suppose $P \longrightarrow P'$. Then $P \models \neg\mathtt{1comp} \vee \mathtt{1amb}$.*

We now define `TestCap`:

$$
\begin{array}{rcl}
\mathtt{TestCap} & \stackrel{\text{def}}{=} & \mathtt{1comm} \\
 & & \wedge\ \mathtt{1comm} \rhd \Box(\mathtt{2comm} \vee m[\mathtt{1comp}]) \\
 & & \wedge\ \mathtt{1comm} \blacktriangleright \Diamond\, m[(\!|\mathsf{in}\, n|\!)] \\
 & & \wedge\ \mathtt{1comm} \blacktriangleright \Diamond\, m[0]
\end{array}
$$

where n and m are different names.

The correctness of this definition is proved along the lines of that of `TestComm`. The formula $\mathcal{F}_{\langle M \rangle}$, where M is any closed capability, is then

$$
\mathcal{F}_{\langle M \rangle} \stackrel{\text{def}}{=} \mathtt{1comm} \wedge \Big(\mathtt{TestCap} \rhd \Diamond\, m[(\!|M|\!)]\Big)
$$

A.5.2 Synchronous Ambients

Since the modal logic does not talk about the I/O primitives, it is interesting to examine variations of these primitives, to see the effect on the equality induced by the logic. In MA communication is asynchronous: since a message has no continuation, no process is blocked until the message is consumed. The most natural variation consists in making communication *synchronous*. For this the production $\langle \eta \rangle$ for messages in the grammar of MA in Table A.1 is replaced by the production $\langle \eta \rangle.\, P$. Reduction rule `Red-Com` becomes:

$$
\frac{}{\langle M \rangle.\, Q \mid (x)\, P \longrightarrow Q \mid P\{^{M}\!/\!_{x}\}}\ \mathtt{Red\text{-}Com}
$$

The communication act liberates, at the same time, both the continuation P of the abstraction *and* the continuation Q of the message. We write $\mathrm{MA}^{\mathrm{sync}}$ for the resulting synchronous calculus.

Synchrony leads to some important modifications in the assertions and in the proofs of the results in the paper. In $\mathrm{MA}^{\mathrm{sync}}$, the eta law fails in the sense that the logic can separate eta equivalent terms (see Def. A.4.2). Indeed, we will define a formula $\langle\langle n\rangle\rangle.\,\mathcal{A}$ whose models are processes $\langle n\rangle.\,P$ with $P \Rightarrow \models\mathcal{A}$. Then, returning to the eta law, formula $\mathtt{1input} \wedge (\langle\langle n\rangle\rangle.\,n[0]) \blacktriangleright \Box\neg\mathtt{3}$ is satisfied by $(x)\big(\langle x\rangle \mid (y)\mathbf{0}\big)$, and not by $(x)\mathbf{0}$.

Remark A.5.2 *An important change induced by this remark is that logical equivalence coincides with structural congruence in* MA^{sync} *on finite processes (see appendix B).*

We will focus now on the characterisation of this new form of communication. In asynchronous MA, our separation of messages from abstractions exploited their asymmetry: abstractions, but not messages, have a continuation. In the synchronous case the asymmetry disappears, therefore we have to use a different route for the proof, which makes it a bit more involved.

Again, the most delicate point is to find a replacement for the formula `TestComm`. We only quickly sketch how the new definition is obtained.

- We first define a formula, `OnlyCom`, that is satisfied only by abstractions $(x)\,P$ and messages $\langle M\rangle.\,P$ in which capability prefixes and ambients do not appear in the continuation P and, moreover, no subterm of P contains more than two non-trivial parallel components:

$$\boxed{\begin{array}{ll}\mathtt{OnlyCom} \stackrel{\mathrm{def}}{=} & \mathtt{1comm} \blacktriangleright (\ \ \Box(\mathtt{2comm} \vee 0) \\ & \qquad\qquad\quad \wedge \Diamond\ 0)\end{array}}$$

- Using `OnlyCom` we define a formula, `ComAmb`, that is satisfied only by processes defined as those that satisfy `OnlyCom` except that the innermost operator is an ambient $\eta[y[\mathbf{0}]]$:

$$\boxed{\begin{array}{l}\mathtt{ComAmb} \stackrel{\mathrm{def}}{=} \\ \quad \exists x.\Big(\ \mathtt{OnlyCom} \blacktriangleright \Big(\ \Box(\ \mathtt{2comm}\ \vee\ x[n[0]]) \\ \qquad\qquad\qquad\qquad\qquad \wedge \Diamond\ x[n[0]]\Big) \\ \qquad\quad \wedge\ \mathtt{OnlyCom} \blacktriangleright \Diamond\ x[m[0]]\Big) \\ \text{where } n \text{ and } m \text{ are different names.}\end{array}}$$

- We then define a formula that characterises the abstraction $(x)\,h[x[\mathbf{0}]]$; we write `3comm` for `1comm` $\mid$ `1comm` $\mid$ `1comm`:

$$\boxed{\begin{array}{ll}\mathtt{Imm}\,h \stackrel{\mathrm{def}}{=} & \mathtt{ComAmb} \\ & \wedge\ \mathtt{OnlyCom} \rhd (\Box\neg\mathtt{1comm}\ \wedge \Box\neg\mathtt{3comm}) \\ & \wedge\ \mathtt{OnlyCom} \blacktriangleright \Diamond\ h[n[\top]] \\ & \wedge\ \mathtt{OnlyCom} \blacktriangleright \Diamond\ h[m[\top]]\,, \\ \multicolumn{2}{l}{\text{where } n \text{ and } m \text{ are different names.}}\end{array}}$$

Roughly, the first $\wedge$-component implies that a process that satisfies $\mathtt{Imm}\, h$ has an abstraction or a message as its outermost operator, and an ambient $\eta[x[\mathbf{0}]]$ as the innermost. The second $\wedge$-component, call it $\mathcal{F}$, ensures that the process does not have any other operators; that is, the ambient $\eta[x[\mathbf{0}]]$ is reached immediately after the initial communication. For instance, the process $R \stackrel{\text{def}}{=} \langle M\rangle.\,(x)\ h[x[\mathbf{0}]]$ does not satisfy $\mathcal{F}$ because $R \mid (x)\,\mathbf{0} \Longrightarrow (x)\ h[x[\mathbf{0}]]$ and $(x)\ h[x[\mathbf{0}]]$ satisfies $\mathtt{1comm}$. Finally, the third and fourth $\wedge$-components rule out the messages and the abstraction $(x)\ x[x[\mathbf{0}]]$.

To interpret formula $\mathtt{OnlyCom}$, we introduce the following grammars:

$$
\begin{array}{rcl}
H & ::= & \langle\eta\rangle.\,\mathbf{0} \;\mid\; (x)\,\mathbf{0} \;\mid\; \langle\eta\rangle.\,(H \mid H) \;\mid\; (x)\,(H \mid H) \;\mid\; \langle\eta\rangle.\,H \;\mid\; (x)\,H \\
K & ::= & \langle\eta'\rangle.\,\eta[y[0]] \;\mid\; (x)\,\eta[y[0]] \;\mid\; \langle\eta'\rangle.\,(H \mid K) \;\mid\; (x)\,(H \mid K) \;\mid\; \langle\eta'\rangle K \;\mid\; (x)\,K \\
H^* & ::= & H \;\mid\; \mathbf{0} \\
K^* & ::= & \langle\eta'\rangle.\,\eta[\eta''[0]] \;\mid\; (x)\,\eta[\eta''[0]] \\
 & & \mid\; \langle\eta'\rangle.\,(H \mid K) \;\mid\; (x)\,(H \mid K) \\
 & & \mid\; \langle\eta'\rangle K \;\mid\; (x)\,K \;\mid\; \eta[\eta'[\mathbf{0}]]
\end{array}
$$

We write

- $\mathcal{GH}$ for the set of terms described by H,
- $\mathcal{GK}$ for those described by K,
- $\mathcal{GH}^*$ for those described by H^*, and
- $\mathcal{GK}^*$ for those described by K^*.

(The grammar for K^*, w.r.t. that for K, has the additional production for $\eta[\eta'[\mathbf{0}]]$, and has $\eta[\eta'[\mathbf{0}]]$ in place of $\eta[y[\mathbf{0}]]$ in the other productions.)

Lemma A.5.3 *Suppose P is a MA$^{\text{sync}}$ process. $P\{{}^{n}\!/\!{}_{x}\} \in \mathcal{GH}$ iff $P \in \mathcal{GH}$.*

Lemma A.5.4 *Suppose P is a MA$^{\text{sync}}$ process and $P \models \mathtt{OnlyCom}$. Then $P \equiv P'$ for some $P' \in \mathcal{GH}$.*

Proof: Suppose $P_1 \models \mathtt{OnlyCom}$. Then there is a process P_2 with $P_2 \models \mathtt{1comm}$ such that

$$P_1 \mid P_2 \models (\Box(\mathtt{2comm} \vee 0)\ \wedge \Diamond\ 0).$$

In particular, it holds that $P_1 \mid P_2 \models \mathtt{2comm}$, hence $P_1 \models \mathtt{1comm}$.

We show that if Q_1, Q_2 are processes that satisfy $\mathtt{1comm}$ and such that

$$Q_1 \mid Q_2 \models (\Box(\mathtt{2comm} \vee 0)\ \wedge \Diamond\ 0),$$

then $Q_1, Q_2 \in \mathcal{GH}$.

The proof is by induction on the maximal depth of Q_1, Q_2. The case when this depth is 1 is easy. If this depth is greater than 1, then $Q_1 \mid Q_2 \longrightarrow Q_1' \mid Q_2'$, using the $\mathtt{com}$ rule, where Q_i is the continuation of Q_i. We have three cases:

- $Q_1' \equiv \mathbf{0}$, $Q_2' \equiv R_1 \mid R_2$ for some non-trivial R_1, R_2;
- the symmetric case;

- none of Q'_1 and Q'_2 is structurally congruent to $\mathbf{0}$.

In the first two cases, we deduce that R_1, R_2 satisfy $\mathtt{1comm}$, and then use induction to infer $R_1, R_2 \in \mathcal{GH}$. Then using the first 4 productions of the grammar, and Lemma A.5.3, $Q_1, Q_2 \in \mathcal{GH}$. In the third case, use induction to infer $Q'_1, Q'_2 \in \mathcal{GH}$. Hence also $Q_1, Q_2 \in \mathcal{GH}$, using the last 2 productions of the grammar and Lemma A.5.3. $\Box$

Lemma A.5.5 *Suppose P is a MA$^{\text{sync}}$ process, and $P \models \Box(\mathtt{2comm} \vee h[n[0]]) \wedge \Diamond\, h[n[0]]$, where h, n are any names. Then $P \equiv P_1 \mid P_2$ with $P_1 \in \mathcal{GH}$ and $P_2 \in \mathcal{GK}^*$.*

Proof: By induction on the size of P, where the size is the number of operators in P. The size cannot be 0 or 1. If the size is 2 then $P = h[n[0]]$, and $P \equiv \mathbf{0} \mid h[n[0]]$, hence the assertion of the lemma, for $P_1 \stackrel{\text{def}}{=} \mathbf{0}$.

Suppose the size if greater than 2. Call

$$\mathcal{F} \stackrel{\text{def}}{=} \Box(\mathtt{2comm} \vee h[n[0]]) \wedge \Diamond\, h[n[0]]$$

Then

$$P \equiv P_1 \mid P_2 \quad \text{where, for } i = 1, 2, \text{we have} \quad P_i \models \mathtt{1comm}$$

Since P must reduce,

$$P_1 \mid P_2 \equiv \langle m \rangle.\, Q_1 \mid (x)\, Q_2$$

and

$$Q_1 \mid Q_2\{{}^{m}\!/\!{}_{x}\} \models \mathcal{F}\,.$$

The size of $Q_1 \mid Q_2\{{}^{m}\!/\!{}_{x}\}$ is smaller.

It can be that Q_1 or Q_2 are $\mathbf{0}$, or none is $\mathbf{0}$ (they cannot both be $\mathbf{0}$). In both cases we can conclude by reffering to the appropriate grammar productions and by using the inductive hypothesis. $\Box$

Define now:

$$\mathtt{ComAmb} \stackrel{\text{def}}{=} \exists x. \Big(\mathtt{OnlyCom} \blacktriangleright \Big(\Box(\mathtt{2comm} \vee x[n[0]]) \wedge \Diamond\, x[n[0]]\Big) \wedge \mathtt{OnlyCom} \blacktriangleright \Diamond\, x[m[0]]\Big)$$

where n and m are different names.

Lemma A.5.6 *Suppose P is a MA$^{\text{sync}}$ process, and $P \models \mathtt{ComAmb}$. Then $P \equiv P'$ for some $P' \in \mathcal{GK}$.*

Proof: Suppose $P_1 \models \mathtt{ComAmb}$. Then there is a process P_2 and some h with $P_2 \models \mathtt{OnlyCom}$ such that $P_1 \mid P_2 \models \Box(\mathtt{2comm} \vee h[n[0]]) \wedge \Diamond\, h[n[0]]$. By Lemma A.5.4, $P_2 \equiv\in \mathcal{GH}$. By Lemma A.5.5, $P_1 \in \mathcal{GK}^*$. Moreover, since $P_1 \models \mathtt{1comm}$, it holds that $P_1 \not\equiv h[n[\mathbf{0}]]$; from this and $P_1 \models \mathtt{OnlyCom} \blacktriangleright \Diamond\, h[m[0]]$, we deduce $P_1 \in \mathcal{GK}$.

$\Box$

Define

$$\begin{array}{ll} \mathtt{Imm}\, h \stackrel{\text{def}}{=} & \mathtt{ComAmb} \wedge \\ & \mathtt{OnlyCom} \rhd (\Box\neg \mathtt{1comm} \wedge \Box\neg \mathtt{3comm}) \\ & \mathtt{OnlyCom} \blacktriangleright \Diamond\, h[n[\top]] \\ & \mathtt{OnlyCom} \blacktriangleright \Diamond\, h[m[\top]] \\ \text{where } m \neq n. & \end{array}$$

Lemma A.5.7 *Suppose* $P \models \mathtt{Imm}\, h$. *Then* $P \equiv (x)\; h[x[\mathbf{0}]]$.

Proof: By Lemma A.5.6, $P \equiv P' \in \mathcal{GK}$. One then shows that $(x)\; h[x[\mathbf{0}]]$ satisfies $\mathtt{Imm}\, h$, whereas the other terms in $\mathcal{GK}$ do not (chosing the appropriate $\mathtt{OnlyCom}$). $\Box$

Now we can define the formula:

$$\begin{array}{ll} \langle\langle n\rangle\rangle.\, \mathcal{A} \stackrel{\text{def}}{=} & \mathtt{1comm} \\ & \wedge\, \forall x.\, (\mathtt{Imm}\, x \rhd \Diamond\, (x[n[0]] \mid \mathcal{A})) \end{array}$$

Lemma A.5.8 *Suppose* P *is a* MA^{sync} *process. It holds that* $P \models \langle\langle n\rangle\rangle.\, \mathcal{A}$ *iff* $P \equiv \langle n\rangle.\, Q$ *and* $Q \Longrightarrow Q'$ *and* $Q' \models \mathcal{A}$.

Proof: Take h fresh. Then by Lemma A.5.7,

$$P \mid (x)\; h[x[\mathbf{0}]] \models \Diamond\, (h[x[\mathbf{0}]] \mid \mathcal{A})\, .$$

From this, and $P \models \mathtt{1comm}$, we deduce $P \equiv \langle m\rangle.\, P'$, for some m, P'. We also deduce that

$$P' \mid h[m[\mathbf{0}]] \models \Diamond\, (h[m[\mathbf{0}]] \mid \mathcal{A})\, .$$

Since h is fresh, P' cannot interact with h. Hence $m = n$, and moreover $P' \Longrightarrow\models \mathcal{A}$. $\Box$

Lemma A.5.9 *Suppose* P *is a* MA^{sync} *process. It holds that* $P \models \langle ?n\rangle.\, A$ *iff* $P \equiv (x)\; P'$ *and* $P'\{^{M}\!/\!_{x}\} \Longrightarrow P'' \models A$.

Appendix B

Separability in the Ambient Logic

This appendix presents the characterisation of the logical equivalence ($=_L$) on the Ambient calculus, as presented in chapter 7 and in [HLS02]. The calculus is however richer, namely it includes asynchronous communications of names as in the original presentation of the calculus [CG98]. We moreover present the proof of undecidability of $=_L$ sketched in chapter 9.

First we introduce the intensional bisimilarity $\approx_{int}$, aiming to give a coinductive operational characterisation of the logical equivalence; we prove its congruence and its soundness for $=_L$ (section B.1). The we turn this characterisation in an inductive one and use this fact to derive a proof of completeness without any image-finiteness hypothesis (section B.2). We then give a more precise descprition of $=_L$ comparing it to the extensional and intensional equivalences, and we prove intensionality for the calculus MA_{IF}^{syn}. We conclude by the proof of the undecidability of $=_L$ (section B.4).

B.1 Intensional bisimilarity

In order to be able to carry out our programme for $=_L$, as discussed in the introduction, we look for a co-inductive characterisation of this relation, as a form of labelled bisimilarity. Before introducing the bisimilarity relation, we define labelled transitions on MA, including what we call relations of *stuttering*.

B.1.1 Definitions

Labelled transitions and stuttering

We recall first the definition A.1.5. Let P be a closed process. We write:

- $P \overset{\mathsf{cap}}{\to} P'$, where cap is a capability, if $P \equiv \mathsf{cap}.\, P_1 \mid P_2$ and $P' = P_1 \mid P_2$.
- $P \overset{!n}{\to} P'$ if $P \equiv \langle n \rangle \mid P'$.
- $P \overset{?n}{\to} P'$ if $P \equiv (m)\, P_1 \mid P_2$ and $P' \equiv P_1\{^n\!/\!_m\} \mid P_2$.

- $P \stackrel{\mu}{\Longrightarrow} P'$, where μ is one of the above labels, if $P \Longrightarrow \stackrel{\mu}{\to} \Longrightarrow P'$ (where $\Longrightarrow \stackrel{\mu}{\to} \Longrightarrow$ is relation composition).
- **(stuttering)** $P \xRightarrow{(M_1,M_2)^*} P'$ if there is $i \geq 1$ and processes $P_1, \ldots, P_i$ with $P = P_1$ and $P' = P_i$ such that $P_r \stackrel{M_1}{\Longrightarrow} \stackrel{M_2}{\Longrightarrow} P_{r+1}$ for all $1 \leq r < i$.
- Finally, $\stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow}$ is a convenient notation for compacting statements involving capability transitions. $\stackrel{\langle \mathsf{in\,n} \rangle}{\Rightarrow}$ is $\xRightarrow{(\mathsf{out}\,n,\mathsf{in}\,n)^*}$; similarly $\stackrel{\langle \mathsf{out\,n} \rangle}{\Rightarrow}$ is $\xRightarrow{(\mathsf{in}\,n,\mathsf{out}\,n)^*}$; and $\stackrel{\langle \mathsf{open\,n} \rangle}{\Rightarrow}$ is $\Rightarrow$.

Example B.1.1 (Stuttering Loop) *Consider the processes*

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & !\mathsf{open}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}] \mid n[\mathbf{0}] \\ Q & \stackrel{\text{def}}{=} & !\mathsf{open}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,\mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}] \mid \mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}]\,. \end{array}$$

We have the following loop, modulo stuttering:

$$P \xRightarrow{(\mathsf{in}\,n,\mathsf{out}\,n)^*} Q \xRightarrow{(\mathsf{in}\,n,\mathsf{out}\,n)^*} P\,.$$

The existence of such pairs of processes that reduce one to each other modulo stuttering will play an important role in the axiomatization of $=_L$. We call such a situation a loop.

The following equation between processes, that we call the *eta law*, can also be seen as a form of stuttering (in communication, as opposed to stuttering in movements). We shall see in Section B.3 that the logic is insensitive to this form of stuttering.

Definition B.1.2 (Eta law) *The* eta law *is given by the following equation:*

$$(x)\,((x)\,P \mid \langle x \rangle) = (x)\,P\,.$$

A process P is eta-normalised *if there is no context C and process Q such that $P \equiv C[(x)\,(\langle x \rangle \mid (x)\,Q)]$ (that is, modulo $\equiv$, P does not contain any subcomponent of the form $(x)\,(\langle x \rangle \mid (x)\,Q)$).*

Intensional relation, $\approx_{int}$

We present here our main labelled bisimilarity, written $\approx_{int}$, which we call intensional bisimilarity, and that will be used to capture $=_L$'s separative power.

The definition of $\approx_{int}$ is rather robust, in the sense that it is based on the observations made available by the logic either using built-in operators or through derived formulas for capabilities (see below), and hence does not depend too much on the syntactical operators of the underlying calculus. This robustness makes it easy for instance to derive that $\approx_{int}$ is an equivalence relation.

By contrast, some important properties, like congruence, are not immediate. We shall introduce below another relation, whose definition follows a more syntactical approach, and that will make the proof of congruence easier.

Definition B.1.3 Intensional bisimilarity *is the largest symmetric relation $\approx_{int}$ on processes such that $P \approx_{int} Q$ implies:*

1. *If* $P \equiv P_1 \mid P_2$ *then there are* Q_1, Q_2 *such that* $Q \equiv Q_1 \mid Q_2$ *and* $P_i \approx_{int} Q_i$, *for* $i = 1, 2$.

2. *If* $P \equiv \mathbf{0}$ *then* $Q \equiv \mathbf{0}$.

3. *If* $P \longrightarrow P'$ *then there is* Q' *such that* $Q \Longrightarrow Q'$ *and* $P' \approx_{int} Q'$.

4. *If* $P \xrightarrow{\mathsf{in}\,n} P'$ *then there is* Q' *such that* $Q \xRightarrow{\mathsf{in}\,n} \xRightarrow{(\mathsf{out}\,n, \mathsf{in}\,n)^*} Q'$ *and* $P' \approx_{int} Q'$.

5. *If* $P \xrightarrow{\mathsf{out}\,n} P'$ *then there is* Q' *such that* $Q \xRightarrow{\mathsf{out}\,n} \xRightarrow{(\mathsf{in}\,n, \mathsf{out}\,n)^*} Q'$ *and* $P' \approx_{int} Q'$.

6. *If* $P \xrightarrow{\mathsf{open}\,n} P'$ *then there is* Q' *such that* $Q \xRightarrow{\mathsf{open}\,n} Q'$ *and* $P' \approx_{int} Q'$.

7. *If* $P \xrightarrow{!n} P'$ *then there is* Q' *such that* $Q \xRightarrow{!n} Q'$ *and* $P' \approx_{int} Q'$.

8. *If* $P \xrightarrow{?n} P'$ *then there is* Q' *such that* $Q \mid \langle n \rangle \Longrightarrow Q'$ *and* $P' \approx_{int} Q'$.

9. *If* $P \equiv n[P']$ *then there is* Q' *such that* $Q \equiv n[Q']$ *and* $P' \approx_{int} Q'$.

The definition of $\approx_{int}$ has (at least) two intensional clauses, namely (1) and (2), which allow us to observe parallel compositions and the terminated process. These clauses correspond to the intensional connectives '|' and '0' of the logic. The clause (8) for abstraction is similar to the input clause of bisimilarity in asynchronous message-passing calculi [ACS98]. This is so because communication in MA is asynchronous (see also Subsection B.3.2 below).

The most peculiar aspect of the definition of $\approx_{int}$ is the use of the stuttering relations. Although they could be avoided on finite processes, they cannot in the full calculus, as illustrated by the following example.

Example B.1.4 (Bisimilar processes with stuttering) *Consider the processes* P, Q *of Example B.1.1. It holds that* $P \not\approx_{int} Q$*; however, since* $P \xRightarrow{(\mathsf{in}\,n, \mathsf{out}\,n)^*} Q$ *and* $Q \xRightarrow{(\mathsf{in}\,n, \mathsf{out}\,n)^*} P$*, we have*

$$\mathsf{out}\,n.\,P \;\approx_{int}\; \mathsf{out}\,n.\,Q\,.$$

Actually, $\mathsf{out}\,n.\,P \;\approx\; \mathsf{out}\,n.\,Q.$*, that is, these two processes are extensionally equivalent, and they are also equated by the logic (i.e.,* $\mathsf{out}\,n.\,P =_L \mathsf{out}\,n.\,Q$*). But they would not be intensionally bisimilar without the stuttering relations.*

The reason for this peculiarity is that, intuitively, these processes have the same behaviour in any testing context. To see why the extra capabilities of Q *do not affect its behaviour, consider a reduction involving* $\mathsf{out}\,n.\,P$*, of the following shape:*

$$n[m[\mathsf{out}\,n.\,P \mid R]] \longrightarrow n[\mathbf{0}] \mid m[P \mid R]\,.$$

Process $\mathsf{out}\,n.\,Q$ *can match this transition using three reductions:*

$$\begin{array}{rcl} n[m[Q \mid R]] & \longrightarrow & n[\mathbf{0}] \mid m[\mathsf{in}\,n.\,\mathsf{out}\,n.\,n[\mathbf{0}] \mid Q' \mid R] \\ & \longrightarrow & n[m[\mathsf{out}\,n.\,n[\mathbf{0}] \mid Q' \mid R]] \\ & \longrightarrow & n[\mathbf{0}] \mid m[P \mid R]\,, \end{array}$$

where Q' *is* $!\mathsf{open}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}]$. *Conversely, the process* $\mathsf{out}\, n.\, Q$ *may be involved in the following scenario:*

$$n[m[\mathsf{out}\, n.\, Q \mid R]] \longrightarrow n[\mathbf{0}] \mid m[Q \mid R]\,,$$

and the process $\mathsf{out}\, n.\, P$ *may mimic this reduction. If we set*

$$Q' = !\mathsf{open}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}],$$

we have

$$\begin{array}{rcl} n[m[\mathsf{out}\, n.\, P \mid R]] & \longrightarrow & n[\mathbf{0}] \mid m[n[\mathbf{0}] \mid Q' \mid R] \\ & \longrightarrow & n[\mathbf{0}] \mid m[Q' \mid \mathsf{in}\, n.\, \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}] \mid R] \\ & \longrightarrow & n[m[Q' \mid \mathsf{out}\, n.\, \mathsf{in}\, n.\, \mathsf{out}\, n.\, n[\mathbf{0}] \mid R]] \\ & \longrightarrow & n[\mathbf{0}] \mid m[Q \mid R]\,. \end{array}$$

By contrast, stuttering does not show up in Safe Ambients [LS00b], where movements are achieved by means of synchronisations between a capability and a co-capability.

The following result follows easily from the definition of $\approx_{int}$:

Lemma B.1.5 $\approx_{int}$ *is an equivalence relation.*

However, it is not obvious that $\approx_{int}$ is preserved by all operators of the calculus, due to the fact that $\approx_{int}$ is, intrinsically, higher-order. Formally, $\approx_{int}$ is not higher-order, in that the labels of actions do not contain terms. 'Higher-orderness' is however hidden in clause (3) of Definition B.1.3, for a reduction may involve movement of terms (for instance, if the reduction uses rules `Red-In` or `Red-Out`). This, as usual in higher-order forms of bisimilarity, complicates the proof that bisimilarity is preserved by parallel composition. As announced, we introduce a second relation on processes, in order to be able to prove congruence.

Syntactical relation, $\approx_{syn}$

Our proof of congruence makes use of a second bisimilarity, $\approx_{syn}$, that, by construction, is preserved by all operators of the calculus, and that is defined as follows:

Etienne: Ici ca a un peu change par rapport a Davide, j'ai mis $\equiv$ a la place de $=$ dans la definition. Tous les resultats me semblent se prouver aussi facilement, donc ca va, mais surtout si on fait pas ca il faut prendre en compte le ! et ca devient affreux. $\boxed{*}$

Definition B.1.6 $\approx_{syn}$ *is the largest symmetric relation on processes such that* $P \approx_{syn} Q$ *implies*

1. *If* $P \equiv P_1 \mid P_2$ *then there are* Q_1, Q_2 *such that* $Q \equiv Q_1 \mid Q_2$ *and* $P_i \approx_{syn} Q_i$, $i = 1, 2$.

2. *If* $P \equiv \mathsf{cap}.\, P'$ *then there are* Q', Q'' *such that* $Q \equiv \mathsf{cap}.\, Q'$, $Q' \overset{\langle \mathsf{cap} \rangle}{\Rightarrow} Q''$, *and* $P' \approx_{syn} Q''$.

3. *If $P \equiv \langle n \rangle$ then $Q \equiv \langle n \rangle$.*

4. *If $P \equiv (x)\, P'$ then there is Q' such that $Q \equiv (x)\, Q'$ and for all m there is Q'' such that $\langle m \rangle \mid Q \Longrightarrow Q''$ and $P'\{m/x\} \approx_{syn} Q''$.*

5. *If $P \equiv n[P']$ then there is Q' such that $Q \equiv n[Q']$ and $P' \approx_{syn} Q'$.*

We extend the relation $\approx_{syn}$ to open terms by saying that $P \approx_{syn} Q$ iff for any closing substitution σ, $P\sigma \approx_{syn} Q\sigma$.

Clause 4 is typical of asynchronous calculi; its statement will entail that $\approx_{syn}$ validates the eta law (see Sec. B.3).

The differences between the definitions of $\approx_{int}$ and $\approx_{syn}$ are the following. First, labelled transitions are replaced by structural congruence in the hypothesis of the corresponding clause. Second, clause (3) is removed.

Transitivity of $\approx_{syn}$ is not obvious, because it is not immediate that $\approx_{syn}$ is preserved under reductions (there is no clause for matching τ-transitions, and, moreover, $\Longrightarrow$ is used in a few places, such as the clauses for movement, for the definition of the stuttering relation).

We shall prove that $\approx_{int}$ and $\approx_{syn}$ coincide (Corollary B.1.16 below). Thus, transitivity of $\approx_{syn}$ will hold because of $\approx_{int}$'s transitivity, and conversely, congruence of $\approx_{syn}$ will ensure congruence of $\approx_{int}$. This proof method, which exploits an auxiliary relation that is manifestly preserved by the operators of the calculus but that is not manifestly preserved under reductions, brings to mind Howe's proof technique for proving congruence of bisimilarity in higher-order languages [How96]. In our case, however, the problem is simpler because of the intensional clauses (1) and (2) of the bisimilarity and because MA is not a fully higher-order calculus: terms may move during a computation, but they may not be copied as a consequence of a movement. We may say that MA is a *linear* higher-order calculus (indeed the congruence of $\approx_{int}$ could also be proved directly, with a little more work).

B.1.2 Congruence

In this section, we establish congruence of intensional bisimilarity, using the auxiliary relation $\approx_{syn}$.

Some results about $\approx_{syn}$

To prove congruence of $\approx_{syn}$, we need the following results.

Lemma B.1.7 *If $\mathbf{0} \approx_{syn} Q$ then $Q \equiv \mathbf{0}$.*

Proof: Suppose $Q \equiv \mathbf{0}$ is not true. This means $Q \equiv Q' \mid Q''$ for some Q', Q'' where Q' is of the form $(x)\, R$, $\langle p \rangle$, $M.\, R$, or $n[R]$. We should then have $P \approx_{syn} Q'$, for some $P \equiv \mathbf{0}$, which is impossible for P cannot match the demand by Q'. □

Lemma B.1.8 $\equiv \subseteq \approx_{syn}$ *and* $\equiv\approx_{syn}\equiv \subseteq \approx_{syn}$.

Proof: Straightforward from the definition of $\approx_{syn}$. □

Lemma B.1.9 *If $P \approx_{syn} Q$, then $P\{n/m\} \approx_{syn} Q\{n/m\}$.*

Proof: If $\mathcal{R}$ is a $\approx_{syn}$-bisimulation, then $\mathcal{R}\{n/m\}$ is a $\approx_{syn}$-bisimulation too. □

Lemma B.1.10 *If $P \approx_{syn} Q$ then $C[P] \approx_{syn} C[Q]$, for all contexts C.*

Proof: By induction on C, using the definition of $\approx_{syn}$ and Lemma B.1.9. □

To prove that $\approx_{int}$ and $\approx_{syn}$ coincide, the main result we need is that $\approx_{syn}$ is preserved under reductions:

Lemma B.1.11 *Suppose $P \approx_{syn} Q$ and $P \longrightarrow P'$. Then there is Q' such that $Q \Longrightarrow Q'$ and $P' \approx_{syn} Q'$.*

Proof: By induction on the depth of the derivation proof of $P \longrightarrow P'$. We proceed by case analysis on the last rule used in the derivation.

- Rule `Red-struct`:

$$\frac{P \equiv P_1 \quad P_1 \longrightarrow P_2 \quad P_2 \equiv P_3}{P \longrightarrow P_3}$$

  By Lemma B.1.8, $P_1 \approx_{syn} Q$; by induction $Q \Longrightarrow Q' \approx_{syn} P_2$; again by Lemma B.1.8, $Q' \approx_{syn} P_3$.

- Rule `Red-Par`:

$$\frac{P_1 \longrightarrow P_1'}{P_1 \mid P_2 \longrightarrow P_1' \mid P_2}$$

  By definition of $\approx_{syn}$ there are Q_i such that $Q \equiv Q_1 \mid Q_2$ and $P_i \approx_{syn} Q_i$. Then we conclude, using induction and Lemma B.1.10.

- Rule `Red-Amb`: use induction and Lemma B.1.10.

- Rule `Red-Com`: Immediate by clauses (1), (3), and (4) of Definition B.1.6.

- Rule `Red-Open`:

$$\frac{}{\mathsf{open}\, n.\, P_1 \mid n[P_2] \longrightarrow P_1 \mid P_2}$$

  By definition of $\approx_{syn}$, $Q \equiv \mathsf{open}\, n.\, Q_1 \mid n[Q_2]$, and for some Q_1' with $Q_1 \Longrightarrow Q_1'$, we have: $P_2 \approx_{syn} Q_2$, $P_1 \approx_{syn} Q_1'$. We also have $Q \Longrightarrow Q_1' \mid Q_2$. Using Lemma B.1.10, we derive $P_1 \mid P_2 \approx_{syn} Q_1' \mid Q_2$, which concludes the case.

- Rule `Red-In`:

$$\frac{}{n[\mathsf{in}\, m.\, P_1 \mid P_2] \mid m[P_3] \longrightarrow m[n[P_1 \mid P_2] \mid P_3]}$$

  By definition of $\approx_{syn}$, $Q \equiv n[\mathsf{in}\, m.\, Q_1 \mid Q_2] \mid m[Q_3]$, and there is Q_1' such that $Q_1 \xRightarrow{(\mathsf{out}\, n, \mathsf{in}\, n)^*} Q_1'$ and we have: $P_2 \approx_{syn} Q_2$, $P_3 \approx_{syn} Q_3$, and $P_1 \approx_{syn} Q_1'$.

  We also have $Q \Longrightarrow m[n[Q_1' \mid Q_2] \mid Q_3]$. Using Lemma B.1.10, we derive

$$m[n[P_1 \mid P_2] \mid P_3] \approx_{syn} m[n[Q_1' \mid Q_2] \mid Q_3],$$

  which concludes the case.

- Rule `Red-Out`: similar to the previous case.

□

Corollary B.1.12 *Suppose $P \approx_{syn} Q$ and $P \Longrightarrow P'$. Then there is Q' such that $Q \Longrightarrow Q'$ and $P' \approx_{syn} Q'$.*

Proof: By induction on the number of transitions in $P \Longrightarrow P'$, using Lemma B.1.11 for the inductive case. □

Correspondence between the two bisimilarities

Having proved Lemma B.1.8 and Corollary B.1.12, we can derive transitivity, and hence conclude that $\approx_{syn}$ is a congruence relation. This can be proved directly, or by going through the correspondence with $\approx_{int}$. Here we follow the latter approach.

Lemma B.1.13

- $\mathsf{cap}.\, P \approx_{int} Q$ *implies* $Q \equiv \mathsf{cap}.\, Q'$, *for some* Q'.

- $\langle n \rangle \approx_{int} Q$ *implies* $Q \equiv \langle n \rangle$.

- $(x)\, P \approx_{int} Q$ *implies* $Q \equiv (x)\, Q'$, *for some* Q'.

Proof: Suppose $Q \equiv Q_1 \mid Q_2$ where both Q_i are not structurally congruent to $\mathbf{0}$. Then P and Q could be distinguished using the clauses of $\approx_{int}$ for parallel composition and $\mathbf{0}$.

Similarly, P and Q can be distinguished if Q is single but not of the same shape as P. □

Lemma B.1.14 $\approx_{int} \subseteq \approx_{syn}$.

Proof: By proving that $\approx_{int}$ is a $\approx_{syn}$-bisimulation. The proof is easy, using Lemma B.1.13. □

Lemma B.1.15 $\approx_{syn} \subseteq \approx_{int}$.

Proof: By proving that $\approx_{syn}$ is a $\approx_{int}$-bisimulation.

We need Lemma B.1.10 (precisely, the fact that $\approx_{syn}$ is preserved by parallel composition), Lemma B.1.8, Corollary B.1.12, and Lemma B.1.7. □

Corollary B.1.16 *Relations $\approx_{int}$ and $\approx_{syn}$ coincide.*

Corollary B.1.17 *Relations $\approx_{int}$ and $\approx_{syn}$ are congruence relations.*

Proof: Follows from Corollary B.1.16, and Lemmas B.1.5 and B.1.10 □

B.1.3 Inversion results for $\approx_{int}$

We prove some more properties about $\approx_{int}$. The inversion results we derive below represent a crucial step in justifying the construction of the inductive characterisation of $\approx_{int}$ we present in Section B.2.

Lemma B.1.18 (Inversion results for $\approx_{int}$) *Let P, P_1, P_2, Q be processes of* MA. *Then*

1. $\mathbf{0} \approx_{int} Q$ *iff* $Q \equiv \mathbf{0}$.
2. $n[P] \approx_{int} Q$ *iff there exists* Q' *such that* $Q \equiv n[Q']$ *and* $P \approx_{int} Q'$.
3. $P_1 \mid P_2 \approx_{int} Q$ *iff there exist* Q_1, Q_2 *such that* $Q \equiv Q_1 \mid Q_2$ *and* $P_i \approx_{int} Q_i$ *for* $i = 1, 2$.
4. $!P \approx_{int} Q$ *iff there exist* $r \geq 1, s \geq r, Q_i$ $(1 \leq i \leq s)$ *such that* $Q \equiv \Pi_{1\leq i\leq r}!Q_i \mid \Pi_{r+1\leq i\leq s}Q_i$, *and* $P \approx_{int} Q_i$ *for* $i = 1 \ldots s$.
5. $\mathsf{cap}.\,P \approx_{int} Q$ *iff there exists* Q' *such that* $Q \equiv \mathsf{cap}.\,Q'$ *with* $P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow}\approx_{int} Q'$ *and* $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow}\approx_{int} P$.
6. $\langle n\rangle \approx_{int} Q$ *iff* $Q \equiv \langle n\rangle$
7. $(x)\,P \approx_{int} Q$ *iff there exists* $Q', m.$ *such that* $m \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q)$, $Q \equiv (x)\,Q'$ $Q \mid \langle m\rangle \Rightarrow\approx_{int} P$ *and* $(x)\,P \mid \langle m\rangle \Rightarrow\approx_{int} Q'$.

Proof: Except for the fourth item, the left to right implications hold because $\approx_{syn}$ and $\approx_{int}$ have equivalent definitions. For the right to left implication, cases 1 and 6 hold by reflexivity of $\approx_{int}$, and cases 2 and 3 follow from congruence of $\approx_{int}$ (Corrolary B.1.17), and case 5 comes from a case analysis in the definition of $\approx_{int}$.

We are thus left with case 4: it follows, using clauses 1 and 2, by reasoning over the cardinality of single components in P and Q. We consider each process as a multiset of single components, and we remark that in case there are infinitely many copies of a given process on P's side, there are also infinitely many processes of bisimilar processes on Q's side. □

B.1.4 Soundness and completeness of $\approx_{int}$ for $=_L$

We now establish soundness and completeness of $\approx_{int}$ w.r.t. $=_L$. Soundness means that for any processes P and Q, $P \approx_{int} Q$ implies $P =_L Q$. We defer the presentation of the proof of completeness in the general case to the next section, and only show that $\approx_{int} \supseteq =_L$ on finite processes. We chose to do this for the sake of clarity: the proof in the finite case is much simpler, and exposes the basic ideas of the argument in the full calculus.

Soundness

In order to prove soundness (on the whole calculus), we use the definition of $\approx_{syn}$ and the congruence property to establish that bisimilar processes satisfy the same formulas.

Theorem B.1.19 (Soundness of $\approx_{int}$) *Assume $P, Q \in \mathsf{MA}$, and suppose $P \approx_{int} Q$. Then, for all $\mathcal{A}$, it holds that $P \models \mathcal{A}$ iff $Q \models \mathcal{A}$.*

Proof: By induction on the depth of $\mathcal{A}$.

- $\mathcal{A} = \top$.

  Nothing to prove.

- $\mathcal{A} = \neg\mathcal{B}$ or $\mathcal{A} = \mathcal{B}_1 \vee \mathcal{B}_2$.

  By induction and the definition of satisfiability.

- $\mathcal{A} = 0$.

  By definition of satistability and clause (2) of the definition of $\approx_{int}$.

- $\mathcal{A} = n[\mathcal{B}]$.

  Then $P \equiv n[P']$ and $P' \models \mathcal{B}$. Hence $Q \equiv n[Q']$ for some $Q' \approx_{int} P'$. By induction, $Q' \models \mathcal{B}$; we can therefore conclude that also $Q \models n[\mathcal{B}]$ holds.

- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$.

  Then $P \equiv P_1 \mid P_2$ and $P_i \models \mathcal{A}_i$. By clause (1) of Definition B.1.3, $Q \equiv Q_1 \mid Q_2$ for some $Q_i \approx_{int} P_i$. By induction, $Q_i \models \mathcal{A}_i$; we can therefore conclude that also $Q \models \mathcal{A}_1 \mid \mathcal{A}_2$ holds.

- $\mathcal{A} = \forall x.\, \mathcal{B}$.

  By definition of satisfaction, $P \models \mathcal{A}\{^m\!/\!_x\}$ for all m. The result for Q then follows by induction, for $\mathcal{A}\{^m\!/\!_x\}$ has one fewer operator.

- $\mathcal{A} = \Diamond\, \mathcal{B}$.

  By definition of satisfaction, there is P' such that $P \Longrightarrow P'$ and $P' \models \mathcal{B}$. Using clause (3) of the definition of $\approx_{int}$, there is Q' such that $Q \Longrightarrow Q' \approx_{int} P'$. By induction, $Q' \models \mathcal{B}$; hence $Q \models \mathcal{A}$.

- $\mathcal{A} = @\mathcal{B}n$ or $\mathcal{A} = \mathcal{A}_1 \rhd \mathcal{A}_2$.

  Follows using induction and the congruence of $\approx_{int}$.

$\square$

Completeness (on finite processes)

The proof of completeness we develop here is based on the construction of a sequence of approximants of $\approx_{syn}$, which is a standard approach for image-finite calculi. This works in the finite case (finiteness implies image-finiteness), but not on the whole calculus of Mobile Ambients. The proof is however interesting on its own, and gives a much simpler account on how the logic expresses the clauses of $\approx_{int}$ than the whole calculus.

We remark that we are able to give two other completeness results, apart from this one. The first one holds as a corollary of the definability of characteristic formulas for $\approx_{int}$ (see appendix A), and is valid for a (non finite) subcalculus of MA, called MA_{IF}, that enjoys an image-finiteness property (see Remark B.1.21 below). The other completeness result is for the whole calculus: its proof is much more difficult and is presented in Section B.2.

The idea to derive completeness on finite processes is to express in the logic the syntactic requirements that belong to the defining clauses of $\approx_{int}$ (Definition B.1.3). How to achieve this is clear for all clauses of the definition, except for capabilities, inputs, and messages. For this, we rely on the following expressiveness results for AL:

Lemma B.1.20 (cf. Appendix A) *For any* $\mathsf{cap}, \mathcal{A}$, *there exists a formula* $\langle\langle\mathsf{cap}\rangle\rangle.\,\mathcal{A}$ *such that for all process* P,

$$P \models \langle\langle\mathsf{cap}\rangle\rangle.\,\mathcal{A} \qquad \textit{iff} \qquad \exists P', P''.\ P \equiv \mathsf{cap}.\,P'\,,\ P' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} P'' \textit{ and } P'' \models \mathcal{A}\,.$$

For all n, *there is a formula* $\langle n \rangle$ *such that*

$$P \models \langle n \rangle \qquad \textit{iff} \qquad P \equiv \langle n \rangle\,.$$

For all $n, \mathcal{A}$, *there is a formula context* $\langle\langle ?n \rangle\rangle.\,\mathcal{A}$ *such that for all process* P,

$$P \models \langle\langle ?n \rangle\rangle.\,\mathcal{A} \qquad \textit{iff} \qquad \exists m, P', P''.\ P \equiv (m)P'\,,\ (m)P' \mid \langle n \rangle \Rightarrow P'' \textit{ and } P'' \models \mathcal{A}\,.$$

These results, that are presented in appendix A, represent the key steps towards the derivability of characteristic formulas for a subcalculus of MA. We now make a short digression on this.

Remark B.1.21 (Completeness via characteristic formulas) *Completeness can be obtained as a consequence of the definability of characteristic formulas. This holds on a subcalculus of* MA *that we define as follows:*

Definition B.1.22 (Subcalculus MA_{IF}) *We call* image-finite processes *processes* P *such that in any subterm of the form* $\mathsf{cap}.\,P'$ *(resp.* $(x)P'$*), the process* P' *admit only finitely many distinguishable reducts under* $\stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow}$ *(resp.* $P'\{n/x\}$ *admit only finitely many distinguishable reducts under* $\Rightarrow$ *for some* $n \notin \mathrm{fn}(P)$*).*

MA_{IF} *is the set of image-finite MA processes.*

Lemma B.1.23 (cf. Appendix A) *For any* MA_{IF} *process* P, *there exists a formula* $\mathcal{A}_P$ *s.t. for any* Q, $Q \models \mathcal{A}_P$ *iff* $P \approx_{syn} Q$.

Corollary B.1.24 (Completeness on MA_{IF}) *On* MA_{IF}, $=_L\ \subseteq \approx_{int}$.

The derivability of characteristic formulas for $\approx_{syn}$ (Lemma B.1.23) entails completeness on MA_{IF}, that contains the finite MA processes. We are however interested here in a different route for the completeness proof, that uses i-th approximants $\approxeq_i$ of relation $\approx_{syn}$, and the fact that $\approxeq_\omega \stackrel{\text{def}}{=} \bigcap_i \approxeq_i$ coincides with $\approx_{syn}$.

Definition B.1.25 *We define the relations* $\approxeq_i$ *between processes, for all* $i \geq 0$. $\approxeq_0$ *is the universal relation, and* $\approxeq_{i+1}$ *is defined by saying that* $P \approxeq_{i+1} Q$ *holds if we have:*

1. *If* $P \equiv P_1 \mid P_2$ *then there are* Q_s *such that* $Q \equiv Q_1 \mid Q_2$ *and for all* s $P_s \approxeq_i Q_s$.
2. *If* $P \equiv \mathsf{cap}.\,P'$ *then there are* Q', Q'' *such that*

(a) $Q \equiv \mathsf{cap}.\,Q'$,

(b) $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} Q''$, and

(c) $P' \approxeq_i Q'$.

3. If $P \equiv \langle n \rangle$ then $Q \equiv \langle n \rangle$.

4. If $P \equiv (x)\, P'$ then there is Q' such that

(a) $Q \equiv (x)\, Q'$ and

(b) for all n there is Q'' such that $\langle n \rangle \mid Q \Longrightarrow Q''$ and $P'\{^n\!/\!_x\} \approxeq_i Q''$.

5. If $P \equiv n[P']$ then there is Q' such that $Q \equiv n[Q']$ and $P' \approxeq_i Q'$.

We set $\approxeq_\omega \stackrel{\text{def}}{=} \bigcap_{i \geq 0} \approxeq_i$.

Lemma B.1.26 $\approxeq_\omega$ *coincides with* $\approx_{syn}$ *on finite processes.*

Proof: By a standard approximation result, using image finiteness, we get $\approxeq_\omega = \equiv\approx_{syn}\equiv$. We then conclude with $\approx_{syn} = \equiv\approx_{syn}\equiv$. □

Lemma B.1.27 *Let P, Q be two* finite *processes. If $P =_L Q$ then $P \approxeq_\omega Q$.*

Proof: Suppose $P \not\approxeq_\omega Q$. Then there is i such that $P \not\approxeq_i Q$. We prove, by induction on i, that in this case we could find a formula $\mathcal{A}$ such that $P \models \mathcal{A}$ holds but $Q \models \mathcal{A}$ does not.

For $i = 0$, this trivialy holds since the hypothesis $P \not\approxeq_0 Q$ is absurd for $\approxeq_0$ being the universal relation.

Now the case $i + 1$, for $i \geq 0$. We proceed by case analysis:

1. $P \equiv P_1 \mid P_2$, and for all Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ there is t $(1 \leq t \leq 2)$ such that $P_t \not\approxeq_i Q_t$.

   Modulo $\equiv$, there is a finite number, say s, of pairs of processes Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ (note that by hypothesis P is finite, and identify two pairs Q_1, Q_2 and Q_1', Q_2' if $Q_t \equiv Q_t'$ for all t). Call $Q_{t,u}$ the t-th process of the u-th pair. Then for all u $(1 \leq u \leq s)$ there is t such that $P_t \not\approxeq_i Q_{t,u}$. By induction, there is $\mathcal{A}_{t,u}$ such that

$$P_t \models \mathcal{A}_{t,u} \text{ but } Q_{t,u} \not\models \mathcal{A}_{t,u}\,.$$

   Define

$$B_t \stackrel{\text{def}}{=} \wedge_{1 \leq u \leq s \text{ and such that } \mathcal{A}_{t,u} \text{ is defined}} \mathcal{A}_{t,u}\,.$$

   Then

$$P \models B_1 \mid B_2\,,$$

   whereas

$$Q \not\models B_1 \mid B_2\,.$$

2. $P \equiv \mathsf{cap}.\, P'$, $Q \equiv \mathsf{cap}.\, Q'$, and for all Q_t such that $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} nQ_t$, it holds that $P' \not\approxeq_i Q_t$.

   By induction, for all t there is $\mathcal{A}_t$ such that $P' \models \mathcal{A}_t$ but $Q_t \not\models \mathcal{A}_t$. Since Q is finite, there is only a finite number of these processes Q_t (up to $\equiv$). Write $(Q_t)_{t\in I}$ this set of processes up to $\equiv$, and call $\mathcal{A}_t$ the formula corresponding to each Q_t. Define

$$\mathcal{A} \stackrel{\mathrm{def}}{=} \langle\langle \mathsf{cap} \rangle\rangle. \bigwedge_{t\in I} \mathcal{A}_t \,,$$

   using an obvious notation for the (finite) conjunction of the $\mathcal{A}_t$s. Then $P \models \mathcal{A}$ but $Q \not\models \mathcal{A}$.

3. $P \equiv \langle n \rangle$, and $Q \not\equiv \langle n \rangle$: then $P {\models} \langle n \rangle$, and $Q \not\models \langle n \rangle$.

4. $P \equiv (x)\, P'$, $Q \equiv (x)\, Q'$ and there is n such that for all Q_t such that $\langle n \rangle \mid Q \Longrightarrow Q_t$ it holds that $P'' \not\approxeq_i Q_t$, for $P'' \stackrel{\mathrm{def}}{=} P'\{n/x\}$.

   Modulo $\equiv$, there is only a finite number of such Q_t, say $Q_1, \ldots, Q_s$. By induction, there are formulas $\mathcal{A}_1, \ldots, \mathcal{A}_s$ with $P'' \models \mathcal{A}_t$ and $Q_t \not\models \mathcal{A}_t$. We introduce as above the notation $(Q_t)_{t\in I}$, and we define

$$\mathcal{A} \stackrel{\mathrm{def}}{=} \langle\langle ?n \rangle\rangle. \bigwedge_{t\in I} \mathcal{A}_t \,.$$

   Then $P \models \mathcal{A}$; but $Q \not\models \mathcal{A}$, because whenever $\langle n \rangle \mid Q \Longrightarrow Q_t$ it holds that $Q_t \not\models \mathcal{A}_t$.

5. $P \equiv n[P']$, $Q \equiv n[Q']$ and $P' \not\approxeq_i Q'$.

   By induction there is $\mathcal{A}'$ with $P' \models \mathcal{A}'$ but $Q' \not\models \mathcal{A}'$. Define $\mathcal{A} \stackrel{\mathrm{def}}{=} n[\mathcal{A}']$; then $P \models \mathcal{A}$ but $Q \not\models \mathcal{A}$.

$\square$

Theorem B.1.28 (Completeness on finite processes) *Let P, Q be finite processes. If $P{=_L}Q$ then $P \approx_{int} Q$.*

Proof: Follows from Lemma B.1.26 and B.1.27. $\square$

B.2 Completeness of $\approx_{int}$ in the full calculus

The proof we have presented in the finite case cannot be used directly in the full MA calculus, because we lack the image-finiteness hypothesis, which allowed us to show that the limit $\approxeq_\omega$ coincides with $\approx_{syn}$. In this section, we establish completeness via another characterisation of intensional bisimilarity. We introduce an inductive relation, written $\doteq$ (Definition B.2.13), and we prove that it is equal to $\approx_{int}$ and $\approx_{syn}$ (Subsection B.2.2). We then show completeness of $\doteq$ with respect to logical equivalence by induction on the derivation of a $\doteq$ statement, in Subsection B.2.4. Technically, this kind of reasoning is enabled by the ability to capture within the logic the order on terms that is used in the induction. We achieve this by introducing the notion of *active context decomposition* (Subsection B.2.3).

B.2.1 Preliminary definitions

As we will see, Lemma B.1.18 gives a point of view on processes that allows us to define an inductive presentation of $\approx_{int}$. For this, an important role is played by two measures on terms, the *sequentiality degree* and the *depth degree*, that we recall from appendix A. We also introduce the well-founded order that is used in appendix A for the definition of characteristic formulas on a subcalculus of MA (see Lemma B.1.23 above), and that will serve as a basis for the definition of $\simeq$.

Intuitively, the sequentiality degree corresponds to the maximal depth of nesting of capabilities in a process.

Definition B.2.1 (Sequentiality degree, $\mathsf{ds}()$) *The sequentiality degree of a term P is defined as follows:*

- $\mathsf{ds}(()\mathbf{0}) = 0$, $\mathsf{ds}(()P \mid Q) = \max\big(\mathsf{ds}(()P), \mathsf{ds}(()Q)\big)$*;*
- $\mathsf{ds}(()n[P]) = \mathsf{ds}(()!P) = \mathsf{ds}(()P)$*;*
- $\mathsf{ds}(()\mathsf{cap}.\, P) = 1 + \mathsf{ds}(()P)$*;*
- $\mathsf{ds}(()\langle n\rangle) = 1$*;*
- $\mathsf{ds}(()(x)\ P) = \mathsf{ds}(()P') + 1$ *if $(x)\ P'$ is the eta normal form of $(x)\ P$.*

Note that this definition relies on the presence of the ! operator (instead of a recursion operator) in the calculus, so that we do not have processes that are infinite 'in depth'. Two important properties of $\mathsf{ds}(()P)$ are the following:

Lemma B.2.2 *Let P, Q be two terms of MA. Then:*

1. *if $P \equiv Q$, then $\mathsf{ds}(()P) = \mathsf{ds}(()Q)$*
2. *if $P \rightarrow Q$ or $P \stackrel{\mu}{\rightarrow} Q$ then $\mathsf{ds}(()P) \geq \mathsf{ds}(()Q)$.*

Proof: 1 is immediate, as is the result on $\stackrel{\mu}{\rightarrow}$ in 2. For $P \rightarrow Q$, we reason by induction on the height of the derivation of $P \rightarrow Q$. $\square$

Corollary B.2.3 *For all cap, if $P \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} Q$, then $\mathsf{ds}(()P) \geq \mathsf{ds}(()Q)$.*

This result will be important for the justification of Definition B.2.13 below.

We show now how to characterise the sequentiality degree of a process through formulas. This characterisation will be helpful in some part of the completeness proof.

Lemma B.2.4 (Sequentiality degree constraint lemma) *For any process $P \in \mathsf{MA}$, there exists a formula $F_{\mathsf{ds}(()P)}$ such that:*

- $P \models F_{\mathsf{ds}(()P)}$
- *for any term Q, if $Q \models F_{\mathsf{ds}(()P)}$, then $\mathsf{ds}(()Q) \geq \mathsf{ds}(()P)$.*

Proof: We may freely assume that P is eta normalised. Let us first reason by induction on $\mathsf{ds}(()P)$:

- for $\mathsf{ds}(()P) = 0$, $F_{\mathsf{ds}(()P)} = \top$ is sufficient.
- for $\mathsf{ds}(()P) > 0$, let us assume we may find formulas $F_{\mathsf{ds}(()P')}$ for any P' such that $\mathsf{ds}(()P') < \mathsf{ds}(()P)$. Moreover, let us reason by induction on P.
  - the case $P = \mathbf{0}$ is impossible.
  - for $P = P_1 \mid P_2$, there is some i such that $\mathsf{ds}(()P) = \mathsf{ds}(()P_i)$. Then we may choose $F_{\mathsf{ds}(()P)} = F_{\mathsf{ds}(()P_i)} \mid \top$. In the same way, let us set $F_{\mathsf{ds}(()\langle n\rangle)} = \mathcal{F}_{\langle n\rangle}$, $F_{\mathsf{ds}(()!P)} = F_{\mathsf{ds}(()P)} \mid \top$ and $F_{\mathsf{ds}(()n[P])} = n[F_{\mathsf{ds}(()P)}]$.
  - for $P = \mathsf{cap}.\, P'$, we use the general induction hypothesis to get some $F_{\mathsf{ds}(()P')}$. Let us then take $F_{\mathsf{ds}(()P)} = \langle\langle\mathsf{cap}\rangle\rangle.\, F_{\mathsf{ds}(()P')}$. Then $P \models F_{\mathsf{ds}(()P)}$ obviously, and for any Q such that $Q \models F_{\mathsf{ds}(()P)}$, we deduce (from Lemma B.1.20) that there are some Q', Q'' such that $Q \equiv \mathsf{cap}.\, Q'$ and $Q' \overset{\langle\mathsf{cap}\rangle}{\Rightarrow} Q''$ with $Q'' \models F_{\mathsf{ds}(()P')}$. Now by Lemma B.2.2, $\mathsf{ds}(()Q) - 1 = \mathsf{ds}(()Q') \geq \mathsf{ds}(()Q'')$, and by induction hypothesis $\mathsf{ds}(()Q'') \geq \mathsf{ds}(()P') = \mathsf{ds}(()P) - 1$, so that finally $\mathsf{ds}(()Q) \geq \mathsf{ds}(()P)$.
  - for $P = (x)\, P'$, we use the general induction hypothesis to get some $F_{\mathsf{ds}(()P')}$. Let us then take $F_{\mathsf{ds}(()P)} = \exists x.\ \langle\langle ?x\rangle\rangle.\, F_{\mathsf{ds}(()P')}$. Then $P \models F_{\mathsf{ds}(()P)}$ obviously, and for any Q such that $Q \models F_{\mathsf{ds}(()P)}$, we deduce (from Lemma B.1.20) that there are some n, Q', Q'' such that $Q \equiv (x)\, Q'$ and $Q_1 = \langle n\rangle \mid (x)Q' \Rightarrow Q''$ with $Q'' \models F_{\mathsf{ds}(()P')}$. Now by Lemma B.2.2, $\mathsf{ds}(()Q_1) - 1 = \mathsf{ds}(()Q) - 1 = \mathsf{ds}(()Q'\{n/x\}) \geq \mathsf{ds}(()Q'')$, and by induction hypothesis $\mathsf{ds}(()Q'') \geq \mathsf{ds}(()P') = \mathsf{ds}(()P) - 1$, so that finally $\mathsf{ds}(()Q) \geq \mathsf{ds}(()P)$.

□

Corollary B.2.5 *If $P \approx_{int} Q$, then $\mathsf{ds}(P) = \mathsf{ds}(Q)$*

Proof: By Theorem B.1.19, $P{=_L}Q$, hence the result. □

The sequentiality degree is moreover handfull to reason inductively on processes up to reductions of some subterms. We formalize now this induction principle.

Definition B.2.6 (Well-founded order) *Given two processes P and Q , we note $P < Q$ if either $\mathsf{ds}(()P) < \mathsf{ds}(()Q)$ or P is a strict subterm of Q. We shall sometimes write $Q > P$ as a synonym for $P < Q$.*

Lemma B.2.7

- *$<$ is well-founded.*
- *Suppose P is of the form either $\mathsf{cap}.\, P'$ or $(x)P'$, and suppose moreover $P > Q$ and $Q \overset{\langle\mathsf{cap}\rangle}{\Rightarrow} Q'$ for some cap. Then $P > Q'$.*

Proof:

- $<$ is well-founded: if P is a strict subterm of Q, then $\mathsf{ds}(()P) \leq \mathsf{ds}(()Q)$.
- $P > Q'$: follows from Lemma B.2.2.

□

We end this section mentioning another notion of measure on processes that will be necessary to consider in the interpretation of the replicated formula $!n[.\,]$ (see later).

Definition B.2.8 (Depth degree) *The depth degree of a process is given by a function* deep *from* MA *processes to natural numbers, inductively defined by:*

- $\mathsf{deep}(\mathbf{0}) \stackrel{\text{def}}{=} 0$, $\mathsf{deep}(\mathsf{cap}.\,P) \stackrel{\text{def}}{=} 0$;
- $\mathsf{deep}((x)\,P) \stackrel{\text{def}}{=} 0$, $\mathsf{deep}(\langle n\rangle) \stackrel{\text{def}}{=} 0$;
- $\mathsf{deep}(n[P]) \stackrel{\text{def}}{=} \mathsf{deep}(P) + 1$;
- $\mathsf{deep}((!)P_1 \mid \ldots \mid (!)P_r) \stackrel{\text{def}}{=} \max_{1\leq i\leq r} \mathsf{deep}(P_i)$.

B.2.2 Inductive characterisation of $\approx_{int}$

As hinted above, following the inversion principles given by Lemma B.1.18, we are able to define the inductive characterisation of $\approx_{int}$ (the clauses for capabilities and input are less simple than the others, but in these cases, we exploit the fact that the sequentiality degree decreases along transitions). Accordingly, in order to define the inductive relation, we decompose processes into what we call their *active context* and their *frozen subterms* (these are frozen in the sense that they do not participate in the immediate interactions of the process).

To illustrate the point, the active context of the process $P = \mathsf{in}\,n.\,\mathsf{out}\,m.\,m[\mathbf{0}] \mid n[\mathsf{open}\,n]$ is $\mathsf{in}\,n.\,[\,]_1 \mid n[\mathsf{open}\,n.\,[\,]_2]$ (notice the presence of two different —numbered— holes), and its frozen subterms of order one are $\mathsf{out}\,m.\,m[\mathbf{0}]$ and $\mathbf{0}$. If we iterate, we have that $m[\mathbf{0}]$ is a frozen subterm of order one of $\mathsf{out}\,m.\,m[\mathbf{0}]$, and we will hence say that $m[\mathbf{0}]$ is a frozen subterm of order two of P.

We now present the formal definition of active contexts. The set of frozen terms will be studied later on.

Active contexts

A *n-ary context*, ranged over with $\mathcal{C}, \mathcal{D}, \ldots$, is a process term with some holes $[\,]_i$ in it, $i \in \mathbb{N}$, with the requirement for each hole to occur at most once (note that a hole $[\,]_i$ cannot be confused with ambient constructs thanks to the i subscript). i will be called an *index* of $\mathcal{C}$ if $\mathcal{C}$ contains the hole $[\,]_i$. Two contexts are *disjoint* if no hole $[\,]_i$ occurs in both. Structural congruence is extended on holes by adding the terminal rule $[]_i \equiv []_i$. The application of a context to a vector of terms is inductively defined by $\mathcal{C}[\epsilon] \stackrel{\text{def}}{=} \mathcal{C}$ and $\mathcal{C}[P.\,\tilde{P}] \stackrel{\text{def}}{=} \mathcal{C}\{P/[\,]_i\}[\tilde{P}]$, where i is the smallest index of $\mathcal{C}$ (for the case $\mathcal{C}$ is not a term, otherwise $\mathcal{C}[\tilde{P}]$ is simply $\mathcal{C}$). In the remainder of the paper, we will actually restrict ourselves to expressions of the form $\mathcal{C}[\tilde{P}]$ with $\tilde{P}$ and $\mathcal{C}$ having the same arity, that is "fully applied" contexts.

Definition B.2.9 (Active contexts, normalised contexts, expansion) *Let* $\mathcal{C}, \mathcal{D}$ *be two contexts.*

- $\mathcal{C}$ *is an* active context *if each hole appears underneath exactly one capability or input in* $\mathcal{C}$.

- $\mathcal{C}$ *is* single *if its topmost constructor is an ambient or a capability construct. A context* $\mathcal{C}$ *is* normalised *if every replication occurring in* $\mathcal{C}$ *is applied to a single context.*

- *If* σ *is a function that associates an index of* $\mathcal{C}$ *to any index of* $\mathcal{D}$*, we will say that* $\mathcal{D}$ *is a* σ-expansion *of* $\mathcal{C}$*, (written* $\mathcal{C} \lhd_\sigma \mathcal{D}$*), or equivalently that* $\mathcal{C}$ *is a* σ-contraction *of* $\mathcal{D}$*, if* $\mathcal{C} \equiv \mathcal{D}(\sigma)$*, where* $\mathcal{D}(\sigma)$ *is* $\mathcal{D}\{[]_{\sigma(j)}/[]_j\}_{j\in I_\mathcal{D}}$*,* $I_\mathcal{D}$ *being the (finite) set of integers indexing the holes that occur in* $\mathcal{D}$*.*

Remark B.2.10 *We can make the following observations about active contexts:*

- *Active contexts can be described by the following grammar:*

$$\mathcal{C} \quad ::= \quad \mathbf{0} \quad \Big| \quad n[\mathcal{C}] \quad \Big| \quad !\mathcal{C} \quad \Big| \quad \mathcal{C} \mid \mathcal{C} \quad \Big| \quad \mathsf{cap}.\,[\,]_i \quad \Big| \quad \langle n \rangle \quad \Big| \quad (x)\,[\,]_i$$

where each $[\,]_i$ *has a single occurrence. We sometimes use* cap_i *to refer to the capability occurring immediately above the hole* $[\,]_i$ *in an active context.*

- *For any process* P*, there exists a unique active context* $\mathcal{C}$ *and a unique vector of terms* $\tilde{P}$ *such that* $P = \mathcal{C}[\tilde{P}]$ *(where* $=$ *denotes syntactic equality). This will be referred to as the* active context decomposition *of* P*. Moreover, we can remark that* $\mathsf{ds}(()P) = 1 + \max_{P_i\in\tilde{P}} \mathsf{ds}(()P_i)$ *when* P *is in eta normal form.*

- *Given a process* P*, and using the axioms* $!\mathbf{0} \equiv \mathbf{0}$ *and* $!(P \mid Q) \equiv\, !P \mid !Q$*, we can always exhibit* $Q \equiv P$ *such that* $Q = \mathcal{C}[\tilde{Q}]$ *where* $\mathcal{C}$ *is a normalised active context.*

- *Relation* $\lhd_\sigma$ *is defined up to* $\equiv$*, i.e.* $\equiv\lhd_\sigma\equiv$ *coincides with* $\lhd_\sigma$*.*

Intuitively, we expect that two intensionally bisimilar processes induce structurally congruent active contexts. Having this property would allow us to reason inductively on the corresponding frozen subterms (those terms that are used to fill the holes in the active contexts). However, due to the replication operator, this is not exactly the case, and bisimilar processes only exhibit the same active context up to replication unfolding, which is what we express through expansion.

The following properties about expansion of active contexts are the counterpart of the corresponding results for $\approx_{int}$ (Lemma B.1.18).

Lemma B.2.11 (Inversion results for expansion) *Let* $\mathcal{C}, \mathcal{D}, \mathcal{C}_1, \mathcal{C}_2$ *be active contexts, and* σ *an expansion function whose domain contains the indexes of* $\mathcal{D}$*. Then:*

1. $\mathbf{0} \lhd_\sigma \mathcal{D}$ *iff* $\mathcal{D} \equiv \mathbf{0}$*.*

2. $n[\mathcal{C}] \lhd_\sigma \mathcal{D}$ *iff there exists some* $\mathcal{D}'$ *such that* $\mathcal{D} \equiv n[\mathcal{D}']$ *and* $\mathcal{C}' \lhd_\sigma \mathcal{D}'$

3. $\mathcal{C}_1 \mid \mathcal{C}_2 \lhd_\sigma \mathcal{D}$ *iff there exist* $\mathcal{D}_1, \mathcal{D}_2$ *such that* $\mathcal{D} \equiv \mathcal{D}_1 \mid \mathcal{D}_2$ *and* $\mathcal{C}_i \lhd_\sigma \mathcal{D}_i$ *(*$i = 1, 2$*).*

4. $!\mathcal{C} \equiv \mathcal{D}$ *iff there exist* $r \geq 1, s \geq r, \mathcal{D}_i$ *(*$1 \leq i \leq s$*) such that* $\mathcal{D} \equiv \Pi_{1\leq i\leq r}!\mathcal{D}_i \mid \Pi_{r+1\leq i\leq s}\mathcal{D}_i$*, and* $\mathcal{C} \lhd_\sigma \mathcal{D}_i$ *for* $i = 1 \ldots s$*.*

5. $\mathsf{cap}.[]_i \ \lhd_\sigma \ \mathcal{D}$ *iff* $\mathcal{D} \ \equiv \ \mathsf{cap}.[]_j$ *and* $\sigma(j) = i$.

6. $\langle n \rangle \lhd_\sigma \ \mathcal{D}$ *iff* $\mathcal{D} \ \equiv \ \langle n \rangle$.

7. $(x)\,[\,]_i \lhd_\sigma \ \mathcal{D}$ *iff* $\mathcal{D} \ \equiv \ (x)\,[\,]_j$ *and* $\sigma(j) = i$.

Proof: Cases 1,2,5,6,7 are easy, and so are the reverse implications for the other cases. The only points to prove are the direct implications for results 3 and 4.

We begin by establishing a useful result, in the case where two contexts $\mathcal{C}$ and $\mathcal{D}$ are related by $\mathcal{C} = \mathcal{D}(\sigma)$ for some σ. Suppose we have two families of single active contexts $(\mathcal{C}_i)_i, (\mathcal{D}_j)_j$ such that within any family, every two contexts belong to distinct congruence classes. Suppose further that we have two corresponding families $(\alpha_i)_i, (\beta_j)_j$ of "exponents", taken from $\mathbb{N}^* \cup \{\infty\}$, such that $\mathcal{C} \equiv \Pi_{i=1\ldots l}\mathcal{C}_i^{\alpha_i}$ and $\mathcal{D} \equiv \Pi_{j=1\ldots l'}\mathcal{D}_j^{\beta_j}$ (where $\mathcal{C}^\infty$ denotes $!\mathcal{C}$). Since expansion does not duplicate single terms, we have that for any $j \leq l'$, there exists a unique $i \leq l$ such that $\mathcal{D}_j(\sigma) = \mathcal{C}_i$. Moreover, if we call J_i the set of indices j such that $\mathcal{C}_i \equiv \mathcal{D}_j(\sigma)$, we have that $J_i \neq \emptyset$, and

$$\alpha_i = \sum_{j \in J_i} \beta_j \,.$$

Let us now prove the direct implication of case 3. Using the notations we have introduced, we may write a decomposition of $(\alpha_i)_i$ into two families $(\alpha_i^1), (\alpha_i^2)$ of indices in $\mathbb{N} \cup \{\infty\}$ such that $\mathcal{C}_\delta \equiv \Pi_{i=1\ldots l}\mathcal{C}_i^{\alpha_i^\delta}$, $\delta = 1, 2$, with $\alpha_i = \alpha_i^1 + \alpha_i^2$. We then mimick this decomposition on the family $(\beta_j)_j$, and define two families $(\beta_j^1), (\beta_j^2)$ such that $\beta_j = \beta_j^1 + \beta_j^2$ and $\alpha_i^\delta = \sum_{j \in J_i} \beta_j^\delta$ for any i. We conclude by defining $\mathcal{D}_\delta$ as $\Pi_{j=1\ldots r'}\mathcal{D}_j^{\beta_j^\delta}$.

To establish case 4, we first remark that in this case every α_i is equal to ∞, so that every J_i contains at least one j such that $\beta_j = \infty$. This is enough to prove case 4 for a single context $\mathcal{C}$ (i.e. with $r = 1$).

In the general case where $\mathcal{C}$ is not single, the sets J_is need not necessarily have the same cardinality. We therefore define:

$$J^\infty = \{j.\ \beta_j = \infty\}\,, \quad r = \max_i\{\sharp(J_i \cap J^\infty)\}\,, \quad \text{and} \quad s = \max_{j \notin J^\infty} \beta_j \,.$$

Then using the structural congruence laws $!\mathcal{C} \equiv !\mathcal{C} \mid !\mathcal{C}$ and $!\mathcal{C} \equiv !\mathcal{C} \mid \mathcal{C}$ in order to "unfold" copies of some components of $\mathcal{D}$, one may make cardinalities match r and s. $\square$

The following result allows us to relate the active context decomposition of two structurally congruent processes.

Lemma B.2.12 (Active context decomposition) *Let P, Q be two terms. Then $P \equiv Q$ iff there exist two active contexts $\mathcal{C}$ and $\mathcal{D}$, two vectors of processes $\tilde{P}, \tilde{Q}$ such that $P \equiv \mathcal{C}[\tilde{P}], Q \equiv \mathcal{D}[\tilde{Q}]$, and a function σ such that:*

1. $\mathcal{C} \ \lhd_\sigma \ \mathcal{D}$;

2. *for any j,*

   - *either $[\,]_{\sigma(j)}$ and $[\,]_j$ are underneath the same capability cap_j in $\mathcal{C}$ and $\mathcal{D}$ respectively, and $P_{\sigma(j)} \ \equiv \ Q_j$*

- *or* $[\,]_{\sigma(j)}$ *and* $[\,]_j$ *are underneath inputs* $(x)[\,]\sigma(j)$ *and* $(x)[\,]_j$*, and* $P_{\sigma(j)} \equiv Q_j$

Proof: The direct implication is trivial, by taking $\mathcal{C} = \mathcal{D}$. For the reverse implication, we reason by induction on $\mathcal{C}$ using Lemma B.2.11 and an analogous of Lemma B.1.18 for $\equiv$. □

The characterization

We now exploit the definitions introduced above to provide an inductive characterisation of $\approx_{int}$, based on the notion of active context decomposition. We start by defining the inductive relation $\doteq$.

Definition B.2.13 (Inductive relation) *Given a process P, let $P \doteq (.)$ be the (least) predicate defined by induction on $\mathsf{ds}(()P)$ as follows:*
if there exist two active contexts $\mathcal{C}$ and $\mathcal{D}$, and two vectors of processes $\tilde{P}$, $\tilde{Q}$, such that the following conditions hold:

1. $P \equiv \mathcal{C}[\tilde{P}]$ *and* $Q \equiv \mathcal{D}[\tilde{Q}]$*;*
2. *there exists σ such that:*
   - $\mathcal{C} \ \lhd_\sigma \ \mathcal{D}$*;*
   - *for any j,*
     - *either $[\,]_{\sigma(j)}$ and $[\,]_j$ are underneath the same capability cap_j. Moreover, there are P'_j, Q'_j such that $P_{\sigma(j)} \stackrel{\mathsf{cap}_j}{\Longrightarrow} P'_j \doteq Q_j$, $Q_j \stackrel{\mathsf{cap}_j}{\Longrightarrow} Q'_j$, and $P_{\sigma(j)} \ \doteq \ Q'_j$;*
     - *or $[\,]\sigma(j)$ and $[\,]_j$ are underneath inputs $(x)[\,]\sigma(j)$ and $(x)[\,]_j$. Moreover, there are P'_j, Q'_j, $m \notin \mathrm{fn}(P) \cup \mathrm{fn}(Q)$, such that $\langle m\rangle \mid (x)P_{\sigma(j)} \stackrel{\mathsf{cap}_j}{\Longrightarrow} P'_j \doteq Q_j$, $\langle m\rangle \mid (x)Q_j \stackrel{\mathsf{cap}_j}{\Longrightarrow} Q'_j$, and $P_{\sigma(j)} \ \doteq \ Q'_j$,*

then $P \ \doteq \ Q$.

Lemma B.2.14 (Correctness of Definition B.2.13) *Definition B.2.13 is inductive.*

Proof: The point to check is that $\mathsf{ds}(()P'_j) < \mathsf{ds}(()P)$. This is a consequence of $\mathsf{ds}(()P_{\sigma(j)}) < \mathsf{ds}(()P)$ and $\mathsf{ds}(()P'_j) \leq \mathsf{ds}(()P_{\sigma(j)})$ (by Lemma B.2.2). □

Remark B.2.15

- *The fact that $\doteq$ is inductive does not imply that it is a decidable relation. Indeed, in the second clause, first case of the requirement on j, there is a search for P'_j, Q'_j in the (potentially infinite) set of images of $P_\sigma(j)$ and Q_j. We will show in Section B.4 that $\doteq$ is actually not decidable in general.*
- *The intuitive idea behind the use of expansion in the definition of $\doteq$ is that we must be able to satisfy clause 4 in Lemma B.1.18.*

The following important result can now be assessed:

Theorem B.2.16 *Relations $\approx_{int}$ and $\doteq$ coincide on MA.*

Proof: Let us first assume that $P \approx_{int} Q$. We reason by induction on $\mathsf{ds}(()P)$ and consider a decomposition of P under the form $\mathcal{C}[\tilde{P}]$ such that $\mathcal{C}$ is normalised. We then proceed by induction on $\mathcal{C}$; using Lemma B.1.18 we may find some $\mathcal{C}', \tilde{Q}$ such that $Q \equiv \mathcal{C}'[\tilde{Q}]$ and some function σ satisfying the following conditions: (i) $\mathcal{C} \lhd_\sigma \mathcal{C}'$; (ii) for any j,

- either $[]_{\sigma(j)}$ and $[]_j$ are underneath the same capability cap_j, and there are P'_j, Q'_j such that $P_{\sigma(j)} \stackrel{\mathsf{cap}_j}{\Longrightarrow} P'_j \approx_{int} Q_j$, $Q_j \stackrel{\mathsf{cap}_j}{\Longrightarrow} Q'_j$, and $P_{\sigma(j)} \approx_{int} Q'_j$.
- or $[]_{\sigma(j)}$ and $[]_j$ are underneath input (x), and there are P'_j, Q'_j such that $\langle m\rangle \mid (x)P_{\sigma(j)} \stackrel{\mathsf{cap}_j}{\Longrightarrow} P'_j \simeq Q_j$, $\langle m\rangle \mid (x)Q_j \stackrel{\mathsf{cap}_j}{\Longrightarrow} Q'_j$, and $P_{\sigma(j)} \simeq Q'_j$.

The function σ is defined as in the case of replication in Lemma B.1.18.

Now if $\mathsf{ds}(()P) = 0$, this means that $\tilde{P}, \tilde{Q}$ are empty and then $P \simeq Q$. Otherwise, we have by induction that $P_{\sigma(j)} \stackrel{\mathsf{cap}_j}{\Longrightarrow} P'_j \simeq Q_j$, and $Q_j \stackrel{\mathsf{cap}_j}{\Longrightarrow} Q'_j$ and $P_{\sigma(j)} \simeq Q'_j$, and again $P \simeq Q$. So the first implication is proved.

The other implication follows the same pattern (induction on $\mathsf{ds}(()P)$, then on the active context of P), using the results of Lemma B.2.11 instead of Lemma B.1.18. $\square$

B.2.3 Characteristic formulas for active contexts

The next step in order to establish completeness is to show that the logic allows one to capture the active context of a term. This is done by defining a function that computes, starting from a normalized active context, a *formula context* (i.e. a formula with holes in it, along the lines of the definition of plain contexts). In order to do that, we first need some more results from appendix A

Auxiliary results about the expressiveness of AL

We have recalled in B.1.4 the definability of modalities $\langle\langle \mathsf{cap}\rangle\rangle.\,\mathcal{A}$ and $\langle\langle ?n\rangle\rangle.\,\mathcal{A}$. We will also need the *duals* of these modalities, given by the following result:

Lemma B.2.17 (cf. Appendix A) *For all* cap*, there is a formula context* $[\![\mathsf{cap}]\!].\,[]$ *such that for all process* P *and formula* $\mathcal{A}$*,*

$$P \models [\![\mathsf{cap}]\!].\,\mathcal{A} \quad \textit{iff} \quad \exists P'.\ P \equiv \mathsf{cap}.\,P' \textit{ and } \forall P''.\ P' \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow} P'' \textit{ implies } P'' \models \mathcal{A}\,.$$

For all n*, there is a formula context* $[\![?n]\!].\,[]$ *such that for all process* P *and formula* $\mathcal{A}$*,*

$$P \models [\![?n]\!].\,\mathcal{A} \quad \textit{iff} \quad \exists P'.\ P \equiv (x)P' \textit{ and } \forall P''.\,(x)P' \mid \langle n\rangle \Rightarrow P'' \textit{ implies } P'' \models \mathcal{A}\,.$$

In order to express all operators that can appear in active contexts, we need to be able to handle replication. AL allows one to express a restricted form of replication on formulas. We would like to be able to define a formula $!\mathcal{A}$ expressing that infinitely many components in parallel satisfy $\mathcal{A}$. However, this is only possible for certain formulas $\mathcal{A}$, under the condition that the models of $\mathcal{A}$ belong to some restricted class of processes.

More precisely, let $\mathcal{E}$ be a set of processes. We say that a formula is *sequentially selective* on $\mathcal{E}$ if all its models in $\mathcal{E}$ have the same sequentiality degree, and *depth selective* on $\mathcal{E}$ if all its models in $\mathcal{E}$ have the same depth degree. These forms of selectiveness are necessary in the construction of formulas for persistent terms.

Lemma B.2.18 (cf Appendix A) *For all* cap*, there exists a formula context* $\mathit{Rep}_{\mathsf{cap}}()$ *such that for all process* $P \in \mathcal{E}$ *and for all formula* $\mathcal{A}$ *sequentially selective on* $\mathcal{E}$*, whose models are only of the form* $\mathsf{cap}.\,R$*,*

$$P \models \mathit{Rep}_{\mathsf{cap}}(\mathcal{A}) \quad \text{iff} \quad \exists P_1, \ldots, P_r.\ P \equiv !P_1 \mid (!)P_2 \mid \ldots \mid (!)P_r \text{ and, } P_i \models \mathcal{A}\,, i = 1 \ldots r\,.$$

For all n*, there is a formula* $\langle !n \rangle$ *such that*

$$P \models \langle n \rangle \quad \text{iff} \quad P \equiv !\langle n \rangle\,.$$

For all n*, there is a formula context* $\mathit{Rep}_{input}()$ *such that for all process* P *and for all formula* $\mathcal{A}$ *sequentially selective whose models are only of the form* $(x)P$*,*

$$P \models \mathit{Rep}_{input}(\mathcal{A}) \quad \text{iff} \quad \exists P_1, \ldots, P_r.\ P \equiv !P_1 \mid (!)P_2 \mid \ldots \mid (!)P_r \text{ and, } P_i \models \mathcal{A}\,, i = 1 \ldots r\,.$$

We have similar results for the replicated version of the *dual* modalities. We shall also need ability to test free name occurrences in a process, which is given by the following result:

Lemma B.2.19 (cf. Appendix A) *For any name* n*, there exists a formula* Ⓒn *such that for any* P*,* $P \models$ Ⓒn *iff* $n \in \mathrm{fn}(P)$*.*

The notion of depth selectiveness allows us to build formulas that capture replicated ambients:

Lemma B.2.20 (cf Appendix A) *For all* n*, there is a formula context* $!n[.]$ *such that for all process* $P \in \mathcal{E}$ *and for all formula* $\mathcal{A}$ *depth selective on* $\mathcal{E}$*,*

$$P \models !n[\mathcal{A}] \quad \text{iff} \quad \exists P_1, \ldots, P_r.\ P \equiv !P_1 \mid (!)P_2 \mid \ldots \mid (!)P_r \text{ and, } P_i \models n[\mathcal{A}]\,, i = 1 \ldots r\,.$$

Capturing active contexts

Formulas for active contexts. The active contexts we consider now are *normalised*, as defined above, in the sense that replication is only applied to single terms (we can suppose this without loss of generality thanks to the $\equiv$ laws we have for replication). The formula context we produce from an active context of this form associates two holes $[]_i^{\Diamond}$ and $[]_i^{\Box}$ to each hole $[]_i$ in the initial context. The formula contexts we define are then to be filled with two vectors of formulas $\widetilde{F^{\Diamond}}$ and $\widetilde{F^{\Box}}$, matching the corresponding holes.

Definition B.2.21 (Active context formula) *Consider an active normalised context* $\mathcal{C}$*. We define its associated context formula* $\mathcal{C}^*$ *by the equations given below. In particular, we associate to every prefixed hole* $[]_i$ *of* $\mathcal{C}$ *a context formula having two holes, that we denote* $[]_i^{\Diamond}$ *and* $[]_i^{\Box}$*. Hence the arity of* $\mathcal{C}^*$ *is twice the* *arity of $\mathcal{C}$.*

$$
\begin{array}{rcl}
(\mathbf{0})^* & \stackrel{\text{def}}{=} & 0 \\
(\mathcal{C}_1 \mid \mathcal{C}_2)^* & \stackrel{\text{def}}{=} & \mathcal{C}_1^* \mid \mathcal{C}_2^* \\
(n[\mathcal{C}])^* & \stackrel{\text{def}}{=} & n[\mathcal{C}^*] \\
(\mathsf{cap}.\,[]_i)^* & \stackrel{\text{def}}{=} & \langle\langle \mathsf{cap} \rangle\rangle.\,[]_i^\diamond \;\wedge\; [\![\mathsf{cap}]\!].\,[]_i^\Box \\
(\langle n \rangle)^* & \stackrel{\text{def}}{=} & \langle n \rangle \\
\big((x)\,P\big)^* & \stackrel{\text{def}}{=} & \exists x.\ \neg ©x \;\wedge\; \langle\langle ?x \rangle\rangle\ \big(\neg\langle x \rangle \mid \langle\langle ?x \rangle\rangle.\top\big) \;\wedge\; []_i^\diamond\big) \\
 & & \qquad \wedge\ [\![?x]\!].\,\big((\langle x \rangle \mid \langle\langle ?x \rangle\rangle.\top\big) \;\vee\; []_i^\Box\big) \\
(!n[\mathcal{C}])^* & \stackrel{\text{def}}{=} & !n[\mathcal{C}^*] \\
(!\mathsf{cap}.\,[]_i)^* & \stackrel{\text{def}}{=} & \mathit{Rep}_{\mathsf{cap}}((\mathsf{cap}.\,[]_i)^*) \\
(!\langle n \rangle)^* & \stackrel{\text{def}}{=} & !\langle n \rangle \\
(!(x).\,[]_i)^* & \stackrel{\text{def}}{=} & \mathit{Rep}_{input}\big(\big((x)[]_i\big)^*\big)
\end{array}
$$

The interpretation of the formulas for active contexts depends on hypotheses about both the processes that are used to fill the holes in the active contexts and the formulas that are used to fill the holes in the formula contexts. In order to state the interpretation, we have to introduce formally the notion of frozen process.

Frozen processes.

As we will see, the interpretation of a judgment of the form $P \models \mathcal{C}^*[\tilde{F}]$ depends on two kinds of hypotheses:

- first, the formulas $F_i^\diamond$ and $F_i^\Box$, that are used to fill the holes should be expressive enough on some fixed set of processes $\mathcal{E}$ (as it was the case for replication formulas);

- second, not only should process P belong to $\mathcal{E}$, but we furthermore require that terms issued from P and from its subterms also do.

In the second point above, we basically want to guarantee some properties about those subterms of P that appear under a prefix when P performs a reduction. We now introduce the formal definitions that allow us to state these properties.

Definition B.2.22 (Sets of processes $\mathcal{E}_P$, $\mathsf{frozen}^1(P)$ and $\mathsf{frozen}(P)$)

For any process P, we define $\mathcal{E}_P \stackrel{\text{def}}{=} \{P',\ \exists\ \mathsf{cap}.\ P \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} P'\}$.

Let N be a set of names; the set $\mathsf{frozen}^1(P)$ is defined by structural induction on P as follows:

- $\mathsf{frozen}^1(\mathbf{0}) = \mathsf{frozen}^1(\langle n \rangle) = \emptyset$;
- $\mathsf{frozen}^1(P_1 \mid P_2) = \mathsf{frozen}^1(P_1) \cup \mathsf{frozen}^1(P_2)$;
- $\mathsf{frozen}^1(\mathsf{cap}.\,P) = \{P\}$;
- $\mathsf{frozen}^1((x)P) = \{P\{^n\!/\!_x\}\ :\ n \in N\}$.

The set $\mathsf{frozen}(P)$ *is defined by induction on the sequentiality degree of* P *by setting:*

$$\mathsf{frozen}(P) \stackrel{\text{def}}{=} \mathsf{frozen}^1(P) \cup \bigcup_{P' \in \mathsf{frozen}^1(P)} \mathsf{frozen}(P') .$$

If P, P' are two congruent terms, then, modulo $\equiv$, $\mathsf{frozen}^1(P) = \mathsf{frozen}^1(P')$, and ditto for $\mathsf{frozen}(P) = \mathsf{frozen}(P')$. Hence these sets (in their quotiented version w.r.t. $\equiv$) are uniquely determined by the structural congruence class of P.

Lemma B.2.23 (Finiteness of $\mathsf{frozen}(P)$) *For any* $P \in \mathsf{MA}$*, if* N *is finite, then, up to* $\equiv$*,* $\mathsf{frozen}(P)$ *is finite.*

Proof: By induction on P □

Not only is $\mathsf{frozen}(P)$ finite, but, as expressed by the following result, this construction allows us to somehow 'isolate' the future of P, and to focus on a finite set of terms (hence the name 'predictive lemma').

Lemma B.2.24 (Predictive lemma) *Let* P, Q *be two processes such that* $P \rightarrow Q$ *or* $P \stackrel{\mathsf{cap}}{\rightarrow} Q$ *for some* cap*, and assume* $\mathrm{fn}(P) \subseteq N$*. Then* $\mathsf{frozen}(Q) \subseteq \mathsf{frozen}(P)$ *when quotiented by* $\equiv$*.*

Proof: The result is easy for $\stackrel{\mathsf{cap}}{\rightarrow}$. For $\rightarrow$, we reason by induction on the derivation of $P \rightarrow Q$. The cases corresponding to movement transitions follow from $\stackrel{\mathsf{cap}}{\rightarrow}$. So the only way a reduction could alter the set of frozen terms is through name substitutions generated by communications, and this is handled by the condition $\mathrm{fn}(P) \subseteq N$. □

Remark B.2.25 *We believe that there is no counterpart of this lemma for a calculus having both communications and name restriction, such as, e.g., the* π*-calculus. For this reason, we do not know whether our methodology could be adapted in order to establish completeness of the Hennessy-Milner logic in absence of image-finiteness.*

Interpretation of the context formulas. We are now ready to explain the interpretation of the formulas of Definition B.2.21.

Lemma B.2.26 (Characterisation of active contexts) *Let* $\mathcal{C}$ *be a normalised active context. Let* $\tilde{F^\diamond}$ *and* $\tilde{F^\square}$ *be two vectors of formulas (called the* filling formulas*) whose lengths are equal to the arity of* $\mathcal{C}^*$*. Let us also consider a set of processes* $\mathcal{E}$*, and assume that the filling formulas are such that for any index* i *of a hole* $\mathsf{cap}_i.[]_i$ *or* $(x)\,[]_i$ *of* $\mathcal{C}$*, the formula*

$$F_i \stackrel{\text{def}}{=} \langle\langle \mathsf{cap}_i \rangle\rangle. F_i^\diamond \;\wedge\; [\![\mathsf{cap}_i]\!]. F_i^\square$$

and respectively for input

$$F_i \stackrel{\text{def}}{=} \exists x.\ \neg \copyright x \wedge \langle\langle ?x \rangle\rangle \left(\neg(\mathcal{F}_{\langle x \rangle} \mid \langle\langle ?x \rangle\rangle. \top) \wedge F_i^\diamond\right) \wedge [\![?x]\!]. \left((\mathcal{F}_{\langle x \rangle} \mid \langle\langle ?x \rangle\rangle. \top) \vee F_i^\square\right)$$

is sequentially selective on $\mathcal{E}$*.*

Then for any term P *such that* $N = \mathrm{fn}(P)$ *and* $\mathsf{frozen}(P) \subseteq \mathcal{E}$*,* $P \models \mathcal{C}^*[F^\diamond, F^\square]$ *iff there exists an active context* $\mathcal{C}_1$*, a vector of processes* $\tilde{P_1}$*, and a function* σ *such that* $P \equiv \mathcal{C}_1[\tilde{P_1}]$ *and :*

1. $\mathcal{C} \ \lhd_\sigma \ \mathcal{C}_1$*;*

2. for any index j *of a hole* $\mathsf{cap}_j.\,[]_j$ *of* $\mathcal{C}_1$*,* $\mathsf{cap}_j.\,P_j \models F_{\sigma(j)}$*, and for any index* j *of a hole* $(x)\,[]_j$ *of* $\mathcal{C}_1$*,* $(x)\,P_j \models F_{\sigma(j)}$*.*

Proof: Let us consider P such that $\mathsf{frozen}(P) \subseteq \mathcal{E}$ and reason by induction on $\mathcal{C}$:

- if $\mathcal{C} = \mathbf{0}$ then we can take $\mathcal{C}_1 = \mathbf{0}$ (and we have $P \equiv \mathbf{0}$).

- if $\mathcal{C} = n[\mathcal{C}_0]$ then $P \models \mathcal{C}^*[\tilde{F^\diamond}, \tilde{F^\Box}]$ iff $P \equiv n[P_0]$ and $P_0 \models \mathcal{C}_0^*[\tilde{F^\diamond}, \tilde{F^\Box}]$. By induction, we may find $\mathcal{C}_{1,0}$ such that $P_0 \equiv \mathcal{C}_{1,0}[\tilde{P}]$ for some $\tilde{P}$ and a function σ having the right properties. Then we may establish the result taking $\mathcal{C}_1 = n[\mathcal{C}_{1,0}]$ and σ.

- if $\mathcal{C} = \mathcal{C}_a \mid \mathcal{C}_b$ then $P \equiv P_a \mid P_b$ with $P_i \models \mathcal{C}_i^*[\tilde{F^\diamond}, \tilde{F^\Box}]$ $(i = a, b)$. This gives by induction contexts $\mathcal{C}_{a,1}$ and $\mathcal{C}_{b,1}$ and functions σ_a and σ_b. Without loss of generality, we may suppose the indices of the holes occurring in these contexts to be disjoint, so that we can define σ, a function which extends both σ_a and σ_b. Define then $\mathcal{C}_1 \stackrel{\text{def}}{=} \mathcal{C}_{a,1} \mid \mathcal{C}_{b,1}$. We have $\mathcal{C} = \mathcal{C}_a \mid \mathcal{C}_b \equiv \mathcal{C}_{a,1}(\sigma_a) \mid \mathcal{C}_{b,1}(\sigma_b) = \mathcal{C}_1(\sigma)$, and condition 2 also holds by induction.

- if $\mathcal{C} = \mathsf{cap}_i.\,[]_i$, then vectors $\tilde{F^\diamond}$ and $\tilde{F^\Box}$ have only one component. Moreover, $\mathcal{C}^*[\tilde{F^\diamond}, \tilde{F^\Box}]$ is exactly F_i, and $P \ \equiv \ \mathsf{cap}_i.\,P'$ with $\mathsf{cap}_i.\,P' \models F_i$. So we may take $\mathcal{C}_1 = \mathcal{C}$ and just any function.

- the case $\mathcal{C} = (x)\,[]_i$ is identical.

- if $\mathcal{C} = !\mathsf{cap}_i.\,[]_i$, then $\mathcal{C}^*[\tilde{F^\diamond}, \tilde{F^\Box}]$ is $\mathsf{Rep}_{\mathsf{cap}}(F_i)$ and we may apply Lemma B.2.18 with $\mathcal{E}$ (as by hypothesis F_i is sequentially selective on $\mathcal{E}$), so that there are $r \geq 1, s \geq r, P_i$ $(1 \leq i \leq s)$ such that $P \equiv \Pi_{1 \leq i \leq r}!\mathsf{cap}.\,P_i \mid \Pi_{r+1 \leq i \leq s}\mathsf{cap}.\,P_i$, and $\mathsf{cap}.\,P_i \models \mathcal{A}$ for all $1 \leq i \leq s$. Then we set $\mathcal{C}_1 = \Pi_{1 \leq i \leq r}!\mathsf{cap}.\,[]_i \mid \Pi_{r+1 \leq i \leq s}\mathsf{cap}.\,[]_i$, and σ can be freely chosen as long as it is constant on these indices. Conditions 1 and 2 are then satisfied.

- if $\mathcal{C} = !n[\mathcal{C}_1]$, we reason on the same way. Just note that the depth selective hypothesis is there expressed by the inductive hypothesis.

- the case $\mathcal{C} = !\mathsf{cap}_i[]_i$ is identical.

- the cases $\mathcal{C} = \langle n \rangle$ and $\mathcal{C} = !\langle n \rangle$ are straightforward.

$\Box$

B.2.4 Completeness, infinite case

We can now derive completeness of $\approx_{int}$ w.r.t. $=_L$, and hence deduce that $\approx_{int}$ coincides with $=_L$ on MA. This is achieved by exploiting Lemma B.2.26 (characteristic formulas for active contexts), using Lemma B.2.23. As said above, the latter result says that given a fixed active context, we can restrict ourselves to the exploration of a *finite* set of processes. This makes it possible to consider a notion of *local characteristic formula*, that we define as follows:

Definition B.2.27 (Local characteristic formula) *Let $\mathcal{E}$ be a set of terms, P a term and F a formula. We will say that F is a characteristic formula for P on $\mathcal{E}$ if for any $Q \in \mathcal{E}$, $Q \models F$ iff $Q \approx_{int} P$.*

With this definition, completeness of $\approx_{int}$ boils down to the existence of a characteristic formula of P on $\{P, Q\}$ for any terms P, Q. We shall exploit this idea in a more refined fashion, to establish completeness by reasoning inductively on the sequentiality degree of terms. We will namely work with the sets $\mathcal{E}_P$ (see Def. B.2.22), and prove that for any P and Q, there exists a characteristic formula of P on $\mathcal{E}_Q$.

Lemma B.2.28 (Characteristic formulas on $\mathcal{E}_Q$) *For any two terms P, Q of* MA*, there is a characteristic formula $F_{P,Q}$ for P on $\mathcal{E}_Q$, or in other words:*

- $P \models F_{P,Q}$
- *for any Q' and* cap *such that $Q \overset{\langle \mathsf{cap} \rangle}{\Rightarrow} Q'$, if $Q' \models F_{P,Q}$, then $Q' \approx_{int} P$.*

Proof: Let P, Q be two terms. We reason by induction on $\mathsf{ds}(()P)$ to define $F_{P,Q}$:

- if $\mathsf{ds}(()P) = 0$, then P is a pure tree and has an obvious characteristic formula identical to itself.
- Let us now assume that $\mathsf{ds}(()P) > 0$.
  - if $\mathsf{ds}(()P) > \mathsf{ds}(()Q)$, we take $F_{P,Q} = F_{\mathsf{ds}(()P)}$. Then $P \models F_{P,Q}$. As for any Q' and cap such that $Q \overset{\langle \mathsf{cap} \rangle}{\Rightarrow} Q'$, $\mathsf{ds}(()P) > \mathsf{ds}(()Q) \geq \mathsf{ds}(()Q')$, $Q' \not\models F_{P,Q}$ (and $Q' \not\approx_{int} P$).
  - if $\mathsf{ds}(()P) \leq \mathsf{ds}(()Q)$, the formula we will construct will make an intensive use of the induction hypothesis. We first set $N = \mathrm{fn}(P) \cup \mathrm{fn}(Q) \uplus N'$ where N' is an extra set of fresh names such that $\sharp N' > \max(\mathsf{ds}(()P), \mathsf{ds}(()Q))$. Doing that, when considering any $R \in \mathsf{frozen}(P) \cup \mathsf{frozen}(Q)$ we may find a fresh name for it inside N. Let us precisely tell now how we use the induction hypothesis. By induction, we actually get formulas F_{P_i,Q_j} and F_{Q_j,P_i} for any $P_i \in \mathsf{frozen}(P)$ and $Q_j \in \mathsf{frozen}(Q)$ for these various reasons: we have by induction formulas F_{P_i,Q_j} for any $P_i \in \mathsf{frozen}(P)$ and $Q_j \in \mathsf{frozen}(Q)$; moreover, we may also obtain formulas F_{Q_j,P_i} by induction when $\mathsf{ds}(()Q_j) < \mathsf{ds}(()P)$; and finally, when $\mathsf{ds}(()Q_j) \geq \mathsf{ds}(()P) > \mathsf{ds}(()P_i)$, we may define $F_{Q_j,P_i} \overset{\text{def}}{=} F_{\mathsf{ds}(()Q_j)}$ as above. So we have the formulas F_{P_i,Q_j} and F_{Q_j,P_i} for any $P_i \in \mathsf{frozen}(P)$ and $Q_j \in \mathsf{frozen}(Q)$ with a valid induction hypothesis on them.
    - <u>construction of the formula</u>
      Let us set
      $$\mathcal{E} \overset{\text{def}}{=} \mathsf{frozen}(P) \cup \mathsf{frozen}(Q)\,.$$
      $\mathcal{E}$ is finite because of Fact B.2.23, and we may therefore define the formulas
      $$F_i^{\Diamond} \overset{\text{def}}{=} \bigwedge_{R \in \mathcal{E}} F_{P_i,R} \qquad \text{and} \qquad F_i^{\Box} \overset{\text{def}}{=} \neg \bigvee_{R \in \mathcal{E} \setminus \mathcal{E}_{P_i}} F_{R,P_i}\,.$$

Then for the decomposition $P = \mathcal{C}[\tilde{P}]$ for an active context $\mathcal{C}$ we take

$$F_{P,Q} = \mathcal{C}^*[\tilde{F^\diamond}, \tilde{F^\Box}]\,.$$

* A key result

Let us first consider the formulas $F_i = \langle\langle \mathsf{cap}_i \rangle\rangle.\, F_i^\diamond \wedge [\![\mathsf{cap}_i]\!].\, F_i^\Box$. We first prove that for any $R \in \mathcal{E}$, $\mathsf{cap}_i.\, R \models F_i$ iff $\mathsf{cap}_i.\, R \ \approx_{int}\ \mathsf{cap}_i.\, P_i$. In particular, this will entail that F_i is sequentially selective on $\mathcal{E}$.

  - Let $R \in \mathcal{E}$ be such that $\mathsf{cap}_i.\, R \models F_i$. Then there is some R' such that $R \stackrel{\mathsf{cap}_i}{\Longrightarrow} R'$ and $R' \models F_i^\diamond$. In particular, $R' \models F_{P_i,R}$, and hence (by induction) $R' \ \approx_{int}\ P_i$. Moreover, $R \models F_i^\Box$, and since $R \in \mathcal{E}$, R is in $\mathcal{E}_{P_i}$. This entails that $R \stackrel{\mathsf{cap}_i}{\Longrightarrow} \approx_{int} \ P_i \stackrel{\mathsf{cap}_i}{\Longrightarrow} \approx_{int}\ R$, and by definition of $\approx_{int}$, $\mathsf{cap}_i.\, R \ \approx_{int}\ \mathsf{cap}_i.\, P_i$.
  - Let $R \in \mathcal{E}$ be such that $\mathsf{cap}_i.\, R \ \approx_{int}\ \mathsf{cap}_i.\, P_i$. We want to prove that $\mathsf{cap}_i.\, R \models F_i$. By correction, we can restrict ourselves to prove this for $R = P_i$. $\mathsf{cap}_i.\, P_i \models \langle\langle \mathsf{cap}_i \rangle\rangle.\, F_i^\diamond$ trivially holds. We thus are left with proving $\mathsf{cap}_i.\, P_i \models [\![\mathsf{cap}_i]\!].\, F_i^\Box$. Let P' be such that $P_i \stackrel{\mathsf{cap}_i}{\Longrightarrow} P'$ and assume by absurd there exists some $R \ \in \mathcal{E} \setminus \mathcal{E}_{P_i}$ such that $P' \models F_{R,P_i}$. Then P' would be bisimilar to R, and $P_i \stackrel{\mathsf{cap}_i}{\Longrightarrow} \approx_{int}\ R$, which means $R \ \in \mathcal{E}_{P_i}$, hence a contradiction. Therefore, $P' \models F_i^\Box$. As this can be proved for any P', $\mathsf{cap}_i.\, P_i \models [\![\mathsf{cap}_i]\!].\, F_i^\Box$.

Similarly, for a hole $(x)[]_i$, $F_i = \big((x)[]_i\big)^*[F_i^\diamond, F_i\Box]$, and $R\{n/x\} \in \mathcal{E}$, then $(x)\, R \models F_i$ iff $(x)\, R \approx_{int} (x)\, P_i$, and in particular F_i is sequentially selective. The proof of this goes almost the same as for capabilities, except a tiny trick we have to use with names:

  - If $R\{n/x\} \in \mathcal{E}$ is such that $(x)R \models F_i$, then there is $m \notin \mathrm{fn}(R)$ such that $(x)R \models \langle\langle ?m \rangle\rangle\big(\neg\langle m\rangle \mid \langle\langle ?m\rangle\rangle\top) \ \wedge\ F_i\big)$. Since $N' \subseteq N$ has been taken with $\sharp N' \geq \max(\mathsf{ds}(()P), \mathsf{ds}(()Q))$, there is $m' \in N - (\mathrm{fn}(R) \cup \mathrm{fn}(P) \cup \mathrm{fn}(Q)$. So we have $(x)R \models \langle\langle ?m' \rangle\rangle\big(\neg\langle m\rangle \mid \langle\langle ?m\rangle\rangle\top) \ \wedge\ F_i\big)$ (note that by our construction $\mathrm{fn}(F_{P,Q}) \subseteq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$). So there is R' such that $\langle m' \rangle \mid (x)R \Rightarrow R'$ and $R' \not\equiv \langle m' \rangle \mid (x)R''$ for some R'', so that in particular $R\{m'/x\} \Rightarrow R'$, that is $R' \in \mathcal{E}$. Morevoer, $R' \models F_i^\diamond$, so $R' \approx_{int} P_i\{m'/x\}$. Similarly, using the second clause of the formula $\big((x)[]_i)\big)^*[F_i^\diamond, F_i^\Box]$, one get that $R\{m'/x\} \in \mathcal{E}_{P_i\{m'/x\}}$, which shows that $(x)R \approx_{int} (x)P_i$.
  - Reciproquely, from $(x)R \approx_{int} (x)P_i$ we may deduce $(x)R \models F_i$ choosing m' as a guess for $\exists x$ in F_i.

* Putting it all together

We now come to the proof that formula $F_{P,Q}$ introduced above satisfies the expected properties. First, we show that $P \models F_{P,Q}$: as $P = \mathcal{C}[\tilde{P}]$, this is equivalent by Lemma B.2.26 to $\mathsf{cap}_i.\, P_i \models F_i$ for any i, and is a consequence of the previous reasoning.

Consider now a capability cap and a process Q' such that $Q \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} Q'$; by Lemma B.2.24, $\mathsf{frozen}(Q') \subseteq \mathcal{E}$. Let us moreover assume that

$Q' \models F_{P,Q}$. By Lemma B.2.26, this means there is a decomposition $Q' \equiv \mathcal{C}'[\tilde{Q}']$ and a function σ such that $\mathsf{cap}_j . Q'_j \models F_{\sigma(j)}$. As a consequence of the previous reasoning, we have that $\mathsf{cap}_j . Q_j \approx_{int} \mathsf{cap}_j . P_{\sigma(j)}$. So $Q' \doteq P$, which concludes the proof.

□

We finally have:

Theorem B.2.29 (Completeness of $\approx_{int}$) *In* MA, $=_L \subseteq \approx_{int}$.

Proof: Let P, Q be two terms such that $P \not\approx_{int} Q$. We apply Lemma B.2.28 to processes P and Q. We have $P \models F_{P,Q}$. We then have $Q \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} Q$ (for any cap), and $Q \models F_{P,Q}$ implies $P \approx_{int} Q$. Hence, since by hypothesis $P \not\approx_{int} Q$, $Q \not\models F_{P,Q}$, and $P \neq_L Q$. □

Corollary B.2.30 *In* MA*, relations* $=_L$*,* $\approx_{int}$ *and* $\doteq$ *coincide.*

Remark B.2.31 *As mentioned before, this completeness proof in absence of an image-finiteness hypothesis is rather specific to the calculus we study, the public Mobile Ambients calculus. It seems far from obvious to adapt our method to a different model, for instance to the* π*-calculus. First, it is not so clear how to derive an operational characterisation of the spatial logic equivalence in this case. More importantly, we do not see how to derive the essential result given by Lemma B.2.24 (the* 'predictive lemma'*) in a calculus having both communication and the possibility to supply an infinite number of names (as is the case in the* π*-calculus).*

B.3 Axiomatization of $=_L$ on MA_{IF}^{syn}

In this section, we give an axiomatization of $=_L$ on MA_{IF}^{syn}, a subcalculus of MA in which image-finiteness is guaranteed by a syntactical condition (Definition B.3.2 below). We shall see that AL is very intensional, in the sense that $=_L$ is 'almost equal' to $\equiv$. More precisely, we show that logical equivalence coincides with $\equiv_E$, a relation we define by extending structural congruence with one axiom. We establish the following chain of (dis)equalities, on MA_{IF}^{syn}:

$$\equiv \;\subsetneq\; \equiv_E \;=\; =_L \;=\; \approx_{int} \;\subsetneq\; \approx .$$

We then move to the study of a variant of MA_{IF}^{syn} in which communication is synchronous, and show that logical equivalence coincides with $\equiv$ on this calculus.

B.3.1 Extensionality and intensionality

We use the characterisation of $=_L$ as $\approx_{int}$ to compare logical equivalence with relations $\approx$ (extensionality) and $\equiv$ (intensionality). We start by studying the difference between $=_L$ and $\approx$.

Non-extensionality

Theorem B.3.1 *Relation* $=_L$ *is strictly included in* $\approx$.

Proof: The inclusion follows from $=_L \subseteq \approx_{int}$ and $\approx_{int} \subseteq \approx$ (the second inclusion is essentially a consequence of the congruence of $\approx_{int}$).

The strictness of the inclusion is proved by the following laws, that are valid for $\approx$ but not for $\approx_{int}$:

1. $M.\,P = M \mid P$, where M is a capability, if P is of the form $M.\cdots.M.\mathbf{0}$.
2. $(x)\,P = (x)\,\mathbf{0} \mid P$, if P is of the form $(x)\ \cdots (x)\,\mathbf{0}$.
3. $(x)\,\langle x\rangle = \mathbf{0}$.

$\square$

The third axiom is typical for behavioural equivalences in calculi where communication is asynchronous. The first and second axiom schemata are not valid for $=_L$ because the logic allows one to intensionally observe parallel compositions. Here are two concrete examples of equalities derived from these axiom schemata:

$$\mathsf{in}\,n.\,\mathsf{in}\,n \quad = \quad \mathsf{in}\,n \mid \mathsf{in}\,n \qquad\qquad (x)\,(y)\,\mathbf{0} \quad = \quad (x)\,\mathbf{0} \mid (y)\,\mathbf{0}$$

We do not know whether laws (1-3) are all the laws that make $=_L$ finer than $\approx$.

Intensionality

We now provide a precise account of the difference between $=_L$ and $\equiv$, in the setting of the subcalculus MA_{IF}^{syn}, defined as follows:

Definition B.3.2 (MA_{IF}^{syn}) *The subcalculus* MA_{IF}^{syn} *is defined by the grammar:*

$$P \ ::= \ \mathbf{0} \ \big|\ P \mid P \ \big|\ !P \ \big|\ n[P] \ \big|\ \mathsf{cap}.\,P_0 \ \big|\ \langle n\rangle \ \big|\ (n)P_0$$

where P_0 is a finite process (without !).

MA_{IF}^{syn} strictly contains the finite calculus we considered for the completeness proof in B.1.4. Therefore, Corollary B.1.24 does not apply, but Corollary B.2.30, which holds for the whole calculus, does. Like MA_{IF} (see Remark B.1.21), MA_{IF}^{syn} is image-finite. While in the former subcalculus this property is guaranteed at a semantical level, in MA_{IF}^{syn} it follows from a syntactic restriction. We will see in Section B.4 that MA_{IF}^{syn} is Turing complete.

The only difference we observe between $=_L$ and $\equiv$ on MA_{IF}^{syn} is given by a form of *eta* conversion:

Definition B.3.3 *We let* normalised structural congruence*, written $\equiv_E$, be the relation defined by the rules of $\equiv$ plus the eta law (see Definition B.1.2).*

Lemma B.3.4 $\equiv_E \subseteq \approx_{syn}$.

Proof: It is enough to prove that given P, Q such that $P \rightarrow_\eta Q$, we have $P \approx_{int} Q$. We reason by induction on P, following Lemma B.1.18. In that lemma, the situations corresponding to the operators of parallel composition, ambients and capability prefixes are easy because of commutation properties of $\rightarrow_\eta$. In the cases of $\mathbf{0}$ and of messages, there is no redex for $\rightarrow_\eta$.

So we only have to examine the clause for the input condition in $\approx_{int}$. Let n be a fresh name and write $P \equiv (x)\,P'$, $Q \equiv (x)\,Q'$; then we have to prove that

$P \mid \langle n\rangle \Rightarrow\approx_{int} Q'\{n/x\}$ and $Q \mid \langle n\rangle \Rightarrow\approx_{int} P'\{n/x\}$. The reduction $P \rightarrow_\eta Q$ can follow from two reasons: either $P \equiv (x)\,\big(\langle x\rangle \mid (x)\,Q'\big)$, or $P' \rightarrow_\eta Q'$. In the first case, the proof is straightforward, and in the second case, the induction hypothesis allows us to conclude. □

Intensionality will be established by proving some syntactical properties of *finite* processes w.r.t. $\approx_{syn}$. Let R be a finite process; we write $\texttt{messages}(R)$ for the number of messages in R, and $\texttt{pref}(R)$ for the number of capabilities and abstractions in R.

Lemma B.3.5 *Let P, Q be two finite processes. Suppose $P \longrightarrow P'$. Then*

1. $\texttt{messages}(P) \geq \texttt{messages}(P')$;
2. $\texttt{pref}(P) \geq \texttt{pref}(P')$.

Proof: By induction on the derivation of $P \rightarrow P'$. □

Lemma B.3.6 *Let P, Q be two finite processes. Suppose that $P \approx_{syn} Q$, and that both P and Q are eta-normalised. Then* $\texttt{messages}(P) = \texttt{messages}(Q)$.

Proof: Suppose $\texttt{messages}(P) > \texttt{messages}(Q)$. We prove that we derive a contradiction. We proceed by a case analysis on the shape of P (ie, the number of its operators)

- $P = P_1 \mid P_2$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approx_{syn} Q_i$. Now, for some i, we should have $\texttt{messages}(P_i) \neq \texttt{messages}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\,P'$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv \mathsf{cap}.\,Q'$ and $Q' \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow} Q'' \approx_{syn} P'$. It will then be, by Lemma B.3.5(1), $\texttt{messages}(P) = \texttt{messages}(P') > \texttt{messages}(Q'')$, which is impossible, by the induction on the shape.
- $P = (x)\,P'$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv (x)\,Q'$; moreover, chosen n fresh, there must be Q'' such that $\langle n\rangle \mid (x)\,Q' \Longrightarrow Q'' \approx_{syn} P'\{n/x\}$.

  If the reduction $\langle n\rangle \mid (x)\,Q' \Longrightarrow Q''$ contains at least one step, then we would have $\texttt{messages}(P'\{n/x\}) = \texttt{messages}(P) > \texttt{messages}(Q') \geq \texttt{messages}(Q'')$ and therefore, by induction on the shape, it could not be $Q'' \approx_{syn} P'\{n/x\}$.

  Therefore, suppose $Q'' = \langle n\rangle \mid (x)\,Q'$. Then $Q'' \approx_{syn} P'\{n/x\}$ implies $P'\{n/x\} \equiv \langle n\rangle \mid (x)\,P''$, for some $(x)\,P''$ with n fresh for $(x)\,P''$. Hence, since n was chosen fresh, the original process P must have been of the form $(x)\,(\langle m\rangle \mid (x)\,P'')$. This means that, modulo $\equiv$, P was not eta-normalised, thus contradicting one the hypothesis of the lemma.
- If $P = \langle n\rangle$ then by definition of $\approx_{syn}$ we should have $Q \equiv \langle n\rangle$, which is impossible, since the hypothesis is $\texttt{messages}(P) > \texttt{messages}(Q)$.

□

Lemma B.3.7 *Let P, Q be two finite processes. Suppose $P \approx_{syn} Q$, and that both P and Q are eta-normalised. Then* $\texttt{pref}(P) = \texttt{pref}(Q)$.

Proof: Suppose $\mathtt{pref}(P) > \mathtt{pref}(Q)$. We prove that we derive a contradiction. We proceed by induction on the shape of P.

- If $P = \mathbf{0}$ then $Q \equiv \mathbf{0}$.
- $P = P_1 \mid P_2$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approx_{syn} Q_i$. Now, for some i, it should be $\mathtt{pref}(P_i) \neq \mathtt{pref}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\, P'$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv \mathsf{cap}.\, Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Longrightarrow} Q'' \approx_{syn} P'$. Then

  $$\mathtt{pref}(P') = \mathtt{pref}(P) - 1 > \mathtt{pref}(Q) - 1 = \mathtt{pref}(Q') \geq \mathtt{pref}(Q'')$$

  Hence $\mathtt{pref}(P') > \mathtt{pref}(Q'')$, which is impossible by the induction on the shape.
- $P = (x)\, P'$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv (x)\, Q'$; moreover, chosen n fresh, there must be Q'' such that $\langle n \rangle \mid (x)\; Q' \Longrightarrow Q'' \approx_{syn} P'\{{}^{n}\!/\!{}_{x}\}$.

  Moreover, by the previous lemma we know that $\mathtt{messages}(P) = \mathtt{messages}(Q)$, and we should also have $\mathtt{messages}(P'\{{}^{n}\!/\!{}_{x}\}) = \mathtt{messages}(Q'')$

  The reduction $\langle n \rangle \mid (x)\; Q' \Longrightarrow Q''$ must contain at least one step, for otherwise we could not have $\mathtt{messages}(P'\{{}^{n}\!/\!{}_{x}\}) = \mathtt{messages}(Q'')$. For the same reason, during these reductions only the message $\langle n \rangle$ may have been consumed (no other messages). Thus $\langle n \rangle \mid (x)\; Q' \Longrightarrow Q''$ can be written as

  $$\langle n \rangle \mid (x)\, Q' \longrightarrow Q'\{{}^{n}\!/\!{}_{x}\} \Longrightarrow Q''\,,$$

  where $\mathtt{pref}(Q') = \mathtt{pref}(Q'\{{}^{n}\!/\!{}_{x}\})$ and also $\geq \mathtt{pref}(Q'')$ (Lemma B.3.5(2)).

  Therefore we have $\mathtt{pref}(P'\{{}^{n}\!/\!{}_{x}\}) = \mathtt{pref}(P) - 1 > \mathtt{pref}(Q) - 1 = \mathtt{pref}(Q') \geq \mathtt{pref}(Q'')$. By the induction on the shape, this is in contradiction with $Q'' \approx_{syn} P'\{{}^{n}\!/\!{}_{x}\}$.

$\square$

Lemma B.3.8 *Let P, Q be two finite processes. Suppose $P \approx_{syn} Q$, with both P and Q eta-normalised. If $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q' \approx_{syn} P'$. Similarly, if $P \longrightarrow P'$, then there is Q' such that $Q \longrightarrow Q' \approx_{syn} P'$.*

Proof: From Lemmas B.3.7 and B.3.6: if Q performed more than one actions, then it would consume one more prefix or message than P. $\square$

Theorem B.3.9 *Let P, Q be processes of MA_{IF}^{syn}. Suppose $P \approx_{syn} Q$, with both P and Q eta-normalised. Then $P \equiv Q$.*

Proof: By induction on the shape of P.

- If $P = \mathbf{0}$ then also $Q \equiv \mathbf{0}$.
- Suppose $P = P_1 \mid P_2$. Then, by definition of $\approx_{syn}$, $Q \equiv Q_1 \mid Q_2$ with $P_i \approx_{syn} Q_i$. By induction, $P_i \equiv Q_i$. Hence also $P \equiv Q$.

- Suppose $P = !P'$. Then, by Lemma B.1.18, there are r and some $(Q_i)_{1\leq i\leq r}$ such that $Q \equiv !Q_1 \mid (!)Q_2 \mid \ldots \mid (!)Q_r$, and $P' \approx_{syn} Q_i$ for all i. By induction, $P' \equiv Q_i$ for all i, so finally $Q \equiv !Q_1 \equiv P$.
- $P = \mathsf{cap}.\,P'$. By definition of $\approx_{syn}$, $Q \equiv \mathsf{cap}.\,Q'$ and there is Q'' such that $Q' \stackrel{\langle\mathsf{cap}\rangle}{\Rightarrow} Q'' \approx_{syn} P'$. By construction of MA_{IF}^{syn}, P', Q' are finite, so that we may apply Lemma B.3.15. Then it must be $Q' = Q''$, therefore by induction $Q' \equiv P'$. We conclude that $P \equiv Q$.
- $P = \langle n\rangle, n[P']$: straightforward.
- $P = (x)\ P'$. By definition of $\approx_{syn}$, we have $Q \equiv (x)\ Q'$, and again by construction of MA_{IF}^{syn}, P', Q' are finite. Since $\approx_{syn}$ is a congruence, $\langle n\rangle \mid P \approx_{syn} \langle n\rangle \mid (x)\ Q'$. We have $\langle n\rangle \mid P \longrightarrow P'\{^n\!/\!_x\}$, hence by Lemma B.3.15, $\langle n\rangle \mid (x)\ Q' \longrightarrow Q'\{^n\!/\!_x\} \approx_{syn} P'\{^n\!/\!_x\}$. By induction, $P'\{^n\!/\!_x\} \equiv Q'\{^n\!/\!_x\}$, hence $P' \equiv Q'$.

$\square$

Corollary B.3.10 *Let P, Q be processes of MA_{IF}^{syn}. Then $P{=_L}Q$ iff $P \equiv_E Q$.*

Proof: First, $=_L \subset\approx_{int}$ by Theorem B.1.28, and $\approx_{int}\subset\equiv_E$ by Theorem B.3.9. Conversely, $\equiv_E\subset\approx_{int}$ by Lemma B.3.4, and $\approx_{int}\subset =_L$ by Theorem B.1.19. $\square$

B.3.2 The case of synchronous communications

We now consider a variant of Mobile Ambients where communication is *synchronous*. For this the production $\langle\eta\rangle$ for messages in the grammar of MA in Table A.1 is replaced by the production $\langle\eta\rangle.\,P$. Communication is thus synchronous: in $\langle\eta\rangle.\,P$, the process P is blocked until the message $\langle\eta\rangle$ has been consumed. Reduction rule `Red-Com` becomes:

$$\frac{}{\langle n\rangle.\,Q \mid (x)\ P \longrightarrow Q \mid P\{^n\!/\!_x\}}\ \texttt{Red-Com}$$

In the remainder of this subsection, terms belonging to the synchronous version of the calculus will be referred to simply as 'processes'. Since our goal here is to study how the result given by Corollary B.3.10 evolves when moving to a synchronous calculus, we focus directly on MA_{IF}^{ss}, the set of all terms of the synchronous calculus in which processes guarded by prefixes are finite (along the lines of Definition B.3.2, that introduces MA_{IF}^{syn}). We shall see that in MA_{IF}^{ss}, the eta law fails and, surprisingly, the equality induced by the logic is precisely structural congruence.

In order to show this, we have to port the results about (asynchronous) MA to the synchronous case. The co-inductive characterisation in terms of $\approx_{int}$ (that is, Theorems B.1.19 and B.1.28) remains true, provided that in the definition of the bisimilarity the communication clauses are replaced by the following:

- If $P \stackrel{!n}{\rightarrow} P'$, then there is Q' such that $Q \stackrel{!n}{\Longrightarrow} Q'$ and $P' \approx_{int} Q'$.
- If $P \stackrel{?n}{\rightarrow} P'$ then there is Q' such that $Q \stackrel{?n}{\Longrightarrow} Q'$ and $P' \approx_{int} Q'$.

Accordingly, we have to change the definition of $\approx_{syn}$ by adapting the following clauses for communicating processes:

- If $P \equiv (x)\,P'$ then there is Q' such that $Q \equiv (x)\,Q'$ and for all n there is Q'' such that $Q'\{{}^{n}\!/\!{}_{x}\} \Longrightarrow Q''$ and $P'\{{}^{n}\!/\!{}_{x}\} \approx_{int} Q''$.
- If $P \equiv \langle n\rangle.\,P'$ then there is Q' such that $Q \equiv \langle n\rangle.\,Q'$ and $Q' \Longrightarrow Q'' \approx_{int} P'$.

As we show in appendix A, formulas similar to those that are needed in the asynchronous case can be derived for the synchronous calculus. In particular, we prove the following results:

Lemma B.3.11 (cf. A)

- *For all $\mathcal{A}$, there is a formula $\langle\langle ?n\rangle\rangle.\,\mathcal{A}$ such that for all P, $P \models \langle\langle ?n\rangle\rangle.\,\mathcal{A}$ iff there is P' such that $P \equiv (x)\,P'$ and $P'\{{}^{n}\!/\!{}_{x}\} \Rightarrow \models \mathcal{A}$.*
- *For all $\mathcal{A}$, there is a formula $\langle\langle !n\rangle\rangle.\,\mathcal{A}$ such that for all P, $P \models \langle\langle !n\rangle\rangle.\,\mathcal{A}$ iff there is P' such that $P \equiv \langle n\rangle.\,P'$ and $P' \Rightarrow \models \mathcal{A}$.*

Using this result, the correction and completeness proofs for $\approx_{int}$ w.r.t. $=_L$ follow exactly the same scheme as in the asynchronous case (cf. Sections B.1 and B.2), except that we do not need to reason on eta-normalised terms.

We now derive the counterpart of the properties we have established above for MA_{IF}^{syn} about the number of messages and prefixes in a term. As we will see, the concluding result differs.

Lemma B.3.12 *Suppose $P \longrightarrow P'$, where P is a finite process. Then*

1. $\mathtt{messages}(P) \geq \mathtt{messages}(P')$*;*
2. $\mathtt{pref}(P) \geq \mathtt{pref}(P')$*.*

Proof: By induction on the derivation of $P \rightarrow P'$. □

Lemma B.3.13 *Let P, Q be two finite processes and suppose $P \approx_{syn} Q$. Then* $\mathtt{messages}(P) = \mathtt{messages}(Q)$*.*

Proof: Suppose $\mathtt{messages}(P) > \mathtt{messages}(Q)$. We prove that we derive a contradiction. We proceed by a case analysis on the shape of P (ie, the number of its operators)

- $P = P_1 \mid P_2$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approx_{syn} Q_i$. Now, for some i, it should be $\mathtt{messages}(P_i) \neq \mathtt{messages}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\,P'$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv \mathsf{cap}.\,Q'$ and $Q' \stackrel{\langle \mathsf{cap}\rangle}{\Rightarrow} Q'' \approx_{syn} P'$. It will then be, by Lemma B.3.5(1), $\mathtt{messages}(P) = \mathtt{messages}(P') > \mathtt{messages}(Q'')$, which is impossible, by the induction on the shape.
- $P = \langle n\rangle.\,P'$. Then $Q \equiv \langle n\rangle.\,Q'$ and $P' \approx_{syn} Q'$. But $\mathtt{messages}(P') > \mathtt{messages}(Q')$, which by induction is impossible.
- $P = (x)\,P'$. Then $Q \equiv (x)\,Q'$ and for all h fresh, $Q'\{{}^{h}\!/\!{}_{x}\} \approx_{syn}\Longrightarrow Q''$ and $P'\{{}^{h}\!/\!{}_{x}\} \approx_{syn} Q''$ and $\mathtt{messages}(P'') > \mathtt{messages}(Q'')$, so we can conclude by induction.

□

Lemma B.3.14 *Let P, Q be two finite processes, and suppose $P \approx_{syn} Q$. Then* $\mathtt{pref}(P) = \mathtt{pref}(Q)$.

Proof: Suppose $\mathtt{pref}(P) > \mathtt{pref}(Q)$. We prove that we derive a contradiction. We proceed by induction on the shape of P.

- If $P = \mathbf{0}$ then $Q \equiv \mathbf{0}$.
- $P = P_1 \mid P_2$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv Q_1 \mid Q_2$ with $P_i \approx_{syn} Q_i$. Now, for some i, it should be $\mathtt{pref}(P_i) \neq \mathtt{pref}(Q_i)$, which is impossible, by the induction on the shape.
- $P = \mathsf{cap}.\,P'$. Then, by definition of $\approx_{syn}$, it must be $Q \equiv \mathsf{cap}.\,Q'$ and $Q' \stackrel{\langle \mathsf{cap} \rangle}{\Rightarrow} Q'' \approx_{syn} P'$. Then

  $$\mathtt{pref}(P') = \mathtt{pref}(P) - 1 > \mathtt{pref}(Q) - 1 = \mathtt{pref}(Q') \geq \mathtt{pref}(Q'')$$

  Hence $\mathtt{pref}(P') > \mathtt{pref}(Q'')$, which is impossible by the induction on the shape.
- $P = \langle n \rangle.\,P'$. Similar to capability case.
- $P = (x)\,P'$. Then $Q \equiv (x)\,Q'$ and there is Q'' such that $Q'\{{}^{h}\!/\!_{x}\} \approx_{syn}\Longrightarrow Q''$ and $P'\{{}^{h}\!/\!_{x}\} \approx_{syn} Q''$. There is no consumption of messages, hence $\mathtt{pref}(P'\{{}^{h}\!/\!_{x}\}) > \mathtt{messages}(Q'')$, and we can conclude using induction.

□

Lemma B.3.15 *Let P, Q be two finite processes, and suppose $P \approx_{syn} Q$. If $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q' \approx_{syn} P'$. Similarly, if $P \longrightarrow P'$, then there is Q' such that $Q \longrightarrow Q' \approx_{syn} P'$.*

Proof: From two previous Lemmas: if Q performed more than one actions, then it would consume one more prefix or message than P. □

Theorem B.3.16 *Let P, Q be two processes in MA^{ss}_{IF}, and suppose $P \approx_{syn} Q$. Then $P \equiv Q$.*

Proof: By induction on the shape of P (exactly analogous to Theorem B.3.9).

□

Corollary B.3.17 *Let P, Q be processes of MA^{ss}_{IF}. Then $P =_L Q$ iff $P \equiv Q$.*

B.4 (Un)decidability of logical equivalence

In this section, we define the encoding of a Turing machine in MA^{syn}_{IF}. The definition of the encoding requires the introduction of some macros that will be given as (MA^{syn}_{IF}) contexts. To increase lisibility of our definitions, we introduce a new notation for contexts. The contexts we shall use here are terms containing one or several occurrences of a hole, written [] (with this notation, the hole cannot be confused with an ambient construct). Given a context $\mathcal{C}$, we write $\mathcal{C}\{\!|\ P\ |\!\}$ for the process obtained by filling the hole with a term P in $\mathcal{C}$. To abbreviate definitions, we shall sometimes work with *parametrised contexts*, which are context definitions that depend on some values (names, words, or movements of the head of the Turing machine). Some parametrised definitions shall be written $\texttt{foo}(p); P$: such a definition depends on parameters p and P (P being a process), but we adopt this notation to emphasize sequentiality between the process being introduced and P.

B.4.1 Ribbons

Digits and words We associate to booleans true and false two names tt and $f\!f$. We call these names *digits*, and range over digits with d, d'. A word will be the result of a (possibly empty) concatenation of digits. The empty word shall be written ϵ. We range over words with w, w', w_1, w_2. Given a word w consisting in r digits (with $r \geq 1$), we shall sometimes write w as $w^1 \dots w^r$ to refer to w's digits.

We start by the definition of the support of the Turing machine: ribbons are defined as follows, and can be in differents states (frozen, growing, work ribbon, old).

Cells and Words		
$\texttt{cell}(d)[\,]$	$:=$	$cell[d[\mathbf{0}] \mid\ !\mathsf{open}\, wo \mid [\,]\,]$
$\texttt{word}(w)[\,]$	$:=$	$\texttt{cell}(w^1)\{\!|\ \texttt{cell}(w^2)\{\!|\ \dots \texttt{cell}(w^r)[\,] \dots |\!\}\ |\!\} \qquad (w = w^1 w^2 \dots w^r)$
Ribbon Extensor		
$\texttt{deadextcode}$	$:=$	$!\mathsf{open}\, coin.\, \mathsf{open}\, newcell.\, \mathsf{in}\, cell.\, coin[\mathbf{0}] \mid\ !newcell[\texttt{cell}(f\!f)\{\!|\ \mathsf{out}\, ext\ |\!\}]$
$\texttt{sendstart}$	$:=$	$msg[\mathsf{out}\, ext.\, !\mathsf{out}\, cell \mid \mathsf{out}\, ribbon\_left.\, start[\mathsf{in}\, TM]]$
$\texttt{ExtensorFrozen}$	$:=$	$ext[\texttt{deadextcode} \mid \mathsf{open}\, coin.\, \texttt{sendstart}]$
$\texttt{ExtensorAlive}$	$:=$	$ext[coin[\mathbf{0}] \mid \texttt{deadextcode} \mid \mathsf{open}\, coin.\, \texttt{sendstart}]$
$\texttt{ExtensorDead}$	$:=$	$ext[\texttt{deadextcode}]$
Ribbons		
$\texttt{cleaninst}$	$:=$	$\mathsf{open}\, cleaner.\, \mathsf{open}\, runclean \mid runclean[\texttt{deadcleancode}]$
$\texttt{deadcleancode}$	$:=$	$!\mathsf{open}\, f\!f \mid\ !\mathsf{open}\, tt \mid\ !\mathsf{open}\, cell \mid\ !\mathsf{open}\, wo$
$\texttt{FrozenRibbon}(w)$	$:=$	$ribbon\_left[\texttt{cleaninst} \mid \texttt{word}(w)\{\!|\ \texttt{ExtensorFrozen}\ |\!\}]$
$\texttt{GrowingRibbon}(w)$	$:=$	$ribbon\_left[\texttt{cleaninst} \mid \texttt{word}(w)\{\!|\ \texttt{ExtensorAlive}\ |\!\}]$
$\texttt{WorkRibbon}(w_1, w_2)[\,]$	$:=$	$ribbon\_left[\texttt{cleaninst}] \mid \texttt{word}(w_1)\{\!|\ [\,] \mid \texttt{word}(w_2)\{\!|\ \texttt{ExtensorDead}\ |\!\}\ |\!\}$
$\texttt{OldRibbon}$	$:=$	$ribbon\_left[\texttt{deadcleancode} \mid \texttt{ExtensorDead}]$

All names used in the definitions above are supposed to be pairwise distinct. In particular, TM is the name we shall use for the ambient containing the Turing Machine (see below).

Fact B.4.1 (Ribbon evolution) *For any word w and $n \in \mathbb{N}$, let $P_n = \texttt{GrowingRibbon}(w.\, f\!f^n)$. We have:*

- $P_n \ \Rightarrow\ P_{n+1}$*;*
- $P_n \ \Rightarrow\ R$ *with* $R = \texttt{WorkRibbon}(\epsilon, w.\, f\!f^n)\{\!|\ msg[!\mathsf{out}\, cell \mid \mathsf{out}\, ribbon\_left.\, start[\mathsf{in}\, TM]] \mid\ |\!\}$*;*
- *for any term Q along the reduction paths from P_n to P_{n+1} and from P_n to R, there exists Q' such that $Q \equiv ribbon\_left[Q']$.*

Moreover, for any word w, we have:

$$\texttt{WorkRibbon}(w, \epsilon)\{\!|\ \mathbf{0}\ |\!\} \ \mid \ cleaner[\mathsf{in}\, ribbon\_left] \quad \Rightarrow \quad \texttt{OldRibbon}.$$

Proof: At any step, the extensor can only choose between creating a new *ff* cell or dying and sending up through the ribbon an ambient msg. Note that when extending the ribbon with a new *ff* cell, there are at some point two concurrent actions $\mathsf{in}\, cell$ and $\mathsf{out}\, ext$: these are in causal dependency, since the $\mathsf{in}\, cell$ can only happen once the $\mathsf{out}\, ext$ has taken place, which ensures sequentiality of the execution. □

B.4.2 Turing Machine

Definition B.4.2 ((Ideal) Turing Machine) *We introduce three symbols $\leftarrow$, $\downarrow$ and $\rightarrow$ for the movements of the head of a Turing machine.*

We represent a Turing machine as a quadruplet $(\mathcal{Q}, q_{start}, q_A, \delta)$ where $\mathcal{Q}$ is a set of states, q_{start} is the initial state, q_A is the accepting state, and $\delta : \mathcal{Q} \times \{ff, tt\} \rightarrow \mathcal{Q} \times \{ff, tt\} \times \{\leftarrow, \downarrow, \rightarrow\}$ is the evolution function.

Notation: we shall write

$$(w_1, q, w_2) \twoheadrightarrow\!\!\!\!\twoheadrightarrow (w_1', q', w_2')$$

to denote the fact that the Turing machine in state q with the head on the cell of the last letter of w_1 (which will be referred to as *"the head dividing the ribbon into words w_1 and w_2"*) evolves in one step of computation into the machine in state q', dividing the ribbon into words w_1' and w_2'.

We can now present the encoding of Turing machines, consisting in the execution of transitions and some extra machinery to be executed after recognition of a word.

Turing Machine Transitions

$$\begin{array}{lcl}
\texttt{clear}(d); P & := & wo[\,\mathsf{out}\,head.\,\mathsf{open}\,d.\,cl_ack[\mathsf{in}\,head]\,] \;\mid\; \mathsf{open}\,cl_ack.\,P \\
\texttt{write}(d); P & := & wo[\,\mathsf{out}\,head.\,d[\mathbf{0}] \mid wr_ack[\mathsf{in}\,head]\,] \;\mid\; \mathsf{open}\,wr_ack.\,P \\
\texttt{become}(mo); P & := & mo[\mathsf{out}\,head.\,\mathsf{open}\,head.\,P] \mid \mathsf{in}\,mo \\
\texttt{domove}(mv); P & := & \begin{cases} \mathsf{in}\,cell.\,P & \text{if } mv = \leftarrow \\ P & \text{if } mv = \downarrow \\ \mathsf{out}\,cell.\,P & \text{if } mv = \rightarrow \end{cases} \\
\texttt{transcode}(d_r, q_w, d_w, mv) & := & \texttt{clear}(d_r); \texttt{write}(d_w); \texttt{become}(mo); \mathsf{in}\,TM.\,\texttt{domove}(mv); \mathsf{open}\,q_w
\end{array}$$

State

$$\begin{array}{lcl}
ff \rightarrow P \;+\; tt \rightarrow Q & := & coin[\mathsf{in}\,ff.\,\mathsf{out}\,ff.\,P] \mid coin[\mathsf{in}\,tt.\,\mathsf{out}\,tt.\,Q] \mid \mathsf{open}\,coin \\
\texttt{code}(q) & := & !q[head[\,\mathsf{out}\,TM.\,\big(\;ff \rightarrow \texttt{transcode}(ff, d_{ff}, q_{ff}, mv_{ff}) \\
& & \qquad\qquad + \; tt \rightarrow \texttt{transcode}(tt, d_{tt}, q_{tt}, mv_{tt})\big)\;]] \\
& & \mid\; !coin[\mathsf{in}\,ff.\,\mathsf{out}\,ff.\,\texttt{transcode}(ff, d_{ff}, q_{ff}, mv_{ff})] \\
& & \mid\; !coin[\mathsf{in}\,tt.\,\mathsf{out}\,tt.\,\texttt{transcode}(tt, d_{tt}, q_{tt}, mv_{tt})] \\
\texttt{code}(q_A) & := & !q_A[\,get_out[\mathbf{0}]\,]
\end{array}$$

Turing Machine Behavior after Recognition

$$\begin{array}{l}
\texttt{getout} := \\
\quad !\mathsf{open}\,get_out.\,\mathsf{out}\,cell.\,get_out[\mathbf{0}] \\
\mid\; !\mathsf{open}\,get_out.\,\mathsf{out}\,ribbon_left.\,\big(\;\; cleaner[\mathsf{out}\,TM.\,\mathsf{in}\,ribbon_left] \\
\qquad\qquad\qquad\qquad \mid\; coin[\mathsf{out}\,TM.\,\mathsf{in}\,ribbon_left.\,\mathsf{in}\,cell^{\mathsf{length}(w)}.\,\mathsf{in}\,ext] \\
\qquad\qquad\qquad\qquad \mid\; \mathsf{open}\,start.\,\mathsf{in}\,ribbon_left.\,\mathsf{in}\,cell.\,\mathsf{open}\,q_{start}\;\big)
\end{array}$$

Our Turing Machine encoding is somehow reminiscent of the one presented in [CG98]. This is also the case for some of the macros we use. We can however remark that we work here in a language without name restriction, and with a simpler encoding of choice. Note as well that this encoding is parametric over a word w, whose length (denoted $\mathsf{length}(w)$) is used in the definition of `getout` (in that definition, $\mathsf{in}\,cell^{\mathsf{length}(w)}$ stands for the concatenation of $\mathsf{length}(w)$ copies of the capability $\mathsf{in}\,cell$). This aspect of our encoding is however irrelevant since it is influent only after the end of the execution of the machine, and not within the central part of the simulation. The remainder of this subsection is devoted to establishing the following claim:

Fact B.4.3 *Any Turing Machine computation may be encoded in* MA_{IF}^{syn}.

Remark: Busi and Zavattaro [BZ04] have independently obtained an encoding of Random Acces Machines in MA_{IF}^{syn} (although this sublanguage is not explicitely mentioned in the paper).

Definition B.4.4 (Turing Machine in Mobile Ambients) *The encoding of a Turing Machine is based on an ambient named TM, containing a persistent code named* `tmsoup`*:*

$$\texttt{tmsoup} := \;\; \texttt{code}(q_0) \mid \ldots \mid \texttt{code}(q_n) \;\mid\; \texttt{getout} \;\mid\; !\mathsf{open}\,mo\,.$$

We define two configurations for the encoding of a Turing Machine. Before being active, the machine is in starting state*, defined by:*

$$\texttt{TMStart} := \;\; TM[\;\mathsf{open}\,start.\,\mathsf{in}\,ribbon\_left.\,\mathsf{in}\,cell.\,\mathsf{open}\,q_{start} \;\mid\; \texttt{tmsoup}\;]\,.$$

Once the computation has started, the Turing machine in state q is represented by the term

$$\mathtt{TM}(q) \;:=\; TM[\,\mathsf{open}\,q \;\mid\; \mathtt{tmsoup}\,]\,.$$

Lemma B.4.5 (Turing machine in MA_{IF}^{syn}) *All terms used in the encoding of a Turing machine belong to MA_{IF}^{syn}.*

We now examine the evolution of the terms we have defined in order to simulate Turing machines. We first introduce a useful relation.

Definition B.4.6 (deterministic evolution relation) *We say that a process P* deterministically evolves *to Q, written $P \rightsquigarrow Q$, if and only if $P \rightarrow Q$ and for any Q' s.t. $P \rightarrow Q'$, either $Q' \not\rightarrow$ or $Q \equiv Q'$.*

Notation: We shall write $P \rightsquigarrow^k Q$ to say that P deterministically reduces to Q in k steps ($k \geq 1$). We write $P \rightsquigarrow^* Q$ when $P \rightsquigarrow^k Q$ for some k.

Using $\rightsquigarrow$, we can state some elementary facts about the macros involved in the execution of the machine.

The relation $P \rightsquigarrow^* Q$ captures the fact that P can only reduce to Q up to some blocking states. Actually, such blocking states may only appear due to the firing of the "wrong branch" in a choice encoding ($f\!f \rightarrow \cdots + tt \rightarrow \ldots$).

Fact B.4.7 (state evolution) *For any terms P, Q, names $d, d' \in \{f\!f, tt\}$ and word w, let us suppose $M \;\equiv\; d[\mathbf{0}] \;\mid\; !\mathsf{open}\,wo \;\mid\; \mathtt{word}(w)\{\!\mid \mathtt{ExtensorDead} \mid\!\}$ for any word w. We then have the following deterministic transition sequences:*

- $head[d{\rightarrow}P \;+\; \neg d{\rightarrow}Q] \;\mid\; d[\mathbf{0}] \;\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]$

  $\rightsquigarrow^3 \;\; head[P \mid coin[\mathsf{in}\,\neg d.\,\mathsf{out}\,\neg d.\,Q]] \mid d[\mathbf{0}] \mid !\mathsf{open}\,wo \mid cell[M] \mid TM[\mathtt{tmsoup}]\,;$

- $head[\mathtt{clear}(d); P \;\mid\; coin[\mathsf{in}\,d'.\,Q]] \;\mid\; d[\mathbf{0}] \mid\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]$

  $\rightsquigarrow^5 \;\; head[P \mid coin[\mathsf{in}\,d'.\,Q]] \;\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]\,;$

- $head[\mathtt{write}(d); P \;\mid\; coin[\mathsf{in}\,d'.\,Q]] \;\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]$

  $\rightsquigarrow^4 \;\; head[P \mid coin[\mathsf{in}\,d'.\,Q]] \;\mid\; d[\mathbf{0}] \;\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]\,;$

- $head[\mathsf{become}(mo); P \mid coin[\mathsf{in}\,d'.\,Q]] \;\mid\; d[\mathbf{0}] \;\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]$

  $\rightsquigarrow^3 \;\; mo[P \mid coin[\mathsf{in}\,d'.\,Q]] \;\mid\; d[\mathbf{0}] \;\mid\; !\mathsf{open}\,wo \;\mid\; cell[M] \;\mid\; TM[\mathtt{tmsoup}]\,.$

Moreover, the same results hold with a frozen (instead of dead) extensor in M, the only condition being that ambient ext contains an inactive term.

Proof: Straightforward. From the second statement on, the ambient $coin[\mathsf{in}\,d'.\,Q]$ is frozen: it actually represents the non-chosen branch in the encoding of the choice operator, that will be erased later, when the head of the Turing machine comes back inside ambient TM (see below). □

We can now merge the results above into a property regarding transitions of the Turing machine.

Lemma B.4.8 (One step of Turing machine simulation) *Let $\mathcal{M}$ be a Turing machine, q one of its non accepting states, and w_1, w_2 two words, with $w_2 \neq \epsilon$. Suppose $(w_1, q, w_2) \rightarrowtail\!\!\!\rightarrow (w_1', q', w_2')$. Then*

$$\texttt{WorkRibbon}(w_1, w_2)\{\!|\ \texttt{TM}(q)\ |\!\}\ \rightsquigarrow^*\ \texttt{WorkRibbon}(w_1', w_2')\{\!|\ \texttt{TM}(q')\ |\!\}\,.$$

Proof: We divide the evolution of the term representing the Turing machine into the following steps:

1. From state q, the TM may open the q code, which has the effect of releasing an ambient named *head*. Moreover, this is the only place where some reduction is possible, because first, `Extensor` is inactive and second, in every ambient named *cell*, no reduction occurs. Therefore,

   $$\begin{array}{l}\texttt{WorkRibbon}(w_1, w_2)\{\!|\ \texttt{TM}(q)\ |\!\}\\ \quad \rightsquigarrow^2\ \texttt{WorkRibbon}(w_1, w_2)\{\!|\ \texttt{TMNostate}\ \mid\ head[\mathit{ff} \rightarrow \cdots + tt \rightarrow \ldots]\ |\!\}\end{array}$$

   where the notation `TMNostate` stands for the following configuration of the Turing machine ambient:

   $$TM[\,\mathtt{code}(q_0)\ \mid\ \ldots\ \mid\ \mathtt{code}(q_n)\ \mid\ \mathtt{tmsoup}]$$

   Note that this ambient cannot perform any reduction as long as it is not visited by a *mo* or *getout* ambient.

2. Using the previous fact, and considering that reductions can only take place at *cell* level, we have

   $$\begin{array}{l}\texttt{WorkRibbon}(w_1, w_2)\{\!|\ \texttt{TMNostate}\ \mid\ head[\mathit{ff} \rightarrow \cdots + tt \rightarrow \ldots]\ |\!\}\ \rightsquigarrow^{15}\\ \texttt{WorkRibbon}(w_1^1 \ldots w_1^{r-1} d, w_2)\\ \quad \{\!|\ \texttt{TMNostate}\ \mid\ mo[\mathsf{in}\, TM.\, \mathsf{domove}(mv).\, \mathsf{open}\, q'\ \mid\ coin[\mathsf{in}\, \neg w_1^r.\, P]]\ |\!\}\end{array}$$

   where $\delta(q, w_1^r) = (q', d, mv)$.

3. The ambient *mo* comes back into the Turing machine and is opened by the $\mathsf{open}\, mo$ in the `tmsoup` component. Then the head movement (if any) is performed, which activates the $\mathsf{open}\, q'$, so that the Turing machine gets into $\texttt{TM}(q')$ state.

   $$\begin{array}{l}\texttt{WorkRibbon}(w_1^1 \ldots w_1^{r-1} d, w_2)\{\!|\ \texttt{TMNostate}\\ \qquad \mid\ mo[\mathsf{in}\, TM.\, \mathsf{domove}(mv).\, \mathsf{open}\, q'\ \mid\ coin[\mathsf{in}\, \neg w_1^r.\, P]]\ |\!\}\\ \qquad\qquad \rightsquigarrow^{2(+1)}\ \texttt{WorkRibbon}(w_1', w_2')\{\!|\ \texttt{TM}(q')\ |\!\}\,.\end{array}$$

   Note that opening ambient *mo* triggers the absorbtion of the non-selected branch of the choice (ambient *coin*) by a $!coin[\ldots]$ (from the code for the original state of the machine).

   The 2(+1) above comes from the fact that the head of the machine can also make no movement in its transition from a state to another (case $\downarrow$).

$\square$

We obtain as a corollary of the Lemma above:

Proposition B.4.9 (Turing machine simulation) *Given a Turing machine $\mathcal{M}$, for any word w and $n \in \mathbb{N}$, the Turing machine $\mathcal{M}$ recognises the word w on the ribbon $w.\,\mathit{ff}^n$ iff there exist two words w_1 and w_2 s.t.*

$$\mathtt{WorkRibbon}(\epsilon, w.\,\mathit{ff}^n)\{\!|\ \mathtt{TM}(q_{start})\ |\!\}\ \rightsquigarrow^*\ \mathtt{WorkRibbon}(w_1, w_2)\{\!|\ \mathtt{TM}(q_A)\ |\!\}\,,$$

where the terms above are given by the encoding of $\mathcal{M}$.

Let us finally describe what happens after the machine has reached the accepting state.

Fact B.4.10 (Acceptation) *Let w_1, w_2 be two words. Then*

$$\begin{array}{l}\mathtt{WorkRibbon}(w_1, w_2)\{\!|\ \mathtt{TM}(q_A)\ |\!\}\\ \qquad \Rightarrow\ \mathtt{OldRibbon}\ \mid\ \mathtt{TMStart}\ \mid\ coin[\mathsf{in}\, ribbon\_left.\,\mathsf{in}\, cell^{\mathsf{length}(w)}.\,\mathsf{in}\, ext]\end{array}$$

where w is the word used in the encoding of the machine.

Proof:

1. when the q_A ambient has been opened, the ambient *get_out* is liberated and is present within TM:

   $$\mathtt{WorkRibbon}(w_1, w_2)\{\!|\ \mathtt{TM}(q_A)\ |\!\}\ \Rightarrow\ \mathtt{WorkRibbon}(w_1, w_2)\{\!|\ \mathtt{TMGetout}\ |\!\}$$

   where TMGetout refers to the term

   $$TM[get\_out[\mathbf{0}]\ \mid\ \mathtt{code}(q_0)\ \mid \ldots \mid\ \mathtt{code}(q_n)\ \mid\ \mathtt{tmsoup}]\,.$$

2. this allows the TM ambient to open a *get_out* 'coin' and execute the branch containing the out *cell*, and doing this liberate a new *get_out* ambient:

   $$\mathtt{WorkRibbon}(w_1, w_2)\{\!|\ \mathtt{TMGetout}\ |\!\}\ \Rightarrow\ \mathtt{WorkRibbon}(w_1^1 \ldots w_1^{r-1}, w_1^r.\,w_2)\{\!|\ \mathtt{TMGetout}\ |\!\}$$

   Note that the other subterm starting with open *get_out* could also have been triggered, leading to a blocked state. This is no harm for us, since we want to establish the existence of an execution where the machine exits the ribbon. This way, TM progresses to the left until it is directly inside $ribbon\_left$.

3. Then TM gets out of $ribbon\_left$, choosing the other branch of open *get_out*, which leads to the following state:

   $$\begin{array}{lll}\mathtt{WorkRibbon}(w_1, w_2)\{\!|\ \mathbf{0}\ |\!\}\ \mid\ TM[& cleaner[\mathsf{out}\, TM.\,\mathsf{in}\, ribbon\_left] \\ & \mid\ coin[\mathsf{out}\, TM.\,\mathsf{in}\, ribbon\_left.\,\mathsf{in}\, cell^{\mathsf{length}(w)}.\,\mathsf{in}\, ext] \\ & \mid\ \mathtt{code}(q_0)\ \mid \ldots \mid\ \mathtt{code}(q_n)\ \mid\ \mathtt{tmsoup}]\end{array}$$

4. At this point, the TM ambient may liberate an ambient *cleaner* that enters $ribbon\_left$ and starts the cleaning process. TM may also liberate the ambient *coin* so that we exactly obtain the expected term.

□

Remarks:

- As we already mentioned above, our encoding of the Turing machine is at this point dependent from the word w that we want it to recognize.
- We reason here using $\Rightarrow$ transitions instead of deterministic reduction $\leadsto$: indeed, we are considering states where the machine has already recognized the word, and we only need to prove that *there exists* some way back to its (*exact*) initial state. This will be used for the proof of undecidability in Sec. B.4.3.

B.4.3 Undecidability of Logical Equivalence

We can now exploit the encoding we have studied to establish undecidability of the equivalence induced by the logic $=_L$.

Lemma B.4.11 (Loop lemma) *Given a Turing machine $\mathcal{M}$ and a word w, define the following terms, given from the encoding of $\mathcal{M}$:*

$$Q := \ !\texttt{FrozenRibbon}(w) \mid !\texttt{OldRibbon} \mid !\mathsf{open}\, msg \mid !\mathsf{out}\, cell \mid \texttt{TMStart}\,,$$
$$P_0 := Q \mid \texttt{GrowingRibbon}(w) \quad \textit{and} \quad P_1 := Q \mid \texttt{GrowingRibbon}(w.\, \mathit{ff})\,.$$

Then $P_0 \Rightarrow P_1$. Conversely, $P_1 \Rightarrow P_0$ if and only if the word w may be recognized on a finite (but sufficiently long) ribbon of the shape $w.\, \mathit{ff}^N$, for some $N \in \mathbb{N}$, by the Turing machine $\mathcal{M}$.

Proof: The transition $P_0 \Rightarrow P_1$ follows from Fact B.4.1.

Let us then first assume that w can be recognized on a ribbon of the form $w.\, \mathit{ff}^N$, that is w followed by an arbitrary number of ff digits. Then from Fact B.4.1, we can obtain the corresponding extension of the ribbon from state P_1, i.e. exhibit a transition $P_1 \Rightarrow Q \mid \texttt{WorkRibbon}(w.\, \mathit{ff}^N, \epsilon)\{\!|\, \mathbf{0} \mid \,|\!\} \mid start[\mathsf{in}\, TM]$. At this point, the ambient *start* can enter TM and allow it to get into the work ribbon. Then, using the simulation result (Prop. B.4.9), we know that the Turing machine reaches the acceptation state (this result is obtained by induction over the length of w). At this point, according to Fact B.4.10, the work ribbon is transformed into an old ribbon (collected by the corresponding replicated term in Q), and the Turing machine comes out of the ribbon and waits for a start signal. The liberated *coin* ambient may progress inside a frozen ribbon (containing word w by definition of Q above) until it reaches the frozen extensor and wakes it up. We then exactly obtain P_0.

Now let us assume that w cannot be recognized on any ribbon. As Q is blocked (in particular, `TMStart` is waiting for an ambient *start* to enter TM), the first reducts of P_1 are of the form $Q \mid ribbon\_left[R]$, where $\texttt{GrowingRibbon}(w.\, \mathit{ff}) \Rightarrow ribbon\_left[R]$. If a reduction chain from P_1 to P_0 can be found, then by Fact B.4.1 there exists an integer n such that

$$P_1 \Rightarrow \underbrace{Q \mid \texttt{WorkRibbon}(w.\, \mathit{ff}^n)\{\!|\, \mathbf{0} \,|\!\} \mid start[\mathsf{in}\, TM]}_{T} \Rightarrow P_0\,.$$

In term T the `WorkRibbon` is blocked, so the only evolution can come from the machine entering a ribbon. We distinguish three cases according to the kind of ribbon which is entered by the machine:

1. If it gets into an old ribbon, there can be no more reduction, as the TM is stuck on an $\mathsf{in}\, cell$ action.

2. If it gets into the work ribbon, according to Prop. B.4.9, there is a unique way to evolve, through simulation of the machine. At his point, the machine may have an infinite computation on the finite ribbon, never reaching accepting state: this means that it will not get out of the ribbon, which prevents the system to evolve into P_0. Alternatively, or the machine may try to use more ribbon than what has been created before evolution from `GrowingRibbon` into `WorkRibbon`, and the machine is stuck. So in any case, state P_0 cannot be reached.

3. The latter reasoning also applies if the machine enters a frozen ribbon.

Finally, we have that state P_0 is unreachable if word w cannot be recognised by the machine on a ribbon of the form $w.\mathit{ff}^N$ for some N, which concludes the proof. □

Theorem B.4.12 (Undecidability of $=_L$) *$=_L$ is an undecidable relation on* MA*.*

Proof: Let us first note that the decidability of $=_L$ over MA_{IF} is a consequence of its inductive characterisation $\doteq$ (Definition B.2.13) together with the image finitess hypothesis of MA_{IF}.

Consider processes P_0 and P_1 from Lemma B.4.11. We show that the problem of deciding whether $\mathsf{open}\,n.\,P_0 =_L \mathsf{open}\,n.\,P_1$ is equivalent to decide wether $P_0 \Rightarrow P_1 \Rightarrow P_0$. This will be enough, by Lemma B.4.11, to prove the undecidability of $=_L$.

Let us prove now the undecidability of $=_L$ on MA. Consider processes P_0 and P_1 of Lemma B.4.11. These processes are in MA_{IF}^{syn}. Using Corollary B.2.30, the definition of $\approx_{int}$, and Theorem B.3.9, we have:

$$\begin{array}{rll} \mathsf{open}\,n.\,P_0 =_L \mathsf{open}\,n.\,P_1 & \text{iff} & \mathsf{open}\,n.\,P_0 \;\approx_{int}\; \mathsf{open}\,n.\,P_1 \\ & \text{iff} & P_0 \;\Rightarrow\approx_{int}\; P_1 \;\Rightarrow\approx_{int}\; P_0 \end{array}$$

(from Theorem B.3.9, $\Rightarrow\approx_{int}$ is $\Rightarrow$ on MA_{IF}^{syn}).

The first equivalence follows from soundness and completeness (Theorems B.1.19 and B.2.29). The second is the definition of $\approx_{int}$. Since on MA_{IF}^{syn} $\approx_{int}=\equiv$, the last condition is simply the loop condition, and Then undecidability follows from Lemma B.4.11.

□

Appendix C

Minimalisation in Static Ambients and Separation Logic

This appendix studies the contribution of spatial connectives in the expressiveness of static spatial logics, as done in chapters 5 and 8, and as presented in [Loz04a].

We introduce SA, SAL and its adjunct-free fragment (SAL_{int}) in Sect. C.1. We prove adjunct elimination for quantifier-free formulas in Sect. C.3, based on the notion of intensional bisimilarity, discussed in Sect. C.2. The general result for SAL is then established in Sect. C.4, based on prenex forms. We discuss the adjunct elimination for $SAL^{\exists}$ in Sect. C.5, and show minimality of SAL_{int} in Sect. C.6; in Sect. C.7, we introduce SL and a classical fragment of it (CL), which we prove to be as expressive as SL.

C.1 Background

In this section we define the model of static ambients (SA) and its logic SAL. We also define the intensional fragment (SAL_{int}) of SA.

In all what follows we assume an infinite set $\mathcal{N}$ of names, ranged over by n, m. Tree terms are defined by the following grammar:

$$P \ ::= \ P \mid P \ \big| \ n[P] \ \big| \ (\nu n)P \ \big| \ \mathbf{0}\,.$$

The set $\mathrm{fn}(P) \subset \mathcal{N}$ of free names of P is defined by saying that ν is the only binder on trees. We call *static ambients* tree terms quotiented by the smallest congruence $\equiv$ (called *structural congruence*) satisfying the axioms of Fig C.1. Formulas, ranged over with $\mathcal{A}, \mathcal{B}, \ldots$, are defined in Fig C.2 . These formulas form *the static ambient logic*, and we call *intensional fragment* the subset of the formulas not using the connectives $\triangleright$, @, and $\oslash$ (adjuncts). We note them respectively SAL and SAL_{int}.

We will say that $\mathcal{A}$ is *quantifier-free* if $\mathcal{A}$ does not contain any $\mathsf{И}$ quantification. The set of free names of a formula $\mathcal{A}$, written $\mathrm{fn}(\mathcal{A})$ is the set of names

$P \mid \mathbf{0} \equiv P$	$(\nu n)\,\mathbf{0} \equiv \mathbf{0}$	
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	$(\nu n)\, m[P] \equiv m[(\nu n)P]$	$(n \neq m)$
$P \mid Q \equiv Q \mid P$	$(\nu n)P \mid Q \equiv (\nu n)(P \mid Q)$	$(n \notin \mathrm{fn}(Q))$

Figure C.1: Structural congruence on SA

$\mathcal{A} \;::=\; \mathcal{A}\wedge\mathcal{A} \mid \neg\mathcal{A} \mid И n.\,\mathcal{A} \mid 0$	$\mathcal{A} \mid \mathcal{A}$	$n[\mathcal{A}]$	$n\circledR\mathcal{A}$ (intensional fragment)
	$\mathcal{A} \triangleright \mathcal{A}$	$\mathcal{A}@n$	$\mathcal{A} \oslash n$ (adjuncts)

Figure C.2: SAL and the intensional fragment SAL_{int}

appearing in $\mathcal{A}$ that are not bound by a $И$ quantification. $\mathcal{A}(n \leftrightarrow n)'$ is the formula $\mathcal{A}$ in which names n and n' are swapped.

Definition C.1.1 (Satisfaction) *We define the relation $\models\, \subset (SA \times SAL)$ by induction on the formula as follows:*

- $P \models \mathcal{A}_1 \wedge \mathcal{A}_2$ *if* $P \models \mathcal{A}_1$ *and* $P \models \mathcal{A}_2$
- $P \models \neg\mathcal{A}$ *if* $P \not\models \mathcal{A}$
- $P \models И n.\,\mathcal{A}$ *if* $\forall n' \in \mathcal{N} - (\mathrm{fn}(P) \cup \mathrm{fn}(\mathcal{A}))$, $P \models \mathcal{A}(n \leftrightarrow n)'$
- $P \models \mathcal{A}_1 \mid \mathcal{A}_2$ *if there is* P_1, P_2 *s.t.* $P \equiv P_1 \mid P_2$ *and* $P_i \models \mathcal{A}_i$ *for* $i = 1, 2$
- $P \models 0$ *if* $P \equiv \mathbf{0}$
- $P \models n[\mathcal{A}]$ *if there is* P' *such that* $P \equiv n[P']$ *and* $P' \models \mathcal{A}$
- $P \models n\circledR\mathcal{A}$ *if there is* P' *such that* $P \equiv (\nu n)P'$ *and* $P' \models \mathcal{A}$
- $P \models \mathcal{A}_1 \triangleright \mathcal{A}_2$ *if for all* Q *such that* $Q \models \mathcal{A}_1$, $P \mid Q \models \mathcal{A}_2$
- $P \models \mathcal{A}@n$ *if* $n[P] \models \mathcal{A}$
- $P \models \mathcal{A} \oslash n$ *if* $(\nu n)P \models \mathcal{A}$

We note $\mathcal{A} \dashv\vdash \mathcal{B}$ if for all $P \in SA$, $P \models \mathcal{A}$ iff $P \models \mathcal{B}$. A context is a formula containing a *hole*; if $\mathcal{C}$ is a context, $\mathcal{C}[\mathcal{A}]$ stands for the formula obtained by replacing the hole with $\mathcal{A}$ in $\mathcal{C}$.

Lemma C.1.2 *For all $\mathcal{A}, \mathcal{B}$, and all context $\mathcal{C}$, if $\mathcal{A} \dashv\vdash \mathcal{B}$, then $\mathcal{C}[\mathcal{A}] \dashv\vdash \mathcal{C}[\mathcal{B}]$.*

Remark C.1.3

- *The formula $\perp$, that no process satisfies, can be defined as $0 \wedge \neg 0$. As e.g. in [CG00], other derived connectors include $\vee$, and $\blacktriangleright$: P satisfies $\mathcal{A} \blacktriangleright \mathcal{B}$ iff there exists Q satisfying $\mathcal{A}$ such that $P \mid Q$ satisfies $\mathcal{B}$.*
- *If $P \models \mathcal{A}$ and $P \equiv Q$, then $Q \models \mathcal{A}$. Moreover, $\models$ is* equivariant*, that is $P \models \mathcal{A}$ iff $P(n \leftrightarrow n)' \models \mathcal{A}(n \leftrightarrow n)'$ for any n, n'.*

- *For any P, there is a characteristic formula (for $\equiv$) $\mathcal{A}_P$, using the same tree representation, such that for all Q, $Q \models \mathcal{A}_P$ iff $Q \equiv P$. In particular, two static ambients are logically equivalent if and only if they are structurally congruent.*

C.2 Intensional bisimilarity

In this section and the following, we will give a first illustration of our minimalisation method on the case of SAL and SAL_{int}. This minimalisation transforms a formula from a logic to the other; however, it does not proceed as a dictionary, that is we do not show that the connectives from the original logic are some syntactic sugar for some fixed construction in the target logic. The translation actually goes through the exploration of all behaviours a process may have with respect to a formula. Roughly, we translate a formula $\mathcal{A}$ into an exhaustive disjunction

$$\mathcal{A} \quad \rightsquigarrow \quad \bigvee_{C \in \mathsf{Behaviours}(\mathcal{A})} F_C$$

of all the behaviours that lead to the acceptance of $\mathcal{A}$.

The bottleneck of this embedding is to define what are these behaviours. By behaviours, we refer to equivalence classes of some observational equivalence.

In this section, we will hence introduce a notion of partial observation over trees corresponding to logical testing. This model equivalence can be seen as the adapted *game* for this logic (in the sense of Ehrenfreucht-Fraïssé), or as the static *intensional bisimilarity* ([San01]). Observations are taken from the logic to which we want to reduce to, in this setting SAL_{int}. Each connective defines a simulation rule in a very natural way. Then we show that this observational equivalence is enough to ensure model equivalence with respect to the logic we want to minimalize, that is SAL (Proposition C.2.4) in this setting. We then give a compact representation of the observational equivalence classes as some symbolic sets we call *signatures*.

We will assume in the remainder some fixed set $N \subset \mathcal{N}$.

C.2.1 Definition

We now introduce the intensional bisimilarity. Intuitively, $\approx_{i,N}$ equates processes that may not be distinguished by logical tests involving at most i steps where the names used for the tests are picked in N.

Definition C.2.1 (Intensional bisimilarity) *We define the family $(\approx_{i,N})_{i\in\mathbb{N}}$ of symmetric relations over SA by induction on i: $\approx_{0,N} \stackrel{\text{def}}{=} SA \times SA$, and for any $i \geq 1$, $\approx_{i,N}$ is the greatest relation such that if $P \approx_{i,N} Q$, then the following conditions hold:*

- *if $P \equiv \mathbf{0}$ then $Q \equiv \mathbf{0}$*
- *for all P_1, P_2, if $P \equiv P_1 \mid P_2$ then there is Q_1, Q_2 such that $Q \equiv Q_1 \mid Q_2$ with $P_\epsilon \approx_{i-1,N} Q_\epsilon$, $\epsilon = 1, 2$.*
- *for all $n \in N$ and for all P', if $P \equiv n[P']$, then there is Q' such that $Q \equiv n[Q']$ and $P' \approx_{i-1,N} Q'$.*

- *for all* $n \in N$ *and for all* P', *if* $P \equiv (\nu n)P'$, *then there is* Q' *such that* $Q \equiv (\nu n)Q'$ *and* $P' \approx_{i-1,N} Q'$.

Lemma C.2.2 *For all* i, $\approx_{i,N}$ *is an equivalence relation.*

We shall write $SA_{/\approx_{i,N}}$ for the quotient of SA induced by $\approx_{i,N}$: it will be ranged over by equivalence classes called C, C_1, C_2.

We may observe that the bisimilarities define a stratification of observations on terms, namely $\approx_{i',N'} \subseteq \approx_{i,N}$ for $i \leq i'$ and $N \subseteq N'$. This may be understood in a topological setting. Given a fixed N, we consider the ultrametric distance over models defined by $d(P,Q) = 2^{-i}$ if i is the smallest natural for which $P \not\approx_{i,N} Q$, and $d(P,Q) = 0$ if $P \approx_{\omega,N} Q$ where $\approx_{\omega,N} = \bigcap_{i\in\mathbb{N}} \approx_{i,N}$. We call it the N-topology. It somehow captures the granularity of the logical observations with respect to their cost.

C.2.2 Correction

The key step in proving correction of the intensional bisimilarities with respect to the logic is their congruence properties for the connectives admitting an adjunct.

Lemma C.2.3 *If* $P \approx_{i,N} Q$, *then:*

- *for all* R, $P \mid R \approx_{i,N} Q \mid R$;
- *for all* $n \in \mathcal{N}$, $n[P] \approx_{i,N} n[Q]$;
- *for all* $n \in N$, $(\nu n)P \approx_{i,N} (\nu n)Q$.

Proof: By induction on i. $\square$

Note that the last point cannot be improved: consider $N = \{n\}$, $P \equiv m_1[\mathbf{0}]$, $Q \equiv m_2[\mathbf{0}]$. Then $P \approx_{2,N} Q$, but $(\nu m_1)P \not\approx_{2,N} (\nu m_1)Q$. For this reason, $\approx_{i,N}$ is not a pure congruence.

We note $s(\mathcal{A})$ the size of $\mathcal{A}$, defined as the number of its connectives.

Proposition C.2.4 (Correction) *For all* P, Q, i *such that* $P \approx_{i,N} Q$, *for all quantifier free formula* $\mathcal{A}$ *such that* $s(\mathcal{A}) \leq i$ *and* $\mathrm{fn}(\mathcal{A}) \subseteq N$,

$$P \models \mathcal{A} \quad \textit{iff} \quad Q \models \mathcal{A}.$$

Proof: By induction on $\mathcal{A}$. For the adjuncts, apply the congruence properties of Lemma C.2.3, and for the other connectives use the definition of $\approx_{i,N}$. $\square$

C.2.3 Signature functions

Definition C.2.5 (Signature) *For* $i \geq 1$, *we set:*

- $z_i^N(P) = 0$ *if* $P \equiv \mathbf{0}$, *otherwise* $\neg 0$
- $p_i^N(P) = \{(C_1, C_2) \in (SA_{/\approx_{i-1,N}})^2 : P \equiv P_1 \mid P_2$ *and* $P_i \in C_i\}$
- $a_i^N(P) = [n, C]$ *if there is* P' *s.t.* $P \equiv n[P']$, $n \in N$ *and* $P \in C$, $C \in SA_{/\approx_{i-1,N}}$, *otherwise* $a_i^N(P) = \mathsf{noobs}$, *where* noobs *is a special constant.*

- $r_i^N(P) = \{(n, C) \in N \times SA_{/\approx_{i-1,N}} : \exists P'.\ P \equiv (\nu n)P' \text{ and } P' \in C\}$

We call signature of P at (i, N) *the fourtuple* $\chi_i^N(P) = [z_i^N(P), p_i^N(P), a_i^N(P), r_i^N(P)]$.

The following lemma says that the signature actually collects all the information that may be obtained from the bisimilarity tests.

Lemma C.2.6 *Assume $i \geq 1$. Then $P \approx_{i,N} Q$ iff $\chi_i^N(P) = \chi_i^N(Q)$.*

C.3 Adjuncts elimination on quantifier-free formulas

In this section, we show that the quantifier free formulas of SAL have equivalent formulas in SAL_{int}. This result is then extended to all formulas of SAL in the next section.

In all what follows, we will assume N is a finite subset of $\mathcal{N}$; it is intended to bound the free names of the considered formulas. The encoding result is based on two key properties:

- Precompactness of the N-topology. In other words, when i, N are fixed, only a finite number of behaviours may be observed.
- Existence of intensional characteristic formulas for the classes of $\approx_{i,N}$.

The first property basically says the following: if we fix some formula $\mathcal{A}$, then we may finitely list all the behaviours a process P may have with respect to $\mathcal{A}$. Then we may tag the ones corresponding to an acceptance and the ones corresponding to a rejection, and from the second property, we may express this by some formula in SAL_{int}.

Here is the proof with more details.

Lemma C.3.1 *The codomain of χ_i^N is finite.*

Proof: We reason by induction on i. First notice that the codomain of χ_i^N is:

$$\mathbf{codom}\,\chi_i^N \quad = \quad \{0, \neg 0\} \times \left(SA_{/\approx_{i-1,N}}\right)^2 \times \left(\{\mathsf{noobs}\} + N \times SA_{/\approx_{i-1,N}}\right) \times \mathcal{P}\left(N \times SA_{/\approx_{i-1,N}}\right)$$

hence $\mathbf{codom}\,\chi_i^N$ is finite iff $SA_{/\approx_{i-1,N}}$ is finite too (here we use that N is finite). For $i = 1$, $SA_{/\approx_{0,N}} = \{SA\}$, hence χ_0^N is finite, and so is $\mathbf{codom}\,\chi_1^N$. For $i \geq 2$, we have by induction $\mathbf{codom}\,\chi_{i-1}^N$ finite. By Lemma C.2.6, there is an injection of $SA_{/\approx_{i-1,N}}$ into $\mathbf{codom}\,\chi_{i-1}^N$, so $SA_{/\approx_{i-1,N}}$ is finite, and so is $\mathbf{codom}\,\chi_i^N$. □

Here is an immediate consequence of Lemma C.3.1:

Proposition C.3.2 (Precompactness) *For all i, the number of classes of $\approx_{i,N}$ is finite.*

These results roughly say that only a finite amount of information is needed to capture a given bisimilarity class. The next result makes it more precise: this information may be collected in a single formula of SAL_{int}.

Proposition C.3.3 (Characteristic formulas) *For any $i \in \mathbb{N}$ and for any process P, there is a formula $\mathcal{A}_P^{i,N} \in SAL_{int}$ such that*

$$\forall Q \qquad Q \models \mathcal{A}_P^{i,N} \qquad \Leftrightarrow \qquad Q \approx_{i,N} P\,.$$

Proof: By induction on i. For $i = 0$, we may take $\mathcal{A}_P^{i,N} = \top$. Then assume $i \geq 1$, and we have formulas $\mathcal{A}_P^{i-1,N}$ for all P. This obviously gives a characteristic formula $\mathcal{A}_C^{i-1,N}$ for any class C of $SA_{/\approx_{i-1,N}}$. Let us consider some fixed P. We set

$$\begin{array}{rcl}
\mathcal{A}_z & = & 0 \text{ if } z_i^N(P) = 0, \text{ otherwise } \neg 0 \\
\mathcal{A}_p & = & \bigwedge_{(C_1,C_2)\in p_i^N(P)} \mathcal{A}_{C_1}^{i-1,N} \mid \mathcal{A}_{C_2}^{i-1,N} \ \wedge\ \neg \bigvee_{(C_1,C_2)\notin p_i^N(P)} \mathcal{A}_{C_1}^{i-1,N} \mid \mathcal{A}_{C_2}^{i-1,N} \\
\mathcal{A}_a & = & \left\{ \begin{array}{ll} \bigwedge_{n\in N} \neg n[\top] & \text{if } a_i^N(P) = \mathsf{noobs} \\ n[\mathcal{A}_C^{i-1,N}] & \text{if } a_i^N(P) = [n, C] \end{array} \right. \\
\mathcal{A}_r & = & \bigwedge_{[n,C]\in r_i^N(P)} n \circledR \mathcal{A}_C^{i-1,N} \ \wedge\ \neg \bigvee_{[n,C]\notin r_i^N(P)} n \circledR \mathcal{A}_C^{i-1,N} \\
\mathcal{A}_P^{i,N} & = & \mathcal{A}_z \ \wedge\ \mathcal{A}_p \ \wedge\ \mathcal{A}_a \ \wedge\ \mathcal{A}_r
\end{array}$$

where the finiteness of the conjunctions and disjunctions is ensured by Lemma C.3.1.

Then $Q \models \mathcal{A}_P^{i,N}$ iff $\chi_i^N(Q) = \chi_i^N(P)$, hence the result. □

The precompactness property says that if we bound the granularity of the observations, only finitely many distinct situations may occur. The characteristic formula property says that each of these situations is expressible in the intensional fragment. The idea of the encoding is then just to logically enumerate all these possible situations.

Theorem C.3.4 *For all quantifier-free formula $\mathcal{A} \in SAL$, there is a formula $[\mathcal{A}] \in SAL_{int}$ such that*

$$\mathcal{A} \quad \dashv\vdash \quad [\,\mathcal{A}\,].$$

Proof: We define $[\mathcal{A}]$ as follows:

$$[\,\mathcal{A}\,] \ \stackrel{\text{def}}{=}\ \bigvee \mathcal{A}_C^{i,N} \qquad \text{for } C \in SA_{/\approx_{i,N}}, C \models \mathcal{A}$$

for $i = s(\mathcal{A})$ and $N = \mathrm{fn}(\mathcal{A})$. The disjunction is finite by Proposition C.3.2. $P \models [\,\mathcal{A}\,]$ iff there is Q such that $Q \models \mathcal{A}$ and $P \approx_{i,N} Q$, that is, by Proposition C.2.4, $P \models \mathcal{A}$. □

Effectiveness of the encoding:
Due to its finiteness, the construction of our proof could seem to be effective. However, this cannot be the case due to an undecidability result for the model-checking problem on SAL [GC04]. This is quite surprising, since only an effective enumeration of the bisimilarity classes is missing to make the proof constructive. Moreover, such an enumeration exists for SA without name restriction, via testing sets as defined in [CCG03]. This reveals an unexpected richness of SA compared to pure trees.

C.4 Adjuncts elimination and fresh quantifier

In this section we establish the adjunct elimination for the full SAL. The result we already obtained for quantifier-free formulas easily extends to formulas in prenex forms. So our efforts will focus on establishing the existence of an equivalent formula in prenex form for any formula of SAL. Intuitively, prenex forms can be generated by pulling out the fresh quantifiers. We actually show how to swap the order between a quantifier and another connective without changing the semantic. Except for the $\rhd$ connective, this turns out to be quite natural.

We present our algorithm as a rewriting system in Fig. C.3. The essential result is then

Proposition C.4.1 (Correction of $\rightsquigarrow$) *The term rewriting system $\rightsquigarrow$ defined by the rules of Fig. C.3 preserves the semantics: for any $\mathcal{A}, \mathcal{B} \in SAL$, if $\mathcal{A} \rightsquigarrow \mathcal{B}$, then $\mathcal{A} \dashv\vdash \mathcal{B}$.*

$(\wedge)$	$(\text{И}n.\,\mathcal{A}_1)\ \wedge\ \mathcal{A}_2$	$\rightsquigarrow$	$\text{И}n.\,(\mathcal{A}_1 \wedge \mathcal{A}_2)$	$(n \notin \mathrm{fn}(\mathcal{A}_2))$
$(\neg)$	$\neg\text{И}n.\,\mathcal{A}_1$	$\rightsquigarrow$	$\text{И}n.\,\neg\mathcal{A}_1$	
$(\mid)$	$(\text{И}n.\,\mathcal{A}_1)\ \mid\ \mathcal{A}_2$	$\rightsquigarrow$	$\text{И}n.\,(\mathcal{A}_1 \mid \mathcal{A}_2)$	$(n \notin \mathrm{fn}(\mathcal{A}_2))$
$(\rhd\ L)$	$(\text{И}n.\,\mathcal{A}_1)\ \rhd\ \mathcal{A}_2$	$\rightsquigarrow$	$\text{И}n.\,\Big(\big(n\circledR\top \wedge \mathcal{A}_1\big) \rhd \mathcal{A}_2\Big)$	$(n \notin \mathrm{fn}(\mathcal{A}_2))$
$(\rhd\ R)$	$\mathcal{A}_1\ \rhd\ (\text{И}n.\,\mathcal{A}_2)$	$\rightsquigarrow$	$\text{И}n.\,\Big(\big(n\circledR\top \wedge\ \mathcal{A}_1\big) \rhd \mathcal{A}_2\ \Big)$	$(n \notin \mathrm{fn}(\mathcal{A}_1))$
(Amb)	$m[\text{И}n.\,\mathcal{A}]$	$\rightsquigarrow$	$\text{И}n.\,m[\mathcal{A}]$	$(m \neq n)$
$(@)$	$(\text{И}n.\,\mathcal{A})@m$	$\rightsquigarrow$	$\text{И}n.\,(\mathcal{A}@m)$	$(m \neq n)$
$(\circledR)$	$m\circledR\text{И}n.\,\mathcal{A}$	$\rightsquigarrow$	$\text{И}n.\,m\circledR\mathcal{A}$	$(m \neq n)$
$(\oslash)$	$(\text{И}n.\,\mathcal{A}) \oslash m$	$\rightsquigarrow$	$\text{И}n.\,(\mathcal{A} \oslash m)$	$(m \neq n)$

Figure C.3: Term rewriting system for prenexation

Proof:(sketched) We only detail the proof for rule $(\rhd\ L)$.

$$
\begin{array}{ll}
 & P \models (\text{И}n.\,\mathcal{A}_1) \rhd \mathcal{A}_2 \\
\Leftrightarrow & \forall Q, \forall n' \notin \mathrm{fn}(\mathcal{A}_1) \cup \mathrm{fn}(Q).\ Q{\models}\mathcal{A}_1(n \leftrightarrow n)' \ \Rightarrow\ P \mid Q{\models}\mathcal{A}_2 \\
\Leftrightarrow & \forall Q, \forall n' \notin \mathrm{fn}(\mathcal{A}_1 \rhd \mathcal{A}_2) \cup \mathrm{fn}(P \mid Q).\ Q{\models}\mathcal{A}_1(n \leftrightarrow n)' \ \Rightarrow\ P \mid Q{\models}\mathcal{A}_2 \\
\Leftrightarrow & \forall Q, \forall n' \notin \mathrm{fn}(\mathcal{A}_1 \rhd \mathcal{A}_2) \cup \mathrm{fn}(P \mid Q).\ Q{\models}\mathcal{A}_1(n \leftrightarrow n)' \ \Rightarrow\ P \mid Q{\models}\mathcal{A}_2(n \leftrightarrow n)' \\
\Leftrightarrow & \forall n' \notin \mathrm{fn}(\mathcal{A}_1 \rhd \mathcal{A}_2) \cup \mathrm{fn}(P), \\
 & \forall Q.\, n' \notin \mathrm{fn}(Q) \ \Rightarrow\ Q{\models}\mathcal{A}_1(n \leftrightarrow n)' \ \Rightarrow\ P \mid Q{\models}\mathcal{A}_2(n \leftrightarrow n)' \\
\Leftrightarrow & P \models \text{И}n.\ \big(\mathcal{A}_1\ \wedge\ n\circledR\top\big) \rhd \mathcal{A}_2
\end{array}
$$

$\square$

Remark C.4.2 *Some of the rules above (such as $(Amb), (\neg)$, and a variant of $(\mid L)$) have already been presented in [CG01b], under the form of equalities. The same result is independently developed in [GC04].*

We say that a formula $\mathcal{A}$ is *well-formed* if every variable bound by И is distinct from all other (bound and free) variables in $\mathcal{A}$. For such formulas, the side conditions in $\rightsquigarrow$ are always satisfied.

It is easy to see that $\rightsquigarrow$ defines a terminating rewriting system, and that the normal forms of well-formed formulas are formulas in prenex form. Confluence holds modulo permutation of consecutive И quantifiers.

Proposition C.4.3 (Prenex forms) *For any formula $\mathcal{A}$, there are $\tilde{n}, \mathcal{A}'$ such that $\mathcal{A} \dashv\vdash И\tilde{n}.\,\mathcal{A}'$ and $\mathcal{A}'$ is quantifier free.*

This result directly implies the following extension of Theorem C.3.4:

Theorem C.4.4 (Adjunct elimination) *For any formula $\mathcal{A} \in SAL$, there is a formula $[\mathcal{A}] \in SAL_{int}$ such that*

$$\mathcal{A} \quad \dashv\vdash \quad [\,\mathcal{A}\,].$$

Proof: There is $\mathcal{A}'$ quantifier free and $\tilde{n}$ such that $\mathcal{A} \dashv\vdash И\tilde{n}.\,\mathcal{A}'$ by Proposition C.4.3. Then by Lemma C.1.2 and Theorem C.3.4, we may write

$$\mathcal{A} \;\dashv\vdash\; И\tilde{n}.\,\mathcal{A}' \;\dashv\vdash\; И\tilde{n}.\,[\mathcal{A}']\,.$$

□

Example C.4.5 *: We show an example to illustrate how SAL_{int} formulas can capture non trivial properties expressed using the adjuncts. Let*

$$\mathcal{A} \quad ::= \quad \Big(Hm'.\,m'[\top] \;\blacktriangleright\; \big(Hn_1.\,n_1[0] \mid Hn_2.\,n_2[Hn_3.\,n_3[0]]\big)\Big) \oslash m@m$$

where $Hn.\,\mathcal{A}$ (H being the hidden name quantifier *[CC01]) stands for $Иn.\,n®\mathcal{A}$. The prenex form of $\mathcal{A}$ is*

$$Иm', n_1, n_2, n_3.\; \Big((m'®\top\wedge.\,m'®m'[\top]) \;\blacktriangleright\; \big(n_1®n_1[0] \mid n_2®n_2[n_3®.\,n_3[0]]\big)\Big)\oslash m@m$$

Then $P \models \mathcal{A}$ iff there is Q such that

$$(\nu m)\, m[P] \;\mid\; (\nu m')\, m'[Q] \quad\equiv\quad (\nu n_1)(\nu n_2)(\nu n_3)\big(n_1[\mathbf{0}] \mid n_2[n_3[\mathbf{0}]]\big)$$

The only solutions of this equation are $P \equiv \mathbf{0}$ or $P \equiv (\nu n_3)n_3[\mathbf{0}]$. In other words, $\mathcal{A}$ is equivalent to $\mathcal{B} = 0 \vee Hn_3.\,n_3[0]$.

C.5 Adjuncts elimination and classical quantifiers

In this section we consider a variant of SAL. Instead of fresh quantified formulas, we consider name quantification of the form $\forall x.\,\mathcal{A}$ and $\exists x.\,\mathcal{A}$ with the natural semantics:

$$P \models \forall x.\,\mathcal{A} \qquad \text{if} \qquad \forall n \in \mathcal{N}.\;\; P \models \mathcal{A}\{^n/x\}$$

Let us note $SAL^{\exists}_{int}$ the intensional fragment with classical quantification. We ask the question of adjuncts elimination for extensions of this logic. The undecidability result of [CTZ+04] implies that there is no effective adjunct elimination for $SAL^{\exists}_{int} + \{\triangleright\}$. We establish now a more precise result:

Theorem C.5.1 (Expressiveness of adjuncts in $SAL^{\exists}_{int}$) *$SAL^{\exists}_{int}+\{\triangleright\}$, $SAL^{\exists}_{int}+\{@\}$ and $SAL^{\exists}_{int}+\{\oslash\}$ are strictly more expressive than $SAL^{\exists}_{int}$.*

The proof of this theorem is based on the following observation. In any of the extensions we consider, it is possible to define a formula $\mathcal{A}$ such that

$$P \models \mathcal{A} \qquad \text{iff} \qquad \sharp\,\mathrm{fn}(P) \ \leq\ 1 \tag{C.1}$$

For the $\triangleright$ and @ connectives, we may first encode the formula $n = m$ as $\big(n[\top] \wedge \neg m[\top]\big) \triangleright \bot$ and $(n[\top])@m$. Then (C.1) is satisfied by the formula

$$\exists x.\ \forall y.\ \big(\neg y \circledR \top\big) \rightarrow x = y$$

For the $\oslash$ connective, there is a direct formula satisfying (C.1):

$$\exists x.\ \big(\forall y.\, y \circledR \top\big) \oslash x$$

We are now interested in proving that such a property cannot be expressed in $SAL_{int}^{\exists}$. Our approach consists in studying the stability of $\models$ with respect to substitutions. We actually find some particular processes P for which $P \models \mathcal{A}$ is equivalent to $P \models \mathcal{A}\{^n/m\}$. From this, we deduce processes P such that $P \models \mathcal{A}$ implies $P\{^n/m\} \models \mathcal{A}$. This last result shows that, on certain conditions, a formula may not observe the action of equating two names in a process, which is contradictory with counting the number of free names.

We call *thread context* a context $\mathcal{C}$ of the form

$$\mathcal{C}[\,P\,]\ \equiv\ (\nu\tilde{n})\, n_1[\ldots n_k[\,P\,]\ldots]$$

with $\tilde{n} \subseteq \{n_1, \ldots, n_k\}$. We note $n(\mathcal{C}) \stackrel{\text{def}}{=} \{n_1, \ldots, n_k\}$ and $d(\mathcal{C}) \stackrel{\text{def}}{=} k$. For a formula $\mathcal{A}$, we note $d(\mathcal{A})$ the number of $n[.]$ connectives in $\mathcal{A}$.

Lemma C.5.2 *Let $\mathcal{A}$ be a formula of $SAL_{int}^{\exists}$, and $\mathcal{C}$ a thread context such that $d(\mathcal{C}) > d(\mathcal{A})$. Let n, m be two names such that $\{n, m\} \cap n(\mathcal{C}) = \emptyset$, and*

$$P \ \stackrel{\text{def}}{=}\ \mathcal{C}\big[\,n[\mathbf{0}] \mid m[\mathbf{0}]\,\big]$$

Then $P \models \mathcal{A}$ iff $P \models \mathcal{A}\{^n/m\}$.

Proof: By induction on the size of $\mathcal{A}$:

- the cases $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \neg\mathcal{A}_1$, and $\mathcal{A} = \mathbf{0}$ are trivial.

- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. Assume first $P \models \mathcal{A}$. Since $d(\mathcal{C}) \geq 1$, we may assume by symmetry that $\mathbf{0} \models \mathcal{A}_2$ and $P \models \mathcal{A}_1$. Then $P \models \mathcal{A}_1\{^n/m\}$ by induction, and $P \models \mathcal{A}\{^n/m\}$. The other direction is proved similarly.

- $\mathcal{A} = a[\mathcal{A}_1]$. Assume first $P \models \mathcal{A}$. Then $\mathcal{C} \equiv a[\mathcal{C}']$ and $P' \stackrel{\text{def}}{=} \mathcal{C}'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \mathcal{A}_1$. By induction $P' \models \mathcal{A}_1\{^n/m\}$. Since $\{n, m\} \cap n(\mathcal{C})$, $a \neq m$, so $\mathcal{A}\{^n/m\} = a[\mathcal{A}_1\{^n/m\}]$, and $P \models \mathcal{A}\{^n/m\}$.

  Assume now $P \models \mathcal{A}\{^n/m\}$. Let $b = a\{^n/m\}$. Then $\mathcal{C} \equiv b[\mathcal{C}']$ and $P' \stackrel{\text{def}}{=} \mathcal{C}'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \mathcal{A}_1\{^n/m\}$. Then $b \in n(\mathcal{C})$, so $b \notin \{m, n\}$, and $b = a$. By induction $P' \models \mathcal{A}_1$, so $P \models b[\mathcal{A}_1] = \mathcal{A}$.

- $\mathcal{A} = a \circledR \mathcal{A}_1$. Assume first $P \models \mathcal{A}$. Then $\mathcal{C} \equiv (\nu a)\mathcal{C}'$ and $P' \stackrel{\text{def}}{=} \mathcal{C}'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \mathcal{A}_1$. Since n, m are free in P, $a \neq m$ and $a \neq n$. So $\{n, m\} \cap n(\mathcal{C}') = \emptyset$, and by induction, $P' \models \mathcal{A}_1\{^n/m\}$. $\mathcal{A}\{^n/m\} = a \circledR \mathcal{A}_1\{^n/m\}$, and $P \models \mathcal{A}\{^n/m\}$. The other direction is proved similarly.

- $\mathcal{A} = \forall x.\,\mathcal{A}_1$. Assume first $P \models \mathcal{A}$. Let take $a \in \mathcal{N}$. Then $P \models \mathcal{A}_1\{^a/x\}$, and by induction $P \models \mathcal{A}_1\{^a/x\}\{^n/m\}$. For $a \neq m$, this is also $P \models \mathcal{A}_1\{^n/m\}ax$. For $a = m$, this requires a bit more. Consider that $P \models \mathcal{A}_1\{^n/x\}$. Then $P \models \mathcal{A}_1\{^n/x\}\{^n/m\}$ by induction. But $\mathcal{A}_1\{^n/x\}\{^n/m\} = \big(\mathcal{A}_1\{^n/m\}\{^m/x\}\big)\{^n/m\}$, so by induction $P \models \mathcal{A}_1\{^n/m\}\{^m/x\}$. Hence $P \models \mathcal{A}_1\{^n/m\}\{^a/x\}$ for all a, that is $P \models \forall x.\,\mathcal{A}_1\{^n/m\} = \mathcal{A}\{^n/m\}$.
  Assume now that $P \models \mathcal{A}\{^n/m\}$. Let take $a \in \mathcal{N}$. Then $P \models \mathcal{A}_1\{^n/m\}\{^a/x\}$. If $a \neq m$, this is $P \models \mathcal{A}_1\{^a/x\}\{^n/m\}$, so by induction $P \models \mathcal{A}_1\{^a/x\}$. For $a = m$, consider that $P \models \mathcal{A}_1\{^n/m\}\{^n/x\}$, that is $P \models \mathcal{A}_1\{^m/x\}\{^n/m\}$, so by induction $P \models \mathcal{A}_1\{^m/x\}$. Hence $P \models \mathcal{A}_1\{^a/x\}$ for all a, that is $P \models \mathcal{A}$.

$\square$

Lemma C.5.3 *Let $\mathcal{A}$ be a formula of $SAL^{\exists}_{int}$, and $\mathcal{C}$ a thread context such that $d(\mathcal{C}) > d(\mathcal{A})$. Let n, m be two names such that $\{n, m\} \cap n(\mathcal{C}) = \emptyset$, and moreover $m \notin \mathrm{fn}(\mathcal{A})$. Let*

$$P_1 \stackrel{\text{def}}{=} \mathcal{C}\big[\,n[\mathbf{0}] \mid m[\mathbf{0}]\,\big] \qquad \textit{and} \qquad P_2 \stackrel{\text{def}}{=} \mathcal{C}[\,n[\mathbf{0}] \mid n[\mathbf{0}]\,]$$

If $P_1 \models \mathcal{A}$, then $P_2 \models \mathcal{A}$.

Proof: By induction on the size of $\mathcal{A}$:

- the cases $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2$, $\mathcal{A} = \mathbf{0}$ and $\mathcal{A} = \neg\mathbf{0}$ are trivial.

- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. Since $d(\mathcal{C}) \geq 1$, we may assume by symmetry that $\mathbf{0} \models \mathcal{A}_2$ and $P_1 \models \mathcal{A}_1$. Then $P_2 \models \mathcal{A}_1$ by induction, and $P_2 \models \mathcal{A}$

- $\mathcal{A} = \mathcal{A}_1 \mid\mid \mathcal{A}_2$. Since $d(\mathcal{C}) \geq 1$, $P_1 \models \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathbf{0} \models \mathcal{A}_1 \wedge \mathcal{A}_2$. By induction, $P_2 \models \mathcal{A}_1 \wedge \mathcal{A}_2$, that is $P_2 \models \mathcal{A}$

- $\mathcal{A} = a[\mathcal{A}_1]$. Then $\mathcal{C} \equiv a[\mathcal{C}']$ and $\mathcal{C}'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \mathcal{A}_1$. By induction $\mathcal{C}'[n[\mathbf{0}] \mid n[\mathbf{0}]] \models \mathcal{A}_1$, that is $P_2 \models \mathcal{A}$.

- $\mathcal{A} = \neg a[\mathcal{A}_1]$. Then either $\mathcal{C}$ is not of the form $n[\mathcal{C}']$, and $P_2 \models \neg a[\mathcal{A}_1]$, or $C \equiv n[\mathcal{C}']$ but $\mathcal{C}'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \neg\mathcal{A}_1$. Then by induction $\mathcal{C}'[n[\mathbf{0}] \mid n[\mathbf{0}]] \models \neg\mathcal{A}_1$, that is $P_2 \not\models a[\mathcal{A}_1]$.

- $\mathcal{A} = a \circledR \mathcal{A}_1$. Then $\mathcal{C} \equiv (\nu a)\mathcal{C}'$ and $\mathcal{C}'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \mathcal{A}_1$. Since n, m are free in P, $a \notin \{m, n\}$, so $n(\mathcal{C}') \cap \{m, n\} = \emptyset$. Then by induction, $\mathcal{C}'[n[\mathbf{0}] \mid n[\mathbf{0}]] \models \mathcal{A}_1$, and $P_2 \models \mathcal{A}$.

- $\mathcal{A} = \neg a \circledR \mathcal{A}_1$. Assume first that a is free in P_1. Then $a \neq m$ since $m \notin \mathrm{fn}(\mathcal{A})$ by hypothesis. So a is also free in P_2 and $P_2 \models \mathcal{A}$. Assume now a is fresh for P_1 (and P_2). Let $\mathcal{C}'$ be such that $\mathcal{C} \equiv (\nu a)\mathcal{C}'$. Then $C'[n[\mathbf{0}] \mid n[\mathbf{0}]] \not\models \mathcal{A}_1$, otherwise $C'[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \mathcal{A}_1$ and $P \models \mathcal{A}$. So $P_2 \not\models a \circledR \mathcal{A}_1$.

- $\mathcal{A} = \forall x.\,\mathcal{A}_1$. Let take $a \in \mathcal{N}$. Then $P_1 \models \mathcal{A}_1\{^a/x\}$, and by induction $P_2 \models \mathcal{A}_1\{^a/x\}$ for $a \neq m$. Let take some fresh m'. By equivariance, $P_1(m \leftrightarrow m)' \models \forall x.\,\mathcal{A}_1$, so $P_1(m \leftrightarrow m)' \models \mathcal{A}_1\{^m/x\}$. Applying induction on P_1 and $\mathcal{A}_1\{^m/x\}$ for m' instead of m, we have $P_2 \models \mathcal{A}_1\{^m/x\}$. Hence $P \models \mathcal{A}_1\{^a/x\}$ for all a, that is $P_2 \models \forall x.\,\mathcal{A}_1$.

- $\mathcal{A} = \exists x.\, \mathcal{A}_1$. Let $a \in \mathcal{N}$ be such that $P_1 \models \mathcal{A}_1\{{}^a/x\}$. If $a \neq m$, then we may apply induction on $\mathcal{A}_1\{{}^a/x\}$, and $P_2 \models \mathcal{A}_2\{{}^a/x\}$, that is $P_2 \models \mathcal{A}$. Otherwise $P_1 \models \mathcal{A}_1\{{}^m/x\}$. By Lemma C.5.2, $P_1 \models \mathcal{A}_1\{{}^m/x\}\{{}^n/m\} = \mathcal{A}_1\{{}^n/x\}\{{}^n/m\}$, and again $P_1 \models \mathcal{A}_1\{{}^n/x\}$. Then by induction, $P_2 \models \mathcal{A}_1\{{}^n/x\}$, that is $P_2 \models \mathcal{A}$. □

This last result implies the desired property about $SAL^{\exists}_{int}$:

Proposition C.5.4 *There is no formula in $SAL^{\exists}_{int}$ that satisfies (C.1).*

Proof: Let us assume by absurd we have some $\mathcal{A}$ such that

$$P \models \mathcal{A} \qquad \text{iff} \qquad \sharp\,\mathrm{fn}(P) \ \leq\ 1$$

Then let $\mathcal{C}$ be the thread context of the form $(\nu a)a[\ldots a[.]\ldots]$, and $d(\mathcal{C}) = d(\mathcal{A}) + 1$. Let m, n be two fresh names. Then $\mathcal{C}[n[\mathbf{0}] \mid m[\mathbf{0}]] \models \neg\mathcal{A}$ by definition of $\mathcal{A}$, so by Lemma C.5.3, $\mathcal{C}[n[\mathbf{0}] \mid n[\mathbf{0}]] \models \neg\mathcal{A}$. Moreover, by definition of $\mathcal{A}$, $\mathcal{C}[n[\mathbf{0}] \mid n[\mathbf{0}]] \models \mathcal{A}$, so the contradiction. □

C.6 Minimality of SAL_{int}

In this section, we show minimality w.r.t. expressive power of SAL_{int}.

Theorem C.6.1 (Minimality) *SAL_{int} is a minimal logic, that is all fragments of SAL_{int} are less expressive.*

This result is the consequence of several technical lemmas for each connective. We may distinguish two forms of contribution to the expressiveness of the logic. We will say that a connective κ is *expressive* when there is a property expressed by a formula containing κ that cannot be expressed otherwise. As a consequence, this connective must belong to any minimal fragment. We will also say that a connective κ is *separative* when there exists two models P_1, P_2 and a formula containing κ satisfied by P_1 but not P_2, such that all κ-free formulas equally satisfy P_1 and P_2. Separative connectives are expressive as well, but in a deeper way: removing them, one reduces the separation power of the logic. For SAL_{int}, we will now establish the following classification:

- connectives $. \mid .$, n®$.$, and $n[.]$ are separative,
- connectives $0, \wedge, \neg, И$ are expressive but not separative.

In particular, SAL_{int} is minimal in terms of expressiveness, but as far as separation power is concerned, the minimal fragment is $SAL_{int} - \{И, \neg, \wedge, 0\}$, since for this fragment logical equivalence coincides with intensional bisimilarity.

Notice that we do not show that SAL_{int} is the *unique* minimal fragment of SAL. This is far from being obvious.

Example C.6.2 *The fragment $SAL - \{\wedge\}$ is surprisingly quite expressive, as the formula*

$$\neg И n.\, n ® \neg n ® \ \big(И m_1.\, m_1 ® И m_2.\, m_2 ® m_1[m_2[0]]\big) \oslash n_1 \oslash n_2$$

shows. This formula is equivalent to $n_1[n_2[0]] \vee n_2[n_1[0]]$, and hence the proof of expressiveness of $\wedge$ (see below) must be carried out in a different way. We do not know the exact expressiveness of this fragment, one could think that it captures any finite set of processes. The interested reader may want to look for a formula for $n_1[0] \vee n_2[n_2[0]]$ in this fragment.

C.6.1 Separative connectives

We establish now that the connectives $. \mid .$, n®$.$, and $n[.]$ are separative. Intuitively, $\mid$ carries the ability of SAL_{int} to count, so without this connective it will not be possible to distinguish $n[\mathbf{0}] \mid n[\mathbf{0}]$ from $n[\mathbf{0}] \mid n[\mathbf{0}] \mid n[\mathbf{0}]$; in the same way, $n[.]$ is necessary to separate $n_1[n_2[\mathbf{0}]]$ from $n_2[n_1[\mathbf{0}]]$, and n®. is the only way of specifying properties of hidden names, so it must be required to distinguish $(\nu n)n[\mathbf{0}]$ and $(\nu n)n[n[\mathbf{0}]]$.

Lemma C.6.3 *If $\mathcal{A} \in SAL_{int} - \{\mid\}$, then $P_1 = n[\mathbf{0}] \mid n[\mathbf{0}] \models \mathcal{A}$ iff $P_2 = n[\mathbf{0}] \mid n[\mathbf{0}] \mid n[\mathbf{0}] \models \mathcal{A}$.*

Proof: By absurd, suppose there exists a formula $\mathcal{A}$ telling apart P_1 from P_2, take a minimal such $\mathcal{A}$, and reason by case analysis on $\mathcal{A}$.

- the cases $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \neg\mathcal{A}_1$ and $\mathcal{A} = И m \mathcal{A}_1$ are straightforward.
- if $\mathcal{A} = 0$, then none of P_1, P_2 does satisfy $\mathcal{A}$.
- $\mathcal{A} = m$®$\mathcal{A}_1$: if $m = n$, then none of those processes do satisfy $\mathcal{A}$, otherwise the process satisfying $\mathcal{A}$ does satisfy $\mathcal{A}_1$, and $\mathcal{A}_1$ is a smaller separating formula.
- $\mathcal{A} = m[\mathcal{A}_1]$: none of the two processes do satisfy $\mathcal{A}$.

□

Lemma C.6.4 *If $\mathcal{A} \in SAL_{int} - \{n[.]\}$, then for any names n_1, n_2, we set $P_1 = n_1[n_2[\mathbf{0}]]$ and $P_2 = n_2[n_1[\mathbf{0}]]$. Then $P_1 \models \mathcal{A}$ iff $P_2 \models \mathcal{A}$.*

Proof: As above, by absurd and case analysis on a minimal $\mathcal{A}$:

- the cases $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \neg\mathcal{A}_1$ and $\mathcal{A} = И m \mathcal{A}_1$ are straightforward.
- if $\mathcal{A} = 0$, then none of P_1, P_2 do satisfy $\mathcal{A}$.
- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. We may assume by symmetry that $P_1 \models \mathcal{A}$. Also by symmetry, we may assume $P_1 \models \mathcal{A}_1$ and $\mathbf{0} \models \mathcal{A}_2$. If $P_2 \not\models \mathcal{A}$, then $\mathcal{A}_1$ separates P_1 from P_2 and is a smaller formula: contradiction.
- $\mathcal{A} = m$®$\mathcal{A}_1$: if $m \in \{n_1, n_2\}$, then none of the two processes do satisfy $\mathcal{A}$, otherwise the process satisfying $\mathcal{A}$ also satisfies $\mathcal{A}_1$, and $\mathcal{A}_1$ is a smaller separating formula.

□

Lemma C.6.5 *Assume $\mathcal{A} \in SAL_{int} - \{n[.]\}$, We set $P_1 = (\nu n)n[n[\mathbf{0}]]$ and $P_2 = (\nu n)n[\mathbf{0}]$. Then $P_1 \models \mathcal{A}$ iff $P_2 \models \mathcal{A}$.*

Proof: Again, by absurd and case analysis on a minimal $\mathcal{A}$:

- the cases $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \neg\mathcal{A}_1$ and $\mathcal{A} = И m \mathcal{A}_1$ are straightforward.
- if $\mathcal{A} = 0$, then none of P_1, P_2 do satisfy $\mathcal{A}$.
- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. We may assume by symmetry that $P_1 \models \mathcal{A}$. Also by symmetry, we may assume $P_1 \models \mathcal{A}_1$ and $\mathbf{0} \models \mathcal{A}_2$. If $P_2 \not\models \mathcal{A}$, then $\mathcal{A}_1$ separates P_1 from P_2 and is a smaller formula: contradiction.
- $\mathcal{A} = m[\mathcal{A}_1]$: none of P_1, P_2 do satisfy $\mathcal{A}$.

□

C.6.2 Expressive connectives

We show that the connectives $\wedge, \neg, \text{И}, 0$ are expressive. Expressiveness proofs are more subtle than in the separability cases, since the loss of expressiveness is less sensitive. The scheme of the proof that the connective κ is expressive is to find a property (cardinality, stability by substitution, truncation...) common to all set of models corresponding to any formula without κ, and a formula with κ whose set of models does not have this property.

$\wedge$ is expressive

By duality, $\wedge$ expresses disjunction; we will show that the intensional logic may not express the disjunction present in the formula $n_1[n_2[0]] \vee n_2[n_1[0]]$ without the $\wedge$ connective.

Remark C.6.6 *The $\wedge$ connective is probably the connective whose expressiveness is the most difficult to characterise. It would be even more difficult if one had to take into account adjuncts. As shown in Example C.6.2, we may express the formula $n_1[n_2[0]] \vee n_2[n_1[0]]$ in $SAL - \{\wedge\}$ using adjuncts.*

We note $\mathcal{P}_2(\mathcal{N}) = \{\{n_1, n_2\} : n_1 \neq n_2\}$. We note $K_n = \{\{n, m\} : m \neq n\}$. We say that $K \subseteq \mathcal{P}_2(\mathcal{N})$ is cofinite if there is $N \subseteq \mathcal{N}$, N finite, such that for all $n_1, n_2 \notin N$, if $n_1 \neq n_2$ then $\{n_1, n_2\} \in K$. We may remark that K_1, K_2 are cofinite iff $K_1 \cap K_2$ is cofinite, and K is cofinite iff $K - K_n$ is cofinite.

Lemma C.6.7 *Assume $\mathcal{A}$ is a formula of $SAL_{int} - \{\wedge\}$ such that $\mathbf{0} \not\models \mathcal{A}$. We set*

$$K_{\mathcal{A}} \stackrel{\text{def}}{=} \{\ \{n_1, n_2\} : n_1 \neq n_2,\ n_1[n_2[\mathbf{0}]] \models \mathcal{A} \text{ and } n_2[n_1[\mathbf{0}]] \models \mathcal{A}\ \}.$$

Then either $K_{\mathcal{A}} = \emptyset$ or $K_{\mathcal{A}}$ is cofinite.

Proof: By induction on $\mathcal{A}$:

- $\mathcal{A} = \text{И}n.\, \mathcal{A}_1$. Then $\mathbf{0} \not\models \mathcal{A}_1$, and for any n_1, n_2 s.t. $n_1 \neq n, n_2 \neq n$ and $n_1 \neq n_2$, $\{n_1, n_2\} \in K_{\mathcal{A}_1}$ iff $\{n_1, n_2\} \in K_{\mathcal{A}_1}$. That is $K_{\mathcal{A}} - K_n = K_{\mathcal{A}_1} - K_n$.
- $\mathcal{A} = 0$: $\mathbf{0} \models \mathcal{A}$.
- $\mathcal{A} = \neg 0$: then $K_A = \mathcal{P}_2$
- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$: since $\mathbf{0} \not\models \mathcal{A}$, we may assume by symmetry that $\mathbf{0} \not\models \mathcal{A}_1$. If also $\mathbf{0} \not\models \mathcal{A}_2$, then $K_{\mathcal{A}} = \emptyset$. Otherwise, $K_{\mathcal{A}} = K_{\mathcal{A}_1}$.
- $\mathcal{A} = \mathcal{A}_1 \mid\mid \mathcal{A}_2$: since $\mathbf{0} \not\models \mathcal{A}$, $\mathbf{0} \not\models \mathcal{A}_1$ and $\mathbf{0} \not\models \mathcal{A}_2$. then $K_{\mathcal{A}} = K_{\mathcal{A}_1} \cap K_{\mathcal{A}_2}$.
- $\mathcal{A} = n[\mathcal{A}_1]$: then $K_{\mathcal{A}} = \emptyset$.
- $\mathcal{A} = \neg n[\mathcal{A}_1]$: then $\mathcal{P}_2(\mathcal{N}) - K_n \subseteq K_{\mathcal{A}}$, so $K_{\mathcal{A}}$ is cofinite.
- $\mathcal{A} = n \circledR \mathcal{A}_1$: then $\mathbf{0} \not\models \mathcal{A}_1$, and $K_{\mathcal{A}} - K_n = K_{\mathcal{A}_1} - K_n$.
- $\mathcal{A} = \neg n \circledR \mathcal{A}_1$: then $\mathbf{0} \not\models \mathcal{A}_1$, and $K_{\mathcal{A}} - K_n = K_{\neg\mathcal{A}_1} - K_n$.

□

Lemma C.6.8 *Let n_1, n_2 be two distinct names. Then there is no formula $\mathcal{A} \in SAL_{int} - \{\wedge\}$ equivalent to $n_1[n_2[0]] \vee n_2[n_1[0]]$.*

Proof: By absurd: if there is such a formula $\mathcal{A}$, then $\mathbf{0} \not\models \mathcal{A}$. Then by Lemma C.6.7 $\sharp K_{\mathcal{A}} \neq 1$, and the contradiction. □

¬ is expressive

¬ enriches the expressive power in several ways; here we consider the property that the name n occurs free, expressed by $\neg n Ⓡ \top$, and show that negation is necessary to express it. To prove this, we remark that for a formula $\mathcal{A}$ without negation, there is a height h such that for all P, if $P \models \mathcal{A}$ then so does the truncation of P at height h, so we may find a contradiction by considering a process having an occurrence of n deep enough.

Definition C.6.9 *We define the truncation at height $h \in \mathbb{N}$ as $t_0(P) = \mathbf{0}$, and*

$$t_h\big((\nu\tilde{n})(n_1[P_1] \mid \ldots \mid n_r[P_r])\big) \quad = \quad (\nu\tilde{n})(n_1[t_{h-1}(P_1)] \mid \ldots \mid n_r[t_{h-1}(P_r)]).$$

Note that $\mathrm{fn}(t_h(P)) \subseteq \mathrm{fn}(P)$.

Lemma C.6.10 *If $\mathcal{A}$ is a formula without ¬, $s(\mathcal{A}) \leq h$ and $P \models \mathcal{A}$, then $t_h(P) \models \mathcal{A}$.*

Proof: By induction on $\mathcal{A}$:

- $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$: then by induction $t_h(P) \models \mathcal{A}_1$, $t_h(P) \models \mathcal{A}_2$, so $t_h(P) \models \mathcal{A}_1 \wedge \mathcal{A}_2$.
- $\mathcal{A} = Иn.\,\mathcal{A}_1$: then there is $n' \notin \mathrm{fn}(P)$ s.t. $P \models \mathcal{A}_1(n \leftrightarrow n)'$. By induction $t_h(P) \models \mathcal{A}_1(n \leftrightarrow n)'$, $n' \notin \mathrm{fn}(t_h(P))$, so $t_h(P) \models Иn.\,\mathcal{A}_1$.
- $\mathcal{A} = 0$: then $t_h(P) \equiv P \equiv \mathbf{0}$
- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$: then $P \equiv P_1 \mid P_2$ with $P_\epsilon \models \mathcal{A}_\epsilon$, and by induction $t_h(P_\epsilon) \models \mathcal{A}_\epsilon$, so $t_h(P) \models \mathcal{A}$.
- $\mathcal{A} = n[\mathcal{A}_1]$: then $P \equiv n[P_1]$ and $P_1 \models \mathcal{A}_1$. By induction, $t_{h-1}(P_1) \models \mathcal{A}_1$, and so $t_h(P) \models \mathcal{A}$.
- $\mathcal{A} = n Ⓡ \mathcal{A}_1$: then $P \equiv (\nu n)P_1$ with $P_1 \models \mathcal{A}_1$. Then by induction $t_h(P_1) \models \mathcal{A}_1$, so $t_h(P) \models \mathcal{A}$.

□

Lemma C.6.11 *There is no formula $\mathcal{A} \in SAL_{int} - \{\neg\}$ equivalent to $\neg n Ⓡ \bot$.*

Proof: Suppose $\mathcal{A}$ exists, and take $h = s(A)$. We note $P \equiv m[m[\ldots m[\mathbf{0}] \ldots]]$ and $Q \equiv m[m[\ldots m[n[\mathbf{0}]] \ldots]]$ a nesting of h ambients m, for some $m \neq n$. Then $Q \models \mathcal{A}$, $P \not\models \mathcal{A}$, and $P \equiv t_h(Q)$, which contradicts Lemma C.6.10 □

$\reflectbox{N}$ is expressive

$\reflectbox{N}$ is very useful to deal with an hidden name without making any hypothesis on the free names of processes (which revelation taken alone would do). Here we consider the property of having at least one hidden name, that is the model is congruent to $(\nu n)P'$ with $n \in \mathrm{fn}(P')$. This is expressed by the formula $\reflectbox{N} n.\, n®\neg n®\top$. For $N = \{n_1, \dots n_r\}$ we consider $P_N^n = n[n_1[\mathbf{0}] \mid \dots \mid n_r[\mathbf{0}]]$ for some $n \notin N$.

Lemma C.6.12 *Assume some finite set of names N and a quantifier free formula $\mathcal{A}$ such that $\mathrm{fn}(\mathcal{A}) \subset N$, and $n \notin N$. Then*

$$P_N^n \models \mathcal{A} \quad \textit{iff} \quad (\nu n)P_N^n \models \mathcal{A}$$

Proof: By induction on $\mathcal{A}$:

- the cases $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, and $\mathcal{A} = \neg\mathcal{A}_1$, are straightforward.
- if $\mathcal{A} = 0$: then none of the two processes satisfies $\mathcal{A}$.
- if $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. Assume first that $P_N^n \models \mathcal{A}$. By symmetry, we may assume $P_N^n \models \mathcal{A}_1$ and $\mathbf{0} \models \mathcal{A}_2$. So $(\nu n)P_N^n \models \mathcal{A}_1$ by induction, and $(\nu n)P_N^n \models \mathcal{A}$. If we assume $(\nu n)P_N^n \models \mathcal{A}$, we may do the same reasoning.
- $\mathcal{A} = m[\mathcal{A}_1]$: none of $P_N^n, (\nu n)P_N^n$ does satisfy $\mathcal{A}$.
- $\mathcal{A} = m®\mathcal{A}_1$: then $m \in \mathrm{fn}(\mathcal{A}) \subseteq N$, hence none of $P_N^n, (\nu n)P_N^n$ does satisfy $\mathcal{A}$.

□

Lemma C.6.13 *There is no formula $\mathcal{A} \in SAL_{int} - \{\reflectbox{N}\}$ equivalent to $\reflectbox{N} n.\, n®n®\bot$.*

Proof: By absurd, let $\mathcal{A}$ be such a quantifier free formula, and $\{n_1, \dots, n_r\} = \mathrm{fn}(\mathcal{A})$. Then $P_N^n \not\models \mathcal{A}$, so $(\nu n)P \not\models \mathcal{A}$, by Lemma C.6.12, and the contradiction.
□

0 is expressive

Here we assume we take $\top$ instead of 0 as a primitive formula. Then 0 is not expressible. For this, we remark that for any $\mathcal{A}$ without 0 and for $n \notin \mathrm{fn}(A)$, $\mathbf{0} \models \mathcal{A}$ iff $n[\mathbf{0}] \models \mathcal{A}$.

Lemma C.6.14 *Let $\mathcal{A}$ be a formula without 0, and $n \notin \mathrm{fn}(A)$. Then*

$$\mathbf{0} \models \mathcal{A} \quad \textit{iff} \quad n[\mathbf{0}] \models \mathcal{A}$$

Proof: We reason by induction on $\mathcal{A}$

- $\mathcal{A} = \top$, $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \neg\mathcal{A}_1$: straightforward.
- $\mathcal{A} = \reflectbox{N} m.\, \mathcal{A}_1$: We assume without loss of generality $m \neq n$. If $\mathbf{0} \models \reflectbox{N} m.\, \mathcal{A}_1$, then $\mathbf{0} \models \mathcal{A}_1$. $n[\mathbf{0}] \models \mathcal{A}_1$ by induction, so $n[\mathbf{0}] \models \reflectbox{N} n.\, \mathcal{A}_1$. Conversely, if $n[\mathbf{0}] \models \reflectbox{N} m.\, \mathcal{A}_1$, then $n[\mathbf{0}] \models \mathcal{A}_1$, so $\mathbf{0} \models \mathcal{A}_1$ by induction, and then $\mathbf{0} \models \reflectbox{N} n.\, \mathcal{A}_1$.

- if $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. Assume first that $\mathbf{0}\models\mathcal{A}_1 \mid \mathcal{A}_2$. Then $\mathbf{0}\models\mathcal{A}_1 \wedge \mathcal{A}_2$, hence by induction $n[\mathbf{0}]\models\mathcal{A}_1$, and $n[\mathbf{0}]\models\mathcal{A}_1 \mid \mathcal{A}_2$. If $\mathbf{0} \not\models\mathcal{A}_1 \mid \mathcal{A}_2$, then we may assume by symmetry that $\mathbf{0} \not\models\mathcal{A}_1$. Assume by absurd that $n[\mathbf{0}]\models\mathcal{A}_1 \mid \mathcal{A}_2$. Then $n[\mathbf{0}]\models\mathcal{A}_1$ and $\mathbf{0}\models\mathcal{A}_2$. By induction $\mathbf{0}\models\mathcal{A}_1$ and the contradiction.

- if $\mathcal{A} = m[\mathcal{A}_1]$. Then $m \neq n$ by hypothesis, and both $\mathbf{0} \not\models\mathcal{A}$ and $n[\mathbf{0}] \not\models\mathcal{A}$.

- if $\mathcal{A} = m®\mathcal{A}_1$, $m \neq n$ by hypothesis. If $\mathbf{0}\models\mathcal{A}$, then $\mathbf{0}\models\mathcal{A}_1$, and by induction $n[\mathbf{0}]\models\mathcal{A}_1$ and $n[\mathbf{0}]\models\mathcal{A}$. Conversely, if $n[\mathbf{0}]\models\mathcal{A}$, then $n[\mathbf{0}]\models\mathcal{A}_1$, and $\mathbf{0}\models\mathcal{A}_1$ so $\mathbf{0}\models\mathcal{A}$ by induction.

□

Lemma C.6.15 *There is no formula $\mathcal{A} \in SAL_{int} - \{0\}$ equivalent to* 0.

Proof: By absurd, if $\mathcal{A}$ is such a formula an $n \notin \text{fn}(\mathcal{A})$, then by Lemma C.6.14, $n[\mathbf{0}]\models\mathcal{A}$ and the contradiction. □

C.7 Separation logic and classical logic

In this section, we give a second illustration of our minimalisation method. We consider the assertion language presented in [CYO01], referred as Separation Logic (SL). SL holds spatial connectives $*$ and $-\!\!*$ similar to $\mid$ and $\triangleright$ in SAL, with a light but significant difference for $*$: the composition requires a compatibility condition $h \perp h'$ that is not always satisfied; in particular, it is not possible to compose two copies of the same structure $(h * h)$. As a consequence, the expressiveness of $*$ is quite restricted and essentially express the separation of resources, which equality already expresses. For this reason, we can establish the elimination of both $*$ and $-\!\!*$. We define a classical fragment CL and prove it to be as expressive as SL.

C.7.1 Definitions

We assume a countable set Var of variables, ranged over with x, y, and a set Loc of locations such that $\text{Loc} \subseteq \mathbb{N}$. Expressions and assertions of SL are defined by the following grammar: We write $\mathsf{fv}(P)$ for the set of variables occurring in P.

$$
\begin{array}{lll}
e & ::= & x \mid \mathsf{nil} \mid - \\
P & ::= & (x \mapsto e_1, e_2) \mid x = y \mid \mathsf{emp} \mid \perp \mid P \Rightarrow P \\
 & & \mid P * P \mid P -\!\!* P
\end{array}
$$

Figure C.4: Separation logic (SL)

Assertions express properties of memory states, modelled as a pair consisting of a store and a heap, as follows:

$$
\begin{array}{rcl}
\text{Val} & \stackrel{\text{def}}{=} & \text{Loc} \sqcup \{\mathsf{nil}\} \\
\text{Store} & \stackrel{\text{def}}{=} & \text{Var} \to \text{Val} \\
\text{Heap} & \stackrel{\text{def}}{=} & \text{Loc} \rightharpoonup_{fin} \text{Val} \\
\text{State} & \stackrel{\text{def}}{=} & \text{Stack} \times \text{Heap}
\end{array}
$$

where $\rightharpoonup_{fin}$ stands for a partial function with finite domain. We range over stores with s, over heaps with h, and over states with σ. We note $\sigma_1 \perp \sigma_2$ for $s_1 = s_2$ and $dom(h_1) \cap dom(h_2) = \emptyset$, and, when this holds, $\sigma_1 * \sigma_2$ is the state defined by keeping the same store and by setting $h_1 * h_2(x) = h_1(x)$ or $h_2(x)$.

For a value v, we note $v {\models_\sigma} e$ if either $e = -$, or $v = e = \mathsf{nil}$, or $e = x$ and $v = s(x)$. We then note $(v_1, v_2) {\models_\sigma} (e_1, e_2)$ if $v_1 {\models_\sigma} e_1$ and $v_2 {\models_\sigma} e_2$. The condition for a state σ to match an assertion P, written $\sigma \models P$, is inductively defined as:

$$
\begin{array}{lll}
\sigma \models \perp & \text{never} & \\
\sigma \models (x \mapsto e_1, e_2) & \text{iff} & dom(h) = \{s(x)\} \text{ and} \\
 & & hs(x) {\models_\sigma} (e_1, e_2) \\
\sigma \models x = e_y & \text{iff} & s(x) = s(y) \\
\sigma \models \mathsf{emp} & \text{iff} & dom(h) = \emptyset \\
\sigma \models P_1 \Rightarrow P_2 & \text{iff} & \sigma \models P_1 \text{ implies } \sigma \models P_2 \\
\sigma \models P_1 * P_2 & \text{iff} & \text{there exist } \sigma_1 \text{ and } \sigma_2 \text{ such that} \\
 & & \sigma = \sigma_1 * \sigma_2;\ \sigma_1 \models P_1 \text{ and } \sigma_2 \models P_2 \\
\sigma \models P_1 -\!\!* P_2 & \text{iff} & \text{for all } \sigma_1 \text{ such that } \sigma \perp \sigma_1, \\
 & & \sigma_1 \models P_1 \text{ implies } \sigma * \sigma_1 \models P_2
\end{array}
$$

We may define as usual the connectives $\wedge, \vee, \top, \neg, \Leftrightarrow$ in the obvious way. We also introduce two *monotonic*[1] assertions (cf. Fig C.5). Any assertion of

monotonic assertion	encoding in SL	semantic
$(x \hookrightarrow e_1, e_2)$	$(x \mapsto e_1, e_2) * \top$	$s(x) \in dom(h)$ and $hs(x) {\models_\sigma} (e_1, e_2)$
$\mathsf{size} \geq n$	$\underbrace{\neg\mathsf{emp} * \ldots * \neg\mathsf{emp}}_{n \text{ times}}$	$\sharp dom(h) \geq n$

Figure C.5: Monotonic assertions from SL

this form, or of the form $x = y$ will be said to be *atomic*. In the remainder, we actually take these as primitive, which ensure the encoding of $(x \mapsto e_1, e_2)$ and emp assertions through boolean combinations[2]. We call *classical logic* (CL) the fragment of SL defined by the grammar of Fig C.6. We will note $w(P)$ for the

$$\boxed{P \ ::= \ P \Rightarrow P \ \mid \ \perp \ \mid (x \hookrightarrow e_1, e_2) \ \mid x = y \ \mid \ \mathsf{size} \geq n\,.}$$

Figure C.6: Classical fragment (CL) of SL

maximal n such that $\mathsf{size} \geq n$ is a subassertion of P, and $\mathsf{fv}(P)$ for the set of variables of P.

Our main result is the following:

Theorem C.7.1 *CL is as expressive as SL, i.e. for all assertion P of SL, there exists a classical assertion P' of CL such that $\models P \Leftrightarrow P'$.*

[1] or *intuitionistic*, using the terminology of [Rey02], that is assertions P such that $\sigma \models P$ implies $\sigma' \models P$ for all $\sigma' \geq \sigma$.

[2] On the contrary, it is not possible to encode $(x \hookrightarrow e_1, e_2)$ and $\mathsf{size} \geq n$ from $(x \mapsto e_1, e_2)$ and emp using only boolean combinations; this point is also discussed in conclusion.

At the same time, we also prove the following result: the monotonic (indeed atomic) fragment is as separative as the whole language, that is if two states satisfy the same monotonic assertions, then they satisfy the same assertions.

C.7.2 Proof of the translation

Our proof proceeds in the same way as for SAL: we define an intensional equivalence and prove that it has the precompactness and characteristic formula properties.

Let X be a finite set of variables, and w an integer. We say that two states σ and σ' are intensionally equivalent for X, w, written $\sigma \approx_{X,w} \sigma'$, if for all classical assertion P with $\mathsf{fv}(P) \subseteq X$ and $w(P) \leq w$, $\sigma \models P$ iff $\sigma' \models P$.
Remarks:

1 This definition amounts to say that σ and σ' satisfy the same atomic classical assertions P with $\mathsf{fv}(P) \subseteq X$ and $w(P) \leq w$.

2 Let us write $w(\sigma) = \sharp dom(h)$. Given three natural numbers a, b, w, we write $a =_w b$ if either $a = b$ or $a, b \geq w$. Then for any σ, σ' such that $\sigma \approx_{X,w} \sigma'$, $w(\sigma) =_w w(\sigma')$.

3 Equality assertions $x = y$ only depend on the store. We note $s =_X s'$ if these stores satisfy the same equality assertions with variables in X. Then for any σ, σ' such that $\sigma \approx_{X,w} \sigma'$, $s =_X s'$.

4 Let V be some set of values. We note $v =_V v'$ if either $v = v'$ or $\{v, v'\} \cap V = \emptyset$, and $(v_1, v_2) =_V (v_1', v_2')$ if $v_1 =_V v_1'$ and $v_2 =_V v_2'$. Then for any s, h, h' such that $(s, h) \approx_{X,w} (s, h')$, $dom(h) \cap s(X) = dom(h') \cap s(X)$ due to assertions $x \hookrightarrow -, -$, and for all $l \in s(X) \cap dom(h)$, $h(l) =_{s(X) \cup \{\mathsf{nil}\}} h'(l)$ due to assertions $x \hookrightarrow e_1, e_2$.

Let say more about store equivalence. Consider a store s_0 and a state $\sigma = (s, h)$ such that $s_0 =_X s$. Then we may define a new state $\mathsf{shift}_{s_0 \to \sigma}()$ of store s_0 and heap h' defined such that

- $dom(h) = s_0\big(s^{-1}(dom(h)) \cap X\big) \cup B$ with B some arbitrary set of locations such that $\sharp dom(h) = \sharp dom(h')$ and $B \cap s_0(X) = \emptyset$.
- for all $l \in dom(h')$, if $l = s_0(x)$ and $hs(x) = (s(y), s(z))$ for some $x, y, z \in X$, $h's_0(x)$ is set to be $(s_0(y), s_0(z))$, otherwise $h(l)$ is arbitrarily defined out of $s(X)$.

This is easy to check that σ and $\mathsf{shift}_{s_0 \to \sigma}()$ satisfy the same atomic assertions with variables in X. Moreover, this transformation is compositional, in the sense that $\mathsf{shift}_{s_0 \to (\sigma * \sigma')}(=)\mathsf{shift}_{s_0 \to \sigma}(\,) * \mathsf{shift}_{s_0 \to \sigma}(')$. This transformation is not completely deterministic, but assuming that every choice of a "fresh" value is made different at each time and at each call to $\mathsf{shift}_{\to}()$, $\sigma \bot \tau$ will imply $\mathsf{shift}_{s_0 \to \sigma}(\bot)\mathsf{shift}_{s_0 \to \tau}()$. We actually have the following stronger result:

Lemma C.7.2 *For all assertions $P \in SL$ with $\mathsf{fv}(P) \subseteq X$, $\sigma \models P$ iff $\mathsf{shift}_{s_0 \to \sigma}(\models)P$.*

The proof is straightforward by induction on the assertion P considering previous remarks.

We now recall the equivalence relation defined by Yang in [Yan01] for the decidability proof, and use it to derive the correction of $\approx_{X,w}$.

Definition C.7.3 ($\sim_{s,n,X}$[Yan01]) *Given a stack s, a natural number n and a set X of variables, $\sim_{s,n,X}$ is the relation between heaps such that $h \sim_{s,n,X} h'$ iff*

1 $s(X) \cap dom(h) = s(X) \cap dom(h')$;

2 for all $l \in s(X) \cap dom(h)$, $h(l) =_{s(X)} h'(l)$;

3 $\sharp\big(dom(h) - s(X)\big) \;=_n\; \sharp\big(dom(h') - s(X)\big)$.

The first step of the correction proof is to factorize $\approx_{X,w}$ in $\sim_{s,n,X}$.

Lemma C.7.4 *For any X, w, n such that $n+\sharp X \leq w$, for any $\sigma, \sigma', s, h, h'$ such that $\sigma = (s, h)$, $\sigma \approx_{X,w} \sigma'$, and $\mathsf{shift}_{s\to\sigma}(') = (s, h')$, it holds that $h \sim_{s,n,X} h'$.*

Proof: By Lemma C.7.2, $(s, h) \approx_{X,w} (s, h')$. Then conditions 1 and 2 in Definition C.7.3 holds by Remark 4, so the proof follows from the verification of the condition 3 on the heap size.

Let us assume first that $\sharp\big(dom(h){-}s(X)\big) < n$; then $\sharp dom(h) = k < n{+}\sharp X \leq w$, so $\sigma{\models}P = \mathsf{size} \geq k \wedge \neg\mathsf{size} \geq k+1$, and $w(P) = k+1 \leq w$. By definition of $\approx_{X,w}$, $\sigma'{\models}P$, so $\sharp dom(h') = k = \sharp dom(h)$. Moreover, $s(X) \cap dom(h) = s(X) \cap dom(h')$, so finally $\sharp\big(dom(h) - s(X)\big) \;=\; \sharp\big(dom(h') - s(X)\big)$.

Let us assume now that $\sharp\big(dom(h){-}s(X)\big) \geq n$; and set $k = \min(\sharp dom(h), w)$, so that $\sigma{\models}\mathsf{size} \geq k$, and by definition of $\approx_{X,w}$, $\sigma'{\models}\mathsf{size} \geq k$. Moreover, $dom(h) \geq n{+}\sharp\big(dom(h){\cap}s(X)\big)$, and $w \geq n{+}\sharp X \geq n{+}\sharp\big(dom(h){\cap}s(X)\big)$, so finally $k \geq n{+}\sharp\big(dom(h){\cap}s(X)\big)$. This gives $dom(h') \geq k \geq n{+}\sharp\big(dom(h'){\cap}s(X)\big)$ since $s(X) \cap dom(h) = s(X) \cap dom(h')$, i.e. $\sharp\big(dom(h') - s(X)\big) \geq n$.

$\sharp dom(h) \;\geq\; k \;\geq\; n + \sharp\big(dom(h) \cap s(X)\big)$, where $k = \min(\sharp dom(h), w)$. So $\sigma{\models}\mathsf{size} \geq k$, and by definition of $\approx_{X,w}$, $\sigma'{\models}\mathsf{size} \geq k$, so that finally $\sharp dom(h') \geq n + \sharp\big(dom(h') \cap s(X)\big)$. □

We now recall the correction result obtained by Yang and derive our correction from it. We first recall the notion of formula's size used by Yang:

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\mid (e \mapsto e_1, e_2) \mid & = & 1 & \mid e_1 = e_2 \mid & = & 0 & \mid \mathsf{emp} \mid & = & 1\\
\mid P \Rightarrow Q \mid & = & \max(\mid P \mid, \mid Q \mid) & \mid \bot \mid & = & 0 & & & \\
\mid P * Q \mid & = & \mid P \mid + \mid Q \mid & \mid P \mathrel{-\!\!*} Q \mid & = & \mid Q \mid & & &
\end{array}
$$

Lemma C.7.5 *Take s, h, h', n, X with $h \sim_{s,n,X} h'$. Then for all assertion $P \in SL$ such that $\mathsf{fv}(P) \subseteq X$ and $\mid P \mid \leq n$, $(s, h){\models}P$ iff $(s, h'){\models}P$.*

The proof of this result is detailed in [Yan01].

Corollary C.7.6 (Correction) *Take σ, σ', w, X with $\sigma \approx_{X,w} \sigma'$. Then for all assertion $P \in SL$ such that $\mathsf{fv}(P) \subseteq X$ and $\mid P \mid + \sharp X \leq w$, $\sigma{\models}P$ iff $\sigma'{\models}P$.*

Proof: By Lemma C.7.4, $h \approx_{s,n,X} h'$ with $\sigma = (s, h)$, $\mathsf{shift}_{s\to\sigma}(') = (s, h')$, and $n = w - \sharp X$. Then $\sigma{\models}P$ implies $\mathsf{shift}_{s\to\sigma}('){\models}P$ by Lemma C.7.5, which implies $\sigma'{\models}P$ by Lemma C.7.2. □

We may now end the proof establishing the properties of precompactness and characteristic formula for $\approx_{X,w}$.

We write $\Phi_{X,w}$ for the set of atomic assertions P such that $\mathsf{fv}(P) \subseteq X$ and $w(P) \leq w$. For X finite, $\Phi_{X,w}$ is finite as well. This has two important consequences:

Proposition C.7.7 (Precompactness) *For all w and all finite X, $\approx_{X,w}$ has only finitely many classes.*

Proof: A class is represented by a subset $\Phi \subseteq \Phi_{X,w}$ of atomic assertions that are the ones satisfied by any state of the class. So there are less than $2^{\sharp\Phi_{X,w}}$ distinct classes. $\square$

Proposition C.7.8 (Characteristic formula) *For all states σ, for all X, w, there is a classical assertion $F_\sigma^{(X,w)}$ such that*

$$\forall \sigma'. \quad \sigma' \models F_\sigma^{(X,w)} \quad \textit{iff} \quad \sigma \approx_{X,w} \sigma' .$$

Proof: Take

$$\bigwedge_{\sigma \models P, P \in \Phi_{X,w}} P \quad \wedge \bigwedge_{\sigma \not\models P, P \in \Phi_{X,w}} \neg P .$$

$\square$

We may now establish Theorem C.7.1 noticing that any assertion P of SL is equivalent to the classical assertion:

$$\bigvee_{C \in \mathrm{State}_{/\approx_{X,w}}, C \models P} F_C^{(X,n)} ,$$

where finiteness of this disjunction is ensured by Proposition C.7.7.

Appendix D

Elimination of Quantifiers in spatial logics for concurrency

This appendix presents the proofs of quantifiers elimination for some extensions of the logic $\mathcal{L}_{spat}^{\diamond}$ related to various calculi: CCS, π-calculus and the Mobile Ambients. From this we establish several results, in particular the undecidability of $\mathcal{L}_{spat}^{\diamond}$ on all these calculi. This results have been discussed in chapters 8 and 9, and are also presented in [CL04].

We first recall all the useful definitions (section D.1), then we establish the elimination of quantifiers (section D.2), namely the fact that $\mathcal{L}_{mod}^{(CCS)}$ may be expressed (in a precise sense) by its fragment $\mathcal{L}_{spat}^{\diamond}$. Then we establish the impossibility of the elimination of the composition adjunct (section D.4), and the undecidability of $\models$ for these two logics (section D.5). We conclude saying how these results may be extended to the π-calculus and Mobile Ambients calculi (section D.6

D.1 Preliminaries

In this section, we introduce the process calculus and spatial logics considered in this work. For the process calculus, we pick a fairly small fragment of CCS.

Definition D.1.1 *Assume given an infinite set* Act *of* actions*, ranged over by* α, β. Processes *are defined by the grammar:* $P, Q, R \ ::= \ \mathbf{0} \ \mid \ P \mid Q \ \mid \ \alpha.P.$

Actions are given in pairs of distinct (co)actions, characterized by the involution $\mathbf{co} : \mathsf{Act} \to \mathsf{Act}$ sending α into $\overline{\alpha}$, and such that $\overline{\overline{\alpha}} = \alpha$. The relation of *structural congruence* is defined as the least congruence $\equiv$ on processes such that $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, and $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. Structural congruence represents identity of the spatial structure of processes. Dynamics of processes is captured by labeled transitions.

Definition D.1.2 *Given the set* $\mathcal{L} \stackrel{\text{def}}{=} \{\tau\} \cup \mathsf{Act}$ *of* labels*, the relation of* labeled

$$
\begin{array}{lllll}
P, v & \models M & \neg\mathcal{A} & \text{if} & \textit{not } P, v \models M \mathcal{A} \\
P, v & \models M & \mathcal{A} \wedge \mathcal{B} & \text{if} & P, v \models M \mathcal{A} \textit{ and } P, v \models M \mathcal{B} \\
P, v & \models M & 0 & \text{if} & P \equiv 0 \\
P, v & \models M & \mathcal{A} \mid \mathcal{B} & \text{if} & \exists Q, R.\ P \equiv Q \mid R \textit{ and } Q, v \models M \mathcal{A} \textit{ and } R, v \models M \mathcal{B} \\
P, v & \models M & \mathcal{A} \triangleright \mathcal{B} & \text{if} & \forall Q \in M,\ Q, v \models M \mathcal{A} \textit{ implies } P \mid Q, v \models M \mathcal{B} \\
P, v & \models M & \exists x.\, \mathcal{A} & \text{if} & \exists \alpha \in \mathsf{Act}.\ P, (v\{^x/\alpha\}) \models M\ \mathcal{A} \\
P, v & \models M & \diamond\mathcal{A} & \text{if} & \exists P'.\ P \longrightarrow P' \textit{ and } P', v \models M \mathcal{A} \\
P, v & \models M & \langle x \rangle \mathcal{A} & \text{if} & \exists P'.\ P \xrightarrow{v(x)} P' \textit{ and } P', v \models M \mathcal{A} \\
P, v & \models M & \langle \overline{x} \rangle \mathcal{A} & \text{if} & \exists P'.\ P \xrightarrow{\overline{v(x)}} P' \text{ and } P', v \models M \mathcal{A}
\end{array}
$$

Figure D.1: Semantics of formulas

transition *is defined by the rules*

$$
\alpha.\, P \xrightarrow{\alpha} P \qquad\qquad P \xrightarrow{\ell} P' \Rightarrow P \mid Q \xrightarrow{\ell} P' \mid Q
$$
$$
P \xrightarrow{\alpha} P', Q \xrightarrow{\overline{\alpha}} Q' \Rightarrow P \mid Q \xrightarrow{\tau} P' \mid Q'
$$

Notice that $\xrightarrow{\alpha}$ is closed under $\equiv$, and that $\xrightarrow{\tau}$ corresponds to the usual relation of *reduction*, noted $\longrightarrow$. We define the *depth* of a process P (maximal nesting of actions in a process P) by letting $\mathsf{ds}(\mathbf{0}) = 0$, $\mathsf{ds}(\alpha.\, P) = 1 + \mathsf{ds}(P)$, and $\mathsf{ds}(P \mid Q) = \max(\mathsf{ds}(P), \mathsf{ds}(Q))$. Let M_K denote the set of all processes whose depth does not exceed K: $M_K \stackrel{\text{def}}{=} \{P \mid \mathsf{ds}(P) \leq K\}$. Then $M_\infty \stackrel{\text{def}}{=} \bigcup_{k \in \mathbb{N}} M_k$ coincides with the set of all processes. We also define the projection (by truncation) $\pi_k : M_\infty \rightarrow M_k$, by induction on k by letting $\pi_0(P) = 0$, $\pi_{k+1}\ (\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$, $\pi_k(P \mid Q) \stackrel{\text{def}}{=} \pi_k(P) \mid \pi_k(Q)$, and $\pi_{k+1}(\alpha.\, P) \stackrel{\text{def}}{=} \alpha.\, \pi_k(P)$.

Having defined the intended process model, we turn to logics. The logic we consider includes the basic spatial operators found in all spatial logics namely: the composition operator $\mid$, the void operator 0, and the composition adjunct operator $\triangleright$ (guarantee). To these connectives, we add the temporal operator $\diamond$ (next step), to capture the dynamic behavior of processes. These operators may be considered the core connectives for spatial logics for concurrency. We then consider the extension of the core with modalities for actions (*cf.* Hennessy-Milner logic), and quantifiers ranging over actions.

Definition D.1.3 *Given an infinite set* Var *of variables,* $(x, y \in \mathsf{Var})$ *formulas are given by:*

$$
\begin{array}{lllllllllllll}
\mathcal{A}, \mathcal{B} & ::= & \mathcal{A} \wedge \mathcal{B} & \mid & \mathcal{A} \mid \mathcal{B} & \mid & \neg\mathcal{A} & \mid & \mathcal{A} \triangleright \mathcal{B} & \mid & 0 & \mid & \diamond\mathcal{A} \quad (\mathcal{L}^{\diamond}_{spat}) \\
 & \mid & \langle x \rangle \mathcal{A} & \mid & \langle \overline{x} \rangle \mathcal{A} & \mid & \exists x.\, \mathcal{A} & & & & & & \quad (\mathcal{L}^{(CCS)}_{mod})
\end{array}
$$

We write $\mathcal{L}^{\diamond}_{spat}$ for the set of formulas in the pure spatial fragment, and $\mathcal{L}^{(CCS)}_{mod}$ for the set of all formulas. *Free variables* of formulas are defined as usual; we say a formula is *closed* if it has no free variables. Semantics is defined in Fig. D.1 by a relation of satisfaction. *Satisfaction* is expressed by $P, v \models M \mathcal{A}$ where P is a process, M is a set of processes, $\mathcal{A}$ a formula, and v is a valuation for the free variables of $\mathcal{A}$. A *valuation* is a mapping from a finite subset of Var to Act. For any valuation v, we write $v\{^x/\alpha\}$ for the valuation v' such that $v'(x) = \alpha$, and $v'(y) = v(y)$ if $y \neq x$. By $\emptyset$ we denote the empty valuation. Notice that this definition of satisfaction matches the usual one except for the presence of the index M, which specifies the range of quantification for interpreting the adjunct (see clause for $\triangleright$). This generalization is only a convenience for our technical development; it is clear that $\models_{M_\infty}$ corresponds to the standard non-relativized relation of satisfaction. So, we abbreviate $P, v \models_{M_\infty} \mathcal{A}$ by $P, v \models \mathcal{A}$, moreover, when the formula $\mathcal{A}$ is closed we abbreviate $P, \emptyset \models_M \mathcal{A}$ by $P \models_M \mathcal{A}$. By default, the set of processes M is M_∞, so that we may abbreviate $P{\models}\mathcal{A}$ for $P{\models}M_\infty\mathcal{A}$.

An *action permutation* is a bijection $\sigma : \mathsf{Act}{\rightarrow}\mathsf{Act}$ such that $\sigma(\overline{\alpha}) = \overline{\sigma(\alpha)}$. We write $(\alpha \leftrightarrow \beta)$ for the action permutation that swaps α and β. Satisfaction verifies the fundamental property of equivariance, which in our present setting is formulated as follows.

Definition D.1.4 *Let $\equiv_s$ be the binary relation on processes defined by $P \equiv_s Q$ if and only if there is an action permutation σ such that $P \equiv \sigma(Q)$.*

Proposition D.1.5 (Equivariance) *Let $P, v \models_M \mathcal{A}$. For every action permutation σ, if $P \equiv \sigma(Q)$ then $Q, \sigma(v) \models_M \mathcal{A}$.*

We frequently refer to equivalence $=_L$ induced on processes induced by the logic L (where L is either $\mathcal{L}^{\diamond}_{spat}$ or $\mathcal{L}^{(CCS)}_{mod}$). The relation $=_L$ is defined by setting $P{=_L}Q$ if for all closed formulas $\mathcal{A}$, we have $P, \emptyset \models \mathcal{A}$ if and only if $Q, \emptyset \models \mathcal{A}$.

Besides the basic stock of primitive connectives, we also use a few derived ones: we list their definition and formal meaning in Fig. D.2. By $w(\mathcal{A})$ we denote the maximal level of nesting of composition $|$ in the formula $\mathcal{A}$, and by $\mathsf{ds}(\mathcal{A})$ the maximal nesting of dynamic modalities in the formula $\mathcal{A}$, defined by

$$\begin{array}{l}
w(0) = 0\\
w(\mathcal{A} \wedge \mathcal{B}) = w(\mathcal{A} \triangleright \mathcal{B}) = \max\left(w(\mathcal{A}), w(\mathcal{B})\right)\\
w(\mathcal{A} \mid \mathcal{B}) = 1 + \max\left(w(\mathcal{A}), w(\mathcal{B})\right)\\
w(\diamond\mathcal{A}) = w(\neg\mathcal{A}) = w(\exists x.\, \mathcal{A}) =\\
\qquad w(\langle x\rangle\mathcal{A}) = w(\langle\overline{x}\rangle\mathcal{A}) = w(\mathcal{A})
\end{array}$$

$$\begin{array}{l}
\mathsf{ds}(0) = 0\\
\mathsf{ds}(\mathcal{A} \wedge \mathcal{B}) = \mathsf{ds}(\mathcal{A} \mid \mathcal{B}) =\\
\qquad \mathsf{ds}(\mathcal{A} \triangleright \mathcal{B}) = \max\left(\mathsf{ds}(\mathcal{A}), \mathsf{ds}(\mathcal{B})\right)\\
\mathsf{ds}(\diamond\mathcal{A}) = \mathsf{ds}(\langle x\rangle\mathcal{A}) =\\
\qquad \mathsf{ds}(\langle\overline{x}\rangle\mathcal{A}) = \mathsf{ds}(\mathcal{A}) + 1\\
\mathsf{ds}(\neg\mathcal{A}) = \mathsf{ds}(\exists x.\, \mathcal{A}) = \mathsf{ds}(\mathcal{A})
\end{array}$$

It is easy to see that a formula $\mathcal{A}$ cannot inspect the part of the process that lies deeper than the depth of $\mathcal{A}$. As a consequence, the restriction to M_k of the denotation of a formula of depth k completely charaterizes its denotation, in the precise sense of:

Proposition D.1.6 (Depth finiteness) *For all formulas $\mathcal{A} \in \mathcal{L}^{(CCS)}_{mod}$, for all $k > \mathsf{ds}(\mathcal{A})$, for all processes P, and for all valuations v,*

$$P, v{\models}M_\infty\mathcal{A} \ \text{ if and only if } \ \pi_k(P), v{\models}M_\infty\mathcal{A} \ \text{ if and only if } \ \pi_k(P), v{\models}M_k\mathcal{A}.$$

The following notation will be used. The process $P_1 \mid \ldots \mid P_n$ is abbreviated by $\prod_{i=1\ldots n} P_i$, and by P^n we denote the process $\prod_{i=1\ldots n} P$. In the same way, we abbreviate the formula $\mathcal{A}_1 \mid \ldots \mid \mathcal{A}_n$ by $\prod_{i=1\ldots n} \mathcal{A}_i$, and $\mathcal{A}^n$ then denotes $\prod_{i=1\ldots n} \mathcal{A}$.

$$
\begin{array}{lcl@{\qquad}lcl}
\top & \stackrel{\text{def}}{=} & 0 \vee \neg 0 & \bot & \stackrel{\text{def}}{=} & \neg\top \\
\mathcal{A} \vee \mathcal{B} & \stackrel{\text{def}}{=} & \neg(\neg\mathcal{A} \wedge \neg\mathcal{B}) & \mathcal{A} \to \mathcal{B} & \stackrel{\text{def}}{=} & \neg\mathcal{A} \vee \mathcal{B} \\
\forall x.\, \mathcal{A} & \stackrel{\text{def}}{=} & \neg\exists x.\, \neg\mathcal{A} & \mathcal{A} \| \mathcal{B} & \stackrel{\text{def}}{=} & \neg(\neg\mathcal{A} \mid \neg\mathcal{B}) \\
\mathcal{A} \blacktriangleright \mathcal{B} & \stackrel{\text{def}}{=} & \neg(\mathcal{A} \triangleright \neg\mathcal{B}) & \mathcal{A}^{\forall} & \stackrel{\text{def}}{=} & \mathcal{A} \| \bot \\
\mathcal{A}^{\exists} & \stackrel{\text{def}}{=} & \mathcal{A} \mid \top & \mathcal{A}^{\vdash} & \stackrel{\text{def}}{=} & (\neg\mathcal{A}) \triangleright \bot \\
x = y & \stackrel{\text{def}}{=} & \big((\langle x \rangle 0 \mid \langle \overline{y} \rangle 0) \to \diamond 0\big)^{\vdash} & x = \overline{y} & \stackrel{\text{def}}{=} & \big((\langle x \rangle 0 \mid \langle y \rangle 0) \to \diamond 0\big)^{\vdash}
\end{array}
$$

$$
\begin{array}{lll}
P, v \models M \top & \text{if} & \textit{always} \\
P, v \models M \bot & \text{if} & \textit{never} \\
P, v \models M \mathcal{A} \vee \mathcal{B} & \text{if} & P, v \models M \mathcal{A} \text{ or } P, v \models M \mathcal{B} \\
P, v \models M \mathcal{A} \to \mathcal{B} & \text{if} & P, v \models M \mathcal{A} \textit{ implies } P, v \models M \mathcal{B} \\
P, v \models M \forall x.\, \mathcal{A} & \text{if} & \forall \alpha \in \mathsf{Act}.\ P, (v\{^{x}/\alpha\}) \models M \mathcal{A} \\
P, v \models M \mathcal{A} \| \mathcal{B} & \text{if} & \forall Q, R.\ P \equiv Q \mid R \textit{ implies } Q, v \models M \mathcal{A} \text{ or } R, v \models M \mathcal{B} \\
P, v \models M \mathcal{A} \blacktriangleright \mathcal{B} & \text{if} & \exists Q \in M.\ Q \models M \mathcal{A} \textit{ and } P \mid Q \models M \mathcal{B} \\
P, v \models M \mathcal{A}^{\forall} & \text{if} & \forall Q, R.\ P \equiv Q \mid R \textit{ implies } Q, v \models M \mathcal{A} \\
P, v \models M \mathcal{A}^{\exists} & \text{if} & \exists Q, R.\ P \equiv Q \mid R \text{ and } Q, v \models M \mathcal{A} \\
P, v \models M \mathcal{A}^{\vdash} & \text{if} & \forall Q \in M.\ Q, v \models M \mathcal{A} \\
P, v \models M x = y & \text{if} & v(x) = v(y) \quad (\text{assuming } M_1 \subseteq M) \\
P, v \models M x = \overline{y} & \text{if} & v(x) = \overline{v(y)} \quad (\text{assuming } M_1 \subseteq M)
\end{array}
$$

Figure D.2: Definition and semantics of derived operators.

D.2 Elimination of quantifiers and action modalities

In this section we prove that, quite surprisingly, the logic $\mathcal{L}^{(CCS)}_{mod}$, which contains quantifiers and variables, can be embedded into the core logic $\mathcal{L}^{\diamond}_{spat}$, which does not seem to contain related constructs, in the sense of the following main result:

Theorem D.2.1 *For any closed formula $\mathcal{A} \in \mathcal{L}^{(CCS)}_{mod}$ and any natural number $K > \mathsf{ds}(\mathcal{A})$, we can effectively construct a formula $[\![\mathcal{A}]\!] \in \mathcal{L}^{\diamond}_{spat}$ such that for all processes P:*

$$P \models \mathcal{A} \ \textit{ if and only if } \ \pi_K(P) \models [\![\mathcal{A}]\!]$$

Notice that this result does not state that $\mathcal{L}^{\diamond}_{spat}$ and $\mathcal{L}^{(CCS)}_{mod}$ have the same expressiveness in the usual sense, however, we should note that the denotation of a formula $\mathcal{A}$ is completely characterized by its denotation on some subset of the models M_k, in the sense of Proposition D.1.6. Hence, the denotation of $[\![\mathcal{A}]\!]$ completely characterizes the denotation of $\mathcal{A}$; this close correspondence will be enough to show the undecidability and separability of $\mathcal{L}^{\diamond}_{spat}$, and independence of the composition adjunct.

The proof of Theorem D.2.1 requires considerable build up. In particular, we need to define $\mathcal{L}^{\diamond}_{spat}$ formulas to characterize processes of several quite specific forms, to be used for various purposes in our encoding of $\mathcal{A}$ into $[\![\mathcal{A}]\!]$. This exercise turns out to be quite interesting: by going through it we get a better understanding about what can be expressed in $\mathcal{L}^{\diamond}_{spat}$, in a sometimes not really obvious way.

We want to reduce a satisfaction judgment $P, v \models M_K \mathcal{A}$, where $\mathcal{A}$ is any $\mathcal{L}^{(CCS)}_{mod}$ formula, into a satisfaction judgment for a formula $[\![\mathcal{A}]\!]$ of $\mathcal{L}^{\diamond}_{spat}$ that neither contains quantifiers, nor action modalities (and thus no occurrences of variables whatsoever). The key idea is to represent the valuation v appearing in $P, v \models M_K \mathcal{A}$ by a certain process $\mathbf{val}(e, \nu, w)_K$, to be composed with the process P being tested for satisfaction. With this trick, the introduction of the valuation entry for x introduced in the valuation clause for $\exists x. \mathcal{A}$ can be mimicked by the introduction of a process using $\blacktriangleright$; action modalities are then interpreted by detecting interactions between the process P being tested and the valuation process $\mathbf{val}(e, \nu, w)_K$. More concretely, we encode the pair P, v by a process of the form $P \mid \mathbf{val}(e, \nu, w)_K$, where $\mathbf{val}(e, \nu, w)_K$ encodes the valuation, and $\nu \circ e = v$ is a decomposition of the valuation v into certain maps $e : \mathsf{Var} \to \mathbb{N}$ and $\nu : \mathbb{N} \to \mathsf{Act}$, respectively called *environment* and *naming*, and w is a natural number. The role of these data will be explained below. The encoding of valuations makes use of the notion of *row* process. A row process $\mathbf{row}(n, \alpha)$ is a sequential process of the form $\alpha. \alpha \ldots \alpha. \mathbf{0}$, where the action α occurs precisely n times (so that $\mathsf{ds}(\mathbf{row}(n, \alpha)) = n$). This process is interesting since it can be characterized logically, and we will use rows to represent bindings between variables (represented by rows of different length) and actions α. Moreover, by imposing a bound K on the depth of the process P one considers, we can easily separate the valuation part from the process that represents the "real" model, in the "soup" $P \mid \mathbf{val}(e, \nu, w)_K$.

We start by introducing formulas whose models are precisely the sequential threads with a given number of actions, in the way we also define the derived modality $?. \mathcal{A}$.

$$
\begin{array}{llllll}
1 & \stackrel{\text{def}}{=} & \neg 0 \wedge (0 \,||\, 0) & \mathsf{Thread}(1) & \stackrel{\text{def}}{=} & 1 \wedge (1 \blacktriangleright \diamond 0) \\
?. \mathcal{A} & \stackrel{\text{def}}{=} & 1 \wedge (\mathsf{Thread}(1) \blacktriangleright \diamond \mathcal{A}) & \mathsf{Thread}(n+1) & \stackrel{\text{def}}{=} & ?. \mathsf{Thread}(n)
\end{array}
$$

We have

Lemma D.2.2 *For all processes P, and M such that $M_1 \subseteq M$*

$$
\begin{array}{lll}
P \models M 1 & \textit{iff} & \exists \alpha \in \mathsf{Act}. \exists Q. P \equiv \alpha. Q \\
P \models M ?. \mathcal{A} & \textit{iff} & \exists \alpha \in \mathsf{Act}. \exists Q. \ P \equiv \alpha. Q \ \textit{and} \ Q \models \mathcal{A} \\
P \models M \mathsf{Thread}(1) & \textit{iff} & \exists \alpha \in \mathsf{Act}. \ P \equiv \alpha. \mathbf{0} \\
P \models M \mathsf{Thread}(k) & \textit{iff} & \exists \alpha_1 \in \mathsf{Act}. \ldots. \exists \alpha_k \in \mathsf{Act}. \ P \equiv \alpha_1. \cdots. \alpha_k. \mathbf{0}
\end{array}
$$

We now give (for each $k \geq 0$) a formula $\mathcal{M}_k$ that characterizes the model M_k, that is, such that we have $P \models \mathcal{M}_k$ if and only if $P \in M_k$.

$$
\mathcal{M}_0 \stackrel{\text{def}}{=} 0 \qquad \mathcal{M}_{k+1} \stackrel{\text{def}}{=} (1 \to ?. \mathcal{M}_k)^{\forall}
$$

Using the $\diamond$ modality as an equality tester, we define a formula $\mathsf{Equals}(k)$ that is satisfied by the processes which belong to M_k, and are compositions of guarded processes all with the *same* first action. We may then specify rows using appropriate formulas

$$
\begin{array}{lcl}
\mathsf{Equals}(k) & \stackrel{\text{def}}{=} & \mathcal{M}_k \ \wedge\ (\mathsf{Thread}(k+1)\ \blacktriangleright\ \big((\mathsf{Thread}(k+1) \mid 1) \rightarrow \diamond\top\big)^{\forall}) \\
\mathsf{RowCol}(0) & \stackrel{\text{def}}{=} & 0 \\
\mathsf{RowCol}(n+1) & \stackrel{\text{def}}{=} & \big(\mathsf{Thread}(n+1) \mid \mathsf{Equals}(1)\big) \wedge \diamond\mathsf{RowCol}(n) \\
\mathsf{Row}(n) & \stackrel{\text{def}}{=} & \mathsf{Thread}(n) \wedge (\top\ \blacktriangleright\ \mathsf{RowCol}(n))
\end{array}
$$

We now prove

Lemma D.2.3 *For all k, and process P, we have:*

$$
\begin{array}{lllll}
P \models & \mathcal{M}_k & \textit{iff} & P \in M_k \\
P \models & \mathsf{Equals}(k) & \textit{iff} & P \in M_k \ \textit{and}\ \exists\alpha \in \mathsf{Act}.\, \exists n \geq 0. \\
 & & & \exists P_1, \ldots, P_n.\quad P \ \equiv\ \alpha.\, P_1 \mid \ldots \mid \alpha.\, P_n \\
P \models & \mathsf{Row}(k) & \textit{iff} & \exists\alpha \in \mathsf{Act}.\ P \equiv \mathbf{row}(k, \alpha)
\end{array}
$$

We can now explain our encoding of a valuation v into a certain process. First, we decompose v into two functions ν and e such that $v \ = \ \nu \circ e$. An *environment* e is a partial injective function from variables to naturals. When the translation process crosses a $\exists x$ construction, we want to allocate a fresh number to x of the natural numbers; to do so, we note by $e\{^x/n\}$ the extension of e with $x \mapsto n$, and $\mid e \mid$ is the maximal value of e, that, is the number of variables already allocated. A *naming* ν is a function from $[1, \ldots, n]$ to Act. Notice that the decomposition $v \ = \ \nu \circ e$ is not unique, but will be given by the order in which existential quantified variables are introduced in their scopes. For any naming ν and environment e the process $\mathbf{val}(e, \nu, w)_K$ is

$$\mathbf{val}(e, \nu, w)_K \ \stackrel{\text{def}}{=} \ \textstyle\prod_{i=1\ldots|e|} \mathbf{row}(K+i, \nu_i)^{2^w}$$

The parameter w specifies the number of rows of the appropriate length that are needed to represent the environment entry for a variable x, and is related to the number of occurrences of $\mid$ in the source formulas. Since interpreting $\mid$ also splits the (encoding of the) valuation, we have to provide enough copies (2^w, where w is related to $w(\mathcal{A})$). Note that we can always filter out any undesirable interference of $\mathbf{val}(e, \nu, w)_K$ with the parallel process P, since for any labeled-transition reduct Q of $\mathbf{val}(e, \nu, w)_K$, Q is not an environment since it does not have the right number of rows for each depth. Likewise, for any namings ν, ν', ν'', we have $\mathbf{val}(e, \nu, w+1)_K \equiv \mathbf{val}(e, \nu', w)_K \mid \mathbf{val}(e, \nu'', w)_K$ if and only if $\nu = \nu' = \nu''$. Using already defined properties, we set

$$
\begin{array}{rcl}
\mathsf{Val}(e, w)_K & \stackrel{\text{def}}{=} & \prod_{i=1\ldots|e|} \big(\mathsf{Row}(K+i)^{2^w} \ \wedge\ \mathsf{Equals}(K+i)\big) \\
\mathsf{ProcVal}(e, w)_K & \stackrel{\text{def}}{=} & \mathcal{M}_K \mid \mathsf{Val}(e, w)_K
\end{array}
$$

Lemma D.2.4 *For any process P, environment e and naturals $K, w \geq 1$*

$$
\begin{array}{llll}
P \models & \mathsf{Val}(e, w)_K & \textit{iff} & \exists\nu.\quad P \ \equiv\ \mathbf{val}(e, \nu, w)_K \\
P \models & \mathsf{ProcVal}(e, w)_K & \textit{iff} & \exists Q \in M_K, \exists\nu.\quad P \ \equiv\ Q \mid \mathbf{val}(e, \nu, w)_K
\end{array}
$$

The formula $\mathsf{ProcVal}(e, w)_K$ specifies a pair process-valuation, where the process belongs to M_K. Now we introduce formulas to match specific entries of the

$$
\begin{array}{lcl}
[\![\mathcal{A} \wedge \mathcal{B}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge [\![\mathcal{A}]\!]_{(e,w)} \ \wedge \ [\![\mathcal{B}]\!]_{(e,w)} \\
[\![\neg\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \neg[\![\mathcal{A}]\!]_{(e,w)} \\
[\![0]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \mathsf{Val}(e,w)_K \\
[\![\mathcal{A} \mid \mathcal{B}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \left([\![\mathcal{A}]\!]_{(e,w-1)} \ \middle| \ [\![\mathcal{B}]\!]_{(e,w-1)}\right) \\
[\![\mathcal{A} \triangleright \mathcal{B}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \\
 & & \quad \big([\![\mathcal{A}]\!]_{(e,w)} \ \triangleright \ \big(\mathsf{ProcVal}(e,w+1)_K \rightarrow [\![\mathcal{B}]\!]_{(e,w+1)}\big)\big) \\
[\![\Diamond\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \Diamond[\![\mathcal{A}]\!]_{(e,w)} \\
[\![\exists x.\, \mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \big(\mathsf{EnvX}(x,e',w)_K \ \blacktriangleright \ [\![\mathcal{A}]\!]_{(e',w)}\big) \\
 & & \quad \text{where } e' = e\{^x/ \mid e \mid +1\} \\
[\![\langle x \rangle\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \\
 & & \quad \mathsf{Test}(e)_K \ \blacktriangleright \ \big((\mathsf{TestMatchesX}(x,e,w)_K \mid \top) \wedge \\
 & & \qquad\qquad \Diamond(\mathsf{UsedTest}(e)_K \mid [\![\mathcal{A}]\!]_{(e,w)})\big) \\
[\![\langle \overline{x} \rangle\mathcal{A}]\!]_{(e,w)} & \stackrel{\text{def}}{=} & \mathsf{ProcVal}(e,w)_K \ \wedge \Diamond\big(\mathsf{UsedXRow}(x,e)_K \ \mid \ (\mathsf{XRow}(x,e)_K \ \blacktriangleright \ [\![\mathcal{A}]\!]_{(e,w)})\big)
\end{array}
$$

Figure D.3: Encoding of $\mathcal{L}_{mod}^{(CCS)}$ into $\mathcal{L}_{spat}^{\Diamond}$.

(encoding of the) valuation: selection of the action α associated to the variable x is achieved by filtering the set of row processes of depth $e(x)$.

$$
\begin{array}{rcl}
\mathsf{XRow}(x,e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(K + e(x)) \\
\mathsf{UsedXRow}(x,e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(K + e(x) - 1) \\
\mathsf{EnvX}(x,e,w)_K & \stackrel{\text{def}}{=} & \mathsf{Equals}(K + \mid e \mid) \ \wedge \ (\mathsf{XRow}(x,e)_K)^{2^w}
\end{array}
$$

$\mathsf{XRow}(x,e)_K$ allows us to select one of the rows that represents the environment entry of the variable x. $\mathsf{UsedXRow}(x,e)_K$ checks that such a row has lost an action prefix (after a reduction step takes place). $\mathsf{EnvX}(x,e,w)_K$ matches all the rows that encode the environment entry for the variable x. To encode the modality $\langle x \rangle\mathcal{A}$ we need to check for the presence of the complementary of the action $v(x)$. To this end, we specify a row longer than any other (with $\mathsf{Test}(e)$), and then check (using $\Diamond$) that it may react with some row of depth $e(x)$ (with $\mathsf{UsedTest}(e)$). Let then:

$$
\begin{array}{rcl}
\mathsf{Test}(e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(\mid e \mid + K + 2) \\
\mathsf{UsedTest}(e)_K & \stackrel{\text{def}}{=} & \mathsf{Row}(\mid e \mid + K + 1) \\
\mathsf{TestMatchesX}(x,e,w)_K & \stackrel{\text{def}}{=} & (\mathsf{Test}(e)_K \mid \mathsf{EnvX}(x,e,w)_K) \ \wedge \ \Diamond\top
\end{array}
$$

We are now ready to present our encoding of formulas of $\mathcal{L}_{mod}^{(CCS)}$ into formulas of $\mathcal{L}_{spat}^{\Diamond}$.

Definition D.2.5 *Let $\mathcal{A} \in \mathcal{L}_{mod}^{(CCS)}$ be a formula, e an environment mapping the free variables of $\mathcal{A}$, and w, K be integers such that $w > w(\mathcal{A})$, and $K > 0$. Then, the formula $[\![\mathcal{A}]\!]_{(e,w)} \in \mathcal{L}_{spat}^{\Diamond}$ is inductively defined in Fig. D.3.*

Theorem D.2.1 follows from Lemmas D.2.2, D.2.3, D.2.4, and the following general result:

Lemma D.2.6 (Correctness of the encoding) *For all processes P, all formulas $\mathcal{A} \in \mathcal{L}^{(CCS)}_{mod}$, all environments e declaring the free variables of $\mathcal{A}$, all integers $w > w(A)$, and all $K > 0$ we have:*

$$P, \emptyset \models_{M_\infty} [\![A]\!]_{(e,w)} \quad \textit{if and only if} \quad \exists Q \in M_K, \exists \nu. \begin{cases} P \equiv Q \mid \mathbf{val}(e,\nu,w)_K \\ Q, \nu \circ e \models_{M_K} \mathcal{A} \end{cases}$$

Proof: By induction on $\mathcal{A}$.

- the cases of $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2, \neg\mathcal{A}_1, 0$ are straightforward.

- $\mathcal{A} = \mathcal{A}_a \mid \mathcal{A}_b$. Assume first $P, \emptyset \models_{M_\infty} [\![A]\!](e,w)$. By Lemma D.2.4, there is $P_1 \in M_K$ and ν such that $P \equiv P_1 \mid \mathbf{val}(e,\nu,w)_K$. Moreover, there is a splitting $P_1 \mid \mathbf{val}(e,\nu,w)_K \equiv P_a \mid P_b$ with $P_\epsilon, \emptyset \models_{M_\infty} \mathcal{A}_\epsilon$. By induction, the P_ϵ contain each a $\mathbf{val}(e,\nu,w-1)_K$. Due to the depth of the rows, P_1 do not contribute to that, so $P_\epsilon \equiv P_{1,\epsilon} \mid \mathbf{val}(e,\nu,w-1)_K$ with $P_1 \equiv P_{1,a} \mid P_{1,b}$. By induction, $P_{1,\epsilon}, \nu \circ e \models_{M_K} \mathcal{A}_\epsilon$, hence the result. Conversely, if $P \equiv P_1 \mid \mathbf{val}(e,\nu,w)_K$ with $P_1, \nu \circ e \models_{M_K} \mathcal{A}$, there is $P_{1,a}, P_{1,b}$ such that $P \equiv P_{1,a} \mid P_{1,b}$ and $P_{1,\epsilon}, \nu \circ e \models_{M_K} \mathcal{A}_\epsilon$, hence by induction $P_{1,\epsilon} \models_{M_\infty} [\![\mathcal{A}_\epsilon]\!](e,w)$ and $P \equiv \left(P_{1,a} \mid \mathbf{val}(e,w-1,)_K\right) \mid \left(P_{1,b} \mid \mathbf{val}(e,w-1,)_K\right) \models_{M_\infty} [\![\mathcal{A}]\!](e,w)$

- $\mathcal{A} = \mathcal{A}_1 \rhd \mathcal{A}_2$. Assume first $P, \emptyset \models_{M_{infty}} [\![\mathcal{A}]\!](e,w)$. By Lemma D.2.4, there is $P_1 \in M_K$ and ν such that $P \equiv P_1 \mid \mathbf{val}(e,\nu,w)_K$. To prove that $P_1, \nu \circ e \models_{M_K} \mathcal{A}_1 \rhd \mathcal{A}_2$, we pick some $Q \in M_K$ such that $Q, \nu \circ e \models_{M_K} \mathcal{A}_1$. Then by induction $Q \mid \mathbf{val}(e,\nu,w)_K \models_{M_\infty} [\![\mathcal{A}_1]\!](e,w)$, and $P \mid Q \mid \mathbf{val}(e,\nu,w)_K \models_{M_{infty}} \mathsf{ProcVal}(e,w{+}1)_K$, so $P \mid Q \mid \mathbf{val}(e,\nu,w)_K \models_{M_\infty} [\![\mathcal{A}_2]\!](e,w)$. By induction, $P \mid Q \mid \mathbf{val}(e,\nu,w)_K \equiv R_1 \mid \mathbf{val}(e,\nu',w)_K$ with $R_1, \nu' \circ e \models_{M_K} \mathcal{A}_2$ Due to the depth of the rows in $\mathbf{val}(e,\nu',w)_K$, one has necessarily $\nu = \nu'$ and $R_1 \equiv P_1 \mid Q$, hence the result. Assume now that $P \equiv P_1, \nu \circ e \models_{M_K} \mathcal{A}_1 \rhd \mathcal{A}_2$. To prove that $P, \emptyset \models_{M_\infty} [\![\mathcal{A}_1 \rhd \mathcal{A}_2]\!](e,w)$, we take $Q \in M_\infty$ such that $Q \models_{M_\infty} [\![\mathcal{A}_1]\!](e,w)$. By induction, there is ν' such that $Q \equiv Q_1 \mid \mathbf{val}(e,\nu',w)_K$ and $Q_1, \nu' \circ e \models_{M_K} \mathcal{A}_1$. If $\nu \neq \nu'$, then $\mathbf{val}(e,\nu,w)_K \mid \mathbf{val}(e,\nu',w)_K \not\models_{M_\infty} \mathsf{Val}(e,w{+}1)_K$, so $P \mid Q \models_{M_\infty} \mathsf{Val}(e,w{+}1)_K \rightarrow [\![\mathcal{A}_2]\!](e,w+1)$. Otherwise, $\nu = \nu'$ and by hypothesis $P_1 \mid Q_1, \nu \circ e \models_{M_K} \mathcal{A}_2$, so by induction $P \mid Q \models_{M_\infty} [\![\mathcal{A}_2]\!](e,w+1)$.

- $\mathcal{A} = \diamond\mathcal{A}'$. Assume first that $P \models_{M_{infty}} [\![\diamond\mathcal{A}']\!]$. Then $P \equiv P_1 \mid \mathbf{val}(e,\nu,w)_K$, and there is R such that $P \longrightarrow R \models_{M_\infty} [\![\mathcal{A}']\!](e,w)$. By induction, $R \equiv R_1 \mid \mathbf{val}(e,\nu',w)_K$ for some ν' and $R_1, \nu' \circ e \models_{M_K} \mathcal{A}'$. If $\mathbf{val}(e,\nu,w)_K$ takes part to this reduction, it decreases the size of one row or two rows of different depth. So the number of copies of the deeper one is not 2^k any more, and this process is not congruent to $\mathbf{val}(e,\nu',w)_K$. So $P_1 \longrightarrow R_1$ and the result. Assume now $P_1, \nu \circ e \models_{M_K} \diamond\mathcal{A}'$ and let R_1 be such that $R_1, , \nu \circ e \models_{M_K} \diamond\mathcal{A}'$ and $P_1 \longrightarrow R_1$. Then $P_1 \mid \mathbf{val}(e,\nu,w)_K \longrightarrow R_1 \mid \mathbf{val}(e,\nu,w)_K$, so $P \models_{M_\infty} [\![\diamond\mathcal{A}']\!](e,w)$.

- $\mathcal{A} = \exists x.\mathcal{A}'$. Assume first that $P \models_{M_\infty} [\![\mathcal{A}]\!]$. Then there is an action α such that $\mathbf{row}(K+ \mid e \mid +1, \alpha) \mid P \equiv \mathbf{val}(e',\nu,w)_K \mid P_1$ with $P_1, \nu \circ e' \models_{M_K} \mathcal{A}'$ with $e' = e, x \mapsto \alpha$ by row depth. So $P \equiv \mathbf{val}(e,\nu,w)_K \mid P_1$ and the result. The converse is as straightforward.

- $\mathcal{A} = \langle x \rangle \mathcal{A}'$. Assume first that $P \models_{M_\infty} [\![\mathcal{A}]\!]$. Then there is an action α such that $\mathbf{row}(K+ \mid e \mid +2, \alpha) \mid \mathbf{row}(K + e(x), \nu \circ e(x)) \longrightarrow$. So $\alpha =$

$\overline{\nu \circ e(x)}$. Moreover, there is P' such that $P_1 \mid \mathbf{val}(e, \nu, w)_K \mid \mathbf{row}(K+ \mid e \mid +2, \alpha) \longrightarrow P'$ and $P', \nu \circ e \models M_\infty \mathsf{UsedTest}(e) \mid [\![\mathcal{A}']\!](e, w)$. So P' has a row of depth $K+ \mid e \mid +1$, which is only possible if the reduction involved $\mathbf{row}(K+ \mid e \mid +2, \alpha)$. It cannot involve $\mathbf{val}(e, \nu, w)_K$ since P' contains it unchanged, so it necessarily involve P_1, that is $P' = \mathbf{row}(K+ \mid e \mid +1, \alpha) \mid \mathbf{val}(e, \nu, w)_K \mid P'_1$ with $P_1 \xrightarrow{\overline{\alpha}} P'_1$, hence the result. Assume now that there is P'_1 such that $P_1 \xrightarrow{\nu \circ e(x)} P'_1$; then adding the process $\mathbf{row}(K+ \mid e \mid +2, \overline{\nu \circ e(x)})$ and performing the reduction we just described, we check that $P_1 \mid \mathbf{val}(\nu, e, w)_K, \emptyset \models M_\infty \mathcal{A}$.

- $\mathcal{A} = \langle \overline{x} \rangle \mathcal{A}'$: Assume first that $P \models M_\infty [\![\mathcal{A}]\!]$. Then there is P', α, β such that $P \longrightarrow P' \mid \mathbf{row}(\beta, n-1)$ and $P' \mid \mathbf{row}(\alpha, n) \models M_\infty [\![\mathcal{A}']\!](e, w)$, with $n = e(x)$. Since $P' \mid \mathbf{row}(\alpha, n)$ holds an environment, it must be that a row of size n was absent in P', so that it necessarily contributed to the reduction $P \longrightarrow P' \mid \mathbf{row}(\beta, n-1)$. Hence in the reduction the number of rows of size n decrease of one, the number of rows of size $n-1$ increases of 1, and other rows stay to 2^w copies. To get an environment, the rows of the same size must all have the same name, and we have at least two copies at each size, so necessarily $\alpha = \nu(n)$ and $\mathbf{row}(\beta, n-1)$ is the row that was created by the reduction, that is $\beta = \alpha = \nu(n)$. Since the interaction did not involve any other row from then environment, it actually involved P_1. So there is P'_1 such that $P_1 \xrightarrow{\overline{\nu(n)}} P'_1$ and $P' \equiv P'_1 \mid \mathsf{env}'$, where $textsfenv'$ is the environment $\mathbf{val}(e, \nu, w)_K$ from which a row of size n has been picked out. Then $P' \mid \mathbf{row}(\alpha, n) \equiv P'_1 \mid \mathbf{val}(e, \nu, w)_K$ so by induction $P'_1, \nu \circ e \models M_K \mathcal{A}'$, that is $P_1, \nu \circ e \models M_K \langle \overline{x} \rangle \mathcal{A}'$. Assume now that $P_1, \nu \circ e \models M_K \langle \overline{x} \rangle \mathcal{A}'$. Then there is P'_1 such that $P_1 \xrightarrow{\overline{\nu(e(x))}} P'_1$ and $P'_1, \nu \circ e \models M_K \mathcal{A}'$. Then by induction $P'_1 \mid \mathbf{val}(e, \nu, w)_K \models M_\infty [\![\mathcal{A}]\!]'$, that is ${}_1 \mid \mathbf{val}(e, \nu, w)_K \longrightarrow P'_1 \mid \mathsf{env}' \mid \mathbf{row}(\alpha, n-1)$ where $e(x) = n$, $\alpha = \nu(n)$, and env' is $\mathbf{val}(e, \nu, w)_K$ from which one row of size n has been removed. So $P_1 \mid \mathbf{val}(e, \nu, w)_K \models M_\infty [\![\mathcal{A}]\!]$.

$\square$

We can thus present the proof of Theorem D.2.1. **Proof:** Let $\mathcal{A}$ be a formula of $\mathcal{L}^{(CCS)}_{mod}$. Set $[\![\mathcal{A}]\!] = [\![\mathcal{A}]\!]_{(\emptyset, w)}$ for some w greater than the maximal nesting of $\mid$ connectives in $\mathcal{A}$. Then $\pi_K(P) \equiv \pi_K(P) \mid \mathbf{val}(\emptyset, \emptyset, w)_K$, so by Lemma D.2.6, $\pi_K(P), \emptyset \models M_\infty [\![\mathcal{A}]\!]$ if and only if $\pi_K(P), \emptyset \models M_K \mathcal{A}$, which is equivalent to $P, \emptyset \models M_\infty \mathcal{A}$ by Proposition D.1.6. $\square$

D.3 Separability of $\mathcal{L}^{\diamond}_{spat}$

As a first application of the main Theorem D.2.1, we define characteristic formulas and characterize the separation power of the logic $\mathcal{L}^{\diamond}_{spat}$ (and thus of $\mathcal{L}^{(CCS)}_{mod}$). We conclude that $\mathcal{L}^{\diamond}_{spat}$ is able to describe processes quite precisely, just abstracting away from the identity of the particular names used by processes. We start by introducing a characteristic formula $C(P)$ for any process P. For any complementary pair of actions $\{\alpha, \overline{\alpha}\}$ occurring in P, we reserve a specific variable x_a, collected in the set $\{x_{\alpha_1}, \ldots, x_{\alpha_n}\}$.

$$\chi(\mathbf{0}) \stackrel{\text{def}}{=} 0 \qquad \chi(\alpha.\, P) \stackrel{\text{def}}{=} 1 \wedge \langle x_\alpha \rangle \chi(P)$$
$$\chi(\overline{\alpha}.\, P) \stackrel{\text{def}}{=} 1 \wedge \langle \overline{x_\alpha} \rangle \chi(P) \qquad \chi(P \mid Q) \stackrel{\text{def}}{=} \chi(P) \mid \chi(Q)$$

$$C(P) \stackrel{\text{def}}{=} [\![\exists x_{\alpha_1} \ldots \exists x_{\alpha_n}.\, (\bigwedge_{i \neq j} x_{\alpha_i} \neq x_{\alpha_j} \ \wedge \ x_{\alpha_i} \neq \overline{x_{\alpha_j}}) \wedge \chi(P)]\!]$$

where $K = \mathsf{ds}(P)$. Recall that abbreviations $(x = y)$ and $(x \neq y)$ are defined in Fig.D.2, and notice that $C(P) \in \mathcal{L}^{\diamond}_{spat}$, while $\chi(P) \in \mathcal{L}^{(CCS)}_{mod}$.

Lemma D.3.1 *Let $P \in M_K$, let v be the valuation such that $v(x_{\alpha_i}) = \beta_i$, for pairwise distinct actions $\beta_1, \ldots, \beta_n$, and let σ be the action permutation that sends α_i into β_i. Then we have that $Q, v \models_{M_K} \chi(P)$ if and only $Q \equiv \sigma(P)$.*

Proof: Induction Hypothesis on P. We detail the case of $P = \alpha_j.\, P'$. If $Q, v \models_{M_K} \chi(P)$ then $Q, v \models_{M_K} 1$ and $Q, v \models_{M_K} \langle x_{\alpha_j} \rangle \chi(P')$. This means that $Q \equiv \beta_j.\, Q'$ and $Q', v \models_{M_K} \chi(P')$, where $b_j = v(x_{\alpha_j})$. By inductive hypothesis, $Q' \equiv \sigma(P')$. Since $\sigma(\alpha_j) = \beta_j$ we conclude $Q \equiv \sigma(P)$. Conversely, assume $Q \equiv \sigma(P)$. This means that $\beta_j = \sigma(\alpha_j)$ and $Q = \beta_j.\, Q'$ where $Q' \equiv \sigma(P')$. By inductive hypothesis, $Q', v \models_{M_K} \chi(P')$. Then, we have $Q \xrightarrow{\beta_j} Q'$. Since $Q, v \models 1$, and $v(x_{\alpha_j}) = \beta_j$ we conclude $Q, v \models_{M_K} \langle x_\alpha \rangle \chi(P')$ and then $Q, v \models_{M_K} \chi(P)$. □

Lemma D.3.2 *For all processes Q and P, $Q \models C(P)$ if and only if $Q \equiv_s P$.*

Proof: Let $K = \mathsf{ds}(P)$ and assume $Q, \emptyset \models C(P)$. Then $Q \in M_K$ and thus $\pi_K(Q) = Q$. By Theorem D.2.1 we have $Q, \emptyset \models_{M_K} \exists x_{\alpha_1}.\ldots\exists x_{\alpha_n}.\, \chi(P)$, so there are pairwise distinct actions β_i such that $v(x_{\alpha_i}) = \beta_i$ and $Q, v \models_{M_K} \chi(P)$. By Lemma D.3.1, we conclude that $Q \equiv \sigma(P)$, where $\sigma(\alpha_i) = \beta_i$. Conversely, let $Q \equiv \sigma(P)$ for some action permutation σ; thus if $P \in M_K$ then also $Q \in M_K$. Let $v(\alpha_i) = \beta_i$ whenever $\sigma(\alpha_i) = \beta_i$. By Lemma D.3.1, we conclude $Q, v \models_{M_K} \chi(A)$. Since the actions β_i are pairwise distinct, by Theorem D.2.1 we conclude $Q \models C(P)$. □

We then conclude:

Theorem D.3.3 *The following statements are equivalent:*

(1) $P {=_L}^{mod} Q$ (2) $P {=_L}^{spat} Q$ (3) $Q, \emptyset \models C(P)$ (4) $P \ \equiv_s \ Q$

Proof: (1) $\Rightarrow$ (2) because $\mathcal{L}^{\diamond}_{spat} \subset \mathcal{L}^{(CCS)}_{mod}$, (2) $\Rightarrow$ (3) since $C(P) \in \mathcal{L}^{\diamond}_{spat}$ and $P \models C(P)$, (3) $\Rightarrow$ (4) by Lemma D.3.2, and (4) $\Rightarrow$ (1) by Proposition D.1.5. □

D.4 Expressiveness of Composition Adjunct

It is known that in static spatial logics, that is spatial logics without quantifiers and dynamic operators, the adjunct connective is not independent of the remaining connectives, and can in fact be eliminated, in the sense that for any formula of such a logic we can find a logically equivalent adjunct-free formula [Loz03]. It is not hard to see that adjunct cannot be dispensed with in $\mathcal{L}^{\diamond}_{spat}$, because without adjunct one is not allowed to distinguish threads of different lenght: if we pick $\mathcal{A} \in \mathcal{L}^{\diamond}_{spat} - \{\triangleright\}$, we can verify by an easy induction on $\mathcal{A}$ that $\alpha.\,0 \models \mathcal{A}$ if and only if $\alpha.\,\beta.\,0 \models \mathcal{A}$, for all $\alpha, \beta \in \mathsf{Act}$.

In this section, we prove that the adjunct elimination property does not hold for the spatial logic $\mathcal{L}^{(CCS)}_{mod}$. For this, we adapt a scheme suggested by Yang: on the one hand, we define in $\mathcal{L}^{(CCS)}_{mod}$ a formula that says of a process that its number of toplevel parallel components is even, on the other hand, we show that parity cannot be characterized by adjunct-free formulas. We start by defining a few formulas (where $\Box\mathcal{A} \stackrel{\text{def}}{=} \neg\Diamond\neg\mathcal{A}$):

$$\begin{array}{lcl} \mathsf{Top}(x) & \stackrel{\text{def}}{=} & \langle x\rangle 0 \\ \mathsf{Fam} & \stackrel{\text{def}}{=} & \Box\bot \wedge \big(1 \Rightarrow \exists x.\,\mathsf{Top}(x)\big)^{\forall} \wedge \forall x.\,\forall y.\,(\mathsf{Top}(x) \mid \mathsf{Top}(y) \mid \top) \Rightarrow x \neq y) \end{array}$$

We can verify that $P \models \mathsf{Fam}$ if and only if $P \equiv \alpha_1.\,\mathbf{0} \mid \ldots \mid \alpha_k.\,\mathbf{0}$ for some pairwise distinct k actions $\alpha_1, \ldots, \alpha_k$ such that $P \not\rightarrow$. We call a process of such a form a *family*. The width of such a family P is defined to be the number $w(P) = k$ of parallel threads in P. Now, we can define a formula $\mathsf{Even2}$ that is satisfied by processes that contain exactly an even number of distinct actions at the second level.

$$\begin{array}{lcl} \mathsf{Pair} & \stackrel{\text{def}}{=} & 1 \;\wedge\; \exists xyz.\,\langle x\rangle(\mathsf{Top}(y) \mid \mathsf{Top}(z)) \wedge (y \neq z) \\ \mathsf{Below}(x) & \stackrel{\text{def}}{=} & 1 \wedge \exists z.\,\langle z\rangle\langle x\rangle\top \\ \mathsf{Even2} & \stackrel{\text{def}}{=} & (1 \Rightarrow \mathsf{Pair})^{\forall} \wedge \forall x.\,\forall y.\,(\mathsf{Below}(x) \mid \mathsf{Below}(y) \mid \top) \Rightarrow x \neq y) \end{array}$$

Hence $P \models \mathsf{Even2}$ if and only if $P \equiv \alpha_1.\,(\beta_{1,1}.\,\mathbf{0} \mid \beta_{1,2}.\,\mathbf{0}) \mid \cdots \mid \alpha_k.\,(\beta_{k,1}.\,\mathbf{0} \mid \beta_{k,2}.\,\mathbf{0})$ for some k actions $\alpha_1, \ldots, \alpha_k$, and some pairwise distinct $2k$ actions $\beta_{1,i}, \ldots, \beta_{k,i}$ for $i = 1, 2$. Now, if we compose a process P satisfying Fam in parallel with a process Q satisfying $\mathsf{Even2}$, we can check (in $P \mid Q$) that the actions that occur in the toplevel of P are exactly the same that appear in the second level of Q using the formula Same:

$$\mathsf{Same} \stackrel{\text{def}}{=} \forall x.\,(\mathsf{Top}(x)^{\exists} \Leftrightarrow\; \mathsf{Below}(x)^{\exists})$$

Hence we have the following result

Lemma D.4.1 *There is a closed formula* $\mathsf{Even} \in \mathcal{L}^{(CCS)}_{mod}$ *such that for any process* P*, we have that* $P \models \mathsf{Even}$ *if and only if* P *is a family and* $w(P)$ *is even.*

Proof: Let $\mathsf{Even} \stackrel{\text{def}}{=} \mathsf{Fam} \wedge (\mathsf{Even2} \;\blacktriangleright\; \mathsf{Same})$. $\Box$

A key observation is that the formula Even contains an essential use of the composition adjunct operator. In fact, although the properties denoted by the formulas $\mathsf{Even2}$ and Fam can be expressed by appropriate adjunct-free formulas of $\mathcal{L}^{\diamond}_{spat}$, the same situation does not hold for the parity property expressed by Even. In the remainder of this section, we prove that there is no formula of $\mathcal{L}^{(CCS)}_{mod} - \{\triangleright\}$ able to express the same property. The argument consists in showing that any family P considered in $\mathcal{L}^{(CCS)}_{mod} - \{\triangleright\}$ admits a saturation level from which it is always possible to add an extra parallel component to it while preserving satisfaction. We first define sn (the *sticks number* of the formula $\mathcal{A}$)

to be the natural number defined by induction on $\mathcal{A}$ as follows:

$$\begin{array}{lclclcl} \mathsf{sn} & \stackrel{\text{def}}{=} & \mathsf{sn} & \quad & \mathsf{sn} & \stackrel{\text{def}}{=} & \max(\mathsf{sn}, \mathsf{sn}) \\ \mathsf{sn} & \stackrel{\text{def}}{=} & 1 & & \mathsf{sn} & \stackrel{\text{def}}{=} & \mathsf{sn} + \mathsf{sn} \\ \mathsf{sn} & \stackrel{\text{def}}{=} & 0 & & \mathsf{sn} & \stackrel{\text{def}}{=} & \mathsf{sn} \\ \mathsf{sn} & \stackrel{\text{def}}{=} & \mathsf{sn} + 1 & & \mathsf{sn} & \stackrel{\text{def}}{=} & \mathsf{sn} \end{array}$$

Given a family P and a valuation v, we write $P\backslash v$ for the subfamily of P of the actions α that do not appear in the codomain of the valuation v. More precisely, we define $P\backslash v \stackrel{\text{def}}{=} \prod\{\alpha.\mathbf{0} \;:\; P \equiv \alpha.\mathbf{0} \mid Q \textit{ and } \alpha, \overline{\alpha} \notin \mathsf{codom}(v)\}$. We then have:

Lemma D.4.2 *Let P be a family, let v be a valuation $v : \mathsf{Var} \rightharpoonup_{fin} \mathsf{Act}$, and let $\alpha \in \mathsf{Act}$ be an action such that $\alpha, \overline{\alpha} \notin \mathsf{codom}(v)$ and $(P \mid \alpha.\mathbf{0})$ is a family. Then, for any $\triangleright$-free formula $\mathcal{A} \in \mathcal{L}^{(CCS)}_{mod}$ such that $w(P\backslash v) \geq \mathsf{sn}$ we have*

$$P,\ v \models \mathcal{A} \qquad \textit{if and only if} \qquad P \mid \alpha.\mathbf{0},\ v \models \mathcal{A}.$$

Proof: By induction on $\mathcal{A}$;

- $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, $\mathcal{A} = \neg\mathcal{A}_1$: straightforward.
- $\mathcal{A} = 0$ then $w(P) \geq 1$ so both P and $P \mid \alpha$ do not satisfy $\mathcal{A}$
- $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$. We assume first that $P, v \models \mathcal{A}$. Then there is P_1, P_2 such that $P \equiv P_1 \mid P_2$ and $P_i \models \mathcal{A}_i$. Then

  $$w(P\backslash v) = w(P_1\backslash v) + w(P_2\backslash v) \geq \mathsf{sn} = \mathsf{sn} + \mathsf{sn}$$

  so there is some $e \in \{1,2\}$ such that $w(P_e\backslash v) \geq \mathsf{sn}$. By induction then $P_e \mid \alpha \models \mathcal{A}_e$, so that $P \mid \alpha \models \mathcal{A}$. We assume now that $P \mid \alpha \models \mathcal{A}$. Then there is Q_1, Q_2 such that $P \mid \alpha = Q_1 \mid Q_2$ and $Q_i \models \mathcal{A}_i$. Since $\alpha \notin \mathsf{codom}(v)$, $w(P \mid \alpha\backslash v) = w(P\backslash v) + 1$, so

  $$w(P \mid \alpha\backslash v) = w(Q_1\backslash v) + w(Q_2\backslash v) > \mathsf{sn} = \mathsf{sn} + \mathsf{sn}$$

  and there is some $e \in \{1,2\}$ such that $w(Q_e\backslash v) > \mathsf{sn}$. We pick some $\alpha' \in Q_e$ with $\alpha' \notin \mathsf{codom}(v)$, which is possible since $w(Q_e\backslash v) \geq 1$. We note P_e the family such that $Q_e(\alpha \leftrightarrow \alpha') \equiv P_e \mid \alpha$. Then $w(P_e\backslash v) = w(P_e \mid \alpha\backslash v) - 1 = w(Q_e\backslash(v(\alpha \leftrightarrow \alpha)')) - 1 = w(Q_e\backslash v) - 1$, hence $w(P_e\backslash v) \geq \mathsf{sn}$. By equivariance, we get from $Q_e, v \models \mathcal{A}_e$ that $P_e \mid \alpha, v \models \mathcal{A}_e$, and by induction $P_e, v \models \mathcal{A}_e$. Then we write $P \mid \alpha = Q_1(\alpha \leftrightarrow \alpha') \mid Q_2(\alpha \leftrightarrow \alpha') = P_1 \mid \alpha \mid P_2$, which gives that $P, v \models \mathcal{A}$.
- $\mathcal{A} = \Diamond\mathcal{A}_1$. Then both P and $P \mid \alpha$ do not satisfy $\mathcal{A}$ as deadlock processes.
- $\mathcal{A} = \exists x.\mathcal{A}_1$. We assume first that $P, v \models \mathcal{A}$. Then there is β such that for $v' = v, x \mapsto \beta$, $P, v' \models \mathcal{A}_1$. We may assume that $\beta \notin \{\alpha, \overline{\alpha}\}$, otherwise we pick some fresh action β', and consider instead $v' = x \mapsto \beta'$ for which we also have $P, v' \models \mathcal{A}_1$ by equivariance. We have $w(P \mid \alpha\backslash v') = w(P\backslash v') + 1 \geq w(P\backslash v)$, so $w(P \mid \alpha\backslash v') \geq \mathsf{sn}$, and by induction $P \mid \alpha, v' \models \mathcal{A}_1$, that is $P \mid \alpha, v \models \mathcal{A}$. We assume now that $P \mid \alpha, v \models \mathcal{A}$. Then there is β such that for $v' = v, x \mapsto \beta$, $P \mid \alpha, v' \models \mathcal{A}_1$. If $\beta \in \{\alpha, \overline{\alpha}\}$, we may pick some other β' occurring in P, and by equivariance $P \mid \alpha, v'' \models \mathcal{A}_1$ for $v'' = v, x \mapsto \beta'$. So we may assume $\beta \notin \{\alpha, \overline{\alpha}\}$, and then $w(P\backslash v') \geq w(P\backslash v) - 1 \geq \mathsf{sn}$, so by induction $P, v' \models \mathcal{A}_1$, that is $P, v \models \mathcal{A}$.

- $\mathcal{A} = \langle x \rangle \mathcal{A}_1$. Assume first that $P, v \models \mathcal{A}$. Then there is P' such that $P \xrightarrow{v(x)} P'$ and $P', v \models \mathcal{A}_1$. $w(P' \backslash v) = w(P \backslash v) \geq \mathsf{sn}$, so by induction $P' \mid \alpha, v \models \mathcal{A}_1$, that is $P \mid \alpha, v \models \mathcal{A}$. Assume now that $P \mid \alpha, v \models \mathcal{A}$. Then there is P_1 such that $P \mid \alpha \xrightarrow{v(x)} P_1$ with $P_1, v \models \mathcal{A}_1$. Since $\alpha \notin \mathsf{codom}(v)$, $P_1 \equiv P' \mid \alpha$ with $P \xrightarrow{v(x)} P'$. $w(P' \backslash v) = w(P \backslash v) \geq \mathsf{sn}$, so by induction $P', v \models \mathcal{A}_1$, that is $P, v \models \mathcal{A}$.

- $\mathcal{A} = \langle \overline{x} \rangle . \mathcal{A}_1$ proceeds in the same way.

$\square$

Theorem D.4.3 *There is no closed formula $\mathcal{A} \in \mathcal{L}^{(CCS)}_{mod} - \{\triangleright\}$ that exactly characterizes the set of all families P with $w(P)$ even.*

Proof: By contradiction: if $\mathcal{A}$ was a such formula, then we may take a family P and an extended family $P \mid \alpha$ with $w(P) \geq \mathsf{sn}$. Then by previous lemma, $P, \emptyset \models \mathcal{A}$ if and only if $P \mid \alpha, \emptyset \models \mathcal{A}$, which is a contradiction. $\square$

We thus conclude that in the logic $\mathcal{L}^{(CCS)}_{mod}$ the composition adjunct operator is independent of the remaining operators, in particular there are properties expressible with the composition adjunct that cannot be expressed with action modalities and quantifiers.

D.5 Undecidability

In this section, we show that the validity-, satisfiability- and model-checking problems for the logic $\mathcal{L}^{\diamond}_{spat}$ (and hence for $\mathcal{L}^{(CCS)}_{mod}$) are all undecidable. These results are a consequence of our embedding of $\mathcal{L}^{(CCS)}_{mod}$ into $\mathcal{L}^{\diamond}_{spat}$ (Theorem D.2.1), and of the fact that first-order logic can then be easily encoded into $\mathcal{L}^{(CCS)}_{mod}$ (along the lines of [CTZ+04]). The language of first-order logic (FOL) and its semantics is defined as usual:

$$\mathcal{A}, \mathcal{B} \quad ::= \quad \mathcal{A} \wedge \mathcal{B} \;\mid\; \neg \mathcal{A} \;\mid\; \exists x .\, \mathcal{A} \;\mid\; p(x, y)$$

Formulas of FOL are built from a set $Vars$ of individual variables (x, y), and without loss of generality we consider a single binary predicate symbol p. A model for FOL is a pair (D, I) where D is a set of individuals (the domain of the model), and I is a binary relation $I \subseteq D \times D$. For our purposes it is enough to focus on finite models. Satisfaction of a FOL formula by a model is defined using a valuation v that assigns each individual variable an element of D as follows:

$$\begin{array}{llll}
(D,I) & \models_v & \mathcal{A} \wedge \mathcal{B} & \textit{if} \quad (D,I) \models_v \mathcal{A} \textit{ and } (D,I) \models_v \mathcal{B} \\
(D,I) & \models_v & \neg \mathcal{A} & \textit{if} \quad \textit{not } (D,I) \models_v \mathcal{A} \\
(D,I) & \models_v & \exists x .\, \mathcal{A} & \textit{if} \quad \exists\, d \in D .\, (D,I) \models_{v\{x/d\}} \mathcal{A} \\
(D,I) & \models_v & p(x,y) & \textit{if} \quad (v(x), v(y)) \in I
\end{array}$$

We now show how to encode any FOL satisfaction judgment $(D, I) \models_v \mathcal{A}$ into a $\mathcal{L}^{(CCS)}_{mod}$ satisfaction judgment $\mathcal{M}[\![(D,I)]\!], \mathcal{V}[\![v]\!] \models \mathcal{F}[\![A]\!]$, by means of appropriate translations $\mathcal{M}[\![-]\!]$, $\mathcal{V}[\![-]\!]$ and $\mathcal{F}[\![-]\!]$. We pick natural numbers K, E such that $K > E > 2$. To encode a model (D, I) into a process $\mathcal{M}[\![(D,I)]\!]$, we start by assigning each element $d \in D$ a distinct action $A(d) \in \mathsf{Act}$, and define $\mathcal{E}[\![d]\!] \stackrel{\text{def}}{=} \mathbf{row}(E, A(d))$. The domain $D = \{d_1, \ldots, d_n\}$ is represented by the process $\mathcal{D}[\![D]\!] \stackrel{\text{def}}{=} \mathcal{E}[\![d_1]\!] \mid \ldots \mid \mathcal{E}[\![d_k]\!]$. For the interpretation I, we represent each pair $(d, e) \in I$ by the process $\mathcal{T}[\![(d, e)]\!] \stackrel{\text{def}}{=} A(d).\,A(e).\,\mathbf{0}$. We then let $\mathcal{I}[\![I]\!] \stackrel{\text{def}}{=} \prod_{(d,e)\in I} \mathcal{T}[\![(d, e)]\!]$ and finally set $\mathcal{M}[\![(D, I)]\!] \stackrel{\text{def}}{=} \mathcal{D}[\![D]\!] \mid \mathcal{I}[\![I]\!]$. Notice that, by construction, we always have $\mathcal{M}[\![(D, I)]\!] \in M_K$. Processes encoding our FOL models can be characterized by a formula Model of $\mathcal{L}^{\diamond}_{spat}$ as follows:

$$
\begin{array}{lllllll}
\mathsf{D}(x) & \stackrel{\text{def}}{=} & \mathsf{Row}(E) \wedge \langle x\rangle\top & \mathsf{Diff} & \stackrel{\text{def}}{=} & \forall x.\forall y.\,(\langle x\rangle\top \mid \langle y\rangle\top) \Rightarrow x \neq y) \\
\mathsf{Domain} & \stackrel{\text{def}}{=} & \mathsf{Diff} \wedge \big(1 \Rightarrow \exists x.\,\mathsf{D}(x)\big)^{\forall} & \mathsf{Cmp} & \stackrel{\text{def}}{=} & \forall x.\forall y((\langle x\rangle\langle y\rangle 0)^{\exists} \Rightarrow (\mathsf{D}(x)^{\exists} \wedge \mathsf{D}(y)^{\exists})) \\
\mathsf{Interp} & \stackrel{\text{def}}{=} & \big(1 \Rightarrow \mathsf{Thread}(2)\big)^{\forall} & \mathsf{Model} & \stackrel{\text{def}}{=} & \mathcal{M}_K \wedge \mid (\mathsf{Domain} \mid \mathsf{Interp}) \wedge \mathsf{Cmp} \mid
\end{array}
$$

Lemma D.5.1 *$P \models \mathsf{Model}$ iff there is a finite FOL model (D, I) and $\mathcal{M}[\![(D, I)]\!] \equiv P$.*

Proof: (if) As already remarked, we have $P \equiv \mathcal{M}[\![(D, I)]\!] \in M_K$, hence $P \models \mathcal{M}_K$. We can also check that $P \models_{M_K} (\mathsf{Domain} \mid \mathsf{Interp}) \wedge \mathsf{Cmp})$, because $\mathcal{D}[\![D]\!] \models \mathsf{Domain}$ and $\mathcal{I}[\![I]\!] \models \mathsf{Interp}$. Since $P \in M_K$, by Theorem D.2.1(if) we conclude that $P \models \mathsf{Model}$. (only if) If $P \models \mathsf{Model}$ we conclude that $P \in M_K$ and $P \models \mid (\mathsf{Domain} \mid \mathsf{Interp}) \wedge \mathsf{Cmp} \mid$. By Theorem D.2.1(only if), we have $P \models_{M_K} (\mathsf{Domain} \mid \mathsf{Interp}) \wedge \mathsf{Cmp}$.

This means that $P \equiv P_D \mid P_I$ where $P_D \models \mathsf{Domain}$ and $P_I \models \mathsf{Interp}$. In turn, we conclude that $P_D \equiv \mathbf{row}(E, \alpha_1) \mid \cdots \mid \mathbf{row}(E, \alpha_k)$, where the actions α_i are pairwise distinct. We can then construct a finite FOL model (D, I) from P_D and P_I, by letting $D = \{\alpha_1, \ldots, \alpha_k\}$, and $I(p) = \{(d, e) : \exists R, \alpha, \beta.\, P_I \equiv \alpha.\,(A(d).\,\mathbf{0} \mid A(e).\,A(e).\,\mathbf{0} \mid \mathbf{row}(Code(p), \beta)) \mid R\}$, for all $p \in Preds$. Since $P \models \mathsf{Cmp}$ we indeed have $I(p) \subseteq D \times D$. $\square$

Then, formulas of FOL are encoded into formulas of $\mathcal{L}^{(CCS)}_{mod}$ as follows

$$
\begin{array}{llllll}
\mathcal{F}[\![\neg\mathcal{A}]\!] & \stackrel{\text{def}}{=} & \neg\mathcal{F}[\![\mathcal{A}]\!] & \mathcal{F}[\![\exists x.\,\mathcal{A}]\!] & \stackrel{\text{def}}{=} & \exists x.\,(\mathsf{D}(x)^{\exists} \wedge A) \\
\mathcal{F}[\![\mathcal{A} \wedge \mathcal{B}]\!] & \stackrel{\text{def}}{=} & \mathcal{F}[\![\mathcal{A}]\!] \wedge \mathcal{F}[\![\mathcal{B}]\!] & \mathcal{F}[\![p(x, y)]\!] & \stackrel{\text{def}}{=} & \big(1 \wedge \langle x\rangle\langle y\rangle 0\big)^{\exists}
\end{array}
$$

Finally, for valuations we set $\mathcal{V}[v](x) = A(v(x))$. We can prove

Lemma D.5.2 *Let $v = \{x_1 \mapsto d_1, \ldots, x_k \mapsto d_k\}$ be a valuation for $\mathcal{A}$. Then we have $(D, I) \models_v \mathcal{A}$ if and only if $\mathcal{M}[\![(D, I)]\!], \mathcal{V}[\![v]\!] \models_{M_K} \mathcal{F}[\![\mathcal{A}]\!]$.*

Proof: By an easy induction on the structure of formulas. We detail the case of $A = p(x, y)$. If $(D, I) \models p(x, y)$ then $(v(x), v(y)) \in I(p)$, and thus

$$\mathcal{I}[\![I]\!] \equiv \alpha.\,(\mathbf{row}(Code(p), \beta) \mid A(v(x)).\,\mathbf{0} \mid A(v(y)).\,A(v(y)).\,\mathbf{0}) \mid R$$

for some α, β and R. Hence we have

$$\mathcal{M}[\![(D, I)]\!], \mathcal{V}[\![v]\!] \models_{M_K} \big(\exists z.\,\langle z\rangle(\mathsf{Row}(Code(p)) \mid \langle x\rangle 0 \mid \langle y\rangle\langle y\rangle 0)\big)^{\forall}$$

Conversely, if $\mathcal{M}[\![(D, I)]\!], \mathcal{V}[\![v]\!] \models_{M_K} \mathcal{F}[\![p(x, y)]\!]$, we conclude that

$$\mathcal{M}[\![(D, I)]\!] \equiv \alpha.\,(\mathbf{row}(Code(p), \delta) \mid \beta.\,\mathbf{0} \mid \gamma.\,\gamma.\,\mathbf{0}) \mid R$$

for some R and $\alpha, \beta, \gamma, \delta$ such that $v(x) = \beta$ and $v(y) = \gamma$. We conclude that

$$\mathcal{I}[\![I]\!] \equiv \alpha.\,(\mathbf{row}(Code(p), \delta) \mid \beta.\,\mathbf{0} \mid \gamma.\,\gamma.\,\mathbf{0}) \mid R'$$

, and so there are $d, e \in D$ such that $A(d) = \beta$ and $A(e) = \gamma$ and $(d, e) \in I(p)$, by construction of $\mathcal{I}[\![I]\!]$. Hence $(D, I) \models_v p(x, y)$. $\square$

Proposition D.5.3 *Let $\mathcal{A}$ be a closed formula of FOL. Then the formula $\mathcal{A}$ is satisfiable if and only if the $\mathcal{L}^{\diamond}_{spat}$ formula $\mathsf{Model} \wedge \mid \mathcal{F}[\![\mathcal{A}]\!] \mid$ is satisfiable.*

Proof: Assume that the formula $\mathcal{A}$ is satisfiable. Then there is a FOL model (D, I) such that $(D, I) \models \mathcal{A}$. By Lemma D.5.2, we have that $\mathcal{M}[\![(D, I)]\!] \models_{M_S} \mathcal{F}[\![\mathcal{A}]\!]$. By Theorem D.2.1, we conclude $\mathcal{M}[\![(D, I)]\!] \models \mid \mathcal{F}[\![A]\!] \mid$, for $\mathcal{M}[\![(D, I)]\!] \in M_K$. Since $\mathcal{M}[\![(D, I)]\!] \models \mathsf{Model}$ by Lemma D.5.1, we conclude that $\mathsf{Model} \wedge \mid \mathcal{F}[\![\mathcal{A}]\!] \mid$ is satisfiable. Conversely, if $\mathsf{Model} \wedge \mid \mathcal{F}[\![\mathcal{A}]\!] \mid$ is satisfiable, then there is a process P such that $P \models \mathsf{Model}$ (and thus $P \in M_K$) and $P \models \mid \mathcal{F}[\![\mathcal{A}]\!] \mid$. By Theorem D.2.1, we have that $P, \emptyset \models_{M_K} \mathcal{F}[\![\mathcal{A}]\!]$, and by Lemma D.5.1 we conclude that there is a finite model (D, I) such that $P \equiv \mathcal{M}[\![(D, I)]\!]$. Hence $\mathcal{M}[\![(D, I)]\!] \models_{M_K} \mathcal{F}[\![\mathcal{A}]\!]$, so by Lemma D.5.2 we conclude $(D, I) \models \mathcal{A}$. $\square$

As a corollary of Proposition D.5.3, we conclude

Theorem D.5.4 *The problems of validity-checking, satisfiability-checking, and model-checking of $\mathcal{L}^{\diamond}_{spat}$ formulas are all undecidable.*

Proof: Follows from Proposition D.5.3 and Trakhtenbrot's Theorem [Tra50]. $\square$

D.6 Extension to the π-Calculus and Ambients

In this section, we briefly discuss how our results extend to richer models, namely the π-calculus and the ambient calculus. We may pick any of these calculi as models for the core logic $\mathcal{L}^{\diamond}_{spat}$, which is a fragment of both the ambient logic of [CG00] and the π-calculus logic of [CC01]. We discuss first the case of the ambient calculus without name restriction, and just with the open capability. In this case, we can show that $\mathcal{L}^{\diamond}_{spat}$ can also encode, for processes of bounded depth, its extension with the quantifier $\exists x. \mathcal{A}$, and modalities of the form $\langle \mathsf{open}\ x \rangle. \mathcal{A}$ and $x[\mathcal{A}]$. However, as we might expect, the symmetry between input and output (Theorem D.3.3(4)) does not carry over to ambients: for instance, the formula $1 \ \wedge \ \diamond\top$ may be satisfied by the ambient $n[P]$, but not by the guarded ambient $\mathsf{open}\ n. P$. For the π-calculus, we may consider the extension of $\mathcal{L}^{\diamond}_{spat}$ with the quantifier $\exists x. \mathcal{A}$ and the modalities $\langle x \rangle \mathcal{A}$ and $\langle \overline{x} \rangle \mathcal{A}$, able to observe just the subjects of π-calculus actions. In this case, we may also prove that this extension can be encoded in $\mathcal{L}^{\diamond}_{spat}$ for bounded depth processes, as we did for the other cases. From these results, we conclude

Theorem D.6.1 *The model-checking and validity problems for the π-calculus and the ambient calculus against $\mathcal{L}^{\diamond}_{spat}$ are both undecidable.*

We only sketch our proof, since it is obtained by adapting to the ambient calculus and to the π-calculus the constructions and reasoning shown in detail for the fragment of CCS considered in the paper.

It should be clear that the basic ingredients of our encoding of $\mathcal{L}^{(CCS)}_{mod}$ in $\mathcal{L}^{\diamond}_{spat}$ (Theorem D.2.1), in turn used to prove independence of adjunct (Theorem D.4.3), and then and undecidability (Theorem D.5.4), are the definitions for the formulas $\mathsf{Thread}(k)$, $\mathsf{Row}(k)$ and $\mathcal{M}_k$, characterizing respectively threads, rows, and the submodels M_k.

Thus we just detail how to provide counterparts to these formulas, by taking in consideration each of the models now under consideration. It turns out that for ambients this is quite easy, while for the case of the π-calculus, because of name restriction and passing, it is slightly more involved.

Mobile Ambients

For simplicity, we consider the fragment of the Ambient calculus defined by the following grammar:

$$P, Q \quad ::= \quad \mathsf{open}\, n.\, P \;\big|\; n[P] \;\big|\; P \mid Q \;\big|\; \mathbf{0}$$

where a, n, m, p ranges over the set of names Λ. We assume given the reduction relation $P \to P'$ as defined in [CG98]. We also define the depth $\mathsf{ds}(P)$ of a process P, and the set of models M_k as expected. We also consider the extension $\mathcal{L}^{(MA)}_{mod}$ of $\mathcal{L}^{\diamond}_{spat}$:

$$\begin{array}{lllllllllllll} \mathcal{A}, \mathcal{B} & ::= & \mathcal{A} \wedge \mathcal{B} & \mid & \mathcal{A} \mid \mathcal{B} & \mid & \neg \mathcal{A} & \mid & \mathcal{A} \triangleright \mathcal{B} & \mid & 0 & \mid \diamond \mathcal{A} & (\mathcal{L}^{\diamond}_{spat}) \\ & \mid & \langle \mathsf{open}\, x \rangle \mathcal{A} & \mid & x[\mathcal{A}] & \mid & \exists x.\, \mathcal{A} & & & & & & (\mathcal{L}^{(MA)}_{mod}) \end{array}$$

Semantics for $\mathcal{L}^{(MA)}_{mod}$ specific connectives is defined as expected, relative to a valuation $v : \mathsf{Var} \to \Lambda$. We then have

$$\begin{array}{lll} P, v \models M \exists x.\, \mathcal{A} & \text{iff} & \exists\, n \in \Lambda.\;\, P, v\{^x/n\} \models M \mathcal{A} \\ P, v \models M x[\mathcal{A}] & \text{iff} & \exists Q.\;\, P \equiv v(x)[Q] \text{ and } Q, v \models M \mathcal{A} \\ P, v \models M \langle \mathsf{open}\, x \rangle \mathcal{A} & \text{iff} & \exists Q.\;\, P \overset{\mathsf{open}\, v(x)}{\longrightarrow} Q \text{ and } Q, v \models M \mathcal{A} \end{array}$$

where $P \overset{\mathsf{open}\, n}{\longrightarrow} Q \overset{\text{def}}{=} \exists P.\, P \equiv \mathsf{open}\, n.\, R \mid R'$ and $Q \equiv R \mid R'$. We now show how to mimick the encoding of Section D.2 for the case of ambients: we encode the valuation together with process P to be model-checked by defining "rows processes" that exceed P in depth. Then using such rows we make the model-checked process interact, so that we may encode the modalities $x[\mathcal{A}]$ and $\langle \mathsf{open}\, x \rangle \mathcal{A}$. We then define rows and threads as processes of the form:

$$\begin{array}{lll} \mathsf{threadopen}(\widetilde{a}, n) & \overset{\text{def}}{=} & \mathsf{open}\, a_1.\, \mathsf{open}\, a_2 \ldots \mathsf{open}\, a_n.\, [\mathbf{0}] \\ \mathsf{rowopen}(a, n) & \overset{\text{def}}{=} & \underbrace{\mathsf{open}\, a.\, \mathsf{open}\, a \ldots \mathsf{open}\, a}_{n \text{ times}}.\, [\mathbf{0}] \end{array}$$

The formulas shown in Fig. D.4 show how we may characterise these processes logically. The embedding of $\mathcal{L}^{(MA)}_{mod}$ into $\mathcal{L}^{\diamond}_{spat}$ then follows the one in Section D.2, small differences only appear in encoding of modalities. We set

$$\begin{array}{lll} [\![\exists x.\, \mathcal{A}]\!]_{(e,w)} & \overset{\text{def}}{=} & \mathsf{ProcVal}(e, w)_K \;\wedge \big(\big(\mathsf{RowOpen}(\mid e \mid +1)\big)^{2^w} \;\blacktriangleright\; [\![\mathcal{A}]\!]_{(e',w)}\big) \\ & & \quad \text{where } e' = e\{^x/ \mid e \mid +1\} \\ [\![\langle \mathsf{open}\, x \rangle \mathcal{A}]\!]_{(e,w)} & \overset{\text{def}}{=} & \mathsf{ProcVal}(e, w)_K \;\wedge \\ & & \mathsf{RowOpen}(K + e(x)) \;\big|\; \big(\mathsf{TestRow}(K + e(x)) \;\blacktriangleright\; \diamond [\![\mathcal{A}]\!]_{(e,w)} \\ [\![x[\mathcal{A}]]\!]_{(e,w)} & \overset{\text{def}}{=} & \mathsf{ProcVal}(e, w)_K \;\wedge (\mathsf{Val}(e, w)_K \mid 1) \\ & & \;\wedge \diamond \big(\mathsf{RowOpen}(K + e(x) - 1) \;\big|\; (\mathsf{RowOpen}(K + e(x)) \;\blacktriangleright\; [\![\mathcal{A}]\!]_{(e,w)})\big) \end{array}$$

Formula	Encoding	Interpretation
atom	$1 \ \blacktriangleright\ 1\Diamond 0$	$\exists a.\ \ P \equiv \mathsf{open}\, a$ or $P \equiv a[0]$
$\mathsf{testamb}$	$1\ \wedge \Diamond\mathsf{atom}$	$\exists a, b\ P \equiv a\big[\mathsf{open}\, b \mid b[\mathbf{0}]\big]$
$\mathsf{1open}$	$\mathsf{atom}\ \wedge \mathsf{testamb}\ \blacktriangleright\ \Diamond\Diamond 0$	$\exists a.\ P \equiv \mathsf{open}\, a$
$\mathsf{1amb}$	$\mathsf{atom}\ \wedge \mathsf{1open}\ \blacktriangleright\ \Diamond 0$	$\exists a.\ P \equiv a[0]$
$\mathsf{ThrOpen}(1)$	$\mathsf{1amb}\ \blacktriangleright\ \Diamond 0$	$\exists\widetilde{a}.\ P \equiv \mathsf{threadopen}(\widetilde{a}, 1)$
$\mathsf{ThrOpen}(k+1)$	$1\ \wedge\ \mathsf{1amb}\ \blacktriangleright\ \Diamond\mathsf{ThrOpen}(k)$	$\exists\widetilde{a}.\ P \equiv \mathsf{threadopen}(\widetilde{a}, k+1)$
$\mathcal{M}_0$	0	$P \in M_0$
$\mathcal{M}_{k+1}$	$\big(1\ \Rightarrow\ \mathsf{atom}\ \blacktriangleright\ \Diamond\mathcal{M}_k\big)^{\forall}$	$P \in M_{k+1}$
EqOp	$\big(\mathsf{1open}\ \rhd\ \Box\bot\big)\ \wedge\ \big(\mathsf{1amb}\ \blacktriangleright$ $\big(\mathsf{1amb} \mid 1 \Rightarrow \Diamond\top\big)^{\forall}$	$\exists\ a, \widetilde{P}.$ $P\ \equiv\ \mathsf{open}\, a.\, P_1 \mid \ldots \mid \mathsf{open}\, a.\, P_n$
$\mathsf{RowOpen}(1)$	$\mathsf{1open}$	$\exists a.\ P \equiv \mathsf{rowopen}(a, 1)$
$\mathsf{RowOpen}(k+1)$	$(\mathsf{1amb} \mid \mathsf{1open})\ \blacktriangleright\ \Big((\mathsf{1amb} \mid \mathsf{EqOp})$ $\wedge \Diamond\ \big((\mathsf{RowOpen}(k) \mid \mathsf{1open})\ \wedge\ \mathsf{EqOp}\big)\Big)$	$\exists a.\ P \equiv \mathsf{rowopen}(a, k+1)$

Figure D.4:

where $\mathsf{TestRow}(n)$ is the formula

$$\mathsf{TestRow}(n) \stackrel{\text{def}}{=} 1\ \wedge\ (\mathsf{1amb} \mid \mathsf{1open}\ \wedge \Diamond\top)\ \blacktriangleright\ \Diamond\big(\mathsf{1amb} \mid \mathsf{RowOpen}(n)\ \wedge\ \Diamond\top\big)$$

which characterises processes of the form $\mathsf{testproc}(a, n) \stackrel{\text{def}}{=} a[\mathsf{rowopen}(a, n)]$.

We may check that $\mathsf{testproc}(a, n)$ satisfy this formula for introducing $a[\mathbf{0}] \mid \mathsf{open}\, a$ in parallel. Reciproquely, if P satisfies $\mathsf{TestRow}(n)$, then P starts with an ambient or an open, and there is a process $a[\mathbf{0}] \mid \mathsf{open}\, a$ such that $P \mid a[\mathbf{0}] \mid \mathsf{open}\, a \longrightarrow \mathsf{rowopen}(b, n) \mid b[\mathbf{0}]$. So $\mathsf{open}\, a.\, \mathbf{0}$ disapeared in the reduction ($n \geq 2$), and it cannot have interacted with $a[\mathbf{0}]$ otherwise this would have given P, which is not composed. So $P \equiv a[P']$, and the reduction is $P \mid a[\mathbf{0}] \mid \mathsf{open}\, a \longrightarrow P' \mid a[\mathbf{0}]$, which is to say that $P \equiv \mathsf{TestRow}(n)$.

We again prove the correctness of the embedding by induction on $\mathcal{A}$. Differences with the proof of Lemma D.2.6 happen only for the modality cases:

- $\mathcal{A} = \langle\mathsf{open}\, x\rangle\mathcal{A}'$: Assume first that $P{\models}[\![\mathcal{A}]\!]_{e,w}$. Then $P \equiv Q \mid \mathbf{val}(e, \nu, w)_K$, and if Val' is such that $\mathbf{val}(e, \nu, w)_K \equiv \mathsf{Val}' \mid \mathsf{rowopen}(\nu(K + e(x)),)$, then there is some name a such that $Q \mid \mathsf{Val}' \mid \mathsf{testproc}(a, K + e(x)) \longrightarrow P'$ and $P'{\models}[\![\mathcal{A}']\!]_{e,w}$. By induction, $P'{\models}\mathsf{ProcVal}(e, w)_K$, so the $\mathsf{testproc}$ must have been alterated by the reduction. Moreover, the other partner for the reduction cannot be in Val' since it would consume a row copy which would be visible at end. So this must be the process, that is the reduction is $Q \mid \mathsf{Val}' \mid \mathsf{testproc}(b, K + e(x)) \longrightarrow Q' \mid \mathsf{Val}' \mid \mathsf{rowopen}(a, K + e(x))$ with $Q \stackrel{\mathsf{open}\, a}{\longrightarrow} Q'$. Since $\mathsf{Val}' \mid \mathsf{rowopen}(a, \equiv)\mathbf{val}(e, \nu', w)_K$, this must be that $\nu = \nu'$ and $a = \nu(e(x))$, and the result by induction. Reciproquely, if $P, \nu \circ e{\models}M_K\mathcal{A}$, then there is P' such that $P \stackrel{\mathsf{open}\, a}{\longrightarrow} P'$ with $a = \nu \circ e(x)$ and $P', \nu \circ e{\models}\mathcal{A}'$. Then $P \mid \mathsf{Val}' \mid \mathsf{Testproc}(a, K + e(x)) \longrightarrow P' \mid \mathbf{val}(e, \nu, w)_K$, and $P' \mid \mathbf{val}(e, \nu, w)_K{\models}[\![\mathcal{A}']\!]_{e,w}$ by induction, so that finaly $P{\models}[\![\mathcal{A}]\!]_{e,w}$.
- $\mathcal{A} = x[\mathcal{A}']$: Assume first that $P{\models}[\![\mathcal{A}]\!]_{e,w}$. Then $P \equiv Q \mid \mathbf{val}(e, \nu, w)_K$ with

Q single, there is a, b, P' such that $P \longrightarrow P' \mid \mathsf{rowopen}(a, K{+}e(x){-}1)$, and $P' \mid \mathsf{rowopen}(b, K{+}e(x)) \models [\![\mathcal{A}']\!]_{e,w}$. By induction, $P' \mid \mathsf{rowopen}(b, K{+}e(x))$ contains a process $\mathbf{val}(e, \nu', w)_K$, and again the only way to have this is to have $\nu = \nu'$ and $b = \nu(K + e(x))$. This says that a row of depth $K + e(x)$ was consumed by the reduction but not any other, so that $Q \equiv a[Q']$, $P' \equiv Q' \mid \mathsf{Val}'$ with $\mathbf{val}(e, \nu, w)_K \equiv \mathsf{Val}' \mid \mathsf{rowopen}(a, K{+}e(x){+}1)$, $a = b = \nu(x)$, and the result. Reciproquely, if $P, \nu{\circ}e \models M_K \mathcal{A}$, then there is a, P' such that $P \equiv a[P']$, $P', \nu{\circ}e \models M_K \mathcal{A}'$, and $\nu \circ e(x) = a$. So $P \mid \mathbf{val}(e, \nu, w)_K \longrightarrow P' \mid \mathsf{Val}' \mid \mathsf{rowopen}(a, K{+}e(x){-}1)$, and $P' \mid \mathsf{Val}' \mid \mathsf{rowopen}(a, K{+}e(x)) \models [\![\mathcal{A}']\!]_{e,w}$ by induction, that is $P \mid \mathbf{val}(e, \nu, w)_K \models [\![\mathcal{A}]\!]_{e,w}$.

The π-calculus

We consider the choice-free finite synchronous π-calculus, given by:

$$P \quad ::= \quad n(m).\, P \;\big|\; \overline{n}(m).\, P \;\big|\; (\nu n)P \;\big|\; P \mid P \;\big|\; \mathbf{0}$$

where n, m, p ranges over the set of names Λ, and we assume defined in the standard way the relation $P \to P'$ of reduction, and the relation of $P \xrightarrow{\alpha} P'$ of labeled transition, over the set of labels τ, $n(m)$, and $\overline{n}(m)$, where we assume that the case $P \xrightarrow{\overline{n}(m)} P'$ where $m \notin \mathrm{fn}(P)$ corresponds to a bound output. We then consider the logics $\mathcal{L}^{\diamond}_{spat}$ and $\mathcal{L}^{(CCS)}_{mod}$ exactly as defined in Section C.1, but where we now consider quantifiers and modalities to range over *commitments* $n, \overline{n} \in \mathsf{Com}$, where $n \in \Lambda$. Semantics for $\mathcal{L}^{(CCS)}_{mod}$ specific connectives over the π-calculus is defined as expected, relative to a valuation $v : \mathsf{Var} \to \mathsf{Com}$. We have

$$\begin{array}{llll}
P, v & \models M & \exists x.\, \mathcal{A} & \text{if} \quad \exists \alpha \in \mathsf{Com}.\ P, (v\{^x/\alpha\}) \models M\ \mathcal{A} \\
P, v & \models M & \langle x \rangle.\, \mathcal{A} & \text{if} \quad \exists P', n \in \Lambda.\ P \xrightarrow{v(x)(n)} P' \textit{ and } P', v \models M \mathcal{A} \\
P, v & \models M & \langle \overline{x} \rangle.\, \mathcal{A} & \text{if} \quad \exists P', n \in \Lambda.\ P \xrightarrow{\overline{v(x)}\langle n \rangle} P' \text{ and } P', v \models M \mathcal{A}
\end{array}$$

To embed $\mathcal{L}^{(CCS)}_{mod}$ into $\mathcal{L}^{\diamond}_{spat}$ in the case of π, we use a slightly different (when compared with the one developed in Section D.2) notion of row, but completely equivalent for our purposes, and that makes use of the particulars of the π-calculus model. So, instead of letting a row of size k be a sequential thread on the same action α, in this case we consider a row of size k holding the action α to be a π-process P such that the following hold:

- For all n, β, we have $P \xrightarrow{\beta} P_0 \to P_1 \to \cdots \to P_n \to Q$ if and only if
  - $n = k$, $\beta = \alpha$, $Q \equiv \mathbf{0}$, and
  - For all $i \in \{0, \ldots, k\}$, there are no $\ell \neq \tau$ and R such that $P_i \xrightarrow{\ell} R$, and
  - $P \models 1$, and for all $i \in \{0, \ldots, k\}$ we have $P_i \models 1$.

Processes P satisfying this specification are said to be rows of size k on α. Such processes exist in the π-calculus because of the possibility of synchronizations on restricted channels, *e.g.*,

$$\mathbf{row}(p, 2) \stackrel{\mathrm{def}}{=} p(z).\, (\nu n)(\overline{n}.\, \mathbf{0} \mid n(x).\, (\nu m)(\overline{m}.\, \mathbf{0} \mid m(y).\, \mathbf{0}))$$

Of course, many other implementations exist, for example we could also have

$$\mathbf{row}(p,2) \stackrel{\text{def}}{=} p(z).\,(\nu n)(\overline{n}(n).\,\mathbf{0} \mid n(x).\,(\overline{x}x.\,\mathbf{0} \mid x(y).\,\mathbf{0}))$$

It seems quite difficult to characterize all these processes "intensionally", but we can do it logically as follows (let $\Box\mathcal{A} \stackrel{\text{def}}{=} \neg\Diamond\neg\mathcal{A}$):

$$\begin{array}{lcl}
\mathsf{piAct} & \stackrel{\text{def}}{=} & 1 \ \blacktriangleright\ \Diamond 0 \\
\mathsf{piRow}(k) & \stackrel{\text{def}}{=} & 1 \wedge \Box\bot \wedge (\mathsf{piAct} \ \blacktriangleright\ \Diamond\mathsf{piThread}(k)) \\
\mathsf{Grow} & \stackrel{\text{def}}{=} & \neg 0 \mid \neg 0 \mid \neg 0 \\
\mathsf{piNoExternal} & \stackrel{\text{def}}{=} & ((1 \wedge \Box\bot) \ \triangleright\ \neg\Diamond\mathsf{Grow}) \\
\mathsf{piThread}(0) & \stackrel{\text{def}}{=} & 0 \\
\mathsf{piThread}(n+1) & \stackrel{\text{def}}{=} & 1 \wedge (\Diamond\top) \wedge \mathsf{piNoExternal} \wedge \Box\mathsf{piThread}(n)
\end{array}$$

We can then show that $P \models \mathsf{piRow}(k)$ if and only if P is a row of size k on some action α. We also have that $P_0 \models \mathsf{piThread}(k)$ whenever

- For all n, we have $P_0 \to P_1 \to \cdots \to P_n \to Q$ if and only if
  - $n = k$, $Q \equiv \mathbf{0}$, and
  - For all $i \in \{0,\ldots,k\}$, $P_i \models 1$ and there are no $\ell \neq \tau$ and R such that $P_i \xrightarrow{\ell} R$.

The constructions above gives us suitable notions of row and thread of depth k. Notice that unlike for the rows defined in Section D.2, the name associated to a row is just the first action (the other transitions of a row are always reductions). This is not a problem, because only the first action of a row is actually used in testing for the value of the variable it represents in the encoding of valuations.

From these ingredients, we can then develop a counterpart of Theorem D.2.1; given the previous definitions specific for the π-calculus model, the encoding of $\mathcal{L}^{(CCS)}_{mod}$ into $\mathcal{L}^{\Diamond}_{spat}$ is the same as the one in Fig D.3. We can then obtain results identical to those presented in Section D.4 and D.5.