THÈSE

Présentée à l'École Normale Supérieure de Cachan
pour obtenir le grade de

Docteur de l'École Normale Supérieure de Cachan

Par : Alexandre BOISSEAU
Discipline : INFORMATIQUE

Abstractions pour la vérification de propriétés de sécurité de protocoles cryptographiques

Soutenue le 19 septembre 2003

Composition du jury :

– Michel BIDOIT	Directeur de thèse
– Paul GASTIN	Président du jury
– Jean GOUBAULT-LARRECQ	Examinateur
– Francis KLAY	Examinateur
– Yassine LAKHNECH	Rapporteur
– Jean-François RASKIN	Rapporteur

Remerciements

Je voudrais avant tout remercier les membres du jury.

Jean-François Raskin, ainsi que Steve Kremer, m'ont chaleureusement accueilli au Département d'Informatique de l'Université Libre de Bruxelles, pour travailler avec eux sur les protocoles de type signatures électroniques de contrats. Je garde un excellent souvenir de ces quelques semaines de travail passées en Belgique et je les en remercie. Je remercie d'autant plus Jean-François d'avoir accepté la charge de rapporteur au sujet de cette thèse.

Yassine Lakhnech m'a invité à deux reprises pour donner un séminaire au laboratoire Vérimag à Grenoble. Ces séminaires ont toujours été suivis de longues discussions et d'échanges d'idées très intéressants et mon seul regret est de n'avoir pu développer, au cours de cette thèse, toutes les pistes qui m'avaient été suggérées. Je le remercie pour ces invitations et pour avoir, lui-aussi, accepté la charge de rapporteur.

Jean Goubault-Larrecq a accepté de m'encadrer sur la dernière partie de ma thèse et de faire partie du jury et je l'en remercie. Mais ce pourquoi j'aimerai le remercier également, c'est pour un moment dont, peut-être, il ne se souvient pas. Il s'agit d'une discussion que nous avons eue un jour sur les catégories. Discussion totalement désinterressée, juste pour le plaisir de raconter (pour lui) et d'écouter (pour moi). Merci donc pour ce moment de science pure et pour le reste.

Francis Klay a accepté de faire partie du jury et Paul Gastin en a été le président, je les en remercie tous deux.

Je tiens à remercier mon directeur de thèse, Michel Bidoit pour m'avoir encadré durant cette thèse et le stage de DEA qui l'a précédé tout en me laissant librement décider de suivre telle ou telle direction et faire les choses à ma manière.

Je remercie tous mes camarades Cachanais et assimilés pour leur amitié indéfectible, depuis sept ans (plus ou moins quelques années pour certains). Je voudrais remercier en particulier Nicolas, qui m'a de nombreuses fois aidé à réaliser avec LaTeX mes idées les plus saugrenues et Marie, qui partage et égaye mon bureau depuis plus de trois ans (sans se plaindre).

Bien que je leur ai rarement parlé de mon travail, mes parents et ma sœur m'ont toujours soutenu dans mes choix et ont été extrêmement disponible lorsque j'avais besoin d'eux (en particulier pour l'organisation du pot de thèse). Je les en remercie vivement.

Finalement, je tiens à remercier mes élèves, passés, présents (absents aussi, parfois) et à venir. Grâce à eux, nos salles de cours ont toujours été un lieu agréable et leurs questions m'ont permis de comprendre que rien n'est jamais définitif.

Table des matières

Introduction

Depuis le développement fulgurant de l'utilisation des réseaux informatiques et de l'informatisation des communications (et l'enregistrement des données), rares sont les personnes à n'avoir aujourd'hui aucun contact avec un ordinateur « au sens large » : ordinateurs personnels, terminaux bancaires, informatisation des dossiers médicaux, etc. Chacun, ou presque, est alors confronté à un problème qui ne préoccupait jusqu'à présent que les militaires et diplomates : comment préserver la confidentialité de ses données ? Considérons par exemple des renseignements relatifs au compte bancaire d'un client. Un contrôle d'accès, s'il est correctement mis en place, permet d'assurer la confidentialité de ces renseignements tant qu'ils restent statiques : stockés sur ordinateur dans la banque et sur papier chez le client par exemple. Ce n'est plus le cas si, comme c'est aujourd'hui possible, le client désire consulter son compte ou réaliser des opérations bancaires à distance au moyen d'internet car les données doivent alors circuler sur le réseau et il serait facile, pour quiconque en connaît le fonctionnement, d'y accéder, voire de les modifier. Il est donc apparu, pour le public, un besoin de sécuriser les communications électroniques.

Depuis l'antiquité, d'après [EU00], l'usage de techniques cryptographiques tente d'apporter une réponse à ce problème. Cette méthode consiste à transformer les données au moyen d'un algorithme tel que retrouver les données de départ à partir des données chiffrées (*i.e.* celles obtenues après la transformation) soit algorithmiquement très difficile *a priori* et facile si on dispose d'un autre algorithme dit de déchiffrement. Dans les premiers temps de la cryptographie, le secret reposait sur celui des algorithmes utilisés (essentiellement des transpositions des lettres du message de départ ou des substitutions par d'autres lettres). En 1883, A. Kerckhoffs énonce ce que l'on appelle aujourd'hui les principes de Kerckhoffs [Ker83] que nous reprenons ici :

> « [...] Il faut un système remplissant certaines conditions exceptionnelles, conditions que je résumerai sous les six chefs suivants :
>
> 1. Le système doit être matériellement, sinon mathématiquement, indéchiffrable ;
> 2. Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;
> 3. La clef doit pouvoir en être communiquée et retenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants ;
> 4. Il faut qu'il soit appliquable à la correspondance télégraphique ;
> 5. Il faut qu'il soit portatif, et que son maniement ou son fonctionnement n'exige pas le concours de plusieurs personnes ;
> 6. Enfin, il est nécessaire, vu les circonstances qui en commandent l'application, que le système soit d'un usage facile, ne demandant ni tension d'esprit, ni la connaissance d'une longue série de règles à observer. »

Les deuxième et troisième points sont sans doute les plus novateurs. Suivant ces recommandations, l'algorithme de chiffrement doit maintenant prendre en entrée, non seulement les données à chiffrer, mais aussi une clé. Le chiffrement se fait relativement à cette clé avec pour conséquence que l'algorithme peut tout à fait être public, du moment que la clé reste secrète. La même clé était alors utilisée pour le chiffrement et le déchiffrement (avec, bien entendu, des algorithmes différents pour chiffrer et déchiffrer) ce qui posait, entre autres, le problème de la communication de la clé. On parlait alors de clés (et de cryptographie) symétriques. En 1976, W. Diffie et M. Hellman proposent une nouvelle approche utilisant deux clés dont l'une peut être divulguée [DH76], dont une réalisation pratique est donnée dans [RSA78] : c'est le fameux algorithme RSA de chiffrement. Chaque personne possède alors une clé dite publique qui est potentiellement connue de tous et une clé secrète (on parle alors de clés et de cryptographie asymétriques). Pour faire parvenir à cette personne un message confidentiel, il suffit de la chiffrer en utilisant la clé publique et de lui envoyer. Peu importe si le message chiffré peut être observé ou pas : le destinataire sera le seul à pouvoir le déchiffrer, puisqu'il faut pour cela utiliser la clé secrète. Ce mécanisme permet également de mettre en place une certaine forme d'authentification ou de signature : si quelqu'un chiffre un message connu avec sa clé secrète, on peut vérifier que cette opération a bien été réalisé avec la bonne clé (et par conséquent, par la bonne personne, sous certaines hypothèses) en déchiffrant simplement le message à l'aide de la clé publique. Le problème de la communication des clés est également résolu puisque chaque personne peut, de manière privée, exécuter un algorithme lui fournissant deux clés inverses l'une de l'autre, et en publier une. Néanmoins, ces considérations cryptographiques ne sont qu'un élément de réponse au problème des communications sûres et de nombreuses questions restent en suspens si on veut les mettre en pratique : comment obtient-on, par exemple, la clé publique d'une personne ? Il n'est par ailleurs pas recommandé d'utiliser à outrance les clés privées et publiques. D'une part parce que les temps de chiffrement et de déchiffrement sont plus longs qu'avec des clés symétriques et d'autre part, parce que les clés privées peuvent parfois être découvertes plus facilement lorsque de nombreux couples constitués d'un message en clair et du même message chiffré sont connus. Comment, dans ces conditions, se mettre d'accord avec quelqu'un sur une clé symétrique, qui sera utilisée pour une communication à court terme ? Les protocoles cryptographiques permettent d'intégrer la cryptographie dans des communications réelles en répondant à ces problèmes concrets.

Un protocole cryptographique se présente comme une succession de réceptions et d'envois de messages, entrecoupé de calculs éventuels (déchiffrer des messages, les recombiner, faire des vérifications, etc.) et qui est établi dans un but précis (échanger une clé symétrique, s'assurer de l'identité de son interlocuteur, etc.). Les différents types d'interlocuteurs qui apparaissent dans la description du protocole sont appelés des principaux, par opposition aux participants qui sont les entités instanciant ces principaux dans une session du protocole (*i.e.* une exécution réelle). On emploie parfois les termes plus imagés de rôles et d'acteurs. En plus de la cryptographie classique, les protocoles cryptographiques font généralement usage d'autres notions, par exemple [RSG$^+$01, CJ97] :

– les fonctions de hachage : elles envoient les messages sur un ensemble fini, relativement petit (quelques dizaines d'octets). Elles ne peuvent par conséquent pas être injectives, mais on les choisit de sorte que les collisions accidentelles soient très peu probables et de sorte que, d'une part, toute modification (même minime) du message de départ se traduise par un modification conséquente dans la valeur retournée par la fonction de hachage et, d'autre part, que connaissant un message et son image par la fonction de hachage, il soit très difficile de produire un autre message dont l'image par la fonction de hachage soit la même ;

- les nonces : ce sont des nombres aléatoires qui sont utilisés pour identifier différentes sessions d'un même protocole, ou comme challenges pour vérifier qu'un participant possède bien une certaine clé, ou à d'autres occasions encore. Les nonces créés au cours d'une session d'un protocole ont la propriété d'être imprévisibles et différents de tous les autres nonces créés par ailleurs (que ce soit dans la même session ou dans une autre) ;
- les cachets (*timestamp*) : ce sont des valeurs contenant une référence à la date à laquelle ils sont créés. Ils permettent d'éviter à un participant d'accepter de vieux messages (issus de sessions précédentes du protocole). On notera que les cachets et les nonces peuvent être utilisés pour les mêmes raisons (une discussion sur les relations entre cachets, nonces et d'autres formes de nombres plus ou moins aléatoires utilisés dans les protocoles se trouve dans [Gon93]) ;
- les signatures électroniques : elles peuvent être apposées sur des messages de manière sûre, *i.e.* il ne doit pas être possible de signer à la place de quelqu'un d'autre. En général, les algorithmes de signature électronique utilisent d'une manière ou d'une autre une clé asymétrique privée. Il est alors possible de vérifier le contenu du message et l'identité du signataire.

Les protocoles cryptographiques sont bien entendu sensibles aux attaques visant à décrypter les messages (*i.e.* tenter de découvrir le message de départ à partir du message chiffré sans être en possession de la clé). Mais il ne faudrait pas croire que ce soient les seules attaques possibles. En fait, de nombreux protocoles cryptographiques présentent des failles de conception bien plus facile à exploiter que les faiblesses éventuelles des algorithmes de chiffrement. Citons par exemple (toujours à partir de [RSG$^+$01, CJ97]) :

- les attaques de type entre-deux (*man-in-the-middle*) : un participant malhonnête communique avec deux participants honnêtes, qui jouent, en général deux rôles différents d'un même protocole. L'attaquant, en modifiant subtilement les messages, peut amener chacun (ou au moins un) des deux participants à croire qu'il parle à l'autre, alors que c'est lui qui reçoit et envoie les messages et peut même parfois espionner leur contenu ;
- les attaques au moyen d'oracles : elles consistent, pour l'attaquant, à utiliser les réponses mécaniques que font les participants honnêtes lorsqu'ils reçoivent les messages pour les amener à révéler, sans qu'ils s'en rendent compte, des informations. Typiquement, un protocole qui contient une étape où un principal doit déchiffrer un message reçu et le renvoyer sans avoir pu vérifier son contenu peut être sensible à ce genre d'attaques ;
- le rejeu de messages : il s'agit d'utiliser les messages échangés dans une autre session et à les réinjecter dans la section courante de manière à ce que les participants les acceptent et poursuivent le protocole à partir de là. Cela peut les conduire, par exemple, à se mettre d'accord sur une clé symétrique relativement ancienne, qui a donc pu être dévoilée depuis ;
- les attaques algébriques : elles consistent à tirer parti d'identités éventuelles entre messages (par exemple, le fait que les opérations de chiffrement commutent dans certains systèmes de cryptographie) afin de faire passer des messages pour d'autres.

Les failles qui permettent de telles attaques sont qualifiées de failles logiques et on s'accorde en général pour penser que l'étude des failles logiques des protocoles est orthogonale à l'étude des failles du système de cryptographie sous-jacent. Ces failles sont relativement subtiles, difficiles à déceler à la simple vue du texte du protocole (particulièrement lorsqu'elles mettent en jeu plusieurs sessions du protocole). Ces protocoles intervenant dans virtuellement toute communication électronique exigeant un haut niveau de sécurité et étant difficiles à appréhender, il est nécessaire de ne les employer qu'après une étude minutieuse, ce qui sera pour nous synonyme de vérification formelle.

La vérification formelle d'un problème (et donc en particulier des protocoles cryptographiques) est constituée de trois étapes. Il faut tout d'abord proposer un modèle mathématique pour le problème. On raisonne ensuite mathématiquement sur ce modèle pour en déduire des conséquences et, enfin, on interprète ces conséquences dans le cadre du problème de départ. Parmi ces trois étapes, seule la deuxième peut être formelle. La première et la dernière concernent les rapports entre le monde réel et le modèle qui en a été proposé. Ce modèle est en général choisi de sorte que l'interprétation des conséquences de l'étape formelle dans le monde réel soit simple et naturelle. On se ramène donc essentiellement à deux étapes : la modélisation suivie de la vérification formelle. La modélisation apparaît alors comme un passage obligé extrêmement délicat, puisque c'est sur elle que repose la confiance que l'on peut avoir dans le résultat de la vérification formelle, et qui doit être justifié, informellement (bien entendu) mais précisément. En particulier, en ce qui concerne les protocoles cryptographiques, il existe un certain nombre d'hypothèses simplificatrices qui peuvent être faites sur les modèles qui correspondent chacune à des classes d'attaques, du type de celles évoquées plus haut (autrement dit, faire l'hypothèse simplificatrice sur le modèle empêche ensuite de considérer les attaques qui se trouvent dans la classe correspondante). Les modèles que nous utiliserons dans cette thèse seront issus de modèles classiques de la littérature (à de petites variations près), ce qui nous permettra d'éviter de longues discussions pour les justifier. Toujours dans la domaine de la modélisation, signalons qu'un protocole ne peut jamais être correct en soi. La correction ne peut être envisagée que relativement à une propriété de sécurité attendue, qu'il faut décrire formellement. Voici quelques propriétés typiques des protocoles [RSG$^+$01] :

- les propriétés de secret. Il s'agit probablement des propriétés les plus étudiées sur les protocoles cryptographiques. Informellement, le secret d'une donnée signifie qu'un attaquant ne peut jamais être en mesure de calculer le texte correspondant à cette donnée ;
- les propriétés d'authentification : il s'agit des propriétés les plus étudiées, après les propriétés de secret. Elles sont pourtant, le plus souvent, traitées par le biais du secret. Ceci est sans doute en partie dû au fait qu'elles sont difficiles à énoncer formellement. Intuitivement, une propriété d'authentification énonce qu'il ne peut y avoir de méprise sur le participant réalisant certaines actions. Cependant, dès que l'on tente de donner une formalisation précise, l'authentification se ramifie en plusieurs propriétés formelles, dont la pertinence dépend du protocole ;
- l'intégrité : elle stipule que les données échangées ne pourront pas être modifiées (ou, au pire, que les modifications éventuelles seront détectées) ;
- la non-répudiation : cette propriété énonce que, si un participant a effectivement envoyé un certain message à un autre, ce dernier ne pourra pas nier l'avoir reçu ;
- l'équité : c'est une propriété typique des protocoles électroniques de signature de contrats. Dans un tel protocole, il y a en général un participant qui s'engage le premier en signant d'abord le contrat et qui est alors désavantagé par rapport aux autres. L'équité énonce que, quoi qu'il arrive, ce participant ne pourra pas être floué par les autres ;
- la disponibilité : c'est une propriété de vivacité des protocoles qui a pour but de garantir que chaque requête faite au cours du protocole recevra bien une réponse ;
- l'anonymat d'un principal dans un protocole : cette propriété énonce que, simplement

en observant une session du protocole, un observateur ne peut déduire l'identité du principal en question dans cette session.

Nous traiterons des propriétés de secret et d'authentification (et dans une certaine mesure, d'intégrité, qui est une forme particulière d'authentification, suivant [RSG+01]) dans la première partie de cette thèse. Dans la deuxième partie, nous parlerons des propriétés de type équité (mais la non-répudiation rentrerait aussi tout à fait dans ce cadre) ainsi que d'une certaine forme de disponibilité et d'autres propriétés des protocoles de signature de contrats. Dans la dernière partie, nous étudierons des propriétés généralisant l'anonymat. Pour une description plus précise de ces propriétés et des approches existantes pour les étudier (incluant des références bibliographiques), nous renvoyons le lecteur aux introductions spécifiques à chacune de ces parties. En règle générale, un protocole cryptographique est établi afin d'assurer une certaine propriété et il s'avère que, à chaque type de propriété des protocoles cryptographiques, correspond une classe bien identifiée de protocole englobant les protocoles connus qui tentent d'assurer cette propriété. Il en est de même pour les modèles, autrement dit la modélisation est souvent choisie en fonction des propriétés que l'on désire vérifier. Ces modèles peuvent être présentés sous plusieurs formes, par exemple axiomatiquement (dans des logiques classiques ou dédiées) ou sous forme de systèmes de transitions (quelle que soit la modélisation, il y a de toutes façons presque toujours un système de transitions sous-jacent décrivant les exécutions du protocole). Comme expliqué plus haut, il est important de savoir quelles hypothèses sont faites (parfois implicitement) dans une modélisation donnée. On peut ainsi se demander :

- combien de participants le modèle peut-il gérer ? de sessions ?
- ces sessions sont-elles traitées de manière séquentielle ? concurrente ?
- comment sont gérés les nonces ? les cachets ?
- quels sont les couples de messages indistinguables pour un participant donné ?

Suivant les réponses apportées à ces questions (et d'autres du même type), il se peut que certaines classes d'attaques ne puissent pas être prise en compte dans le modèle (c'est pratiquement toujours le cas, il faut surtout en avoir conscience). Une fois que l'on dispose d'un modèle (représentant un protocole cryptographique) et d'une propriété de ce modèle (représentant une propriété du protocole), on peut passer à l'étape de vérification formelle proprement dite.

La vérification formelle du fait qu'un système satisfait une propriété peut se faire suivant trois approches. La première est le test qui consiste à essayer, sur le modèle, différents scénarios et regarder si l'exécution de l'un d'entre eux ne permettrait pas d'invalider la propriété. Le défaut majeur de cette approche est qu'elle n'est, en général, pas complète (*i.e.* elle ne permet pas de prouver la propriété, mais seulement de la réfuter). Par contre elle est très utile pour découvrir des failles rapidement, particulièrement pendant la phase de mise au point du système. La deuxième approche est la preuve qui consiste à décrire axiomatiquement le modèle, dans le même langage que la propriété, puis à utiliser un système de déduction pour tenter d'en déduire la propriété (éventuellement avec l'aide d'un système d'aide à la preuve). Cette méthode est puissante, permet de vérifier tous les problèmes de vérification qui se posent en pratique, mais reste lourde à mettre en œuvre. La dernière approche, le *model-checking*, consiste à utiliser un algorithme permettant de décider si une propriété est satisfaite dans le modèle (sous réserve qu'un tel algorithme existe). On trouve, dans la littérature concernant l'étude des protocoles cryptographiques, des travaux utilisant ces trois approches. Le test consiste, pour les protocoles cryptographiques, à identifier un sous ensemble des exécutions du protocole, explorable de manière exhaustive, et à vérifier que, sur ces exécutions, la propriété est satisfaite (voir par exemple [MMS97, SS98, CJM98] pour des travaux s'inscrivant dans ce cadre). Lorsqu'elle ne l'est pas, on a trouvé une attaque sur le protocole (c'est pourquoi cette méthode est appelée, dans le cadre de l'étude des protocoles cryptographiques, une recherche d'attaques). Par contre, si aucune attaque n'est trouvée, on ne peut pas en déduire, en toute généralité, que la propriété est satisfaite. La preuve a également été utilisée, car les protocoles cryptographiques se prêtent bien à une représentation axiomatique. On peut citer par exemple [Wei99, CD99, Pau98] (notons qu'une méthode de démonstration automatique, ou au moins un assistant de preuve, sont utilisés dans ces travaux). Pour le *model-checking*, c'est plus délicat. En effet, il est bien connu que les modélisations raisonnables des protocoles cryptographiques conduisent à des systèmes indécidables [CCM01, DLMS99]. La première possibilité consiste à considérer seulement un fragment du modèle, sur lequel il est possible de décider si la propriété considérée est satisfaite ou pas et utiliser alors l'algorithme de décision. On est alors dans le cadre d'une recherche d'attaques (les travaux cités plus haut à propos de la recherche d'attaques font d'ailleurs appel à des outils de *model-checking*). La seconde possibilité consiste à construire une approximation du modèle, préservant les attaques qui pourraient éventuellement être présentes dans le modèle de départ, et sur laquelle la propriété soit décidable. [Bol97] est un exemple typique de cette approche et c'est celle que nous avons choisi de suivre dans cette thèse. Les approximations en question sont réalisées au moyen de techniques d'interprétation abstraite et on conclut donc ensuite au moyen de techniques de *model-checking*. Avant de décrire ces deux composants essentiels de notre approche, signalons que l'étude des protocoles cryptographiques ne se limite pas à la pratique de techniques de vérification. De nombreux travaux, de nature plus théorique, ont pour but de proposer des modèles [AG99, FHG99], étudier les relations entre modèles existants [Aba00, Cor02], justifier les hypothèses simplificatrices des modèles [SMC00, HLS00, Bla02, CLC03] ou encore prouver des résultats d'impossibilité [EY80, CMSS03].

Le but premier de l'interprétation abstraite est d'énoncer des résultats portant sur les exécutions réelles d'un programme, sans pour autant l'exécuter. Le principe est d'utiliser pour cela des exécutions « non-standard » en remplaçant les valeurs qui interviennent dans les calculs et affectations du programme par d'autres valeurs (dites abstraites), plus simples à manipuler [JN95]. Selon cet article, l'idée d'exécuter le programme sur des valeurs abstraites remonte à 1965 [Nau65]. Des techniques, largement utilisées par ailleurs, peuvent tout à fait être considérées comme des techniques d'interprétation abstraite (preuve par neuf et règle des signes en arithmétique [CC92a], typage en informatique [Cou97]). Le cadre des correspondances de Galois, proposé par P. et R. Cousot [CC77, CC92a], est suffisamment général pour englober virtuellement toute technique basée sur l'interprétation abstraite. En particulier, [CGL94, DGG97, HMMR00] utilisent des abstractions pour étudier des systèmes de transitions et entrent tout à fait dans ce cadre, mais il est également possible d'y retrouver des concepts qui ne se réclament *a priori* pas de l'interprétation abstraite, par exemple des notions d'équivalences observationnelles [BT96, HS03]. Une correspondance de Galois est définie au moyen de deux domaines, celui des valeurs concrètes et celui des valeurs abstraites, chacun étant ordonné par une relation d'ordre (dite de précision). On doit également disposer d'une fonction (dite d'abstraction) du domaine concret vers le domaine abstrait ainsi qu'un application en sens inverse (dite de concrétisation), compatibles (en un certain sens) avec les relations d'ordre. Pour obtenir la contrepartie abstraite d'un élément du domaine concret, on peut lui appliquer la fonction d'abstraction, mais toute valeur moins précise doit également convenir. On peut alors répondre aux questions qui se posent en pratique, par exemple :

– si on dispose d'une fonction prenant en entrée la description syntaxique d'un programme et retournant sa sémantique, comment en calculer une abstraction (si possible, la plus précise) ?
– si on dispose d'une fonction entre deux domaines concrets ainsi qu'une abstraction de chaque domaine, comment trouver une contrepartie abstraite à cette fonction ?

Pour notre part, nous nous référerons peu à cette théorie dans cette thèse, mais toutes les abstractions que nous proposerons entrent, bien entendu, dans ce cadre. Nous nous contenterons de proposer des modèles abstraits en montrant qu'ils constituent des approximations correctes des modèles de départ. Nous travaillerons ensuite sur les modèles abstraits au moyen de techniques de *model-checking*.

Le principe du *model-checking* pour la vérification de systèmes de transitions a tout d'abord été proposé dans [CES86] et [QS81]. Plus généralement, le *model-checking* consiste à utiliser un algorithme permettant de décider la relation de satisfaction entre les modèles et leurs propriétés. Souvent, ces modèles sont des systèmes de transitions particuliers et leurs propriétés sont décrites sous formes de logiques temporelles [SBB+99, CGP99], mais, dans cette thèse, nous utiliserons le terme *model-cheking* dans son sens le plus large : les propriétés considérées sur les modèles abstraits seront décidables au moyen d'un algorithme (parfois naïf) que nous utiliserons pour conclure. Il n'y a que dans la deuxième partie que nous utiliserons un « véritable » *model-checker*. Mis à part la mise en pratique de cette technique, les travaux relatifs au *model-checking* comprennent la recherche de modèles pertinents et décidable (ou la description de sous-classes décidables dans des classes de modèles plus vastes), la recherche de langages de propriétés intéressants (là aussi décidables) et l'étude (complexité théorique et pratique) des algorithmes de décision. Ceci comprend aussi l'étude de représentations symboliques, dont le *model-checking* est un grand consommateur.

Comme évoqué plus haut, nous allons ici nous intéresser à la vérification de protocoles cryptographiques, au moyen de techniques d'abstraction et de *model-checking*, en organisant notre étude suivant trois types de propriétés de ces protocoles. Le premier type englobe les propriété d'authentification et de secret. On appelle ces propriétés des propriétés de traces, car il s'agit de propriétés des traces des exécutions du protocole. Les propriétés du deuxième type sont appelées propriétés de jeu et englobent l'équité et la non-répudiation (la terminologie choisie provient du fait que la sémantique des protocoles sera décrite, dans cette partie, en termes de jeux). Le troisième type est celui des propriétés d'opacité, qui généralise l'anonymat. Nous suivrons, dans chacune de ces trois parties, toujours plus ou moins la même approche. Nous commencerons par une introduction spécifique au type de propriétés considéré qui nous permettra de motiver notre étude, présenter les travaux afférents et décrire syntaxiquement la classe des protocoles associés à ces propriétés. Nous présenterons ensuite la classe des modèles qui seront utilisés pour définir la sémantique de ces protocoles (que nous décrirons précisément) et nous montrerons également, au moyen d'exemples, comment utiliser des propriétés de ces modèles pour représenter des propriétés des protocoles. Nous présenterons alors un cadre permettant de réaliser des abstractions sur ces modèles et nous l'appliquerons au cas des protocoles cryptographiques. Nous montrerons enfin, le plus souvent au moyen d'exemples et d'implémentations, comment utiliser ces abstractions pour vérifier, de manière effective, les propriétés considérées. Ces trois parties seront précédées de deux chapitres, en quelque sorte transversaux. Le premier décrit les protocoles cryptographiques de manière plus formelle que ce qui a été fait ici en donnant, en particulier, une syntaxe permettant de les décrire. Le second propose un cadre général pour la vérification au moyen de techniques d'abstractions dans lequel entreront les travaux que nous présenterons dans cette thèse.

Chapitre 1

La syntaxe

1.1 Un exemple de protocole

Le protocole cryptographique suivant est probablement l'un des plus connus. Il a été proposé par R. Needham et M. Schroeder en 1978 [NS78] et son but est d'établir une communication sûre entre A et B (nous allons expliquer comment dans un instant) :

Needham-Schroeder :

$$\begin{array}{llcll} 1. & A & \rightarrow & S & : \langle A, B, N_A \rangle \\ 2. & S & \rightarrow & A & : \{\langle N_A, B, K, \{\langle K, A \rangle\}_{K_{BS}} \rangle\}_{K_{AS}} \\ 3. & A & \rightarrow & B & : \{\langle K, A \rangle\}_{K_{BS}} \\ 4. & B & \rightarrow & A & : \{N_B\}_K \\ 5. & A & \rightarrow & B & : \{N_B - 1\}_K \end{array}$$

A, B et S sont appelés les principaux du protocole. On suppose ici que les principaux A et B partagent une clé symétrique avec le serveur S (ces clés sont notées K_{AS} et K_{BS}). Afin d'entamer une communication avec B, A fait appel à S pour que ce dernier produise une nouvelle clé (symétrique elle aussi, notée K) qui sera consacrée à cette communication entre A et B. Cette étape est réalisée par les échanges de messages 1 et 2 entre A et S. Lors du premier échange de messages, A fait savoir à S qu'il désire communiquer avec B, c'est pourquoi il lui envoie les deux identités en question, ainsi qu'un nonce (on appelle ainsi un nombre aléatoire, créé pour l'occasion et qui peut raisonnablement être supposé différent de tous les autres nonces qui peuvent être créés) qui sert à distinguer cette session des autres. La réponse de S à A est chiffrée avec la clé symétrique commune à A et S. Elle contient, en plus du nonce envoyé par A et de l'identité de B, la clé K nouvellement créée, ainsi qu'un message à destination de B, chiffré avec la clé symétrique commune à B et S et que A ne peut par conséquent pas déchiffrer. Il envoie donc ce message à B qui apprend ainsi que A désire communiquer avec lui et lui propose pour cela la clé K. Les deux dernières étapes du protocole visent à permettre à A et B de vérifier qu'ils sont bien d'accord sur la clé K. Pour cela, B produit un challenge pour A constitué d'un nonce N_B chiffré avec K, à charge pour A de lui renvoyer $N_B - 1$ également chiffré avec K. Un des buts de ce protocole est d'assurer que la clé K reste connue seulement de A, B et S.

Le protocole de Needham et Schroeder présenté ici est donc décrit à la fois par une notation semi-formelle et un court texte descriptif. Si on supprime ce texte, pour ne garder que la partie la plus formelle, on se rend compte que beaucoup de points ne sont pas précisés. Citons à titre d'exemple :

- à quels moments les principaux créent-ils des nonces ?
- à quels moments les principaux effectuent-ils des vérifications ?
- quelle est la nature des clés utilisées ?
- jusqu'à quel point les principaux décomposent-ils les messages reçus ?
- ...

Nous avons donc besoin d'une syntaxe formelle permettant de décrire précisément les protocoles.

1.2 Messages et processus

Afin d'analyser précisément les protocoles cryptographiques et leur propriétés, et s'inspirant du π-calcul proposé par R. Milner, J. Parrow et D. Walker [MPW92a, MPW92b], M. Abadi et A. Gordon proposent en 1997 le spi-calcul [AG99]. Il s'agit d'un calcul de processus (incluant des primitives cryptographiques) muni de notions d'équivalences observationnelles permettant d'écrire les protocoles cryptographiques et de définir leurs propriétés. La syntaxe que nous allons présenter ici et utiliser dans la suite est inspirée d'une des versions du spi-calcul qui se trouvent dans [MPW92a, MPW92b].

On suppose que l'on dispose au départ de deux ensembles Name et Var dont les éléments sont appelés respectivement des noms et des variables. Les noms pourront être utilisés indifféremment comme identités, clés, nonces ou tout autre type de données atomiques dont on pourrait avoir besoin. En particulier, tout nom pourra être utilisé comme une clé symétrique pour chiffrer les messages. Comme on aura également besoin de clés asymétriques, on conviendra qu'à tout nom $n \in \mathrm{Name}$ est associé un couple $(\mathrm{k}^{-1}(n), \mathrm{k}^{+1}(n))$ qui représente la clé privée et la clé publique de n (inverses l'une de l'autre). De même, à tout couple de noms (n, m) est associée une clé $\mathrm{k}(n, m)$ qui représente une clé symétrique partagée par n et m. Ainsi, l'ensemble des clés est défini par la grammaire :

$$
\begin{array}{rcl}
k, k', \ldots \in \mathrm{Key} & ::= & nx \\
& | & \mathrm{k}(nx, nx') \\
& | & \mathrm{k}^{-1}(nx) \\
& | & \mathrm{k}^{+1}(nx)
\end{array}
$$

où nx (respectivement nx') désigne un élément de $\mathrm{Name} \cup \mathrm{Var}$. On définit naturellement une notion d'inverse sur les clés en convenant que si nx et nx' appartiennent à $\mathrm{Name} \cup \mathrm{Var}$, nx et $\mathrm{k}(nx, nx')$ sont leur propre inverse en tant que clés symétriques et $\mathrm{k}^{-1}(nx)$ et $\mathrm{k}^{+1}(nx)$ sont inverses l'une de l'autre. La clé inverse d'une clé k sera notée k^{-1}.

Les messages échangés dans le cadre des protocoles cryptographiques sont construits en appliquant aux messages atomiques (*i.e.* les clés, qui contiennent en particulier les noms et les variables) les opérations suivantes :

- formation de listes : il s'agit, à partir d'une suite de messages $m_1, \ldots, m_n$ de former la liste $\langle m_1, \ldots, m_n \rangle$. Nous nous contenterons plus simplement d'un constructeur de couples (on notera $\langle m_1, m_2 \rangle$ le couple constitué des messages m_1 et m_2) en convenant que par définition $\langle m_1, m_2, \ldots, m_n \rangle$ sera égal à $\langle m_1, \langle m_2, \langle \ldots, m_n \rangle \ldots \rangle \rangle$;
- chiffrement : m étant un message et k une clé, $\{m\}_k$ désignera le message m chiffré au moyen de la clé k ;

– hachage : on désignera par $\mathrm{h}(m)$ le résultat de l'application d'une fonction de hachage (appelée aussi fonction à sens unique) à m ;
– pour pouvoir représenter les entiers à l'intérieur des messages, il suffit de disposer d'un nom particulier $0 \in \mathrm{Name}$ et d'une notion de successeur. À cet effet, il serait possible de définir un message $\mathrm{succ}(m)$ pour chaque message m mais, suivant [Hüt02], on peut également décréter que le successeur de m sera le message $\{m\}_0$. Ainsi, un nom particulier $0 \in \mathrm{Name}$ suffit à définir une notion d'entiers.

En regroupant toutes ces constructions et remarques, on obtient la grammaire suivante définissant les messages :

$$\begin{array}{rcl} m, m', \ldots \in \mathrm{Msg} & ::= & k \\ & | & \langle m, m' \rangle \\ & | & \{m\}_k \\ & | & \mathrm{h}(m) \end{array}$$

Notons que ces constructions sont purement syntaxiques. Par exemple, le résultat du hachage d'un message m (qui est un terme) est simplement le terme $\mathrm{h}(m)$. Il est par contre évident que, bien que des messages comme $\{m\}_0$ et $\mathrm{h}(m)$ possèdent la même structure, ils ne se manipulent pas de la même manière. Donnons tout d'abord quelques explications informelles sur la manière dont on doit manipuler les messages :
– il est toujours possible de former un couple à partir de deux messages, de même qu'il est toujours possible d'extraire les composantes d'un couple donné ;
– pour former un message de la forme $\{m\}_k$, il faut connaître la clé k, alors que pour retrouver m à partir de $\{m\}_k$, il faut connaître l'inverse de k ;
– il est toujours possible à partir de m de former $\mathrm{h}(m)$, mais il est impossible de retrouver m à partir de $\mathrm{h}(m)$.

Ces quelques règles traduisent les hypothèses formulées par D. Dolev et A. Yao [DY83] et communément admises depuis pour l'analyse des protocoles cryptographiques. Pour les formuler précisément, on définit habituellement une relation binaire, notée $\Vdash$, entre ensembles de messages et messages au moyen des règles suivantes :

$$\frac{m \in s}{s \Vdash m}$$

$$\frac{s \Vdash \langle m, m' \rangle}{s \Vdash m} \qquad \frac{s \Vdash \langle m, m' \rangle}{s \Vdash m'} \qquad \frac{s \Vdash \{m\}_k \quad s \Vdash k^{-1}}{s \Vdash m}$$

$$\frac{s \Vdash m \quad s \Vdash k}{s \Vdash \{m\}_k} \qquad \frac{s \Vdash m \quad s \Vdash m'}{s \Vdash \langle m, m' \rangle} \qquad \frac{s \Vdash m}{s \Vdash \mathrm{h}(m)}$$

La première règle signifie que tout message appartenant à s peut être produit à partir de s. Les règles de la deuxième ligne sont appelées règles d'analyse. Elles précisent quand il est possible de décomposer un message construit (noter la condition $s \Vdash k^{-1}$ dans les hypothèses de la troisième règle de cette ligne). Les règles de synthèse (dernière ligne) décrivent quand il est possible d'appliquer des opérations sur les messages pour en former de nouveaux. $s \Vdash m$ signifie alors que, connaissant les messages contenus dans s, on peut former le message m. Suivant [Pau98], on peut remarquer que, si on note $\mathrm{analz}(s)$ (respectivement $\mathrm{synth}(s)$) l'ensemble des messages qui peuvent être obtenus à partir de s utilisant la première règle et les règles d'analyse (respectivement synthèse), alors on a :

$$\{m \in \mathrm{Msg} \mid s \Vdash m\} \quad = \quad \mathrm{synth}(\mathrm{analz}(s))$$

Autrement dit, si $s \Vdash m$, il existe une dérivation de ce résultat dans laquelle aucune application d'une règle de synthèse ne précède l'application d'une règle d'analyse.

Venons-en maintenant aux processus proprement dits décrivant les protocoles. Nous aurons besoin de constructions pour réaliser les tâches suivantes :

– émettre et recevoir des messages : comme dans le π-calcul, il est prévu que ces envois et réceptions puissent se faire sur différents canaux, identifiés simplement par des noms. L'émission du message m sur le canal n, suivie du processus P sera notée $\overline{n}\langle m\rangle.P$ et la réception d'un message x sur le canal n suivie du processus P (qui utilisera la variable x pour faire référence à ce message) sera notée $n(x).P$;
– créer des noms frais (pour pouvoir modéliser la création de nonces par exemple), au moyen de la construction $(\nu n).P$;
– composition en parallèle de processus et choix non-déterministe (notés respectivement $P \parallel Q$ et $P+Q$) ;
– décomposer un message qui est un couple en ses composantes, extraire le message en clair d'un message chiffré et faire des tests. Nous regroupons toutes ces opérations dans une seule construction case m of $m'.P$ dont le rôle consiste informellement à instancier m' en m et à exécuter P (lequel pourra faire référence aux variables présentes dans m', qui auront été instanciées) ;
– une notion de test d'égalité représentée par une construction notée $[m' = m''].P$ qui exécute le processus P uniquement si les messages m' et m'' sont égaux (nous remplacerons parfois le test d'égalité par d'autres conditions, suivant les besoins) ;
– un processus qui ne fait rien, noté 0.

L'ensemble des processus est donc défini par la grammaire suivante :

$$
\begin{array}{rcl}
P, Q, \ldots \in \mathrm{Proc} & ::= & \overline{n}\langle m\rangle.P \\
& | & n(x).P \\
& | & (\nu n).P \\
& | & P \parallel Q \\
& | & P + Q \\
& | & \text{case } m \text{ of } m'.P \\
& | & [m' = m''].P \\
& | & 0
\end{array}
$$

Signalons que, par rapport au spi-calcul de [AG99], nous avons laissé de côté la réplication des processus (ainsi, si on désire prendre en compte les sessions multiples, il faudra le gérer explicitement dans la sémantique) et la possibilité pour un processus de boucler.

Exemple : Le protocole présenté au début de ce chapitre fait intervenir trois principaux. Nous modélisons chacun de ces principaux par un processus (nous avons laissé des variables libres, A, B et S, qui permettront d'instancier ces processus de plusieurs manières différentes si on cherche à décrire plusieurs sessions) :

$$\begin{array}{rcl}
P_A & \hat{=} & (\nu n).\overline{c}\langle\langle A, \langle B, n\rangle\rangle\rangle. \\
& & c(x).\text{case } x \text{ of } \{\langle n, \langle B, \langle k, x'\rangle\rangle\rangle\}_{\mathrm{k}(A,S)}. \\
& & \overline{c}\langle x'\rangle. \\
& & c(y).\text{case } y \text{ of } \{y'\}_k. \\
& & \overline{c}\langle\{\{y'\}_0\}_k\rangle.0 \\
P_B & \hat{=} & c(x).\text{case } x \text{ of } \{\langle k, a\rangle\}_{\mathrm{k}(B,S)}. \\
& & (\nu n).\overline{c}\langle\{n\}_k\rangle. \\
& & c(y).\text{case } y \text{ of } \{\{n\}_0\}_k.0 \\
P_S & \hat{=} & c(x).\text{case } x \text{ of } \langle a, \langle b, n\rangle\rangle. \\
& & (\nu k).\overline{c}\langle\{\langle n, \langle b, \langle k, \{\langle k, a\rangle\}_{\mathrm{k}(b,S)}\rangle\rangle\rangle\}_{\mathrm{k}(a,S)}\rangle.0
\end{array}$$

On voit clairement apparaître ici les différentes étapes de déchiffrement, décomposition des couples que doivent effecteur les principaux, ainsi que les créations de nonces et les tests. Remarquons qu'ici on a fixé un canal c qui est utilisé pour toutes les communications et que la décrémentation par A du nonce reçu de B à la fin du protocole a été remplacée par une incrémentation. □

Terminons cette présentation par quelques remarques. Tout d'abord les constructions $(\nu n).P$ et case m of $m'.P$ agissent comme des lieurs. Nous dirons ainsi que n est lié dans $(\nu n).P$ et que les variables libres de m' sont liées dans case m of $m'.P$. Il est ainsi possible de définir de manière naturelle l'ensemble des noms liés (respectivement des noms libres, des variables liées, des variables libres) qui apparaissent dans un processus P, noté $\mathrm{bn}(P)$ (respectivement $\mathrm{fn}(P)$, $\mathrm{bv}(P)$, $\mathrm{fv}(P)$). Le nom n utilisé dans $(\nu n).P$ peut tout à fait être remplacé par un autre nom n' à condition toutefois que n' ne soit pas un nom libre de P et que ce renommage soit répercuté dans P. Il en est de même pour les variables liées. Un tel renommage s'appelle un α-renommage et deux processus sont dits α-équivalents lorsqu'il est possible de passer de l'un à l'autre en effectuant des α-renommages. Dans la suite, les processus utilisés seront toujours considérés modulo α-équivalence. Soient $m_1, \ldots, m_n$ des messages et $x_1, \ldots, x_n$ des variables distinctes. Si P est un processus dont les variables liées sont distinctes des variables $x_1, \ldots, x_n$ ainsi que des variables apparaissant dans $m_1, \ldots, m_n$ et dont les noms liés sont distincts des noms apparaissant dans $m_1, \ldots, m_n$, on note $P[m_1/x_1, \ldots, m_n/x_n]$ le processus P dans lequel chaque occurrence de x_i a été remplacée par m_i. Si maintenant P est un processus quelconque, on définit $P[m_1/x_1, \ldots, m_n/x_n]$ comme étant égal à $P'[m_1/x_1, \ldots, m_n/x_n]$ où P' est un processus α-équivalent à P, choisi de sorte que les conditions précédentes soient satisfaites (ceci constitue une bonne définition uniquement modulo α-équivalence). Notons qu'une telle substitution ne peut se faire qu'à la condition que les messages qui sont substitués à des variables qui apparaissent dans des clés soient des noms.

1.3 Dans la suite

Maintenant que nous disposons d'une syntaxe, il reste à donner une sémantique aux processus ainsi qu'aux propriétés attendues puis expliquer comment raisonner par abstraction. Pour chaque classe de propriétés considérée, on définira précisément quel sous-langage de notre langage de processus on considère (puisqu'il s'avère que suivant les propriétés désirées, les protocoles proposés présentent des formes différentes) et quelle est la sémantique qui est donnée à ces protocoles.

Chapitre 2

Vérification et abstractions

Nous dessinons ici un cadre général dans lequel pourront s'inscrire les travaux qui seront présentés dans cette thèse. Ce cadre est celui de la vérification de propriétés de systèmes, en combinant des techniques de *model-checking* et d'abstraction (nous nous référons à [SBB⁺99, CGP99] pour ce qui concerne la vérification en général et le *model-checking* en particulier). Nous appellerons problème de vérification la donnée d'un système et d'une propriété de ce système (dont on attend, en principe, qu'elle soit satisfaite dans le système en question). Notons que les problèmes de vérification que l'on étudie appartiennent toujours au monde dit « réel », par opposition au monde « mathématique » dans lequel il faut se placer pour pouvoir travailler de manière formelle. Ceci a deux conséquences majeures pour celui qui cherche à vérifier formellement qu'un système satisfait une propriété. La première est qu'il faut toujours passer par une étape de modélisation pour passer du monde réel au monde mathématique. La seconde est qu'il est impossible d'énoncer quoi que se soit de formel à propos du monde réel. En particulier, il est impossible de montrer formellement que la modélisation choisie est correcte. Pour notre part, nous utiliserons des modélisations adaptées de modélisations classiques qui se trouvent dans la littérature. La question de la pertinence de la modélisation se trouve ainsi en partie résolue par le recours à ce fonds commun, bien accepté depuis plusieurs années. Nous décrivons, dans le reste de chapitre, comment se présentent précisément les étapes de modélisation, d'abstraction et de vérification.

2.1 Modélisation

Comme expliqué plus haut, avant de pouvoir vérifier formellement qu'un système satisfait une propriété, il faut au préalable modéliser ce problème de vérification. Cette modélisation sera donnée, en général, sous la forme d'un couple $(\mathcal{M}, \varphi) \in \mathfrak{M} \times E$, où $\mathfrak{M}$ est appelée la classe des modèles et E est l'ensemble des énoncés sur ces modèles. $\mathfrak{M}$ et E sont reliés au moyen d'une relation binaire $\models \subseteq \mathfrak{M} \times E$ et on dit qu'un modèle $\mathcal{M} \in \mathfrak{M}$ satisfait un énoncé $\varphi \in E$ si $\mathcal{M} \models \varphi$. On dit alors que le système satisfait la propriété (mais, rappelons-le, ceci ne peut pas être une définition formelle) si sa modélisation $(\mathcal{M}, \varphi)$ est telle que $\mathcal{M} \models \varphi$.

Exemple : Comme on l'a indiqué précédemment, on va dans la suite se focaliser sur trois types de propriétés. À chacun de ces types correspondra une modélisation particulière. Décrivons succinctement quelles seront ces modélisations, pour fixer les idées sur ce qui vient d'être dit :

– en ce qui concerne les propriétés de traces, la sémantique d'un protocole cryptographique sera donnée sous la forme d'un modèle de la logique du premier ordre et les énoncés seront

simplement des formules du premier ordre ;
- pour les propriétés de jeux, les protocoles seront modélisés par des systèmes de transitions alternés et les propriétés seront des formules de la logique temporelle alternée ;
- finalement, dans le cas des propriétés d'opacité, nous utiliserons des ensembles munis d'une notion d'observateur pour représenter les protocoles et les propriétés seront des prédicats unaires caractérisant les observations possibles.

$\square$

Les éléments de E sont souvent de nature syntaxique, ce qui signifie seulement qu'ils sont issus d'une construction formelle, sans qu'il leur soit attaché de signification particulière (ceci ne sera pas vrai dans le cas des propriétés d'opacité, mais nous n'en tiendrons pas compte dans cette première approche générale). L'avantage des objets syntaxiques est qu'ils sont très faciles à manipuler, contrairement aux objets de nature sémantique (les éléments de $\mathfrak{M}$). Définir l'élément $\mathcal{M}$ associé au système considéré peut ainsi s'avérer fastidieux. On préfère donc définir un autre ensemble de nature syntaxique M, muni d'une fonction :

$$\begin{array}{rccc} [\![]\!] : & M & \to & \mathfrak{M} \\ & m & \mapsto & [\![m]\!] \end{array}$$

On dit que M est un langage de spécification pour $\mathfrak{M}$. Cet ensemble permet de définir simplement des éléments de $\mathfrak{M}$ au moyen d'éléments de M et la modélisation d'un problème de vérification peut alors se donner sous la forme d'un couple (m, φ) où $m \in M$ et $\varphi \in E$. Ce couple représente $([\![m]\!], \varphi)$. Notons que, dans la littérature, la présence de M est rarement mise en valeur. Si nous insistons ici sur ce point, c'est parce que nous tirerons parti de la présence explicite de M pour construire des abstractions (voir à ce sujet la section suivante).

Exemple : Explicitons ce point en expliquant comment choisir M suivant les différentes modélisations qui seront choisies par la suite pour étudier les protocoles cryptographiques :
- dans le cas des propriétés de traces, nous décrirons les modèles de la logique du premier ordre au moyen d'un langage de spécification algébrique, Casl [ABK$^+$02, Cof] ;
- pour les propriétés de jeux, nous utiliserons une adaptation du langage de commandes gardées présenté dans [HMMR00] pour décrire les systèmes de transitions alternés ;
- les modèles utilisés dans le cas des propriétés d'opacité seront décrits au moyen d'un sous-ensemble de l'ensemble de processus Proc présenté au chapitre précédent.

$\square$

La phase de modélisation consiste donc à associer au système et à la propriété considérés deux objets de nature syntaxique. Néanmoins, dans le cadre de notre étude sur les protocoles cryptographiques, le langage M décrira des objets de nature complètement différente (des modèles de la logique du premier ordre, des systèmes de transition, etc.) de celle des objets étudiés (les protocoles). On préfère donc en général procéder de la manière suivante : on considère un autre langage L, dont les constructions sont adaptées à décrire les systèmes considérés (dans le cadre de cette thèse, les éléments de L seront des processus décrivant le fonctionnement des protocoles). On suppose ensuite les systèmes étudiés décrits sous la forme d'éléments de L, et, pour modéliser un système, il suffit de construire une fonction :

$$\begin{array}{rccc} [] : & L & \to & M \\ & l & \mapsto & [l] \end{array}$$

L'élément $l \in L$, représentant le protocole étudié, est appelé la spécification du protocole et $[l]$ est la spécification de la sémantique associée à ce protocole. On prendra garde à ne pas confondre la fonction de L vers $\mathfrak{M}$ obtenue par composition de $[] : L \to M$ et $[\![]\!] : M \to \mathfrak{M}$ (cet objet, de nature formelle, représente la sémantique de L dans $\mathfrak{M}$) et la correspondance entre systèmes (du monde réel) et éléments de $\mathfrak{M}$ qui correspond à la modélisation des problèmes de vérification. Pour nous, L sera toujours un sous-ensemble de Proc (adapté à la description du type de protocoles qui nous intéresse, pour une classe de propriétés fixée).

Le problème qui nous intéresse est donc de savoir si un système, décrit par $l \in L$, satisfait une propriété, décrite par $\varphi \in E$, autrement dit, savoir si $[\![[l]]\!] \models \varphi$ (on notera de manière plus concise $l \models \varphi$). Répondre à cette question au moyen de techniques de *model-checking* consisterait à utiliser un algorithme de décision pour la relation de satisfaction. Malheureusement, la relation de satisfaction considérée dans le cadre de la modélisation d'un protocole cryptographique est, en général, indécidable. Afin de pouvoir malgré tout obtenir une réponse de manière automatique, on renonce à l'exactitude de cette réponse et on raisonne de manière approchée. Bien entendu, la réponse doit rester sûre, en un certain sens. Les abstractions permettent de mettre en place un tel raisonnent, approché mais correct.

2.2 Des approximations correctes

Avant de placer les abstractions dans le cadre général qui précède, voyons comment il est possible, en pratique, de définir de telles abstractions. Contentons nous d'illustrer ceci par un exemple quelque peu fictif, qui sera suivi par une brève présentation des abstractions qui seront utilisées par la suite.

Exemple : Imaginons que les modèles considérés soient des systèmes de transitions de la forme :

$$\mathcal{M} \ \hat{=}\ (Q, q_0, \to, \mathcal{E})$$

où Q est l'ensemble des états (éventuellement infini), $q_0 \in Q$ est l'état initial, $\to \subseteq Q \times Q$ est la relation de transition et $\mathcal{E} \subseteq Q$ est un ensemble d'états d'erreur. L'ensemble de propriétés considéré est le singleton $\{\mathrm{ok}\}$ et on dit que $\mathcal{M} \models \mathrm{ok}$ si, et seulement si, quel que soit $q \in Q$ atteignable depuis q_0, on a $q \notin \mathcal{E}$. Avec un ensemble d'états infini et une relation de transition arbitraire, savoir si $\mathcal{M} \models \mathrm{ok}$ est naturellement indécidable. Prenons maintenant pour abstraction de $\mathcal{M}$ un système de transitions :

$$\mathcal{M}^\alpha \ \hat{=}\ (Q', q'_0, \to, \mathcal{E}')$$

relié à $\mathcal{M}$ au moyen d'une fonction $f : Q \to Q'$ et satisfaisant les conditions suivantes :
- Q' est fini ;
- $q'_0 = f(q_0)$;
- si $q_1 \to q_2$ (dans $\mathcal{M}$), alors $f(q_1) \to f(q_2)$ (dans $\mathcal{M}^\alpha$) ;
- si $q \in \mathcal{E}$, alors $f(q) \in \mathcal{E}'$.

Pour un état $q \in Q$, atteignable dans $\mathcal{M}$, il est alors clair que $f(q)$ est atteignable dans $\mathcal{M}^\alpha$. Ainsi, si il y a un état q, atteignable dans $\mathcal{M}$ et tel que $q \in \mathcal{E}$, alors $f(q)$ est atteignable dans $\mathcal{M}^\alpha$ et $f(q) \in \mathcal{E}'$. Par contraposée, on a montré que :

$$\mathcal{M}^\alpha \models \mathrm{ok} \ \Rightarrow\ \mathcal{M} \models \mathrm{ok}$$

Cette abstraction est donc correcte (au sens où ce qui est prouvé sur le modèle abstrait reste vrai sur le modèle de départ). Pour qu'elle soit également utile, il faut choisir Q' et f de manière pertinente, mais c'est un autre problème (qui ne peut être résolu de manière efficace qu'avec une connaissance plus approfondie de la structure de $\mathcal{M}$). Nous n'irons pas plus loin dans cet exemple (on pourra, pour plus de détails, consulter [DGG97, CGL94] dont nous nous sommes inspirés). $\square$

Voyons maintenant à quels types d'abstractions nous allons nous intéresser dans la suite :
- en ce qui concerne les propriétés de traces, nous choisirons d'abstraire les modèles de la logique du premier ordre par des modèles finis ;
- pour les propriétés de jeux, nous abstrairons les systèmes de transitions alternés par des systèmes légèrement différents, suivant une construction qui ne sera pas sans rappeler l'exemple qui précède ;
- pour les propriétés d'opacité, nous remplacerons les classes d'équivalence d'éléments indistinguables (pour l'observateur) par des sous-ensembles finis.

Précisons le cadre général qui nous permettra de traiter les problèmes de vérification de protocoles cryptographiques par abstraction. Pour procéder par abstraction, on doit tout d'abord posséder une classe $\mathfrak{M}^\alpha$ dont les éléments sont appelés des modèles abstraits, un ensemble E^α dont les éléments sont appelés des énoncés abstraits et une relation (dite de correction) :

$$\sqsubseteq\ \subseteq\ (\mathfrak{M} \times E) \times (\mathfrak{M}^\alpha \times E^\alpha)$$

Pour plus de simplicité, nous supposerons la relation de correction définie au moyen de deux relations, l'une entre $\mathfrak{M}$ et $\mathfrak{M}^\alpha$ et l'autre entre E et E^α. On suppose également qu'une relation de satisfaction est définie entre $\mathfrak{M}^\alpha$ et E^α, et on la note $\models$.

On dit qu'une telle abstraction est correcte si :

$$\forall(\mathcal{M}, \varphi).\ \forall(\mathcal{M}^\alpha, \varphi^\alpha).\ ((\mathcal{M}, \varphi) \sqsubseteq (\mathcal{M}^\alpha, \varphi^\alpha)\ \Rightarrow\ (\mathcal{M}^\alpha \models \varphi^\alpha\ \Rightarrow\ \mathcal{M} \models \varphi))$$

Ainsi, si on dispose d'une abstraction correcte et si on désire vérifier qu'un système, représenté par $l \in L$, satisfait une propriété, représentée par $\varphi \in E$, il suffit de montrer que $\mathcal{M}^\alpha \models \varphi^\alpha$ pour un couple $(\mathcal{M}^\alpha, \varphi^\alpha)$ tel que $([\![l]\!], \varphi) \sqsubseteq (\mathcal{M}^\alpha, \varphi^\alpha)$. Afin de savoir si raisonner par abstraction est plus intéressant que raisonner directement, on peut regarder si les critères suivants (qui caractérisent, de manière informelle, les abstractions utiles) sont satisfaits.

Le problème de décision de la satisfaction sur $\mathfrak{M}^\alpha \times E^\alpha$ doit être plus simple que sur $\mathfrak{M} \times E$. Le cas le plus intéressant est sans doute celui où la relation de satisfaction est décidable au niveau abstrait alors qu'elle ne l'est pas au niveau concret, mais il ne faut pas dédaigner le cas où les deux sont décidables, mais avec une meilleure complexité au niveau abstrait.

Étant donnés $l \in L$ et $\varphi \in E$, calculer un couple $(\mathcal{M}^\alpha, \varphi^\alpha)$ tel que $([\![l]\!], \varphi) \sqsubseteq (\mathcal{M}^\alpha, \varphi^\alpha)$ doit être facile. En effet, les modèles de $\mathfrak{M}$ seront en général infinis. Il ne faut donc pas espérer calculer tout d'abord $\mathcal{M} = [\![l]\!]$ puis ensuite utiliser $\mathcal{M}$ pour construire $\mathcal{M}^\alpha$, mais fournir un algorithme de calcul direct d'un $\mathcal{M}^\alpha \in \mathfrak{M}^\alpha$ qui convienne, en fonction de l. Notons que le calcul d'un φ^α qui convienne se fait en général très simplement, au moyen d'une fonction (de nature syntaxique) $\alpha : E \to E^\alpha$. L'idée est d'essayer de faire de même pour le calcul effectif d'un $\mathcal{M}^\alpha$. Rappelons que les éléments de $\mathfrak{M}$ sont décrits au moyen d'éléments de M et d'une fonction $[\![]\!] : M \to \mathfrak{M}$. Il est raisonnable de supposer qu'il en est de même pour $\mathfrak{M}^\alpha$, autrement dit il existe un ensemble (syntaxique) M^α et une fonction $[\![]\!] : M^\alpha \to \mathfrak{M}^\alpha$ qui permettent de décrire les éléments de $\mathfrak{M}^\alpha$ au moyen d'éléments de M^α. L'idée naturelle est alors de définir une fonction α de M vers M^α (de nature syntaxique, donc) satisfaisant la condition de correction suivante :

$$\forall m \in M.\ \forall \varphi \in E.\ [\![\alpha(m)]\!] \models \alpha(\varphi) \ \Rightarrow\ [\![m]\!] \models \varphi$$

Nous procéderons dans la suite de la manière suivante : on commencera par travailler sur les modèles afin d'établir une notion de correction $\sqsubseteq\ \subseteq\ \mathfrak{M} \times \mathfrak{M}^\alpha$, puis on expliquera comment obtenir une abstraction correcte à partir des descriptions syntaxiques.

Pour une propriété $\varphi \in E$, typique des systèmes étudiés, l'ensemble :

$$\{l \in L \mid [\![\alpha([l])]\!] \models \alpha(\varphi)\}$$

doit être un sous-ensemble relativement conséquent de $\{l \in L \mid l \models \varphi\}$. En effet, le fait que l'abstraction soit correcte nous assure seulement que, si $[\![\alpha([l])]\!] \models \alpha(\varphi)$, alors $l \models \varphi$, suivant la définition donnée plus haut. Par contre, dans le cas où $l \models \varphi$, il n'est pas du tout certain que ceci puisse être prouvé par abstraction (c'est précisément à ce niveau que se situe l'approximation). Ce critère exprime donc que l'abstraction se comporte bien vis-à-vis d'un nombre conséquent de systèmes. Attardons-nous un moment sur ce problème d'approximation. Étant donnés $l \in L$ et $\varphi \in E$, deux cas peuvent se présenter :

- $[\![\alpha([l])]\!] \models \alpha(\varphi)$: on sait alors que $l \models \varphi$ et tout se passe donc bien ;
- $[\![\alpha([l])]\!] \not\models \alpha(\varphi)$: il y a alors deux sous-cas, soit $l \not\models \varphi$, soit $l \models \varphi$ (et l'abstraction a introduit une trop grande imprécision). Le problème est que l'on ne peut *a priori* pas savoir dans quel sous-cas on se trouve. Dans la suite, nous verrons pour chaque abstraction donnée si il est possible d'apporter une réponse (qui ne pourra être que partielle) à ce problème (au moyen d'exemples).

La figure 2.2.1 récapitule le schéma général de vérification au moyen d'abstractions. Pour conclure, récapitulons les étapes essentielles du processus typique de vérification d'une classe de propriétés de sécurité de protocoles cryptographiques :

- la classe des propriétés considérées sera tout d'abord introduite au moyen d'exemples et nous définirons dans le même temps la classe des protocoles considérés, relativement à ces propriétés ;
- nous présenterons ensuite la classe des modèles et des énoncés choisis pour modéliser ces protocoles et ces propriétés, et nous décrirons précisément la sémantique des protocoles en utilisant ces modèles (et énoncés) ;
- nous donnerons ensuite la classe des modèles qui seront utilisés comme abstractions, en justifiant la correction puis en expliquant comment construire ces abstractions de manière syntaxique. Ceci nous permettra de donner la sémantique abstraite des protocoles et propriétés considérés ;
- nous terminerons en détaillant le processus de vérification effective sur quelques exemples et nous commenterons les résultats obtenus.

$$
\begin{tikzcd}
\textbf{Sys.} \arrow[r, dotted, "(1)"'] & L \arrow[r, "(3)"'] & M \arrow[r, "\alpha", "(7)"'] \arrow[d, "(4)"'] & M^{\alpha} \arrow[d, "(8)"] \\
 & & \mathfrak{M} \arrow[r, dash, "\sqsubseteq", "(5)"'] \arrow[d, dash, "\models"'] & \mathfrak{M}^{\alpha} \arrow[d, dash, "\models"] \\
\textbf{Prop.} \arrow[rr, dotted, "(2)"'] & & E \arrow[r, "\alpha", "(6)"'] & E^{\alpha}
\end{tikzcd}
$$

- (1) et (2) : représentations (formelles et syntaxiques) d'objets du monde réel ;
- (3) suivie de (4) : sémantique de L (dans $\mathfrak{M}$) ;
- (5) : notion de correction entre modèles concrets et modèles abstraits ;
- (6) : abstraction des énoncés ;
- (7) : abstraction des descriptions syntaxiques de modèles, respectant la correction (5) ;
- (3) suivie de (7) puis (8) : sémantique abstraite de L (dans $\mathfrak{M}^{\alpha}$).

FIG. 2.2.1 – *Vérification et abstractions*

Première partie

Propriétés de traces

Chapitre 3

Introduction

3.1 Un exemple

Considérons le protocole suivant[1], proposé par D. Denning et G. Sacco [DS81] :

Denning-Sacco :

$$\begin{array}{lllll} 1. & A & \rightarrow & S & : & \langle A, B\rangle \\ 2. & S & \rightarrow & A & : & \{\langle B, K_{AB}, T, \{\langle K_{AB}, A, T\rangle\}_{\mathrm{k}(B,S)}\rangle\}_{\mathrm{k}(A,S)} \\ 3. & A & \rightarrow & B & : & \{\langle K_{AB}, A, T\rangle\}_{\mathrm{k}(B,S)} \end{array}$$

Le but de ce protocole est d'établir une communication privée entre A et B, au moyen de la clé symétrique K_{AB}, nouvellement créée par le serveur de confiance S. Analysons ce protocole étape par étape. A commence par informer S qu'il désire établir une communication avec B, en lui envoyant pour cela un message constitué de son identité et de celle de B Recevant ce message, S crée une nouvelle clé symétrique K_{AB} ainsi qu'un cachet T (un tel objet, appelé *timestamp* dans la littérature, contient une référence à la date à laquelle il a été créé et possède généralement une certaine durée de validité, dépendant de l'implémentation du protocole. Nous considérerons plus simplement qu'il s'agit d'un nonce). Il envoie alors à A un message, chiffré avec $\mathrm{k}(A, S)$ et constitué de deux parties. La première, $\langle B, K_{AB}, T\rangle$, a pour but de fournir à A la nouvelle clé K_{AB} et contient le cachet T pour éviter que A accepte de cette manière un vieux message de S, contenant une clé qui pourrait avoir été compromise. La seconde partie du message, $\{\langle K_{AB}, A, T\rangle\}_{\mathrm{k}(B,S)}$, est à destination de B (A ne peut d'ailleurs pas la lire, puisqu'elle est chiffrée avec $\mathrm{k}(B, S)$ et se contente de la renvoyer à B dans la dernière étape du protocole). Ce message permet à B de prendre à son tour connaissance de K_{AB} et T.

Quelles propriétés attend-on de ce protocole ? Tout d'abord la clé K_{AB} doit bien entendu rester secrète (c'est à dire connue uniquement de A, B et S). Cette propriété n'est pas suffisante, car elle ne permet pas de s'assurer, par exemple, que, à la fin d'une session du protocole, en apparence jouée avec A, B parle effectivement à A (il s'agit là d'une propriété d'authentification).

Nous sommes donc en présence de deux types de propriétés. De plus, lorsque l'on cherche à formaliser ces propriétés, il apparaît que, pour chacune d'elles, plusieurs définitions sont

[1] Les protocoles cités dans cette thèse sont extraits de [Spo].

possibles. Pour le secret, par exemple, les deux définitions suivantes décrites dans [Aba00] sont communément employées :

- une variable libre x apparaissant dans la description d'un protocole P est dite secrète si l'instanciation de x par des messages clos quelconques conduit à des systèmes observationnellement équivalents ;
- un message clos m est secret dans un protocole P si, lors des exécutions de P, il est impossible à un intrus actif de produire m.

La première définition est adaptée dans le cas d'une sémantique des protocoles donnée par des processus munis d'une notion d'équivalence observationnelle tandis que la seconde convient bien pour une sémantique de protocoles en termes de traces, et c'est celle que nous utiliserons ici. En ce qui concerne l'authentification, les définitions possibles sont encore plus nombreuses (on pourra consulter [Low97b] à ce propos) :

- *(aliveness)* on dit qu'un protocole garantit à a jouant le rôle de A la présence de b si lorsque a termine une session du protocole, apparemment avec b, alors b a auparavant participé à une session du protocole ;
- *(weak agreement)* on dit qu'un protocole garantit à a jouant le rôle de A la présence forte de b si lorsque a termine une session du protocole, apparemment avec b, alors b a auparavant participé à une session du protocole, apparemment avec a.

On peut également vouloir imposer un accord sur certaines valeurs établies au cours du protocole (il pourrait s'agir de K_{AB} dans l'exemple qui précède) :

- *(non-injective agreement)* on dit qu'un protocole garantit à a jouant le rôle de A l'accord de b sur un ensemble de données $x_1, \ldots, x_n$ si lorsque a termine une session du protocole, en apparence avec b, alors b a auparavant participé à une session du protocole, en tant que B, en apparence avec a, et a et b ont instancié $x_1, \ldots, x_n$ par des valeurs égales ;
- *(agreement)* on dit qu'un protocole garantit à a jouant le rôle de A l'accord fort de b sur un ensemble de données $x_1, \ldots, x_n$ si pour toute session du protocole terminée par a , en apparence avec b, il existe une unique session du protocole dans laquelle b a participé, en tant que B, en apparence avec a, dans laquelle a et b ont instancié $x_1, \ldots, x_n$ par des valeurs égales.

À la différence des propriétés de secret qui précèdent, il est clair que ces définitions ne sont pas équivalentes mais qu'elles forment plutôt (comme annoncé dans [Low97b]) une hiérarchie. Ces propriétés sont également plus compliquées que les propriétés de secret, puisque pour les vérifier il faut mettre en correspondance divers évènements apparaissant lors d'une exécution de plusieurs sessions concurrentes et séquentielles du protocole.

3.2 Les approches existantes

Les propriétés de secret (et d'authentification, mais à moindre échelle) font partie des propriétés qui ont été les plus étudiées. La littérature sur ce sujet est extrêmement abondante et il serait illusoire d'en faire ici une présentation exhaustive. Aussi, nous nous contenterons d'une classification des approches existantes, jalonnée de repères bibliographiques.

Considérons tout d'abord la recherche d'attaques dans un protocole. Cette méthode consiste à modéliser les exécutions du protocole par un système de transitions (qui peut être fini [Mil90, Low97a, MMS97, SS98] ou infini (il pourra alors être représenté par un système de réécriture [JRV00], ou plus généralement un système de déduction logique [Mea96])) puis la négation de la propriété de sécurité considérée (une propriété de secret en général) est représentée par un but à atteindre et on utilise un outil (par exemple les *model-checkers* MURφ et FDR ont été employés dans ce but, ainsi que des systèmes logiques comme PROLOG et DaTaC) pour réaliser l'exploration du système. Un chemin permettant d'atteindre le but en question correspond à une attaque pour le protocole. Les avantages qui ont contribué au succès de cette méthode sont d'une part la simplicité de la modélisation (en général) et des techniques utilisées. De plus de nombreuses attaques [Low96, LR97] ont été découvertes de cette manière. Son inconvénient majeur est qu'elle ne permet pas de certifier qu'un protocole est correct, puisque pour mener à bien l'exploration, il faut imposer des limitations (nombre de sessions du protocole, nombre de participants, nonces, etc.).

À l'opposé, la preuve de protocoles permet de certifier qu'un protocole satisfait bien une propriété de sécurité. La méthode consiste à décrire dans un formalisme logique (qu'il s'agisse de logiques classiques [GK00, Wei99, Pau98], moins classiques comme la logique linéaire qui capture naturellement certains aspects essentiels des protocoles [CD99] ou adéquates [BAN89, AG97]) les exécutions possibles du protocole, ainsi que la propriété attendue, puis ensuite à montrer en utilisant un système de déduction adapté que la propriété est satisfaite. Notons que l'auteur d'une éventuelle démonstration peut être aidé dans cette démarche par des outils d'aide à la preuve comme ISABELLE ou SPASS, mais que cette preuve ne pourra pas, en toute généralité, être obtenue de manière complètement automatique car ces problèmes sont en général indécidables [CCM01, DLMS99]. Comparée à la recherche d'attaques, les formalismes employés sont parfois un peu plus compliqués et cette technique nécessite une plus grande interaction avec l'utilisateur.

Plusieurs voies ont ensuite été explorées pour rendre cette dernière méthode plus praticable. La première consiste à déterminer des classes de protocoles et de propriétés pour lesquelles le problème de la satisfaction est décidable et, lorsque c'est le cas, étudier sa complexité. On peut aussi remarquer que les systèmes qui sont utilisés pour la recherche d'attaques s'obtiennent à partir des systèmes utilisés pour la preuve en ajoutant des contraintes (nombre fini de sessions, de participants, etc.). Il est par conséquent intéressant de déterminer quelles contraintes permettent d'obtenir un système décidable [CCM01, RT01] (et là encore étudier la complexité du processus de décision) ou bien d'étudier quelles contraintes peuvent être ajoutées (ou supprimées) tout en gardant la validité du raisonnement [Low98, Sto01]. Cette approche a de plus l'avantage de rendre explicites les hypothèses qui sont parfois implicites dans les travaux de recherche d'attaques. Enfin, la dernière approche est celle que nous avons choisi de suivre : elle consiste à utiliser l'une des formalisations employées pour la preuve de protocoles mais à renoncer (de manière sûre) à l'exactitude de la réponse : on pourra faire confiance à une réponse « oui » à la question « tel protocole satisfait-il telle propriété ? », mais une réponse « non » signifiera seulement que l'algorithme a échoué dans sa tentative de prouver la propriété (sans savoir si c'est parce que, intrinsèquement, le protocole ne la satisfait pas, ou si ce sont les approximations qui sont trop fortes) [Bol97, Bol99, GL00, BLP03].

Remarquons qu'il est tout à fait possible de combiner plusieurs approches. Il peut par exemple être intéressant de combiner une approche consistant à imposer des contraintes (de type nombre fini de sessions par exemple) avec des abstractions (dans le but d'obtenir de meilleures performances). Notons aussi pour finir que recherche d'attaques et preuves de protocoles sont des approches complémentaires (comme il est dit dans [SS98]) présentant chacune avantages et inconvénients.

3.3 Notre approche

Dans l'approche que nous avons choisie, un protocole est modélisé par une structure de la logique du premier ordre. Si nous avons choisi un tel modèle, c'est parce que la logique du premier ordre fournit un cadre général, suffisamment expressif pour donner une sémantique aux protocoles (et, par conséquent, indécidable). Nous utiliserons donc des abstractions afin de savoir si un protocole, modélisé de cette façon, satisfait les propriétés attendues. Habituellement, les abstractions concernant les protocoles cryptographiques sont définies de manière *ad hoc* et leur correction est prouvée au cas par cas. Nous avons préféré proposer un cadre général permettant de définir des abstractions en logique du premier ordre qui seront correctes par construction. Nous exhiberons ensuite quelques abstractions utiles à l'intérieur de ce cadre. Ce travail prend ses sources dans [Bol97, Bol99] et a été publié dans [BB01] (excepté les différentes abstractions utiles au protocoles, qui n'étaient pas détaillées, et l'implémentation, qui est plus récente).

Le reste de cette partie est organisé de la manière suivante : nous allons tout d'abord rappeler quelles sont la syntaxe et la sémantique de la logique du premier ordre, puis nous utiliserons les modèles de cette logique pour donner une sémantique aux protocoles cryptographiques. Nous expliquerons ensuite comment réaliser des abstractions et nous donnerons quelques instanciations possibles dans le cadre des protocoles cryptographiques. Nous terminerons en énonçant et démontrant un résultat traduisant le caractère optimal des abstractions ainsi définies.

Signalons que, comme les protocoles cryptographiques sont des objets relativement compliqués, nous ne les utiliserons pas toujours lorsqu'il s'agira de fixer l'intuition à propos des définitions et des propriétés à venir. Nous utiliserons plutôt l'exemple plus simple consistant à définir, manipuler et énoncer des propriétés sur les listes d'entiers.

3.4 Les processus considérés

Il s'avère que, dans les protocoles cryptographiques dont le but est d'assurer le secret ou l'authentification, chaque principal doit en général effectuer une séquence d'actions qui consistent soit à créer de nouveaux noms et envoyer un message, soit recevoir un message et vérifier son format. Nous allons donc étudier des protocoles cryptographiques représentés par un processus d'un sous-ensemble de Proc, noté $\mathrm{Proc}_{\mathrm{Tr}}$ et défini au moyen des grammaires suivantes :

$$\begin{array}{rcl}
L, L', \ldots \in \mathrm{Proc}_{\mathrm{Lin}} & ::= & (\nu n_1).\ldots.(\nu n_k).\overline{c}\langle m\rangle.L \\
& \mid & c(x).\mathrm{case}\ x\ \mathrm{of}\ m.L \\
& \mid & 0 \\
P, P', \ldots \in \mathrm{Proc}_{\mathrm{Tr}} & ::= & L \\
& \mid & L \parallel P
\end{array}$$

(un processus $L \in \mathrm{Proc}_{\mathrm{Lin}}$ sera dit linéaire). Les conventions sont les suivantes : c est un nom fixé une fois pour toutes (autrement dit, toutes les émissions et réceptions de messages se font sur le même canal), $n_1, \ldots, n_k$ doivent apparaître dans m et x ne doit pas apparaître dans L (pour faire référence au message x dans L, il faut utiliser m).

Chapitre 4

Modélisation algébrique de protocoles

Comme expliqué dans le chapitre 2, afin de pouvoir vérifier si un protocole cryptographique satisfait une propriété donnée, il faut avant tout modéliser ce problème. Nous avons choisi dans cette partie d'effectuer cette étape de modélisation dans le cadre de la logique du premier ordre. Ce chapitre est par conséquent organisé de la manière suivante. On rappelle tout d'abord, dans le première section, comment sont définies la syntaxe et la sémantique de la logique du premier ordre. La deuxième section présente, au moyen d'exemples, le langage de spécifications algébriques CASL, qui permet de décrire (syntaxiquement et simplement) les modèles de la logique du premier ordre. Nous utilisons, dans la dernière partie, le langage CASL pour donner la sémantique d'un protocole cryptographique représenté par un processus P (élément de $\mathrm{Proc}_{\mathrm{Tr}}$) et nous donnons quelques exemples de propriétés de sécurité écrites en logique du premier ordre.

4.1 Logique du premier ordre

Nous allons définir ici de manière succincte la syntaxe et la sémantique de la logique du premier ordre. Pour une présentation plus détaillée, incluant les spécifications algébriques, le lecteur pourra se référer à [LEW96], ou encore à [AKKB99], plus proche de l'état de l'art. La définition d'un type de données (comme les listes d'entiers) comporte un certain nombre d'opérations (par exemple, prendre le successeur d'un entier, ou ajouter un entier à une liste). Ces opérations sont définies à la fois par leur nom et par leur arité (qui indique le type de leurs arguments ainsi que celui de leur valeur). La donnée d'un ensemble de sortes (qui représentent les noms des types possibles pour les arguments et valeurs) ainsi que d'un ensemble de symboles, chacun muni d'une arité constitue une signature algébrique.

Définition – 4.1.1

On appelle signature algébrique tout couple $\Theta = (S, \Omega)$ où S est un ensemble fini, dont les éléments sont appelés des sortes et Ω est un ensemble fini dont les éléments sont appelés symboles de fonctions. Un élément de Ω est un triplet de la forme :

$$(f, (s_1, \ldots, s_n), s_0)$$

où f est le nom de la fonction, $(s_1, \ldots, s_n) \in S^*$ représente à la fois le nombre des arguments et leur type et $s_0 \in S$ est le type de la valeur de retour.

Remarque : Un symbole de fonction $(f, (s_1, \ldots, s_n), s_0) \in \Omega$ sera noté plus simplement :

$$f : s_1 \times \ldots \times s_n \rightarrow s_0$$

Le cas $n = 0$ est tout à fait possible, on dit dans ce cas que la fonction est constante et on la note $f : s_0$. On ne supposera pas que les fonctions sont identifiées de manière unique par leur nom, autrement dit deux symboles de fonctions pourront posséder le même nom, ils seront considérés comme distincts du moment que leurs arités le sont (on fera cependant parfois référence à un symbole de fonction simplement par son nom, lorsque l'arité sera claire en fonction du contexte, ou bien importera peu). □

Exemple : La signature algébrique suivante est adaptée dans le cas des listes d'entiers :

$$\begin{array}{rcl} \Theta & \hat{=} & (S, \Omega) \\ S & \hat{=} & \{Nat, List\} \\ \Omega & \hat{=} & \{zero : Nat, \\ & & \ \ succ : Nat \rightarrow Nat, \\ & & \ \ nil : List, \\ & & \ \ cons : Nat \times List \rightarrow List\} \end{array}$$

Notons qu'aucune signification particulière n'est, pour l'instant, associée à ces symboles. Bien entendu, on pense à interpréter *zero* comme le zéro des entiers, *nil* comme la liste vide, etc. mais il serait tout à fait possible d'interpréter ces symboles d'une autre manière. □

Si on ajoute à une telle signature un ensemble de symboles de prédicats chacun muni d'une arité indiquant le type de leurs arguments, on obtient une signature du premier ordre.

Définition – 4.1.2

On appelle signature du premier ordre un triplet $\Sigma = (S, \Omega, \Pi)$ où (S, Ω) est une signature algébrique (on l'appelle signature algébrique sous-jacente à Σ et on la note $[\Sigma]$) et Π est un ensemble fini dont les éléments sont appelés symboles de prédicats. Un élément de Π est un couple :

$$(P, (s_1, \ldots, s_n))$$

où P est le nom du prédicat et $(s_1, \ldots, s_n) \in S^*$ est son arité.

Remarque : Un symbole de prédicat $(P, (s_1, \ldots, s_n)) \in \Pi$ sera noté plus simplement :

$$P : s_1 \times \ldots \times s_n$$

Le cas $n = 0$ est là encore possible. De même que pour les fonctions, on ne supposera pas que les prédicats sont identifiées de manière unique par leur nom, ce qui ne nous empêchera pas, là aussi, d'utiliser parfois ce dernier pour faire référence au symbole de prédicat. □

Exemple : En ce qui concerne les listes d'entiers, on peut vouloir étendre la signature algébrique précédente au moyen de l'ensemble de symboles de prédicats suivant, de manière à obtenir une signature du premier ordre :

$$\begin{array}{rcl} \Pi & \hat{=} & \{eq : Nat \times Nat, \\ & & \ \ inf : Nat \times Nat, \\ & & \ \ eq : List \times List, \\ & & \ \ in : Nat \times List\} \end{array}$$

Là encore, aucune signification particulière n'est, pour l'instant, associée à ces symboles (mais on peut penser que *eq* représentera l'égalité, *inf* la relation d'ordre sur les entiers et *in* le prédicat d'appartenance d'un entier à une liste). □

Partant d'un ensemble de variables et d'une signature algébrique (respectivement une signature du premier ordre), nous pouvons former des termes (respectivement des atomes). Les termes peuvent être vus comme des arbres, qui représentent des successions d'applications (formelles) de symboles de fonctions à partir de variables et de constantes. L'application formelle d'un symbole de prédicat à un ensemble de termes est un atome. Tout ceci doit bien sûr se faire en respectant les sortes. Pour cela, les variables doivent elles-aussi être associées à des sortes.

DÉFINITION – 4.1.3

> *Soit S un ensemble fini de sortes. Un ensemble de variables sur S est un ensemble (fini ou dénombrable) X dont les éléments sont des couples (x, s) où x est le nom de la variable et $s \in S$. Un tel couple sera noté $x : s$. Si $\Theta = (S, \Omega)$ est une signature algébrique et X un ensemble de variables sur S, l'ensemble des termes sur Θ et X, noté $\mathrm{T}_\Theta(X)$, est le plus petit ensemble satisfaisant les conditions suivantes :*
> - *si $x : s \in X$, x est un terme (de sorte s) ;*
> - *si $t_1, \ldots, t_n$ sont des termes de sortes respectives $s_1, \ldots, s_n$ et si $f : s_1 \times \ldots \times s_n \to s_0 \in \Omega$, alors $ft_1 \ldots t_n$ est un terme de sorte s_0.*
>
> *L'ensemble des atomes sur Σ et X, noté $\mathrm{At}_\Sigma(X)$, est défini au moyen de l'unique règle suivante :*
> - *si $t_1, \ldots, t_n$ sont des termes de sortes respectives $s_1, \ldots, s_n$ et si $P : s_1 \times \ldots \times s_n \in \Pi$, alors $Pt_1 \ldots t_n$ est un atome.*

Remarque : On omettra souvent de préciser à quel ensemble de sortes se réfère un ensemble de variables. De même on fera souvent référence à une variable $x : s$ par son seul nom x. Signalons enfin que, par abus de langage, on notera souvent $\mathrm{T}_\Sigma(X)$ à la place de $\mathrm{T}_{[\Sigma]}(X)$. □

Les atomes constituent les énoncés les plus simples qui puissent être formulés. Nous allons maintenant les utiliser comme briques de base pour construire des énoncés plus compliqués.

DÉFINITION – 4.1.4

> *Soient $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre et X un ensemble de variables. L'ensemble des énoncés (*sentences*) du premier ordre sur Σ et X, noté $\mathrm{Sen}(\Sigma)$ (on omettra la référence à X), est le plus petit ensemble satisfaisant les conditions suivantes :*
> - *si φ est un atome, alors φ est un énoncé ;*
> - *si φ est un énoncé, alors $\neg\varphi$ également ;*
> - *si φ, ψ sont des énoncés, alors $\varphi \wedge \psi$ et $\varphi \vee \psi$ également ;*
> - *si φ est un énoncé et $x : s \in X$, alors $\forall x : s.\ \varphi$ et $\exists x : s.\ \varphi$ également.*
>
> *Si φ et ψ sont deux énoncés, on posera :*
>
> $$\varphi \Rightarrow \psi \ \hat{=}\ (\neg\varphi) \vee \psi \quad \text{et} \quad \varphi \Leftrightarrow \psi \ \hat{=}\ (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$$

Exemple : Voici quelques énoncés à propos des listes d'entiers :

$$\begin{array}{l}
inf(x, succ(x)) \\
inf(succ(x), x) \\
\forall x : Nat.\ inf(x, succ(x)) \\
\forall x : Nat.\ \forall l : List.\ in(x, cons(x, l)) \\
\forall l : List.\ eq(l, nil) \vee (\exists x : Nat.\ in(x, l))
\end{array}$$

□

Signatures, termes et énoncés constituent la syntaxe de la logique du premier ordre. Nous allons maintenant en donner la sémantique. Pour ce faire, il suffit simplement d'associer à chaque sorte son domaine (qui sera simplement un ensemble), à chaque symbole de fonction (respectivement de prédicat) une fonction (respectivement une relation) définie sur les domaines correspondant à son arité.

Définition – 4.1.5

Soit $\Theta = (S, \Omega)$ une signature algébrique. On appelle Θ-algèbre $\mathcal{M}$ la donnée :
- *pour chaque $s \in S$, d'un ensemble non vide $\mathcal{M}_s$;*
- *pour chaque $f : s_1 \times \ldots \times s_n \to s_0 \in \Omega$, d'une fonction $f^{\mathcal{M}} : \mathcal{M}_{s_1} \times \ldots \times \mathcal{M}_{s_n} \to \mathcal{M}_{s_0}$.*

La classe des Θ-algèbres sera notée $\mathfrak{Alg}(\Theta)$. Si $\Sigma = (S, \Omega, \Pi)$ est une signature du premier ordre, on appelle Σ-structure $\mathcal{M}$ la donnée :
- *d'une $[\Sigma]$-algèbre notée $[\mathcal{M}]$;*
- *pour chaque $P : s_1 \times \ldots \times s_n \in \Pi$, d'une relation $P^{\mathcal{M}} \subseteq \mathcal{M}_{s_1} \times \ldots \times \mathcal{M}_{s_n}$.*

La classe des Σ-structures sera notée $\mathfrak{Str}(\Sigma)$.

Exemple : Définissons une structure $\mathcal{M}$ pour les listes d'entiers. On pose tout d'abord :

$$\begin{array}{rcl}
\mathcal{M}_{Nat} & \hat{=} & \mathbb{N} \\
\mathcal{M}_{List} & \hat{=} & \mathbb{N}^*
\end{array}$$

où l'étoile est prise au sens de la théorie des langages. On définit ensuite :

$$\begin{array}{rcl}
zero^{\mathcal{M}} & \hat{=} & 0 \\
succ^{\mathcal{M}} & : & \begin{array}{ccc} \mathbb{N} & \to & \mathbb{N} \\ n & \mapsto & n+1 \end{array} \\
nil^{\mathcal{M}} & \hat{=} & \varepsilon \\
cons^{\mathcal{M}} & : & \begin{array}{ccc} \mathbb{N} \times \mathbb{N}^* & \to & \mathbb{N}^* \\ (n, l) & \mapsto & n.l \end{array}
\end{array}$$

où ε représente le mot vide et . désigne la concaténation des mots. Ceci nous donne une $[\Sigma]$-algèbre, que l'on étend en une Σ-structure en posant :

$$\begin{array}{rcl}
eq^{\mathcal{M}} & \hat{=} & \{(n, n) \mid n \in \mathbb{N}\} \subseteq \mathbb{N} \times \mathbb{N} \\
inf^{\mathcal{M}} & \hat{=} & \{(n, n+k) \mid n, k \in \mathbb{N}\} \subseteq \mathbb{N} \times \mathbb{N} \\
eq^{\mathcal{M}} & \hat{=} & \{(l, l) \mid l \in \mathbb{N}^*\} \subseteq \mathbb{N}^* \times \mathbb{N}^* \\
in^{\mathcal{M}} & \hat{=} & \{(x, l'.x.l'') \mid x \in \mathbb{N}, l', l'' \in \mathbb{N}^*\} \subseteq \mathbb{N} \times \mathbb{N}^*
\end{array}$$

(ce choix est naturel pour traiter les listes d'entiers, mais rien n'empêche de considérer d'autres interprétations). □

Une algèbre permet, entre autres, de donner un sens aux termes (qui ne sont pour l'instant que des objets syntaxiques). De la même manière, une structure permet de donner un sens aux énoncés. Avant de définir ces interprétations, il faut préciser comment vont être interprétées les variables qui apparaissent dans les termes et les énoncés.

DÉFINITION – 4.1.6

Soient $\Theta = (S, \Omega)$ une signature algébrique, X un ensemble de variables et $\mathcal{M}$ une Θ-algèbre. On appelle valuation (de X vers $\mathcal{M}$) toute application $\nu : X \to \cup_{s \in S} \mathcal{M}_s$ telle que si $x : s \in X$, alors $\nu(x) \in \mathcal{M}_s$. Une telle valuation permet de définir, pour chaque terme $t \in \mathrm{T}_\Theta(X)$, son interprétation dans $\mathcal{M}$ (relativement à ν) qui sera notée $t\nu$. Cette définition se fait par induction sur t de la manière suivante :

- *si t est une variable x, alors $t\nu \hat{=} \nu(x)$;*
- *si $t = ft_1 \dots t_n$, alors $t\nu \hat{=} f^{\mathcal{M}}(t_1\nu, \dots, t_n\nu)$.*

Remarque : Il est clair que $t\nu$ ne dépend que des valeurs prises par ν sur les variables apparaissant dans t. Soient $x_1, \dots, x_n$ les variables apparaissant dans t et $m_1, \dots, m_n$ leur valeur par l'application ν. On notera parfois $t[m_1/x_1, \dots, m_n/x_n]$ à la place de $t\nu$. De plus, si $x : s \in X$ et $m \in \mathcal{M}_s$, $\nu[m/x]$ désignera la valuation qui envoie x sur m et qui est égale partout ailleurs à ν. □

DÉFINITION – 4.1.7

Soient $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre, X un ensemble de variables, $\mathcal{M}$ une Σ-structure et ν une valuation (de X vers $\mathcal{M}$). La relation de satisfaction dans $\mathcal{M}$ (relativement à ν) d'un énoncé $\varphi \in \mathrm{Sen}(\Sigma)$, notée $\mathcal{M}, \nu \models \varphi$, est définie par induction sur φ de la manière suivante :

$$
\begin{array}{rcl}
\mathcal{M}, \nu \models Pt_1 \dots t_n & \Leftrightarrow & (t_1\nu, \dots, t_n\nu) \in P^{\mathcal{M}} \\
\mathcal{M}, \nu \models \neg\varphi & \Leftrightarrow & \mathcal{M}, \nu \not\models \varphi \\
\mathcal{M}, \nu \models \varphi \wedge \psi & \Leftrightarrow & \mathcal{M}, \nu \models \varphi \textit{ et } \mathcal{M}, \nu \models \psi \\
\mathcal{M}, \nu \models \varphi \vee \psi & \Leftrightarrow & \mathcal{M}, \nu \models \varphi \textit{ ou } \mathcal{M}, \nu \models \psi \\
\mathcal{M}, \nu \models \forall x : s.\ \varphi & \Leftrightarrow & \textit{quel que soit } m \in \mathcal{M}_s \textit{ on a } \mathcal{M}, \nu[m/x] \models \varphi \\
\mathcal{M}, \nu \models \exists x : s.\ \varphi & \Leftrightarrow & \textit{il existe } m \in \mathcal{M}_s \textit{ tel que } \mathcal{M}, \nu[m/x] \models \varphi
\end{array}
$$

On dit que $\mathcal{M}$ satisfait φ (ou que φ est valide dans $\mathcal{M}$) et on note $\mathcal{M} \models \varphi$ si, et seulement si, $\mathcal{M}, \nu \models \varphi$ pour toute valuation ν.

Exemple : Dans la Σ-structure $\mathcal{M}$ associée aux listes d'entiers (exemple page 38), et reprenant les exemples d'énoncés donnés page 38, on a :

$$
\begin{array}{rcl}
\mathcal{M} & \models & \mathit{inf}(x, \mathit{succ}(x)) \\
\mathcal{M} & \not\models & \mathit{inf}(\mathit{succ}(x), x) \\
\mathcal{M} & \models & \forall x : \mathit{Nat}.\ \mathit{inf}(x, \mathit{succ}(x)) \\
\mathcal{M} & \models & \forall x : \mathit{Nat}.\ \forall l : \mathit{List}.\ \mathit{in}(x, \mathit{cons}(x, l)) \\
\mathcal{M} & \models & \forall l : \mathit{List}.\ \mathit{eq}(l, \mathit{nil}) \vee (\exists x : \mathit{Nat}.\ \mathit{in}(x, l))
\end{array}
$$

□

Terminons cette partie par la définition de morphisme entre deux algèbres sur la même signature. Informellement, un morphisme est une application qui respecte les opérations de l'algèbre. C'est ce respect de la structure qui nous conduira à utiliser les morphismes comme fonctions d'abstraction.

Définition – 4.1.8

Soit $\Theta = (S, \Omega)$ une signature algébrique et $\mathcal{M}, \mathcal{N}$ deux Θ-algèbres. On appelle morphisme de $\mathcal{M}$ vers $\mathcal{N}$ toute famille d'applications :

$$\alpha \;=\; (\alpha_s)_{s \in S} \text{ avec } \alpha_s : \mathcal{M}_s \to \mathcal{N}_s$$

telle que, quel que soit $f : s_1 \times \ldots \times s_n \to s_0 \in \Omega$, quels que soient $m_1 \in \mathcal{M}_{s_1}, \ldots, m_n \in \mathcal{M}_{s_n}$, on ait :

$$\alpha_{s_0}(f^{\mathcal{M}}(m_1, \ldots, m_n)) \;=\; f^{\mathcal{N}}(\alpha_{s_1}(m_1), \ldots, \alpha_{s_n}(m_n))$$

Dans la suite, on notera α à la place de α_s.

Remarque : Il existe aussi une notion de morphisme entre deux structures $\mathcal{M}$ et $\mathcal{N}$ sur la même signature $\Sigma = (S, \Omega, \Pi)$. Il s'agit d'un morphisme α de $[\mathcal{M}]$ vers $[\mathcal{N}]$ tel que, quels que soient $P : s_1 \times \ldots \times s_n \in \Pi$ et $m_1 \in \mathcal{M}_{s_1}, \ldots, m_n \in \mathcal{M}_{s_n}$, on ait :

$$P^{\mathcal{M}}(m_1, \ldots, m_n) \;\Rightarrow\; P^{\mathcal{N}}(\alpha(m_1), \ldots, \alpha(m_n))$$

Cette définition n'apparaîtra pas dans la suite (sauf, en filigrane, dans la définition de modèle initial d'une spécification). En particulier, elle ne sera pas utilisée dans la définition des abstractions, où les seuls morphismes qui entrent en jeu sont des morphismes d'algèbres. □

4.2 Égalité et spécifications

Nous avons vu plus haut que, même dans le cas de la signature assez simple des listes d'entiers, définir une structure pouvait se révéler fastidieux. Nous allons expliquer ici comment décrire ces structures de manière plus succincte. Tout d'abord, on peut éviter de traiter explicitement le prédicat d'égalité au moyen de la définition suivante.

Définition – 4.2.1

Soit $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre. On note Σ^{Eq} la signature définie par :

$$\begin{array}{rcl}\Sigma^{\mathrm{Eq}} & \hat{=} & (S, \Omega, \Pi^{\mathrm{Eq}}) \\ \Pi^{\mathrm{Eq}} & \hat{=} & \Pi \cup \{\_\_ = \_\_ : s \times s \mid s \in S\}\end{array}$$

(la notation $\_\_ = \_\_$ sert simplement à indiquer que l'opérateur $=$ sera noté de manière infixe). On appelle Σ-structure avec égalité toute Σ^{Eq}-structure $\mathcal{M}$ telle que quel que soit $s \in S$ on ait :

$$=^{\mathcal{M}} \hat{=} \{(m, m) \mid m \in \mathcal{M}_s\}$$

et les énoncés du premier ordre avec égalité sur Σ et un ensemble X de variables sont simplement les éléments de $\mathrm{Sen}(\Sigma^{\mathrm{Eq}})$. On notera $\mathfrak{Str}^{\mathrm{Eq}}(\Sigma)$ (respectivement $\mathrm{Sen}^{\mathrm{Eq}}(\Sigma)$) les Σ-structures (respectivement les énoncés du premier ordre sur Σ et X) avec égalité. Notons que toute Σ-structure s'étend de manière unique n une Σ-structure avec égalité.

Cette définition nous permet donc d'écrire des signatures et des structures de manière plus simple, puisqu'il n'y a plus besoin de décrire explicitement l'égalité. Afin de définir des structures de manière encore plus concise, nous allons maintenant présenter quelques rudiments de spécifications algébriques, au moyen du langage Casl. Une spécification en Casl décrit une signature du premier ordre ainsi qu'un ensemble de structures avec égalité sur cette signature (ses modèles). Nous n'en définirons ni la syntaxe précise, ni la sémantique, qui sortiraient du cadre de cette thèse, et nous laissons au lecteur le soin de se reporter à [BM01, ABK+02] pour une introduction et [Cof] pour une description plus précise. Nous nous contenterons d'expliquer les constructions employées au moyen de quelques exemples et, dans la suite, nous n'utiliserons pas de nouvelle construction sans en donner en même temps la signification. Commençons avec la spécification suivante :

spec NatList =

  sorts *Nat*, *List*
  ops *0* : *Nat*;
    succ : *Nat* → *Nat*;
    [] : *List*;
    $\_\_ :: \_\_$: *Nat* × *List* → *List*
  preds $\_\_ \leq \_\_$: *Nat* × *Nat*;
    $\_\_ \in \_\_$: *Nat* × *List*

end

Cette spécification se contente de définir les sortes *Nat* et *List*, les symboles de fonctions 0, *succ*, [], $\_\_ :: \_\_$ et les symboles de prédicats $\_\_ \leq \_\_$ et $\_\_ \in \_\_$. Notons que nous nous sommes autorisés à employer des notations un peu plus parlantes, 0 pour le zéro des entiers, [] pour la constante désignant la liste vide et :: pour le constructeur de listes, qui sera noté de manière infixe, ainsi que $\leq$ pour la relation d'ordre sur les entiers et $\in$ pour l'appartenance. La signature du premier ordre associée est la signature Σ utilisée jusqu'à présent pour traiter les listes d'entiers, modulo ces quelques changements de notations et la disparition des prédicats d'égalité. Les modèles de cette spécification sont toutes les Σ-structures du premier ordre avec égalité. On peut abréger cette spécification en :

spec NATLIST =

types $Nat ::= 0 \mid succ(Nat)$;
$List ::= [] \mid \_\_ :: \_\_(Nat; List)$

preds $\_\_ \leq \_\_ : Nat \times Nat$;
$\_\_ \in \_\_ : Nat \times List$

end

La déclaration **types** permet de regrouper la déclaration d'une sorte et de ses constructeurs (*i.e.* des symboles de fonctions dont le type de la valeur de retour est la sorte en question). La première chose à faire pour restreindre cet ensemble de modèles jusqu'à obtenir la Σ-structure des listes d'entiers, est d'imposer que le domaine d'interprétation de *Nat* (respectivement *List*) soit l'ensemble des entiers (respectivement des suites finies d'entiers). Remarquons pour cela que les entiers ont la propriété d'être engendrés par 0 et le successeur et ce de manière libre (*i.e.* il n'y a pas d'équations entre des entiers définis par 0 et successeur autres que les équations triviales). Il en est d'ailleurs de même pour les listes. CASL propose une notation pour spécifier qu'une sorte est engendrée de manière libre par ces constructeurs, et c'est la suivante :

spec NATLIST =

free types $Nat ::= 0 \mid succ(Nat)$;
$List ::= [] \mid \_\_ :: \_\_(Nat; List)$

preds $\_\_ \leq \_\_ : Nat \times Nat$;
$\_\_ \in \_\_ : Nat \times List$

end

De cette manière, non seulement les domaines d'interprétation de *Nat* et *List* sont fixés, mais aussi l'interprétation des constructeurs. Les modèles de cette spécification sont les Σ-structures avec égalité, pour lesquelles la $[\Sigma]$-algèbre sous-jacente est la même que pour la Σ-structure des listes d'entiers. Il ne reste donc plus qu'à définir l'interprétation des symboles de prédicats. Pour cela, introduisons des axiomes dans notre spécification :

spec NATLIST =

free types $Nat ::= 0 \mid succ(Nat)$;
$List ::= [] \mid \_\_ :: \_\_(Nat; List)$

preds $\_\_ \leq \_\_ : Nat \times Nat$;
$\_\_ \in \_\_ : Nat \times List$

$\forall x, y : Nat; l : List$

- $0 \leq x$
- $x \leq y \Rightarrow succ(x) \leq succ(y)$
- $x \in x :: l$
- $x \in l \Rightarrow x \in y :: l$

end

Par rapport à la spécification précédente, celle-ci n'a pour modèles que ceux qui satisfont ces quatre axiomes. Nous avons ainsi éliminé quelques interprétations des symboles de prédicats qui ne convenaient pas (par exemple les interprétations où les prédicats sont toujours faux), mais il est toujours possible de les interpréter de manière autre que dans la Σ-structure des listes d'entiers (par exemple, l'interprétation où les prédicats sont toujours vrais reste possible). L'idée est alors de choisir pour chaque symbole de prédicat une interprétation satisfaisant ces axiomes et qui soit de plus minimale. Ceci est possible dans notre cas particulier car les axiomes sont des clauses de Horn, et ils possèdent par conséquent un plus petit modèle. La syntaxe CASL permettant de faire ce choix est la suivante :

spec NATLIST = **free {**

  free types $Nat ::= 0 \mid succ(Nat);$
  $List ::= [] \mid \_\_ :: \_\_(Nat; List)$
  preds $\_\_ \leq \_\_ : Nat \times Nat;$
  $\_\_ \in \_\_ : Nat \times List$
  $\forall x, y : Nat; l : List$
  - $0 \leq x$
  - $x \leq y \Rightarrow succ(x) \leq succ(y)$
  - $x \in x :: l$
  - $x \in l \Rightarrow x \in y :: l$

} end

La construction **free** autour d'une spécification a pour but de choisir, parmi les modèles de cette spécification, le modèle initial (dans le cas qui nous intéresse, c'est le modèle pour lequel l'interprétation des prédicats est minimale, ce qui est cohérent avec la définition de morphisme entre structures donnée plus haut). Signalons que cette construction est redondante avec la déclaration **free types**, et on pourrait donc plus simplement écrire :

spec NATLIST =

  free types $Nat ::= 0 \mid succ(Nat);$
  $List ::= [] \mid \_\_ :: \_\_(Nat; List)$
  preds $\_\_ \leq \_\_ : Nat \times Nat;$
  $\_\_ \in \_\_ : Nat \times List$

then free {
  $\forall x, y : Nat; l : List$
  - $0 \leq x$
  - $x \leq y \Rightarrow succ(x) \leq succ(y)$
  - $x \in x :: l$
  - $x \in l \Rightarrow x \in y :: l$ **}**

end

L'emploi de listes d'éléments d'une sorte donnée est suffisamment fréquent pour justifier l'utilisation d'une spécification générique de liste, de la forme :

spec LIST[**sort** Elem] =

  free type $List[Elem] ::= [] \mid \_\_ :: \_\_(Elem; List[Elem])$
  pred $\_\_ \in \_\_ : Elem \times List$

then free {
  $\forall x, y : Elem; l : List$
  - $x \in x :: l$
  - $x \in l \Rightarrow x \in y :: l$ **}**

end

Cette spécification prend en paramètre une spécification déclarant une sorte *Elem* et construit, sur cette spécification, les listes d'éléments de sorte *Elem* (en définissant par la même occasion le prédicat d'appartenance). Si on dispose d'une spécification des entiers, par exemple :

spec NAT =
    free type $Nat ::= 0 \mid succ(Nat)$
    pred $\_\_ \leq \_\_ : Nat \times Nat$
then free {
    $\forall x, y : Nat$
    • $0 \leq x$
    • $x \leq y \Rightarrow succ(x) \leq succ(y)$ **}**
end

On peut alors obtenir la spécification NATLIST en instanciant LIST avec NAT :

spec NATLIST = LIST[NAT **fit** $Elem \mapsto Nat$]

La déclaration **fit** $Elem \mapsto Nat$ permet d'utiliser *Nat* à la place de *Elem*. La spécification NATLIST ainsi obtenue définit en particulier une sorte *List*[*Nat*] qui représente les listes d'entiers. La spécification LIST sera beaucoup utilisée dans la partie suivante. Nous aurons également l'occasion de voir plus loin d'autres exemples de constructions CASL ainsi que leur signification.

4.3 Sémantique concrète des protocoles

Expliquons maintenant comment donner à un protocole cryptographique, représenté par un processus $P = L_1 \parallel \ldots \parallel L_n \in \mathrm{Proc}_{\mathrm{Tr}}$, une sémantique sous la forme d'une structure du premier ordre. Nous allons décrire cette structure au moyen d'une spécification en CASL qui sera composée de plusieurs parties, celles-ci étant, pour la plupart, indépendantes du protocole considéré. Signalons que cette modélisation est fortement inspirée de [Pau98], avec quelques différences toutefois.

Pour décrire les noms, il nous faut, tout d'abord, une sorte *Name*, et l'interprétation de cette sorte doit satisfaire les contraintes suivantes :

- le domaine d'interprétation de *Name* doit être égal à Name ;
- ce domaine doit être partitionné en deux sous-domaines : les nonces créés seront pris dans le premier domaine alors que le second contiendra les constantes de l'exécution du protocole (les identités des participants, des données atomiques et le nom distingué 0).

Pour ce faire, nous allons supposer que *Name* est infini et dénombrable et définir deux types libres *Cst* et *Nonce* qui seront tous deux identiques à celui des entiers. On définira *Name* comme étant leur réunion disjointe, de sorte que son domaine d'interprétation soit infini dénombrable, donc en bijection avec Name. Cette réunion disjointe pourrait être définie comme un type libre de la forme :

free type $Name := of\_cst(Cst) \mid of\_nonce(Nonce)$

mais CASL propose une notation plus concise qui permet en plus d'éviter de déclarer des constructeurs explicites (techniquement, il s'agit de sous-sortage, mais on retiendra simplement que c'est une construction libre où les constructeurs sont implicites). Comme les opérations qui engendrent les sortes *Cst* et *Nonce* n'ont pas de sens pour les protocoles cryptographiques (excepté 0 : *Cst*), nous les cacherons dans la spécification au moyen du mot-clé **hide**. Nous aurons besoin, plus tard, de listes de nonces, mais nous utiliserons à ce moment là la spécification générique LIST définie plus haut. Il nous faudra également une constante i qui représentera l'identité de l'intrus [1] (on la définira comme étant égale à $next(0)$, elle sera ainsi différente de 0). On obtient alors la spécification suivante :

spec NAME =
  { free types $Cst := 0 \mid next(Cst);$
    $Nonce := 0 \mid next(Nonce);$
    $Name := sort\ Cst \mid sort\ Nonce$
   op $i : Cst \;=\; next(0)$ **}**
  hide $next : Cst \rightarrow Cst;\ 0 : Nonce;\ next : Nonce \rightarrow Nonce$
end

Les clés étant définies de manière syntaxique par une grammaire, à partir des noms, on ne sera pas surpris de les voir ici définies au moyen d'un type libre (nous utilisons ici le même raccourci que dans la spécification précédente pour éviter de déclarer explicitement un constructeur de type $Name \rightarrow Key$). L'opération consistant à prendre l'inverse d'une clé est définie au moyens d'axiomes décrivant quelle est sa valeur suivant la forme de son paramètre :

spec KEY = NAME
then **free type** $Key ::=$
    $sort\ Name \mid k(Name, Name) \mid k^{-1}(Name) \mid k^{+1}(Name)$
  op $\_\_^{-1} : Key \rightarrow Key$
  $\forall n, n' : Name$
  • $n^{-1} \;=\; n$
  • $k(n, n')^{-1} \;=\; k(n, n')$
  • $k^{-1}(n)^{-1} \;=\; k^{+1}(n)$
  • $k^{+1}(n)^{-1} \;=\; k^{-1}(n)$
end

Après les spécifications précédentes, la spécification suivante décrivant les messages ne présente rien de nouveau ni de surprenant (notons que nous aurons besoin, plus tard, de listes de messages).

spec MSG = KEY
then **free type** $Msg ::= sort\ Key \mid \langle \_\_, \_\_ \rangle(Msg; Msg) \mid$
    $\{\_\_\}_{\_\_}(Msg; Key) \mid h(Msg)$
end

Dans notre modèle, nous supposerons que l'unique canal de communication utilisé est contrôlé par une entité appelée intrus et qui représente à la fois tous les participants malhonnêtes ainsi que les défaillances possibles du réseau. Cet intrus est caractérisé par :

- son pouvoir sur le canal de communication. Dans notre cas, on supposera qu'il peut enregistrer tous les messages émis, les supprimer et éventuellement en ajouter de nouveaux qu'il aura calculé à partir de ceux qu'il a enregistré ;

[1] Il est habituel d'identifier tous les participants malhonnêtes, ainsi que les canaux de communication en une seule entité appelée *intrus*. C'est en quelque sorte déjà une abstraction.

– son pouvoir calculatoire, qui permet de décrire quels messages il peut former à partir d'une liste de messages donnés. Nous représenterons ce pouvoir au moyen d'un prédicat $\Vdash$ d'arité $List[Msg] \times Msg$ défini au moyen des règles suivantes (dans lesquelles m, m', m'' désignent des variables de sorte Msg, k une variable de sorte Key et l une variable de sorte $List[Msg]$) :
  – tout message enregistré peut être produit :

$$\varphi_{\text{incl}} \;\hat{=}\; m \in l \;\Rightarrow\; l \Vdash m$$

  – à partir de deux messages, il est possible de former le couple constitué de ces deux messages et, réciproquement, à partir d'un couple il est possible de retrouver chacune des deux composantes :

$$\begin{aligned} \varphi_{\text{pair}} &\;\hat{=}\; (l \Vdash m' \wedge l \Vdash m'') \;\Rightarrow\; l \Vdash \langle m', m'' \rangle \\ \varphi_{\text{proj1}} &\;\hat{=}\; l \Vdash \langle m', m'' \rangle \;\Rightarrow\; l \Vdash m' \\ \varphi_{\text{proj2}} &\;\hat{=}\; l \Vdash \langle m', m'' \rangle \;\Rightarrow\; l \Vdash m'' \end{aligned}$$

  – il est possible de chiffrer un message (connu) par une clé (connue), mais pour déchiffrer un message, il faut connaître l'inverse de la clé qui a servi à le chiffrer :

$$\begin{aligned} \varphi_{\text{decrypt}} &\;\hat{=}\; (l \Vdash \{m\}_k \wedge l \Vdash k^{-1}) \;\Rightarrow\; l \Vdash m \\ \varphi_{\text{crypt}} &\;\hat{=}\; (l \Vdash m \wedge l \Vdash k) \;\Rightarrow\; l \Vdash \{m\}_k \end{aligned}$$

  – il est possible de hacher un message connu, mais à partir d'un message haché il est impossible de retrouver le message de départ :

$$\varphi_{\text{hash}} \;\hat{=}\; l \Vdash m \;\Rightarrow\; l \Vdash \mathrm{h}(m)$$

spec $\textsc{Intr} = \textsc{List}[\textsc{Msg}$ **fit** $Elem \mapsto Msg]$
then free {
  pred $\_\_ \Vdash \_\_ : List[Msg] \times Msg$
  $\forall m, m', m'' : Msg; k : Key; l : List[Msg]$
  • φ_{incl}
  • φ_{pair}
  • φ_{proj1}
  • φ_{proj2}
  • φ_{crypt}
  • φ_{decrypt}
  • φ_{hash} **}**
end

Supposons maintenant que le protocole P considéré soit défini par des processus $L_1, \ldots, L_n \in \mathrm{Proc}_{\mathrm{Lin}}$. Nous allons représenter l'état courant de l'exécution de plusieurs instances de ce protocole au moyen d'un couple constitué de deux listes : la première représente les messages qui ont été émis et la seconde les nonces qui ont été créés. Un prédicat tr d'arité $List[Msg] \times List[Nonce]$ représentera les couples de listes qui correspondent à des exécutions du protocole. Nous allons définir ce prédicat de manière inductive. Pour chaque i entre 1 et n, on va associer à L_i un ensemble d'axiomes qui permettront de définir les traces du protocole. L_i est une suite d'actions qui consistent à émettre des messages (éventuellement après avoir créé de nouveaux noms) ou à en recevoir, lorsqu'ils vérifient certaines conditions. Supposons que L_i est constitué de k_i actions d'émission et considérons la j-ième. Cette action doit être précédée d'autres qui sont soit des émissions (on note $m_1, \ldots, m_r$ les messages émis lors de ces actions) soit des réceptions (on note $m'_1, \ldots, m'_s$ les motifs servant à filtrer les messages reçus). Le participant qui instancie ce processus pourra réaliser cette j-ième émission dès que les actions précédentes auront été réalisées, c'est à dire dès que :

– les messages $m_1, \ldots, m_r$ auront été émis, ce que l'on modélise par la condition :

$$\zeta_{i,j} \ \hat{=}\ m_1 \in l \wedge \ldots \wedge m_r \in l$$

(où l est une variable de sorte *List*[*Msg*] qui représente la liste des messages émis) ;

– les messages $m'_1, \ldots, m'_s$ auront été reçus, ce que l'on représente par la condition :

$$\eta_{i,j} \ \hat{=}\ l \Vdash m'_1 \wedge \ldots \wedge l \Vdash m'_s$$

Notons maintenant la j-ième émission de L_i sous la forme $(\nu n_1). \ \ldots \ (\nu n_q).\bar{c}\langle m\rangle. \ \ldots$ Pour pouvoir réaliser cette émission, il faut tout d'abord créer de nouveaux nonces, deux à deux distincts. Si on note $n_1, \ldots, n_q$ des variables de sorte *Nonce* et l' une variable de sorte *List*[*Nonce*] qui représente les nonces qui ont déjà été créés, la création de nouveaux nonces $n_1, \ldots, n_q$ est représentée par l'énoncé :

$$\xi_{i,j} \ \hat{=}\ n_1 \notin l' \wedge n_2 \notin n_1 :: l' \wedge \ldots \wedge n_q \notin n_{q-1} :: \ldots :: n_1 :: l'$$

Finalement, si (l, l') correspond à une exécution valide du protocole (autrement dit, si on a $tr(l, l')$) et si les énoncés $\zeta_{i,j}, \eta_{i,j}, \xi_{i,j}$ sont satisfaits, alors on peut réaliser la j-ième émission, tout en enregistrant les nonces qui viennent d'être créés (autrement dit, on a $tr(m :: l, n_q :: \ldots :: n_1 :: l')$).

Afin de donner la spécification de tr, il faut associer des sortes aux variables qui apparaissent dans la description du protocole. On commence par renommer les variables de manière à obtenir des processus α-équivalents tels que deux variables liées par des constructions distinctes soient distinctes. On procède ensuite de la manière suivante :

– aux variables libres du protocole (qui désignent en général des identités), $x_1, \ldots, x_p$, on associe la sorte *Cst* ;
– aux noms liés par une création de nonce, $n_1, \ldots, n_q$, on associe la sorte *Nonce* ;
– aux variables liées par un motif et qui peuvent être utilisées comme des clés, ou comme paramètre dans des clés, $y_1, \ldots, y_r$, on donne la sorte *Name* (sinon, on ne pourrait pas leur appliquer les constructeurs de clés) ;
– aux autres variables liées par un motif, $z_1, \ldots, z_s$, on donne la sorte *Msg*.

On obtient la spécification suivante (qui est la seule à être spécifique au protocole) :

spec $\quad$ TR$_P$ = INTR **and** LIST[NAME **fit** *Elem* $\mapsto$ *Nonce*]
then free {
  pred $\ tr : List[Msg] \times List[Nonce]$
  $\forall x_1, \ldots, x_p : Cst; n_1, \ldots, n_q : Nonce; y_1, \ldots, y_r : Name; z_1, \ldots, z_s : Msg$
  • $tr([], [])$
  %% Et pour $i \in \{1, \ldots, n\}$ et $j \in \{1, \ldots, k_i\}$:
  • $(tr(l, l') \wedge \zeta_{i,j} \wedge \eta_{i,j} \wedge \xi_{i,j}) \ \Rightarrow\ tr(m :: l, n_q :: \ldots :: n_1 :: l')$ **}**
end

Notons que, d'un certain point de vue, cette description est déjà une abstraction. En effet, de nombreux détails ont été omis : qui a envoyé quel message, à qui ce message était-il destiné, etc. Cependant, nous sommes sûrs de capturer dans le prédicat *tr* toutes les exécutions possibles. En ce sens, notre spécification est sûre. Un point qui n'a pas été abordé est le contenu de la connaissance initiale de l'intrus. Il est en fait possible de la coder en prenant comme axiome (pour la construction du prédicat *tr*) non pas $tr([], [])$ mais $tr(l, [])$ où l est la liste de messages initialement connus de l'intrus (en général : des clés publiques, ses propres clés, un nonce, etc.). Une autre possibilité est d'intégrer ces connaissances dans la définition du prédicat $\Vdash$. Les adaptations à faire sont de toutes façons mineures.

Exemple : Considérons le protocole de Denning et Sacco (page 31). Ce protocole est décrit au moyen des processus suivants (pour différencier les variables qui apparaissent dans des processus différents, tout en gardant des notations explicites, nous les avons préfixées par le nom du rôle représenté par le processus en question) :

$$\begin{array}{rcl}
L_A & \hat{=} & \overline{c}\langle\langle A.a, A.b\rangle\rangle. \\
 & & c(A.x).\mathrm{case}\ A.x\ \mathrm{of}\ \{\langle A.b, \langle A.k, \langle A.t, A.x'\rangle\rangle\rangle\}_{\mathrm{k}(A.a,A.s)}. \\
 & & \overline{c}\langle A.x'\rangle.0 \\
L_B & \hat{=} & c(B.x).\mathrm{case}\ B.x\ \mathrm{of}\ \{\langle B.k, \langle B.a, B.t\rangle\rangle\}_{\mathrm{k}(B.b,B.s)}.0 \\
L_S & \hat{=} & c(S.x).\mathrm{case}\ S.x\ \mathrm{of}\ \langle S.a, S.b\rangle. \\
 & & (\nu S.k).(\nu S.t). \\
 & & \quad \overline{c}\langle\{\langle S.b, \langle S.k, \langle S.t, \{\langle S.k, \langle S.a, S.t\rangle\rangle\}_{\mathrm{k}(S.b,S.s)}\rangle\rangle\rangle\}_{\mathrm{k}(S.a,S.s)}\rangle.0
\end{array}$$

Voyons comment chacun de ces processus intervient dans la définition du prédicat *tr* (pour plus de lisibilité, on note sous forme de règles d'inférence les énoncés définissant *tr*) :

- les variables définies par L_A sont :

  $$A.a : Cst; A.s : Name; A.b, A.k, A.t, A.x' : Msg$$

  et les deux règles associées à ce processus sont :

  $$\frac{tr(l, l')}{tr(\langle A.a, A.b\rangle :: l, l')}$$

  $$\frac{tr(l, l') \quad \langle A.a, A.b\rangle \in l \quad l \Vdash \{\langle A.b, \langle A.k, \langle A.t, A.x'\rangle\rangle\rangle\}_{\mathrm{k}(A.a,A.s)}}{tr(A.x' :: l, l')}$$

- les variables de L_B sont :

  $$B.b : Cst; B.s : Name; B.a, B.k, B.t : Msg$$

  mais il n'y a pas de règle associée, puisque B n'envoie jamais de message dans le cadre de ce protocole ;
- finalement, pour L_S on a les variables :

  $$S.s : Cst; S.a, S.b : Name; S.k, S.t : Nonce$$

  et la règle :

$$\frac{tr(l,l') \quad l \Vdash \langle S.a, S.b\rangle \quad S.k \notin l' \quad S.t \notin S.k :: l'}{tr(\{\langle S.b, \langle S.k, \langle S.t, \{\langle S.k, \langle S.a, S.t\rangle\rangle\}_{\mathrm{k}(S.b,S.s)}\rangle\rangle\rangle\}_{\mathrm{k}(S.a,S.s)} :: l, S.k :: S.t :: l')}$$

□

Remarque : Notons que, dans les spécifications qui précèdent :

- la première spécification, NAME, n'a, à isomorphisme près, qu'un seul modèle ;
- chaque extension de spécification ne change pas l'interprétation des symboles de la spécification de départ et définit de nouveaux symboles de manière unique (toujours à isomorphisme près).

On en déduit alors que TR_P ne possède, à isomorphisme près, qu'un seul modèle et on le notera $\mathcal{C}_P$. □

DÉFINITION – 4.3.1

Soit $P \in \mathrm{Proc}_{\mathrm{Tr}}$ un protocole. La sémantique de P (au sens des traces), notée $[\![P]\!]_{\mathrm{Tr}}$ est le couple $(\Sigma_P, \mathcal{C}_P)$ où Σ_P est la signature décrite par la spécification TR_P et $\mathcal{C}_P$ est l'unique structure du premier ordre avec égalité qui soit un modèle de cette spécification (à isomorphisme près).

Nous pouvons maintenant utiliser des énoncés $\varphi \in \mathrm{Sen}(\Sigma_P)$ pour représenter des propriétés du protocole P. Nous nous contenterons de donner quelques exemples, pour des propriétés typiques des protocoles.

Exemple : Considérons toujours le protocole de Denning et Sacco (page 31). Une propriété de secret intéressante à étudier est le secret de la clé fournie par le serveur, à la fin du protocole. Cette propriété de secret se déclinera en deux versions, une pour le rôle A et l'autre pour B (on pourrait également considérer S). Pour le rôle A, par exemple, cette propriété s'énonce en disant que si A a accompli une session complète du protocole (y compris le dernier envoi de message, car la clé pourrait être perdue à ce moment là) en ayant reçu à la fin une clé k, alors cette clé est secrète. Il faut bien entendu exclure les cas où A est l'intrus, ainsi que les cas où A est engagé dans une session avec l'intrus jouant le rôle de B ou S. Formellement, si on note $compl_A$ le fait que dans une exécution du protocole représentée par une liste de messages l, $A.a$ jouant le rôle de A ait accompli une session de P avec $A.b$ dans le rôle de B, $A.s$ dans le rôle de S, un nonce $A.t$, une clé $A.k$, et un message $A.x'$ reçu de $A.s$ et à envoyer à $A.b$, c'est à dire :

$$compl_A \ \hat{=}\ \langle A.a, A.b\rangle \in l \wedge l \Vdash \{\langle A.b, \langle A.k, \langle A.t, A.x'\rangle\rangle\rangle\}_{\mathrm{k}(A.a,A.s)} \wedge A.x' \in l$$

alors le secret de $A.k$ dans cette session s'écrit :

$$secret_A \ \hat{=}\ (tr(l,l') \wedge A.a \neq i \wedge A.b \neq i \wedge A.s \neq i \wedge compl_A) \ \Rightarrow\ \neg(l \Vdash A.k)$$

(en quantifiant universellement sur toutes les variables présentes). Le protocole garantit à $B.b$ jouant le rôle de B la présence forte de $B.a$ si, lorsque $B.b$ a accompli une session en jouant le rôle de B avec $B.a$ jouant le rôle de A, ce qui est représenté par l'énoncé :

$$compl_B \ \hat{=}\ l \Vdash \{\langle B.k, \langle B.a, B.t\rangle\rangle\}_{\mathrm{k}(B.b,B.s)}$$

alors $B.a$ a participé à une session du protocole avec $B.b$. Si on note cette dernière proposition sous la forme :

$$\begin{aligned} part_A \;\hat{=}\; \exists A.a : Cst.\ \exists A.s : Name.\ \exists A.b, A.k, A.t, A.x' : Msg.& \\ A.a = B.a \;\wedge\; A.b = B.b \;\wedge\; \langle A.a, A.b\rangle \in l& \\ \wedge\; l \Vdash \{\langle A.b, \langle A.k, \langle A.t, A.x'\rangle\rangle\rangle\}_{\mathrm{k}(A.a,A.s)}& \\ \wedge\; A.x' \in l& \end{aligned}$$

alors l'authentification forte s'écrit :

$$auth_B \;\hat{=}\; (tr(l,l') \wedge compl_B) \;\Rightarrow\; part_A$$

(en quantifiant universellement sur toutes les variables libres). Remarquons que :

- il n'y a plus de cas particulier pour l'intrus. En effet, un protocole peut tout à fait assurer une propriété d'authentification à l'intrus, mais il est impossible de lui assurer une propriété de secret (puisque par définition, être secret c'est être inconnu de l'intrus) ;
- dans la définition d'authentification forte, il est précisé que la participation de a doit avoir lieu avant que b ait terminé le protocole. Ceci est capturé dans notre formulation par le fait que l'énoncé doit être vrai quel que soit (l,l') avec $tr(l,l')$. En particulier, a doit avoir participé au protocole au moment même où b le termine, donc avant.

Nous dirons que le protocole P satisfait le secret de K_{AB} pour A (respectivement la présence forte de A pour B) si, et seulement si, $\mathcal{C}_P \models secret_A$ (respectivement $\mathcal{C}_P \models auth_B$). □

Nous avons associé, dans ce chapitre, une structure du premier ordre à chaque protocole P (élément de $\mathrm{Proc}_{\mathrm{Tr}}$) et nous avons expliqué comment utiliser des énoncés écrits dans le langage de la logique du premier ordre pour décrire des propriétés de sécurité de ces protocoles. Le problème consistant à savoir si un protocole satisfait une certaine propriété est donc modélisé par un problème de satisfaction d'un énoncé du premier ordre dans une structure. Nous allons montrer dans le chapitre suivant comment des abstractions permettent de répondre, au moins partiellement, à de telles questions.

Chapitre 5

Interprétation abstraite algébrique

Nous allons décrire maintenant notre approche générale permettant de définir des abstractions pour des problèmes de la forme $\mathcal{C} \models \varphi$ où $\mathcal{C}$ est une structure du premier ordre et φ un énoncé sur une signature donnée. Rappelons que $\mathcal{C}$ est constitué d'ensembles (respectivement de fonctions, de relations) interprétant les sortes (respectivement les symboles de fonctions, les symboles de prédicats) de Σ. Le travail consistant à définir des abstractions se divise donc en :

– définir l'abstraction des domaines d'interprétation des sortes de Σ dans $\mathcal{C}$;
– définir l'abstraction des fonctions de $\mathcal{C}$;
– définir l'abstraction des relations de $\mathcal{C}$;
– définir l'abstraction de l'énoncé φ.

En ce qui concerne les deux premières tâches, nous choisissons d'abstraire l'algèbre $[\mathcal{C}]$ (sous-jacente à $\mathcal{C}$) par une algèbre $\mathcal{B}$ sur la même signature $[\Sigma]$, ces deux algèbres étant reliées par un morphisme $\alpha : [\mathcal{C}] \to \mathcal{B}$. Afin de voir comment étendre cette algèbre en une structure du premier ordre, commençons par regarder un exemple.

Exemple : Considérons la spécification des entiers et listes d'entiers du chapitre précédent, avec seulement le prédicat d'appartenance :

spec NatList =

    free types $\mathit{Nat} ::= \mathit{0} \mid \mathit{succ}(\mathit{Nat})$;
                    $\mathit{List} ::= [] \mid \_\_ :: \_\_(\mathit{Nat}\ \mathit{List})$
    pred $\_\_ \in \_\_ : \mathit{Nat} \times \mathit{List}$

then free {
    $\forall x, y : \mathit{Nat}\ l : \mathit{List}$
    • $x \in x :: l$
    • $x \in l \Rightarrow x \in y :: l$ **}**

end

Cette spécification n'a, à isomorphisme près, qu'un seul modèle $\mathcal{C}$, où les interprétations des sortes, des symboles de fonctions et de prédicats se font de manière habituelle. Considérons maintenant la spécification suivante :

spec NatList' =

free { **types** $\mathit{Nat} ::= \mathit{0} \mid \mathit{succ}(\mathit{Nat})$;
                $\mathit{List} ::= [] \mid \_\_ :: \_\_(\mathit{Nat}\ \mathit{List})$

pred $\_\_ \in \_\_ : Nat \times List$

$\forall x, y : Nat\ l : List$

- $succ(succ(x)) \ = \ succ(x)$
- $x :: x :: l \ = \ x :: l$
- $x :: y :: l \ = \ y :: x :: l$
- $x \in x :: l$
- $x \in l \Rightarrow x \in y :: l$ **}**

end

Cette spécification également n'a, à isomorphisme près, qu'un seul modèle $\mathcal{A}$ défini de la manière suivante :

$$\begin{array}{rcl}
\mathcal{A}_{Nat} & \hat{=} & \{\texttt{nul}, \texttt{pos}\} \\
\mathcal{A}_{List} & \hat{=} & 2^{\{\texttt{nul},\texttt{pos}\}} \\
0^{\mathcal{A}} & \hat{=} & \texttt{nul} \\
succ^{\mathcal{A}}(\texttt{nul}) & \hat{=} & \texttt{pos} \\
succ^{\mathcal{A}}(\texttt{pos}) & \hat{=} & \texttt{pos} \\
\in^{\mathcal{A}} & & \text{est le prédicat d'appartenance classique.}
\end{array}$$

Il est clair qu'il existe un unique morphisme α de $[\mathcal{C}]$ vers $[\mathcal{A}]$ et il est défini de la manière suivante :

- un entier $n \in \mathcal{C}_{Nat}$ est envoyé sur `nul` s'il est égal à 0 et `pos` sinon ;
- une liste $l \in \mathcal{C}_{List}$ est envoyée sur l'ensemble de ses éléments (dont on a au préalable pris l'image dans $\{\texttt{nul}, \texttt{pos}\}$).

Maintenant, il est clair que si $n \in^{\mathcal{C}} l$, alors $\alpha(n) \in^{\mathcal{A}} \alpha(l)$. En prenant pour n et l des termes clos quelconques de sortes respectives *Nat* et *List*, on en déduit que :

$$\mathcal{A} \models n \notin l \ \Rightarrow \ \mathcal{C} \models n \notin l$$

La réciproque n'est évidemment pas vraie (considérer par exemple $n = succ(succ(0))$ et $l = succ(0) :: []$). On peut par contre considérer une propriété plus forte :

$$\mathcal{A} \models n \in l \wedge n = 0 \ \Rightarrow \ \mathcal{C} \models n \in l$$

□

Ceci suggère que l'abstraction d'un prédicat pourra prendre deux formes, l'une servira pour le valider et l'autre pour le réfuter. C'est ce que nous allons formaliser dans la suite, en étendant l'algèbre $[\mathcal{A}]$ reliée à $[\mathcal{C}]$ par un morphisme α en une structure du premier ordre relativement à une signature obtenue à partir de Σ en dédoublant les symboles de prédicats.

Notons que des idées semblables ont déjà été employées dans des travaux sur l'abstraction des systèmes réactifs [DGG97, CGL94, Mer97] dans le cadre de logiques temporelles. Nous formalisons ici cette approche dans le cadre différent de la logique du premier ordre.

Remarque : Si nous voulons prouver un énoncé de la forme $\varphi \hat{=} \exists x.\ \psi$ sachant que cet énoncé est vrai au niveau abstrait (*i.e.* $\mathcal{A} \models \varphi'$, φ' désignant l'abstraction de φ qui sera définie plus loin), il est naturel de vouloir utiliser un élément a d'un des domaines de $\mathcal{A}$ tel que $\mathcal{A}, [a/x] \models \psi'$ afin d'obtenir un élément c (d'un des domaines de $\mathcal{C}$) tel que $\mathcal{C}, [c/x] \models \psi$. Pour cela, nous supposerons que $[\mathcal{C}]$ et $[\mathcal{A}]$ ne sont pas seulement reliées par un morphisme arbitraire, mais par un morphisme surjectif. □

5.1 Abstraction des structures

DÉFINITION – 5.1.1

Soit $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre. La signature dédoublée $\Sigma^\bullet$ associée à Σ est définie par :

$$\Sigma^\bullet \quad \hat{=} \quad (S, \Omega, \Pi^\bullet)$$

où $\Pi^\bullet$ est la réunion disjointe de deux copies de Π que l'on notera :

$$\begin{aligned} \Pi^\bullet \quad \hat{=} \quad & \{P_\ominus : s_1 \times \ldots \times s_n \mid P : s_1 \times \ldots \times s_n \in \Pi\} \\ & \cup \{P_\oplus : s_1 \times \ldots \times s_n \mid P : s_1 \times \ldots \times s_n \in \Pi\} \end{aligned}$$

Partant d'un morphisme surjectif $\alpha : [\mathcal{C}] \to \mathcal{B}$ entre $[\Sigma]$-algèbres, nous prendrons donc comme abstraction de $\mathcal{C}$ une $\Sigma^\bullet$-structure $\mathcal{A}$ telle que $[\mathcal{A}] = \mathcal{B}$. Bien entendu, toutes les extensions de $\mathcal{B}$ ne conduisent pas à des abstractions pertinentes de $\mathcal{C}$. La définition suivante indique quand les interprétations $P_\ominus^\mathcal{A}$ et $P_\oplus^\mathcal{A}$ permettent respectivement de prouver et réfuter $P^\mathcal{C}$.

DÉFINITION – 5.1.2

Étant donnés une signature $\Sigma = (S, \Omega, \Pi)$, une Σ-structure $\mathcal{C}$, une $\Sigma^\bullet$-structure $\mathcal{A}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \to \mathcal{B}$, on dit que $\mathcal{A}$ est correcte relativement à $\mathcal{C}$ et α, et on note $\mathcal{C} \sqsubseteq_\alpha \mathcal{A}$, si, et seulement si, $[\mathcal{A}] = \mathcal{B}$ et quels que soient $P : s_1 \times \ldots \times s_n \in \Pi$ et $c_1 \in \mathcal{C}_{s_1}, \ldots, c_n \in \mathcal{C}_{s_n}$ on a :

$$\begin{aligned} & P_\ominus^\mathcal{A}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P^\mathcal{C}(c_1, \ldots, c_n) \\ \textit{et} \quad & P^\mathcal{C}(c_1, \ldots, c_n) \;\Rightarrow\; P_\oplus^\mathcal{A}(\alpha(c_1), \ldots, \alpha(c_n)) \end{aligned}$$

Remarque : Nous avons défini la correction au moyen des deux conditions :

$$\begin{aligned} & P_\ominus^\mathcal{A}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P^\mathcal{C}(c_1, \ldots, c_n) \\ \text{et} \quad & P^\mathcal{C}(c_1, \ldots, c_n) \;\Rightarrow\; P_\oplus^\mathcal{A}(\alpha(c_1), \ldots, \alpha(c_n)) \end{aligned}$$

La première recouvre ce que l'on entend habituellement par « correction » d'une abstraction : pouvoir déduire des résultats sur le modèle concret à partir de résultats sur le modèle abstrait. La deuxième condition pourrait alors sembler surprenante, puisqu'elle permet de passer du modèle concret au modèle abstrait. Mais il ne faut pas oublier que, dans le langage qui nous intéresse ici, à savoir le langage de la logique du premier ordre, nous disposons de négations. Si nous réécrivons la deuxième condition sous la forme :

$$\neg P_\oplus^\mathcal{A}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; \neg P^\mathcal{C}(c_1, \ldots, c_n)$$

on peut alors interpréter cette condition comme la correction vis-à-vis des interprétations des négations des symboles de prédicats. Autrement dit, les deux conditions données correspondent à la notion de correction usuelle, vis-à-vis des littéraux (*i.e.* les prédicats atomiques et leur négation). □

Il est en général possible d'étendre une $[\Sigma]$-algèbre $\mathcal{B}$ en plusieurs $\Sigma^\bullet$-structures correctes relativement à une Σ-structure $\mathcal{C}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \to \mathcal{B}$. Pour comparer ces différentes extensions, nous introduisons la notion de précision. Notons que, comme toutes ces extensions possèdent la même algèbre sous-jacente $\mathcal{B}$, la définition de la notion de précision concerne uniquement les interprétations des symboles de prédicats.

Définition – 5.1.3

Soient $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre et $\mathcal{A}, \mathcal{A}'$ deux $\Sigma^\bullet$-structures. On dit que $\mathcal{A}$ est plus précise que $\mathcal{A}'$, et on note $\mathcal{A} \sqsubseteq \mathcal{A}'$, si, et seulement si, $[\mathcal{A}] = [\mathcal{A}']$ et quels que soient $P : s_1 \times \ldots \times s_n \in \Pi$ et $a_1 \in \mathcal{A}_{s_1}, \ldots, a_n \in \mathcal{A}_{s_n}$ on a :

$$\begin{array}{rcl} P_\ominus^{\mathcal{A}'}(a_1, \ldots, a_n) & \Rightarrow & P_\ominus^{\mathcal{A}}(a_1, \ldots, a_n) \\ \text{et} \quad P_\oplus^{\mathcal{A}}(a_1, \ldots, a_n) & \Rightarrow & P_\oplus^{\mathcal{A}'}(a_1, \ldots, a_n) \end{array}$$

On obtient ainsi un ordre (partiel) sur $\mathfrak{Str}(\Sigma^\bullet)$.

Il est alors naturel de s'intéresser aux éléments de $\mathfrak{Str}(\Sigma^\bullet)$ qui sont corrects, relativement à $\mathcal{C}$ et un morphisme surjectif α, et minimaux pour la relation de précision.

Définition – 5.1.4

Étant donnés une signature du premier ordre $\Sigma = (S, \Omega, \Pi)$, une Σ-structure $\mathcal{C}$, une $[\Sigma]$-algèbre $\mathcal{B}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \to \mathcal{B}$, la $\Sigma^\bullet$-structure canonique associée à $\mathcal{C}$ et α, notée $\mathcal{C}^\alpha$, est définie de la manière suivante :

- $[\mathcal{C}^\alpha] \mathrel{\hat{=}} \mathcal{B}$;
- *quels que soient $P : s_1 \times \ldots \times s_n \in \Pi$ et $a_1 \in \mathcal{B}_{s_1}, \ldots, a_n \in \mathcal{B}_{s_n}$:*
  - *$P_\ominus^{\mathcal{C}^\alpha}(a_1, \ldots, a_n)$ est vrai si, et seulement si, quels que soient $c_1 \in \alpha^{-1}(a_1), \ldots, c_n \in \alpha^{-1}(c_n)$, on a $P^{\mathcal{C}}(c_1, \ldots, c_n)$,*
  - *$P_\oplus^{\mathcal{C}^\alpha}(a_1, \ldots, a_n)$ est vrai si, et seulement si, il existe $c_1 \in \alpha^{-1}(a_1), \ldots, c_n \in \alpha^{-1}(c_n)$ tels que $P^{\mathcal{C}}(c_1, \ldots, c_n)$.*

La proposition suivante permet de relier entre-elles les notions de précision, correction et abstraction canonique.

Proposition – 5.1.1

Soient Σ une signature du premier ordre, $\mathcal{C}$ une Σ-structure, $\mathcal{A}$ et $\mathcal{A}'$ deux $\Sigma^\bullet$-structures et un morphisme surjectif α défini sur $[\mathcal{C}]$. On a les résultats suivants :

1. *Si $\mathcal{C} \sqsubseteq_\alpha \mathcal{A}$ et $\mathcal{A} \sqsubseteq \mathcal{A}'$, alors $\mathcal{C} \sqsubseteq_\alpha \mathcal{A}'$;*
2. *$\mathcal{C} \sqsubseteq_\alpha \mathcal{A}$ si, et seulement si, $\mathcal{C}^\alpha \sqsubseteq \mathcal{A}$;*
3. *$\mathcal{C}^\alpha$ est la $\Sigma^\bullet$-structure la plus précise qui soit correcte relativement à $\mathcal{C}$ et α.*

Démonstration :

Pour le premier point, étant donnés $P : s_1 \times \ldots \times s_n \in \Pi$ et $c_1 \in \mathcal{C}_{s_1}, \ldots, c_n \in \mathcal{C}_{s_n}$, on a les chaînes d'implications :

$$\begin{array}{l} P_{\ominus}^{\mathcal{A}'}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P_{\ominus}^{\mathcal{A}}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P^{\mathcal{C}}(c_1, \ldots, c_n) \\ P^{\mathcal{C}}(c_1, \ldots, c_n) \;\Rightarrow\; P_{\oplus}^{\mathcal{A}}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P_{\oplus}^{\mathcal{A}'}(\alpha(c_1), \ldots, \alpha(c_n)) \end{array}$$

(en utilisant les définitions de correction et précision). On en déduit donc que $\mathcal{C} \sqsubseteq_{\alpha} \mathcal{A}$. En ce qui concerne le deuxième point, il suffit de remarquer que, pour $P : s_1 \times \ldots \times s_n \in \Pi$, on a équivalence entre :

$$\forall c_1 \in \mathcal{C}_{s_1}, \ldots, c_n \in \mathcal{C}_{s_n}.\ P_{\ominus}^{\mathcal{A}}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P^{\mathcal{C}}(c_1, \ldots, c_n)$$

et :

$$\begin{array}{l} \forall a_1 \in \mathcal{A}_{s_1}, \ldots, a_n \in \mathcal{A}_{s_n}.\ \forall c_1 \in \alpha^{-1}(a_1), \ldots, c_n \in \alpha^{-1}(a_n). \\ \qquad\qquad P_{\ominus}^{\mathcal{A}}(\alpha(c_1), \ldots, \alpha(c_n)) \;\Rightarrow\; P^{\mathcal{C}}(c_1, \ldots, c_n) \end{array}$$

et il en est de même avec $P_{\oplus}$. Le troisième point s'en déduit aisément. ■

Remarque : Lorsqu'une $\Sigma^{\bullet}$-structure $\mathcal{A}$ est correcte relativement à $\mathcal{C}$ et α, on a $P_{\ominus}^{\mathcal{A}} \subseteq P_{\oplus}^{\mathcal{A}}$ pour tout $P : s_1 \times \ldots \times s_n \in \Pi$. □

Étant donnés une signature Σ et une Σ-structure $\mathcal{C}$, nous pouvons utiliser une $\Sigma^{\bullet}$-structure $\mathcal{A}$ (correcte relativement à $\mathcal{C}$ et un morphisme surjectif α) de manière à prouver ou réfuter des énoncés atomiques. Nous allons étendre cette propriété à des énoncés plus généraux.

Définition – 5.1.5

À un énoncé φ sur une signature Σ, on associe un couple d'énoncés $(\varphi_{\ominus}, \varphi_{\oplus})$ sur la signature $\Sigma^{\bullet}$, définis par induction sur φ de la manière suivante :

- *pour $P : s_1 \times \ldots \times s_n \in \Pi$, $t_1 \in \mathrm{T}_{\Sigma}(X), \ldots, t_n \in \mathrm{T}_{\Sigma}(X)$ de sortes respectives $s_1, \ldots, s_n$, on pose :*

$$(Pt_1 \ldots t_n)_{\ominus} \;\hat{=}\; P_{\ominus}t_1 \ldots t_n \quad \textit{et} \quad (Pt_1 \ldots t_n)_{\oplus} \;\hat{=}\; P_{\oplus}t_1 \ldots t_n$$

- *pour $\varphi \in \mathrm{Sen}(\Sigma)$, on pose :*

$$(\neg\varphi)_{\ominus} \;\hat{=}\; \neg(\varphi_{\oplus}) \quad \textit{et} \quad (\neg\varphi)_{\oplus} \;\hat{=}\; \neg(\varphi_{\ominus})$$

- *pour $\varphi, \psi \in \mathrm{Sen}(\Sigma)$, on pose :*

$$\begin{array}{l} (\varphi \wedge \psi)_{\ominus} \;\hat{=}\; \varphi_{\ominus} \wedge \psi_{\ominus} \quad \textit{et} \quad (\varphi \wedge \psi)_{\oplus} \;\hat{=}\; \varphi_{\oplus} \wedge \psi_{\oplus} \\ (\varphi \vee \psi)_{\ominus} \;\hat{=}\; \varphi_{\ominus} \vee \psi_{\ominus} \quad \textit{et} \quad (\varphi \vee \psi)_{\oplus} \;\hat{=}\; \varphi_{\oplus} \vee \psi_{\oplus} \end{array}$$

- *pour $\varphi \in \mathrm{Sen}(\Sigma)$ et $x : s \in X$, on pose :*

$$\begin{array}{l} (\forall x : s.\ \varphi)_{\ominus} \;\hat{=}\; \forall x : s.\ (\varphi_{\ominus}) \quad \textit{et} \quad (\forall x : s.\ \varphi)_{\oplus} \;\hat{=}\; \forall x : s.\ (\varphi_{\oplus}) \\ (\exists x : s.\ \varphi)_{\ominus} \;\hat{=}\; \exists x : s.\ (\varphi_{\ominus}) \quad \textit{et} \quad (\exists x : s.\ \varphi)_{\oplus} \;\hat{=}\; \exists x : s.\ (\varphi_{\oplus}) \end{array}$$

Le résultat principal de ce chapitre est le suivant.

THÉORÈME – 5.1.1

Soient Σ une signature du premier ordre, $\mathcal{C}$ une Σ-structure, $\mathcal{A}$ une $\Sigma^\bullet$-structure et un morphisme surjectif $\alpha : [\mathcal{C}] \to [\mathcal{A}]$. On suppose $\mathcal{C} \sqsubseteq_\alpha \mathcal{A}$. Alors, quel que soit $\varphi \in \mathrm{Sen}(\Sigma)$, on a :

$$\begin{array}{rl} & \mathcal{A} \models \varphi_\ominus \;\Rightarrow\; \mathcal{C} \models \varphi \\ \textit{et} & \mathcal{C} \models \varphi \;\Rightarrow\; \mathcal{A} \models \varphi_\oplus \end{array}$$

Démonstration :

Commençons par montrer que, pour tout $\varphi \in \mathrm{Sen}(\Sigma)$ et pour toute valuation ν de X dans $\mathcal{C}$, on a :

$$\mathcal{A}, \alpha\nu \models \varphi_\ominus \;\Rightarrow\; \mathcal{C}, \nu \models \varphi \quad \text{et} \quad \mathcal{C}, \nu \models \varphi \;\Rightarrow\; \mathcal{A}, \alpha\nu \models \varphi_\oplus$$

(où $\alpha\nu$ désigne la valuation de X dans $\mathcal{A}$ obtenue en composant ν par α). On procède pour cela par induction sur φ (on peut sans perte de généralité supposer que les énoncés considérés ne contiennent aucune occurrence de $\vee$ ni $\exists$) :

- si φ est un énoncé atomique de la forme $Pt_1 \dots t_n$ et si $\mathcal{A}, \alpha\nu \models \varphi_\ominus$, alors on a $P_\ominus^{\mathcal{A}}(\alpha(t_1\nu), \dots, \alpha(t_n\nu))$. Donc, par correction de $\mathcal{A}$ par rapport à $\mathcal{C}$ et α, on en déduit que $P^{\mathcal{C}}(t_1\nu, \dots, t_n\nu)$, et par conséquent que $\mathcal{C}, \nu \models \varphi$. Si maintenant $\mathcal{C}, \nu \models \varphi$, alors on a $P^{\mathcal{C}}(t_1\nu, \dots, t_n\nu)$ et, par correction de $\mathcal{A}$ par rapport à $\mathcal{C}$ et α, on en déduit que $P_\oplus^{\mathcal{A}}(\alpha(t_1\nu), \dots, \alpha(t_n\nu))$ et par conséquent que $\mathcal{A}, \alpha\nu \models \varphi_\oplus$;
- si $\mathcal{A}, \alpha\nu \models (\varphi \wedge \psi)_\ominus$, alors $\mathcal{A}, \alpha\nu \models \varphi_\ominus$ et $\mathcal{A}, \alpha\nu \models \psi_\ominus$. Donc, par hypothèse d'induction, on a $\mathcal{C}, \nu \models \varphi$ et $\mathcal{C}, \nu \models \psi$, d'où l'on déduit que $\mathcal{C}, \nu \models \varphi \wedge \psi$. Si maintenant $\mathcal{C}, \nu \models \varphi \wedge \psi$, alors $\mathcal{C}, \nu \models \varphi$ et $\mathcal{C}, \nu \models \psi$. Par hypothèse d'induction on a donc $\mathcal{A}, \alpha\nu \models \varphi_\oplus$ et $\mathcal{A}, \alpha\nu \models \psi_\oplus$, d'où l'on déduit que $\mathcal{A}, \alpha\nu \models (\varphi \wedge \psi)_\oplus$;
- si $\mathcal{A}, \alpha\nu \models (\neg\varphi)_\ominus$, alors $\mathcal{A}, \alpha\nu \not\models \varphi_\oplus$ et, par hypothèse d'induction, $\mathcal{C}, \nu \not\models \varphi$. Donc $\mathcal{C}, \nu \models \neg\varphi$. Si maintenant $\mathcal{C}, \nu \models \neg\varphi$, alors $\mathcal{C}, \nu \not\models \varphi$, donc, par hypothèse d'induction, $\mathcal{A}, \alpha\nu \not\models \varphi_\ominus$ et $\mathcal{A}, \alpha\nu \models (\neg\varphi)_\oplus$;
- si $\mathcal{A}, \alpha\nu \models (\forall x : s.\ \varphi)_\ominus$ et si $c \in \mathcal{C}_s$, alors on a $\mathcal{A}, \alpha\nu[\alpha(c)/x] \models \varphi$. Mais, comme $\alpha\nu[\alpha(c)/x] = \alpha(\nu[c/x])$, on peut appliquer l'hypothèse d'induction, ce qui nous donne $\mathcal{C}, \nu[c/x] \models \varphi$ et ce, quel que soit $c \in \mathcal{C}_s$. Par conséquent $\mathcal{C}, \nu \models \forall x : s.\ \varphi$. Si maintenant $\mathcal{C}, \nu \models \forall x : s.\ \varphi$ et si $a \in \mathcal{A}_s$, on choisit $c \in \alpha^{-1}(a)$ (ce qui est possible car α est surjectif) et on a $\mathcal{C}, \nu[c/x] \models \varphi$. Par hypothèse d'induction, on a $\mathcal{A}, \alpha(\nu[c/x]) \models \varphi$, ce qui revient à dire $\mathcal{A}, \alpha\nu[a/x] \models \varphi$ et ce, quel que soit $a \in \mathcal{A}_s$. On en déduit que $\mathcal{A}, \alpha\nu \models \forall x : s.\ \varphi$.

Supposons maintenant $\mathcal{A} \models \varphi_\ominus$ et montrons $\mathcal{C} \models \varphi$. Pour cela, considérons une valuation ν de X vers $\mathcal{C}$. On a alors $\mathcal{A}, \alpha\nu \models \varphi_\ominus$ et, d'après ce qui précède, $\mathcal{C}, \nu \models \varphi$. Ceci étant vrai pour toute valuation ν, on a $\mathcal{C} \models \varphi$. Si maintenant $\mathcal{C} \models \varphi$ et si ν' est une valuation de X vers $\mathcal{A}$, alors il existe une valuation ν de X vers $\mathcal{C}$ telle que $\nu' = \alpha\nu$ (car α est surjectif). On a $\mathcal{C}, \nu \models \varphi$, d'où l'on déduit, avec ce qui précède, $\mathcal{A}, \nu' \models \varphi_\oplus$ et ce, quelle que soit la valuation ν'. Par conséquent $\mathcal{A} \models \varphi_\oplus$. ■

Par un raisonnement par induction sur φ, du même type que le précédent, on obtient le résultat suivant.

Proposition – 5.1.2

Soient une signature Σ, deux $\Sigma^\bullet$-structures $\mathcal{A}$ et $\mathcal{A}'$ telles que $\mathcal{A} \sqsubseteq \mathcal{A}'$ et $\varphi \in \mathrm{Sen}(\Sigma)$. On a alors :

$$\begin{array}{rl} & \mathcal{A}' \models \varphi_\ominus \;\Rightarrow\; \mathcal{A} \models \varphi \\ \textit{et} & \mathcal{A} \models \varphi \;\Rightarrow\; \mathcal{A}' \models \varphi_\oplus \end{array}$$

Ces résultats suggèrent une méthodologie pour répondre aux problèmes consistant à savoir si une structure $\mathcal{C}$ satisfait un énoncé φ :

1. Trouver une $[\Sigma]$-algèbre $\mathcal{B}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \rightarrow \mathcal{B}$;
2. Étendre $\mathcal{B}$ en une $\Sigma^\bullet$-structure $\mathcal{A}$ correcte relativement à $\mathcal{C}$ et α ;
3. Si $\mathcal{A} \models \varphi_\ominus$, alors $\mathcal{C} \models \varphi$;
4. Si $\mathcal{A} \not\models \varphi_\oplus$, alors $\mathcal{C} \not\models \varphi$,

(remarquons qu'il est tout à fait possible que ni $\mathcal{A} \models \varphi_\ominus$ ni $\mathcal{A} \not\models \varphi_\oplus$ ne soit vrai). Fournir des réponses pertinentes au premier point ne peut se faire qu'avec une connaissance approfondie du problème dont $\mathcal{C}$ et φ sont issus (des exemples de telles abstractions dans le cadre des protocoles cryptographiques seront donnés plus loin). Concentrons-nous pour l'instant sur le second point et montrons comment utiliser les définitions des interprétations des symboles de prédicats dans $\mathcal{C}$ pour étendre $\mathcal{B}$ en une structure qui convient.

5.2 Guide pour les abstractions

Nous allons ici considérer trois cas important de définition d'interprétation des symboles de prédicats : le cas où le prédicat est défini par une énoncé du premier ordre (sans occurrence du symbole de prédicat considéré à l'intérieur de cet énoncé), celui où le prédicat est défini comme plus petit prédicat engendré par un ensemble de règles et, finalement, le cas du prédicat d'égalité dans les structures du premier ordre avec égalité.

Prédicat défini par un énoncé. La proposition suivante est une conséquence immédiate du théorème 5.1.1.

Proposition – 5.2.1

Soient Σ une signature du premier ordre, $\mathcal{C}$ une Σ-structure et $\mathcal{A}$ une $\Sigma^\bullet$-structure correcte relativement à $\mathcal{C}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \rightarrow [\mathcal{A}]$. Soient $P : s_1 \times \ldots \times s_n$ un nouveau symbole de prédicat et $\varphi \in \mathrm{Sen}(\Sigma)$. On note :

- *Σ' la signature Σ étendue avec ce nouveau symbole ;*
- *$\mathcal{C}'$ la structure $\mathcal{C}$ étendue en une Σ'-structure en convenant que :*

$$\mathcal{C}' \;\models\; Px_1 \ldots x_n \Leftrightarrow \varphi$$

($x_1, \ldots, x_n$ désignant des variables distinctes). Alors l'unique $(\Sigma')^\bullet$-structure $\mathcal{A}'$ étendant $\mathcal{A}$ et telle que :

$$\begin{array}{rl} & \mathcal{A}' \;\models\; P_\ominus x_1 \ldots x_n \Leftrightarrow \varphi_\ominus \\ \textit{et} & \mathcal{A}' \;\models\; P_\oplus x_1 \ldots x_n \Leftrightarrow \varphi_\oplus \end{array}$$

est correcte relativement à $\mathcal{C}'$ et α.

Il est très fréquent de rencontrer, dans des spécifications, des prédicats définis de cette manière. Cette proposition permet de proposer simplement des contreparties abstraites pour un tel prédicat.

Prédicat défini par un ensemble de règles. Il est fréquent que l'interprétation d'un prédicat P soit définie comme la plus petite interprétation satisfaisant un ensemble de règles. La proposition suivante montre, dans ce cas, comment définir $P_\oplus$ de manière correcte. On ne peut malheureusement pas faire de même pour $P_\ominus$ dont l'interprétation devra être être donnée au cas par cas.

PROPOSITION – 5.2.2

Soient Σ une signature du premier ordre, $\mathcal{C}$ une Σ-structure et $\mathcal{A}$ une $\Sigma^\bullet$-structure correcte relativement à $\mathcal{C}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \to [\mathcal{A}]$. Soient $P : s_1 \times \ldots \times s_n$ un nouveau symbole de prédicat et $\mathcal{R}$ un ensemble de règles de la forme :

$$R \;=\; \frac{Pt_1^1 \ldots t_n^1 \quad \ldots \quad Pt_1^m \ldots t_n^m \quad \varphi}{Pt_1^0 \ldots t_n^0}$$

(avec $\varphi \in \mathrm{Sen}(\Sigma)$, $t_1^0, \ldots, t_1^m \in \mathrm{T}_\Sigma(X)_{s_1}, \ldots, t_n^0, \ldots, t_n^m \in \mathrm{T}_\Sigma(X)_{s_n}$). On note :

- *Σ' la signature Σ étendue avec ce nouveau symbole ;*
- *$\mathcal{C}'$ la structure $\mathcal{C}$ étendue en une Σ'-structure en convenant que l'interprétation de P dans $\mathcal{C}'$ est la plus petite relation satisfaisant chaque règle de $\mathcal{R}$.*

Alors toute $(\Sigma')^\bullet$-structure $\mathcal{A}'$ étendant $\mathcal{A}$ et telle que :

- *quels que soient $c_1 \in \mathcal{C}_{s_1}, \ldots, c_n \in \mathcal{C}_{s_n}$, $P_\ominus^{\mathcal{A}'}(\alpha(c_1), \ldots, \alpha(c_n))$ implique $P^{\mathcal{C}'}(c_1, \ldots, c_n)$;*
- *l'interprétation de $P_\oplus$ dans $\mathcal{A}'$ est la plus petite relation satisfaisant chaque règle de l'ensemble $\mathcal{R}_\oplus$ obtenu à partir de $\mathcal{R}$ en remplaçant chaque règle R par :*

$$R_\oplus \;=\; \frac{P_\oplus t_1^1 \ldots t_n^1 \quad \ldots \quad P_\oplus t_1^m \ldots t_n^m \quad \varphi_\oplus}{P_\oplus t_1^0 \ldots t_n^0}$$

est correcte relativement à $\mathcal{C}'$ et α.

Démonstration :

Nous allons montrer que, quels que soient $c_1 \in \mathcal{C}_{s_1}, \ldots, c_n \in \mathcal{C}_{s_n}$, on a :

$$P^{\mathcal{C}'}(c_1, \ldots, c_n) \;\Rightarrow\; P_\oplus^{\mathcal{A}'}(\alpha(c_1), \ldots, \alpha(c_n))$$

et ce, par induction sur la dérivation de $P^{\mathcal{C}'}(c_1, \ldots, c_n)$. Supposons donc qu'il existe une règle :

$$R \;=\; \frac{Pt_1^1 \ldots t_n^1 \quad \ldots \quad Pt_1^m \ldots t_n^m \quad \varphi}{Pt_1^0 \ldots t_n^0} \in \mathcal{R}$$

et une valuation ν de X vers $\mathcal{C}'$ telles que :

$$\mathcal{C}', \nu \models P(t_1^1, \ldots, t_n^1) \wedge \ldots \wedge P(t_1^m, \ldots, t_n^m) \wedge \varphi$$

on a par hypothèse d'induction :

$$\mathcal{A}', \alpha\nu \models P_\oplus(t_1^1, \ldots, t_n^1) \wedge \ldots \wedge P_\oplus(t_1^m, \ldots, t_n^m)$$

et par le théorème 5.1.1 :

$$\mathcal{A}', \alpha\nu \models \varphi_\oplus$$

Appliquant la règle $R_\oplus$, il vient alors $P_\oplus^{\mathcal{A}'}(\alpha(t_1^0\nu), \ldots, \alpha(t_n^0\nu))$, d'où le résultat. ■

Prédicat d'égalité. Le prédicat d'égalité des structures avec égalité s'abstrait très simplement, mais en faisant référence au morphisme surjectif, comme le montre la proposition suivante.

Proposition – 5.2.3

Soient Σ une signature du premier ordre, $\mathcal{C}$ une Σ-structure et $\mathcal{A}$ une $\Sigma^\bullet$-structure correcte relativement à $\mathcal{C}$ et un morphisme surjectif $\alpha : [\mathcal{C}] \to [\mathcal{A}]$. On note $\mathcal{C}'$ l'unique Σ-structure avec égalité étendant $\mathcal{C}$. Alors, l'unique $(\Sigma^{\mathrm{Eq}})^\bullet$-structure $\mathcal{A}'$ étendant $\mathcal{A}$ et telle que :

- *pour $a_1, a_2 \in \mathcal{A}$, $a_1 =_\oplus^{\mathcal{A}'} a_2$ si, et seulement si, $a_1 = a_2$;*
- *pour $a_1, a_2 \in \mathcal{A}$, $a_1 =_\ominus^{\mathcal{A}'} a_2$ si, et seulement si, $a_1 = a_2$ et $\alpha^{-1}(a_1)$ est un singleton,*

est correcte relativement à $\mathcal{C}'$ (en tant que Σ^{Eq}-structure) et α.

Démonstration :

Il suffit de montrer que quels que soient $c_1, c_2 \in \mathcal{C}$, on a :

$$\alpha(c_1) =_\ominus^{\mathcal{A}'} \alpha(c_2) \;\Rightarrow c_1 = c_2 \quad \text{et} \quad c_1 = c_2 \;\Rightarrow\; \alpha(c_1) =_\oplus^{\mathcal{A}'} \alpha(c_2)$$

Supposons tout d'abord $\alpha(c_1) =_\ominus^{\mathcal{A}'} \alpha(c_2)$. On a donc $\alpha(c_1) = \alpha(c2)$, d'une part, et $\alpha^{-1}(\alpha(c_1)) = \{c_1\}$, d'autre part. Or $c_1, c_2 \in \alpha^{-1}(\alpha(c_1)) = \alpha^{-1}(\alpha(c_2))$, donc $c_1 = c_2$. La deuxième implication est évidente. ■

Cette proposition nous explique comment abstraire la relation d'égalité dans une Σ-structure avec égalité. En pratique, l'abstraction des énoncés atomiques de la forme $t_1 = t_2$ se fera de la manière suivante :

- $(t_1 = t_2)_\oplus$ est simplement $t_1 = t_2$;
- $(t_1 = t_2)_\ominus$ est l'énoncé $t_1 = t_2 \wedge \mathrm{inj}(t_1)$, où on a défini pour chaque sorte s un prédicat $\mathrm{inj} : s$ dont l'interprétation dans $\mathcal{A}'$ est : $\mathrm{inj}^{\mathcal{A}'}(a)$ si, et seulement si, $\alpha^{-1}(a)$ est un singleton.

Remarquons pour finir que la définition de la proposition précédente est optimale puisqu'elle vérifie :

$$\begin{aligned} a_1 =_\ominus^{\mathcal{A}'} a_2 \;&\Leftrightarrow\; \forall c_1 \in \alpha^{-1}(a_1), c_2 \in \alpha^{-1}(a_2).\ c_1 = c_2 \\ a_1 =_\oplus^{\mathcal{A}'} a_2 \;&\Leftrightarrow\; \exists c_1 \in \alpha^{-1}(a_1), c_2 \in \alpha^{-1}(a_2).\ c_1 = c_2 \end{aligned}$$

Nous avons expliqué ici comment, étant donnés une structure $\mathcal{C}$, un énoncé φ et un morphisme surjectif α défini sur $[\mathcal{C}]$, définir une structure $\mathcal{A}$ et un énoncé $\varphi_\oplus$ tels que :

$$\mathcal{A} \models \varphi_\ominus \;\Rightarrow\; \mathcal{C} \models \varphi$$

À ce niveau de généralité, il n'est pas possible d'aller plus loin et il faut donc restreindre le problème et considérer des structures particulières. Celles auxquelles on s'intéresse ici sont, bien entendu, les structures obtenues à partir de protocoles cryptographiques. Nous expliquons donc dans le chapitre suivant comment, dans le cas de telles structures, proposer des morphismes surjectifs α pertinents et obtenir, à partir de là, des abstractions correctes des structures de départ.

Chapitre 6

En pratique

Dans le chapitre 4, nous avons décrit la sémantique d'un protocole cryptographique au moyen d'une structure du premier ordre. Dans le chapitre 5, nous avons défini une approche générique permettant de raisonner par abstraction sur de telles structures. Nous allons donner dans ce chapitre quelques instanciations possibles de cette approche générique, adaptées au cas particulier des protocoles cryptographiques.

La première abstraction n'est qu'une étape préliminaire, car elle ne permet pas directement d'obtenir une structure finie. Elle consiste à projeter les noms qui interviennent dans la sémantique sur un sous-ensemble fini. Si elle ne permet pas d'obtenir un ensemble de messages abstraits fini, il se peut en revanche que l'ensemble des messages intervenant dans les exécutions du protocole soit, lui, fini. Plutôt que de chercher à justifier formellement une telle propriété, nous proposons une deuxième abstraction qui consiste à ne garder qu'un sous-ensemble fini de l'ensemble des messages et à identifier tous les autres. Cette abstraction a l'avantage d'être simple mais nous montrerons qu'elle s'avère trop imprécise pour un certain nombre de protocoles. Afin de pallier cet inconvénient, nous définirons une troisième et dernière abstraction agissant sur les messages de manière plus homogène.

Remarque : Supposons donné un protocole P, dont la sémantique est $(\Sigma_P, \mathcal{C}_P)$. D'après le chapitre précédent, pour définir une $\Sigma_P^\bullet$-structure $\mathcal{A}$ qui soit correcte relativement à $\mathcal{C}_P$ et un morphisme surjectif α, il faut tout d'abord définir une $[\Sigma_P]$-algèbre $[\mathcal{A}]$ ainsi qu'un morphisme surjectif $\alpha : [\mathcal{C}_P] \to \mathcal{A}$, puis ensuite donner aux symboles de prédicats de $\Sigma_P^\bullet$ une interprétation correcte dans $[\mathcal{A}]$. Nous procéderons exactement de cette façon pour décrire chaque abstraction (et nous utiliserons, à chaque fois que ce sera possible, les résultats de la section 5.2 pour donner des interprétations correctes aux symboles de prédicats). Chaque présentation d'une abstraction correcte sera suivie d'une étude du domaine d'applications et des limitations de l'abstraction en question, le plus souvent au moyen d'exemples. $\square$

6.1 Projection des noms

Exemple : Regardons la propriété de secret portant sur le protocole de Denning et Sacco (page 31) telle que présentée dans le chapitre 4 (nous écrivons cette fois explicitement la quantification universelle) :

$$
\begin{array}{l}
\mathit{secret}_A \ \hat{=} \\
\quad \forall A.a : \mathit{Cst}.\ \forall A.s : \mathit{Name}.\ \forall A.b, A.k, A.t, A.x' : \mathit{Msg}.\ \forall l : \mathit{List}[\mathit{Msg}].\ \forall l' : \mathit{List}[\mathit{Nonce}]. \\
\qquad\qquad\qquad (\mathit{tr}(l,l') \wedge A.a \neq i \wedge A.b \neq i \wedge A.s \neq i \wedge \mathit{compl}_A) \ \Rightarrow\ \neg(l \Vdash A.k)
\end{array}
$$

avec :

$$
\mathit{compl}_A \ \hat{=}\ \langle A.a, A.b\rangle \in l \wedge l \Vdash \{\langle A.b, \langle A.k, \langle A.t, A.x'\rangle\rangle\rangle\}_{\mathrm{k}(A.a,A.s)} \wedge A.x' \in l
$$

Comme cette quantification universelle risque de nous gêner dans nos abstractions (par exemple si une clé de session échangée entre le serveur et un participant honnête était identifiée avec une clé de session échangée entre le serveur et l'intrus, on ne pourrait plus montrer le secret de cette clé), nous allons tout d'abord l'éliminer. Le principe est le même que celui qui était appliqué dans [Bol97] : il s'agit de d'associer à chacune des variables $A.a, A.b, A.s, A.k, A.t$ une nouvelle constante, dont l'interprétation dans $\mathcal{C}_P$ sera arbitraire. Formellement, tout ceci est basé sur l'équivalence entre :

- $\mathcal{C}_P \models \mathit{secret}_A$;
- quels que soient $a \in (\mathcal{C}_P)_{\mathit{Cst}}$, $s \in (\mathcal{C}_P)_{\mathit{Name}}$ et $b, k, t \in (\mathcal{C}_P)_{\mathit{Msg}}$, on a :

  $$\mathcal{C}_P, [a/A.a, s/A.s, b/A.b, k/A.k, t/A.t] \models \mathit{secret}'_A$$

  où on a posé :

  $$
  \begin{array}{l}
  \mathit{secret}'_A \ \hat{=}\ \forall A.x' : \mathit{Msg}.\ \forall l : \mathit{List}[\mathit{Msg}].\ \forall l' : \mathit{List}[\mathit{Nonce}]. \\
  \qquad\qquad (\mathit{tr}(l,l') \wedge A.a \neq i \wedge A.b \neq i \wedge A.s \neq i \wedge \mathit{compl}_A) \ \Rightarrow\ \neg(l \Vdash A.k)
  \end{array}
  $$

Une autre manière équivalente de présenter les choses consiste à étendre la signature Σ_P associée au protocole au moyen de nouveaux symboles de constantes :

$$a : \mathit{Cst}, s : \mathit{Name}, b, k, t : \mathit{Msg}$$

On note Σ'_P la signature obtenue. On a alors équivalence entre :

- $\mathcal{C}_P \models \mathit{secret}_A$;
- pour toute Σ'_P-structure $\mathcal{C}'$, coïncidant avec $\mathcal{C}_P$ sur Σ_P, on a :

  $$\mathcal{C}' \models \mathit{secret}''_A$$

  où secret''_A est obtenue en remplaçant (syntaxiquement) chaque occurrence de $A.a$ (respectivement $A.s, A.b, A.k, A.t$) dans secret'_A par a (respectivement s, b, k, t).

Nous garderons donc à l'esprit dans la suite que la sémantique d'un protocole pourra avoir été étendue au moyen de constantes arbitraires. $\square$

$[\Sigma^{\bullet}_P]$-algèbre $[\mathcal{A}]$. L'abstraction que nous allons présenter maintenant consiste à garder intact un ensemble fini de noms et à identifier tous les autres (il s'agit donc de la technique d'abstraction de [Bol97] replacée dans notre cadre de travail). Pour ce faire, nous considérons deux sous-ensembles finis $C \subseteq (\mathcal{C}_P)_{\mathit{Cst}}$ et $N \subseteq (\mathcal{C}_P)_{\mathit{Nonce}}$ qui indiquent quels noms nous allons garder distincts, ainsi que deux symboles (notés $\perp_{\mathrm{Cst}}$ et $\perp_{\mathrm{Nonce}}$) qui serviront à identifier d'une part les éléments de $(\mathcal{C}_P)_{Cst}$ qui ne sont pas dans C et d'autre part les éléments de $(\mathcal{C}_P)_{Nonce}$ qui ne sont pas dans N. On considère donc :

$$\begin{array}{rcl} \bot_{\mathrm{Cst}} & \in & (\mathcal{C}_P)_{Cst} \\ C & \subseteq & (\mathcal{C}_P)_{Cst} \setminus \{\bot_{\mathrm{Cst}}\} \\ \bot_{\mathrm{Nonce}} & \in & (\mathcal{C}_P)_{Nonce} \\ N & \subseteq & (\mathcal{C}_P)_{Nonce} \setminus \{\bot_{\mathrm{Nonce}}\} \end{array}$$

et on commence à construire une $[\Sigma_P^\bullet]$-algèbre $[\mathcal{A}]$ en posant :

$$\begin{array}{rcl} \mathcal{A}_{Cst} & \hat{=} & C \cup \{\bot_{\mathrm{Cst}}\} \\ \mathcal{A}_{Nonce} & \hat{=} & N \cup \{\bot_{\mathrm{Nonce}}\} \end{array}$$

De la même manière que $(\mathcal{C}_P)_{Name}$ est la réunion disjointe de $(\mathcal{C}_P)_{Cst}$ et $(\mathcal{C}_P)_{Nonce}$, nous allons considérer que $\mathcal{A}_{Name}$ est la réunion disjointe de $\mathcal{A}_{Cst}$ et $\mathcal{A}_{Nonce}$. Nous définissons également des applications surjectives :

$$\begin{array}{rlcl} \alpha_{Cst}: & (\mathcal{C}_P)_{Cst} & \rightarrow & \mathcal{A}_{Cst} \\ & c \in C & \mapsto & c \\ & c \notin C & \mapsto & \bot_{\mathrm{Cst}} \end{array} \quad \text{et} \quad \begin{array}{rlcl} \alpha_{Nonce}: & (\mathcal{C}_P)_{Nonce} & \rightarrow & \mathcal{A}_{Nonce} \\ & n \in N & \mapsto & n \\ & n \notin N & \mapsto & \bot_{\mathrm{Nonce}} \end{array}$$

Ces applications induisent naturellement une surjection α_{Name} de $(\mathcal{C}_P)_{Name}$ vers $\mathcal{A}_{Name}$. Nous posons également :

$$i^{\mathcal{A}} \ \hat{=}\ \alpha_{Cst}(i^{\mathcal{C}_P})$$

(en général, on choisira C de sorte que $i^{\mathcal{C}_P} \notin C$ et on aura alors $i^{\mathcal{A}} = \bot_{\mathrm{Cst}}$). En ce qui concerne les listes de nonces, nous les remplaçons dans $\mathcal{A}$ par des ensembles, ce qui conduit à prendre les interprétations suivantes :

$$\begin{array}{rcl} \mathcal{A}_{List[Nonce]} & \hat{=} & 2^{\mathcal{A}_{Nonce}} \\ []^{\mathcal{A}} & \hat{=} & \varnothing \\ ::^{\mathcal{A}} & : & \begin{array}{ccc} \mathcal{A}_{Nonce} \times \mathcal{A}_{List[Nonce]} & \rightarrow & \mathcal{A}_{List[Nonce]} \\ (n, l) & \mapsto & \{n\} \cup l \end{array} \end{array}$$

On prend pour $\alpha_{List[Nonce]}$ l'application qui à un élément $l \in (\mathcal{C}_P)_{List[Nonce]}$ (nécessairement de la forme $n_1 :: \ldots :: n_k :: []^{\mathcal{C}_P}$ d'après la spécification décrivant $\mathcal{C}_P$) associe l'ensemble $\{\alpha_{Nonce}(n_1), \ldots, \alpha_{Nonce}(n_k)\}$. Il est facile de prolonger ensuite ces constructions en une $[\Sigma^\bullet]$-algèbre $[\mathcal{A}]$, en suivant pas à pas les étapes de construction de $[\mathcal{C}_P]$ (chapitre 4, section 4.3), à un seul détail près : on utilise dans $[\mathcal{A}]$ des ensembles de messages pour interpréter les listes de messages.

$\Sigma_P^\bullet$-structure $\mathcal{A}$. Interprétons maintenant dans $\mathcal{A}$ les symboles de prédicats de $\Sigma_P^\bullet$ (afin d'alléger les notations, nous omettrons la référence à $\mathcal{A}$ dans ces interprétations, autrement dit, nous confondrons un symbole de prédicat dans $\Sigma_P^\bullet$ avec son interprétation dans $\mathcal{A}$). Commençons tout d'abord par donner une interprétation explicite du prédicat inj :

– si $n \in \mathcal{A}_{Name}$, $inj(n)$ est vrai si, et seulement si, $n \in C \cup N$;

- si $l \in \mathcal{A}_{List[Nonce]}$ (respectivement $l \in \mathcal{A}_{List[Msg]}$), $inj(l)$ est vrai si, et seulement si, l est l'ensemble vide (car un ensemble non vide possède une infinité de représentations sous forme de listes) ;
- en ce qui concerne les clés et les messages, inj est défini inductivement :

$$
\begin{array}{rcl}
\mathrm{inj}(\mathrm{k}(n, n')) & \Leftrightarrow & \mathrm{inj}(n) \text{ et } \mathrm{inj}(n') \\
\mathrm{inj}(\mathrm{k}^{-1}(n)) & \Leftrightarrow & \mathrm{inj}(n) \\
\mathrm{inj}(\mathrm{k}^{+1}(n)) & \Leftrightarrow & \mathrm{inj}(n) \\
\mathrm{inj}(\langle m, m' \rangle) & \Leftrightarrow & \mathrm{inj}(m) \text{ et } \mathrm{inj}(m') \\
\mathrm{inj}(\{m\}_k) & \Leftrightarrow & \mathrm{inj}(m) \text{ et } \mathrm{inj}(k) \\
\mathrm{inj}(\mathrm{h}(m)) & \Leftrightarrow & \mathrm{inj}(m)
\end{array}
$$

Pour le prédicat $\in_\oplus$ défini entre *Nonce* et *List*[*Nonce*] (ainsi que pour celui définit entre *Msg* et *List*[*Msg*]) on peut utiliser le prédicat d'appartenance usuel (on peut pour s'en convaincre remarquer que ce prédicat est défini dans $\mathcal{C}_P$ au moyen de règles et appliquer la proposition 5.2.2). On pose donc :

$$
\begin{array}{rcl}
\in_\oplus & \hat{=} & \{(n, l) \in \mathcal{A}_{Nonce} \times \mathcal{A}_{List[Nonce]} \mid n \in l\} \text{ (pour les nonces)} \\
\in_\oplus & \hat{=} & \{(m, l) \in \mathcal{A}_{Msg} \times \mathcal{A}_{List[Msg]} \mid m \in l\} \text{ (pour les messages)}
\end{array}
$$

Pour le prédicat $\in_\ominus$, on utilise les définitions suivantes :

$$
\begin{array}{rcl}
\in_\ominus & \hat{=} & \{(n, l) \in \mathcal{A}_{Nonce} \times \mathcal{A}_{List[Nonce]} \mid n \in l \text{ et } \mathrm{inj}^{\mathcal{A}}(n)\} \text{ (pour les nonces)} \\
\in_\ominus & \hat{=} & \{(m, l) \in \mathcal{A}_{Msg} \times \mathcal{A}_{List[Msg]} \mid m \in l \text{ et } \mathrm{inj}^{\mathcal{A}}(m)\} \text{ (pour les messages)}
\end{array}
$$

Les prédicats $\Vdash$ et *tr* étant définis sur $\mathcal{C}_P$ au moyen de règles, nous appliquons une fois encore la proposition 5.2.2 afin d'obtenir les définitions suivantes pour $\Vdash_\oplus$ et $tr_\oplus$:

$$\frac{m \in_\oplus l}{l \Vdash_\oplus m}$$

$$\frac{l \Vdash_\oplus m \quad l \Vdash_\oplus m'}{l \Vdash_\oplus \langle m, m' \rangle}$$

$$\frac{l \Vdash_\oplus \langle m, m' \rangle}{l \Vdash_\oplus m} \qquad \frac{l \Vdash_\oplus \langle m, m' \rangle}{l \Vdash_\oplus m'}$$

$$\frac{l \Vdash_\oplus m \quad l \Vdash_\oplus k}{l \Vdash_\oplus \{m\}_k} \qquad \frac{l \Vdash_\oplus \{m\}_k \quad l \Vdash_\oplus k^{-1}}{l \Vdash_\oplus m}$$

$$\frac{l \Vdash_\oplus m}{l \Vdash_\oplus \mathrm{h}(m)}$$

$$\frac{}{tr_\oplus([], [])} \qquad \frac{tr_\oplus(l, l') \quad (\zeta_{i,j})_\oplus \quad (\eta_{i,j})_\oplus \quad (\xi_{i,j})_\oplus}{tr_\oplus(m :: l, x_q :: \ldots :: x_1 :: l')}$$

(cette dernière règle doit être écrite pour tout $i \in \{1, \ldots, n\}$ et tout $j \in \{1, \ldots, k_i\}$, avec les notations de la section 4.3). En ce qui concerne le prédicat $\Vdash_\ominus$, nous allons montrer que la définition suivante est correcte :

$$\frac{m \in_\ominus l}{l \Vdash_\ominus m}$$

$$\frac{l \Vdash_\ominus m \quad l \Vdash_\ominus m'}{l \Vdash_\ominus \langle m, m' \rangle}$$

$$\frac{l \Vdash_\ominus \langle m, m' \rangle}{l \Vdash_\ominus m} \qquad \frac{l \Vdash_\ominus \langle m, m' \rangle}{l \Vdash_\ominus m'}$$

$$\frac{l \Vdash_\ominus m \quad l \Vdash_\ominus k}{l \Vdash_\ominus \{m\}_k} \qquad \frac{l \Vdash_\ominus \{m\}_k \quad l \Vdash_\ominus k^{-1}}{l \Vdash_\ominus m}$$

$$\frac{l \Vdash_\ominus m}{l \Vdash_\ominus \mathrm{h}(m)}$$

Montrons que, quels que soient $m \in (\mathcal{C}_P)_{Msg}$ et $l \in (\mathcal{C}_P)_{List[Msg]}$, on a :

$$\alpha(l) \Vdash_\ominus \alpha(m) \;\Rightarrow\; l \Vdash m \text{ et } \mathrm{inj}(\alpha(m))$$

On procède par induction sur la dérivation de $\alpha(l) \Vdash_\ominus \alpha(m)$:

- si $\alpha(m) \in_\ominus \alpha(l)$, alors on a $\mathrm{inj}(\alpha(m))$ (par définition de $\in_\ominus$) et $m \in l$, donc $l \Vdash m$;
- si $\alpha(m) = \langle \alpha(m'), \alpha(m'') \rangle = \alpha(\langle m', m'' \rangle)$, avec $\alpha(l) \Vdash_\ominus \alpha(m')$ et $\alpha(l) \Vdash_\ominus \alpha(m'')$, alors par hypothèse d'induction on a $\mathrm{inj}(\alpha(m'))$, $\mathrm{inj}(\alpha(m''))$, $l \Vdash m'$ et $l \Vdash m''$. On en déduit $\mathrm{inj}(\alpha(m))$ (d'après la définition de inj plus haut) et $l \Vdash \langle m', m'' \rangle$. On a nécessairement $\langle m', m'' \rangle = m$ (car $\mathrm{inj}(\alpha(m))$) et par conséquent $l \Vdash m$;
- si $\alpha(l) \Vdash_\ominus \langle \alpha(m), \alpha(m') \rangle$, alors par hypothèse d'induction on a $\mathrm{inj}(\langle \alpha(m), \alpha(m') \rangle)$ et $l \Vdash \langle m, m' \rangle$, donc $\mathrm{inj}(\alpha(m))$ et $l \Vdash m$;
- si $\alpha(l) \Vdash_\ominus \langle \alpha(m'), \alpha(m) \rangle$: *idem* ;
- si $\alpha(m) = \{\alpha(m')\}_{\alpha(k)} = \alpha(\{m'\}_k)$ avec $\alpha(l) \Vdash_\ominus \alpha(m')$ et $\alpha(l) \Vdash_\ominus \alpha(k)$, alors par hypothèse d'induction on a $\mathrm{inj}(\alpha(m'))$, $\mathrm{inj}(\alpha(k))$, $l \Vdash m'$ et $l \Vdash k$. On en déduit $\{m'\}_k = m$ et $l \Vdash m$;
- si $\alpha(l) \Vdash_\ominus \{\alpha(m)\}_{\alpha(k)}$ et $\alpha(l) \Vdash_\ominus \alpha(k')$ avec $\alpha(k') = \alpha(k)^{-1}$, alors par hypothèse d'induction on a $\mathrm{inj}(\{\alpha(m)\}_{\alpha(k)})$, $\mathrm{inj}(\alpha(k'))$, $l \Vdash \{m\}_k$ et $l \Vdash k'$. On en déduit que $k' = k^{-1}$ et donc $l \Vdash m$;
- si $\alpha(m) = \mathrm{h}(\alpha(m')) = \alpha(\mathrm{h}(m'))$ et $\alpha(l) \Vdash_\ominus \alpha(m')$, alors par hypothèse d'induction on a $\mathrm{inj}(\alpha(m'))$ et $l \Vdash m'$, donc $\mathrm{inj}(\mathrm{h}(\alpha(m')))$ et $l \Vdash m$.

On a en particulier :

$$\alpha(l) \Vdash_\ominus \alpha(m) \;\Rightarrow\; l \Vdash m$$

et cette définition de $\Vdash_\ominus$ est donc correcte. Pour $tr_\ominus$ qui n'intervient pas dans les propriétés des protocoles, on posera simplement $tr_\ominus \mathrel{\hat{=}} \varnothing$.

Discussion. Comme il a été dit auparavant, cette abstraction n'a pas pour but d'être utilisée directement pour vérifier des protocoles, car elle ne permet pas d'obtenir un ensemble fini de messages. Nous l'utiliserons comme étape préliminaire nécessaire aux abstractions qui suivront. Cependant, certains protocoles sont construits de telle sorte que, même si l'ensemble des messages est infini, l'ensemble qui interprète $tr_\oplus$ dans $\mathcal{A}$ est fini. Considérons le protocole suivant, dû à M. Tatebayashi, M. Matsuzaki et D.B. Newman [TMN89] :

<u>TMN</u> :

$$\begin{array}{llcll}
1. & A & \rightarrow & S & : \langle B, \{K_A\}_{\mathrm{k}^{+1}(S)} \rangle \\
2. & S & \rightarrow & B & : A \\
3. & B & \rightarrow & S & : \langle A, \{K_B\}_{\mathrm{k}^{+1}(S)} \rangle \\
4. & S & \rightarrow & A & : \langle B, \{K_B\}_{K_A} \rangle
\end{array}$$

Ce protocole a pour but de permettre à A et B d'obtenir une clé symétrique K_B (on ne suppose que l'existence d'une clé privée et d'une clé publique pour le serveur S). Au cours du protocole, S utilisera la clé symétrique K_A créée par A (il s'agit en fait d'un nonce). Ainsi, à l'exception des identités des participants, toutes les variables utilisées ici représentent des clés. Si on fait de plus l'hypothèse que les participants ont la capacité de déterminer les messages qui sont des noms, alors les seuls messages qui circuleront dans les exécutions de ce protocole seront instances (*i.e.* image par une valuation) d'un des termes suivants :

$$\langle b, \{k_a\}_{\mathrm{k}^{+1}(s)}\rangle, a, \langle a, \{k_b\}_{\mathrm{k}^{+1}(s)}\rangle, \langle b, \{k_b\}_{k_a}\rangle$$

avec a, b, s, k_a, k_b : *Name*. Dans $\mathcal{A}$, les instances de ces messages sont en nombre fini et, par conséquent, il en est de même de l'interprétation de $tr_\oplus$. Il faut néanmoins faire un raisonnement à part, pour chaque protocole, afin d'obtenir ce résultat. La raison d'être de l'abstraction suivante est de permettre d'éliminer tous les messages qui ne sont pas instance de certains termes.

6.2 Filtrage des messages

L'abstraction que nous allons définir ici conserve un certain nombre de messages et identifie les autres. Cette abstraction débute, comme celle de la section précédente, par une projection des noms. Nous allons noter $\mathcal{B}$ la $[\Sigma_P^\bullet]$-algèbre de la section précédente et $\beta : [\mathcal{C}_P] \to \mathcal{B}$ le morphisme surjectif associé. Nous allons ensuite définir une nouvelle $[\Sigma_P^\bullet]$-algèbre $[\mathcal{A}]$ ainsi qu'un morphisme surjectif $\beta' : \mathcal{B} \to [\mathcal{A}]$ qui nous donnera par composition un morphisme surjectif α de $[\mathcal{C}_P]$ dans $[\mathcal{A}]$. Nous donnerons ensuite des interprétations dans $\mathcal{A}$ aux prédicats de $\Sigma_P^\bullet$.

$[\Sigma_P^\bullet]$-algèbre $[\mathcal{A}]$. La seule différence entre $[\mathcal{A}]$ et $\mathcal{B}$ porte sur l'interprétation des messages et des listes de messages. On pose donc tout d'abord :

$$\mathcal{A}_{Cst} \ \hat{=}\ \mathcal{B}_{Cst},\ \mathcal{A}_{Nonce} \ \hat{=}\ \mathcal{B}_{Nonce} \text{ et } \mathcal{A}_{Key} \ \hat{=}\ \mathcal{B}_{Key}$$

et les interprétations des symboles de fonctions correspondants se font dans $\mathcal{A}$ comme ils se faisaient dans $\mathcal{B}$. La surjection β' est alors l'identité. Pour ne garder qu'un nombre fini de messages, nous considérons un ensemble fini $T \subseteq \mathrm{T}_\Sigma(X)$, où X ne contient que des variables de sorte *Cst*, *Nonce*, *Name* ou *Key*. On associe à cet ensemble T un sous-ensemble $M \subseteq \mathcal{B}_{Msg}$ défini comme étant l'ensemble des $m \in \mathcal{B}_{Msg}$ pour lesquels il existe $t \in T$, un sous-terme t' de t et une valuation ν de X dans $\mathcal{B}$ tels que $m = t'\nu$. On note $\perp_{\mathrm{Msg}}$ un élément de $\mathcal{B}_{Msg} \setminus M$ et on pose :

$$\begin{array}{rcl} \mathcal{A}_{Msg} & \hat{=} & M \cup \{\perp_{\mathrm{Msg}}\} \\ \mathcal{A}_{List[Msg]} & \hat{=} & 2^{\mathcal{A}_{Msg}} \end{array}$$

On définit également :

$$\begin{array}{rcrcl} \beta'_{Msg} & : & \mathcal{B}_{Msg} & \to & \mathcal{A}_{Msg} \\ & & m \in M & \mapsto & m \\ & & m \notin M & \mapsto & \perp_{\mathrm{Msg}} \\ \beta'_{List[Msg]} & : & \mathcal{B}_{List[Msg]} & \to & \mathcal{A}_{List[Msg]} \\ & & l = \{m_1, \ldots, m_n\} & \mapsto & \{\beta'_{Msg}(m_1), \ldots, \beta'_{Msg}(m_n)\} \end{array}$$

Les opérations sur les messages sont interprétées dans $\mathcal{A}$ de la manière suivante :

- si l'un des arguments est $\perp_{\mathrm{Msg}}$, alors le résultat est $\perp_{\mathrm{Msg}}$;
- sinon, on calcule le résultat de l'opération dans $\mathcal{B}$ et on lui applique β'_{Msg}.

Les opérations sur $\mathcal{A}_{List[Msg]}$ se font de manière naturelle. On note finalement α le morphisme $\beta' \circ \beta$.

$\Sigma_P^\bullet$-structure $\mathcal{A}$. À l'exception de inj et $\Vdash_\ominus$, les symboles de prédicats sont interprétés de la même manière que dans la section précédente. En ce qui concerne inj, la seule différence est que $\mathrm{inj}(\perp_{\mathrm{Msg}})$ ne doit pas être vrai. Pour $\Vdash_\ominus$, une dérivation dans $\mathcal{B}$ reste valable dans $\mathcal{A}$ à condition qu'elle n'utilise que des messages qui sont dans M. Ceci nous conduit à définir $\Vdash_\ominus$ au moyen des règles suivantes :

$$\frac{m \in_\ominus l \quad \mathrm{inj}(m)}{l \Vdash_\ominus m}$$

$$\frac{l \Vdash_\ominus m \quad l \Vdash_\ominus m' \quad \mathrm{inj}(\langle m, m' \rangle)}{l \Vdash_\ominus \langle m, m' \rangle}$$

$$\frac{l \Vdash_\ominus \langle m, m' \rangle \quad \mathrm{inj}(m)}{l \Vdash_\ominus m} \qquad \frac{l \Vdash_\ominus \langle m, m' \rangle \quad \mathrm{inj}(m')}{l \Vdash_\ominus m'}$$

$$\frac{l \Vdash_\ominus m \quad l \Vdash_\ominus k \quad \mathrm{inj}(\{m\}_k)}{l \Vdash_\ominus \{m\}_k} \qquad \frac{l \Vdash_\ominus \{m\}_k \quad l \Vdash_\ominus k^{-1} \quad \mathrm{inj}(m)}{l \Vdash_\ominus m}$$

$$\frac{l \Vdash_\ominus m \quad \mathrm{inj}(\mathrm{h}(m))}{l \Vdash_\ominus \mathrm{h}(m)}$$

Discussion. Avec cette abstraction, nous sommes maintenant en mesure de traiter le protocole TMN (page 65), ainsi que le suivant, proposé par M. Satyanarayanan dans [Sat89] :

<u>Andrew Secure RPC</u> :

$$\begin{array}{llcll} 1. & A & \rightarrow & B & : \langle A, \{N_A\}_{K_{AB}} \rangle \\ 2. & B & \rightarrow & A & : \{\langle \{N_A\}_0, N_B \rangle\}_{K_{AB}} \\ 3. & A & \rightarrow & B & : \{\{N_B\}_0\}_{K_{AB}} \\ 4. & B & \rightarrow & A & : \{\langle K'_{AB}, N'_B \rangle\}_{K_{AB}} \end{array}$$

Le but de ce protocole est de fournir à A et B une nouvelle clé symétrique K'_{AB}, on suppose pour cela qu'ils disposent déjà d'une clé symétrique K_{AB}. Les trois premières étapes permettent à A et B de s'authentifier mutuellement, au moyen de nonces qui servent de challenges. Dans la dernière étape B produit la nouvelle clé symétrique K'_{AB} ainsi qu'un autre nonce N'_B (qui est censé servir dans les sessions futures) et les envoie à A. Afin de pouvoir traiter ces protocoles au moyen de notre abstraction, il faut supposer que les participants ont la capacité de vérifier, quand ils reçoivent des données, qu'il s'agit bien de noms. Expliquons pourquoi cette hypothèse est nécessaire. Si on la fait pas, l'exécution suivante, où l'intrus participe à une session avec B, serait tout à fait valide :

$$\begin{array}{llcll} 1. & I & \rightarrow & B & : \langle I, \{\langle N_I, N'_I \rangle\}_{K_{IB}} \rangle \\ 2. & B & \rightarrow & I & : \{\langle \{\langle N_I, N'_I \rangle\}_0, N_B \rangle\}_{K_{IB}} \\ 3. & I & \rightarrow & B & : \{\{N_B\}_0\}_{K_{IB}} \\ 4. & B & \rightarrow & I & : \{\langle K'_{IB}, N'_B \rangle\}_{K_{IB}} \end{array}$$

B accepte ici le couple de deux nonces $\langle N_I, N'_I \rangle$ comme un seul. Mais si on regarde ce que donne cette exécution dans notre structure abstraite $\mathcal{A}$, les premier message et le deuxième ne sont pas dans M. Ils sont donc projetés sur $\bot_{\mathrm{Msg}}$ et le début de cette exécution se passe de la manière suivante :

- B accepte $\bot_{\mathrm{Msg}}$ de I (dès que la connaissance de l'intrus l est non vide, on a $l \Vdash_{\oplus} \bot_{\mathrm{Msg}}$;
- B renvoie $\bot_{\mathrm{Msg}}$ à I.

Ainsi, la connaissance de l'intrus devient $\bot_{\mathrm{Msg}} :: l$ Dans ce cas, et quel que soit le message m, on a $\bot_{\mathrm{Msg}} :: l \Vdash_{\oplus} m$ et on ne peut par conséquent plus prouver aucune propriété de secret. Empêcher les participants de vérifier si les messages reçus ont bien la bonne forme rend donc cette abstraction inintéressante. Nous retiendrons donc que cette abstraction fournit des résultats intéressants dans le cas où chaque participant au protocole a la possibilité de vérifier que les messages qu'il reçoit sont bien de la forme voulue. Le gros problème est que, si cette hypothèse peut paraître raisonnable sur les protocoles qui précèdent, elle ne l'est plus dans le cas de protocoles, comme celui de Denning et Sacco (page 31) où un des participants (A dans ce cas) doit accepter un message chiffré avec une clé appartenant à un autre (B ici). Il ne serait en effet pas raisonnable de supposer qu'un participant peut vérifier si un message correspond à ce qui est prévu par le protocole, y compris lorsqu'il lui est impossible de lire ce message. Le problème de cette abstraction est que, mis à part un certain nombre de messages, tous les messages sont identifiés, sans faire aucune distinction suivant ce qu'ils contiennent. L'abstraction suivante permet de travailler de manière plus homogène.

6.3 Linéarisation des messages

L'abstraction que nous allons décrire maintenant présente un point commun et une différence majeure avec l'abstraction de la section précédente. Comme l'abstraction précédente, son but est de projeter l'ensemble des messages sur un ensemble fini. Elle a par contre l'avantage de procéder de manière beaucoup plus homogène. Comme pour l'abstraction précédente, nous allons partir de la $[\Sigma_P^\bullet]$-algèbre de la section 6.1 notée $\mathcal{B}$ et du morphisme surjectif associé $\beta : [\mathcal{C}_P] \to \mathcal{B}$. Nous allons là encore nous baser sur cette construction pour produire une $\Sigma_P^\bullet$-structure $\mathcal{A}$.

$[\Sigma_P^\bullet]$-algèbre $[\mathcal{A}]$. $\mathcal{B}$ et $[\mathcal{A}]$ ne diffèrent que par l'interprétation des messages et des listes de messages. Nous ne décrirons donc que cette partie de $\mathcal{A}$. À cet effet, posons :

$$\begin{array}{rcl} V & \hat{=} & 2^{\mathcal{B}_{Key}} \times \mathcal{B}_{Key} \\ \mathcal{A}_{Msg} & \hat{=} & 2^V \setminus \{\varnothing\} \\ \mathcal{A}_{List[Msg]} & \hat{=} & 2^{\mathcal{A}_{Msg}} \end{array}$$

Un élément $(\{k_1, \ldots, k_n\}, k_0)$ de V sera appelé une vue et noté $k_1 \otimes \ldots \otimes k_n \multimap k_0$. Nous avons donc choisi d'interpréter un message dans $\mathcal{A}$ par un ensemble non vide de vues. On trouve les vues associées à un message en l'écrivant sous forme d'un arbre et en le parcourant de la racine aux feuilles : chaque fois qu'une construction de chiffrement est traversée, on garde la clé associée dans la partie gauche de la vue et, quand une feuille est atteinte, la clé qu'elle contient est placée dans la partie droite de la vue.

Exemple : Considérons le message :

$$m \;\hat{=}\; \{\langle \{k_2\}_{k_1}, \langle k_3, \{k_5\}_{k_4} \rangle \rangle\}_{k_0}$$

les vues associées sont :

$$\begin{array}{l} k_0 \otimes k_1 \multimap k_2 \\ k_0 \multimap k_3 \\ k_0 \otimes k_4 \multimap k_5 \end{array}$$

Les vues du message m traduisent le fait que :
- si on dispose de m, k_0 et k_1, alors on peut obtenir k_2 ;
- si on dispose de m et k_0, alors on peut obtenir k_3 ;
- si on dispose de m, k_0 et k_4, alors on peut obtenir k_5.

En revanche, l'ordre dans lequel ces clés ont été appliquées, ainsi que leur position par rapport aux constructeurs de couples disparaissent. □

Formellement, les vues associées à un message se calculent de la manière suivante (qui nous tiendra lieu de définition de β'_{Msg}) :
- l'ensemble des vues d'une clé $k \in \mathcal{B}_{Key}$ considérée comme un message est :

$$\beta'_{Msg}(k) \ \hat{=} \ \{\multimap k\}$$

- si $m, m' \in \mathcal{B}_{Msg}$, on pose :

$$\beta'_{Msg}(\langle m, m' \rangle) \ \hat{=} \ \beta'_{Msg}(m) \cup \beta'_{Msg}(m')$$

- si $m \in \mathcal{B}_{Msg}$ et $k \in \mathcal{B}_{Key}$, alors :

$$\beta'_{Msg}(\{m\}_k) \ \hat{=} \ \{k \otimes k_1 \otimes \ldots \otimes k_n \multimap k_0 \mid k_1 \otimes \ldots \otimes k_n \multimap k_0 \in \beta'_{Msg}(m)\}$$

- le cas du constructeur de hachage, plus difficile, sera traité un peu plus tard.

Cette définition nous indique également comment interpréter les opérations sur les messages (excepté le hachage) dans $\mathcal{A}$:
- le constructeur de couples est interprété par la réunion ensembliste ;
- le chiffrement par une clé k consiste à ajouter k dans la partie gauche de toutes les vues du message de départ.

Les opérations sur les listes de messages sont interprétées de manière naturelle, une fois de plus. Nous obtenons ainsi une $[\Sigma_P^\bullet]$-algèbre $[\mathcal{A}]$ ainsi qu'un morphisme surjectif $\beta' : \mathcal{B} \to [\mathcal{A}]$. On pose $\alpha \mathrel{\hat{=}} \beta' \circ \beta$.

$\Sigma_P^\bullet$-structure $\mathcal{A}$. Cette abstraction identifie beaucoup plus d'éléments que les précédentes. Un ensemble de vues, même très simple comme $\{\multimap k\}$ et même si $\mathrm{inj}(k)$ est vrai possède une infinité d'antécédents par α :

$$k, \langle k, k \rangle, \langle k, \langle k, k \rangle \rangle, \ldots$$

de sorte que, sur les messages, inj sera toujours faux. Une conséquence de ceci est que $\in_\ominus$ (entre messages et listes de messages) et $\Vdash_\ominus$ seront eux aussi toujours faux (en toute rigueur, il serait possible d'en donner une définition non triviale, mais peu intéressante puisque ce prédicat n'intervient pas dans les propriétés de sécurité. On peut trouver cette définition dans [BB01]). Pour ce qui est des autres prédicats de $\Sigma_P^\bullet$, nous savons déjà comment les abstraire, en utilisant le guide du chapitre précédent. Nous allons cependant donner une définition de $\Vdash_\oplus$, un peu différente de celle qui serait obtenue en appliquant la proposition 5.2.2, mais plus adaptée au formalisme des vues. Considérons donc un élément $l \in \mathcal{A}_{List[Msg]}$ ainsi que $m \in \mathcal{A}_{Msg}$. On note $l = \{m'_1, \ldots, m'_k\}$, alors par définition de $\Vdash_\oplus$:

$$l \Vdash_\oplus m \ \Leftrightarrow \ \{\langle m'_1, \langle \ldots, m'_n \rangle \rangle\} = \{m'_1 \cup \ldots \cup m'_k\} \Vdash_\oplus m$$

On note sous forme d'ensemble de vues $m = \{v_1, \ldots, v_n\}$. Si on pose $m_1 = \{v_1\}, \ldots, m_n = \{v_n\}$, on a alors :

$$m \;=\; \langle m_1, \langle \ldots, m_n \rangle \rangle$$

et d'après la définition de $\Vdash_\oplus$:

$$l \Vdash_\oplus m \;\Leftrightarrow\; (l \Vdash_\oplus m_1, \ldots \text{ et } l \Vdash_\oplus m_n)$$

Par conséquent, $l \Vdash_\oplus m$ est vrai si, et seulement si, chacune des vues de m peut être déduite de l'ensemble des vues des messages qui apparaissent dans l. À cet effet, définissons une relation $\Vvdash$ entre ensembles de vues et vues au moyen des règles suivantes :

$$\frac{k_1 \otimes \ldots \otimes k_n \multimap k_0 \in \bar{l}}{\bar{l} \Vvdash k_1 \otimes \ldots \otimes k_n \multimap k_0}$$
$$\frac{\bar{l} \Vvdash k_1 \otimes \ldots \otimes k_n \multimap k_0 \quad \bar{l} \Vvdash \multimap k}{\bar{l} \Vvdash k \otimes k_1 \otimes \ldots \otimes k_n \multimap k_0}$$
$$\frac{\bar{l} \Vvdash k \otimes k_1 \otimes \ldots \otimes k_n \multimap k_0 \quad \bar{l} \Vvdash \multimap k^{-1}}{\bar{l} \Vvdash k_1 \otimes \ldots \otimes k_n \multimap k_0}$$

Nous pouvons ensuite utiliser cette relation pour décider si $l \Vdash_\oplus m$ compte tenu du fait que, si on pose :

$$\bar{l} \;\hat{=}\; \bigcup_{m \in l} m$$

les propositions suivantes sont équivalentes :

- $l \Vdash_\oplus m$;
- quelle que soit $v \in m$, on a $\bar{l} \Vvdash v$.

Revenons, pour terminer la présentation de cette abstraction au cas du constructeur de hachage. On peut en fait considérer que hacher un message est équivalent à le chiffrer avec une clé connue de tous les participants et sans inverse. Soit donc $\mathrm{hk} \notin \mathcal{B}_{Key}$ et modifions la définition des vues comme suit :

$$V \;\hat{=}\; 2^{\mathcal{B}_{Key} \cup \{\mathrm{hk}\}} \times \mathcal{B}_{Key}$$

Le constructeur de hachage sera alors interprété de la manière suivante : la vue de $\mathrm{h}(m) \in \mathcal{B}_{Msg}$ est obtenue en ajoutant hk dans chaque partie gauche d'une vue de m. Comme hk n'est pas réellement une clé, on ne pourra jamais « déchiffrer » un message « chiffré » avec hk. Par contre il faut laisser la possibilité de « chiffrer » avec hk, par exemple au moyen de la règle :

$$\frac{\bar{l} \Vvdash k_1 \otimes \ldots \otimes k_n \multimap k_0}{\bar{l} \Vvdash \mathrm{hk} \otimes k_1 \otimes \ldots \otimes k_n \multimap k_0}$$

Discussion. Afin de mieux juger du pouvoir de cette abstraction, nous avons réalisé une implémentation. En effet, même si l'abstraction est correcte par construction, il est tout à fait possible qu'elle perde trop d'information pour être à même ensuite de prouver les propriétés de sécurité des protocoles considérés. Pour des raisons d'efficacité, cette implémentation présente quelques différences avec ce qui a été présenté ici. Signalons-le tout de suite : si cette abstraction retourne des résultats facilement compréhensibles, qui plus est en un temps raisonnable, elle s'avère impuissante face à un certain nombre de protocoles, en raison des approximations trop importantes qu'elle introduit. Nous allons expliquer tout ceci au moyen de quelques exemples.

Pour commencer, reprenons le protocole Andrew Secure RPC (page 67), cette fois-ci décrit formellement par les processus :

$$\begin{array}{rcl} L_A & \hat{=} & (\nu A.n).\overline{c}\langle\langle A.a, \{A.n\}_{\mathrm{k}(A.a,A.b)}\rangle\rangle. \\ & & c(A.x).\text{case } A.x \text{ of } \{\langle\{A.n\}_0, A.n'\rangle\}_{\mathrm{k}(A.a,A.b)}. \\ & & \overline{c}\langle\{\{A.n'\}_0\}_{\mathrm{k}(A.a,A.b)}\rangle. \\ & & c(A.y).\text{case } A.y \text{ of } \{\langle A.k, 0\rangle\}_{\mathrm{k}(A.a,A.b)}.0 \\ L_B & \hat{=} & c(B.x).\text{case } B.x \text{ of } \langle B.a, \{B.n\}_{\mathrm{k}(B.a,B.b)}\rangle. \\ & & (\nu B.n').\overline{c}\langle\{\langle\{B.n\}_0, B.n'\rangle\}_{\mathrm{k}(B.a,B.b)}\rangle. \\ & & c(B.y).\text{case } B.y \text{ of } \{\{B.n'\}_0\}_{\mathrm{k}(B.a,B.b)}. \\ & & (\nu B.k).\overline{c}\langle\{\langle B.k, 0\rangle\}_{\mathrm{k}(B.a,B.b)}\rangle.0 \end{array}$$

(remarquons que le dernier nonce, normalement envoyé par B à A, et qui apparaît dans le protocole pour des raisons de fraîcheur, a été ici remplacé par 0, pour simplifier). De ces processus, on déduit les règles de la figure 6.3.1 permettant de définir le prédicat $tr_\oplus$. Nous

$$(r_0)\frac{}{tr_\oplus([],[])}$$

$$(r_{A1})\frac{tr_\oplus(l,l') \quad \neg(N \in_\ominus l')}{tr_\oplus(\langle A, \{N\}_{\mathrm{k}(A,B)}\rangle :: l, N :: l')}$$

$$(r_{A2})\frac{tr_\oplus(l,l') \quad \langle A, \{N\}_{\mathrm{k}(A,B)}\rangle \in l \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,B)}}{tr_\oplus(\{\{N'\}_0\}_{\mathrm{k}(A,B)} :: l, l')}$$

$$(r_{B1})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A, \{N\}_{\mathrm{k}(A,B)}\rangle \quad \neg(N' \in_\ominus l')}{tr_\oplus(\{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,B)} :: l, N' :: l')}$$

$$(r_{B2})\frac{tr_\oplus(l,l') \;\; l \Vdash_\oplus \langle A, \{N\}_{\mathrm{k}(A,B)}\rangle \;\; \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,B)} \in_\oplus l \;\; l \Vdash_\oplus \{\{N'\}_0\}_{\mathrm{k}(A,B)} \;\; \neg(K \in_\ominus l')}{tr_\oplus(\{\langle K, 0\rangle\}_{\mathrm{k}(A,B)} :: l, K :: l')}$$

FIG. 6.3.1 – *Première définition du prédicat $tr_\oplus$ pour le protocole Andrew Secure RPC*

noterons ces règles de déduction respectivement r_0, r_{A1}, r_{A2}, r_{B1} et r_{B2}. L'étape préliminaire de projection est quelque peu différente de ce qui a été décrit auparavant. Ici elle consiste à :

– projeter les éléments de sorte *Cst* différents de i sur une constante a ;
– conserver trois éléments de sorte *Nonce*, k, n et n' et projeter les autres sur 0.

Nous allons maintenant nous contenter des exécutions du protocole dans lesquelles n est créé au moyen de la règle r_{A1}, n' au moyen de la règle r_{B1} et k au moyen de la règle r_{B2}. Ces trois constantes n, n' et k étant arbitraires, ceci ne pose pas de problème. Dernière modification : nous allons remplacer les conditions de la forme $m \in_\oplus l$ apparaissant dans les règles $r_{A1}, r_{A2}, r_{B1}, r_{B2}$ par $l \Vdash_\oplus m$ (ce qui est correct, puisque $m \in_\oplus l \Rightarrow l \Vdash_\oplus m$). Muni de ces hypothèses, nous sommes ramenés à considérer le prédicat $tr_\oplus$ défini au moyen des règles de la figure 6.3.2. Nous avons séparé chacune des règles traduisant une création de nonce (par exemple r_{A1}) en deux règles distinctes : la première (r'_{A1} ici) correspond au cas où le nonce créé est préservé par l'abstraction, et la seconde (r''_{A1}) pour le cas où c'est un autre nonce qui est créé. En ce qui concerne précisément ces deux règles, r'_{A1} et r''_{A1}, il est clair qu'il est

$$(r_0)\frac{}{tr_\oplus([],[])}$$

$$(r'_{A1})\frac{tr_\oplus(l,l') \quad n \notin l'}{tr_\oplus(\langle a,\{n\}_{\mathrm{k}(a,B)}\rangle :: l, n :: l')} \qquad \frac{tr_\oplus(l,l')}{tr_\oplus(\langle a,\{0\}_{\mathrm{k}(a,B)}\rangle :: l, 0 :: l')}(r''_{A1})$$

$$(r_{A2})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle a,\{N\}_{\mathrm{k}(a,B)}\rangle \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(a,B)}}{tr_\oplus(\{\{N'\}_0\}_{\mathrm{k}(a,B)} :: l, l')}$$

$$(r'_{B1})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle \quad n' \notin l'}{tr_\oplus(\{\langle\{N\}_0, n'\rangle\}_{\mathrm{k}(A,a)} :: l, n' :: l')} \qquad \frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle}{tr_\oplus(\{\langle\{N\}_0, 0\rangle\}_{\mathrm{k}(A,a)} :: l, 0 :: l')}(r''_{B1})$$

$$(r'_{B2})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,a)} \quad l \Vdash_\oplus \{\{N'\}_0\}_{\mathrm{k}(A,a)} \quad k \notin l'}{tr_\oplus(\{\langle k, 0\rangle\}_{\mathrm{k}(A,a)} :: l, k :: l')}$$

$$(r''_{B2})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,a)} \quad l \Vdash_\oplus \{\{N'\}_0\}_{\mathrm{k}(A,a)}}{tr_\oplus(\{\langle 0, 0\rangle\}_{\mathrm{k}(A,a)} :: l, 0 :: l')}$$

FIG. 6.3.2 – *Deuxième définition du prédicat $tr_\oplus$ pour le protocole Andrew Secure RPC*

correct (mais on y perd en précision) de les appliquer une fois pour toutes, au tout début de l'exécution. Ceci revient à considérer simplement l'ensemble de règles de la figure 6.3.3. Il

$$(r_{0,A1})\frac{}{tr_\oplus(\langle a,\{n\}_{\mathrm{k}(a,a)}\rangle :: \langle a,\{0\}_{\mathrm{k}(a,a)}\rangle :: \langle a,\{n\}_{\mathrm{k}(a,i)}\rangle :: \langle a,\{0\}_{\mathrm{k}(a,i)}\rangle :: [], n :: 0 :: [])}$$

$$(r_{A2})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle a,\{N\}_{\mathrm{k}(a,B)}\rangle \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(a,B)}}{tr_\oplus(\{\{N'\}_0\}_{\mathrm{k}(a,B)} :: l, l')}$$

$$(r'_{B1})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle \quad n' \notin l'}{tr_\oplus(\{\langle\{N\}_0, n'\rangle\}_{\mathrm{k}(A,a)} :: l, n' :: l')} \qquad \frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle}{tr_\oplus(\{\langle\{N\}_0, 0\rangle\}_{\mathrm{k}(A,a)} :: l, 0 :: l')}(r''_{B1})$$

$$(r'_{B2})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,a)} \quad l \Vdash_\oplus \{\{N'\}_0\}_{\mathrm{k}(A,a)} \quad k \notin l'}{tr_\oplus(\{\langle k, 0\rangle\}_{\mathrm{k}(A,a)} :: l, k :: l')}$$

$$(r''_{B2})\frac{tr_\oplus(l,l') \quad l \Vdash_\oplus \langle A,\{N\}_{\mathrm{k}(A,a)}\rangle \quad l \Vdash_\oplus \{\langle\{N\}_0, N'\rangle\}_{\mathrm{k}(A,a)} \quad l \Vdash_\oplus \{\{N'\}_0\}_{\mathrm{k}(A,a)}}{tr_\oplus(\{\langle 0, 0\rangle\}_{\mathrm{k}(A,a)} :: l, 0 :: l')}$$

FIG. 6.3.3 – *Dernière définition du prédicat $tr_\oplus$ pour le protocole Andrew Secure RPC*

ne nous reste donc plus qu'à calculer les exécutions abstraites engendrées par cet ensemble $\{r_{0,A1}, r_{A2}, r'_{B1}, r''_{B1}, r'_{B2}, r''_{B2}\}$. Parmi les règles de cet ensemble, $r_{0,A1}$ peut s'appliquer une seule fois, au tout début. De même, r'_{B1} et r'_{B2} ne peuvent s'appliquer qu'une seule fois. Nous procéderons donc de la manière suivante :

- le point de départ est le couple $p \mathrel{\hat{=}} (l, l')$ engendré par la règle $r_{0,A1}$;
- on calcule p', clôture de p par les règles $r_{A2}, r''_{B1}, r''_{B2}$;
- on calcule P_1 (respectivement P_2), ensemble des exécutions obtenues à partir de p' par une application de la règle r'_{B1} (respectivement r'_{B2}) ;
- on calcule P'_1 (respectivement P'_2) en prenant la clôture de P_1 (respectivement P_2) au

moyen des règles $r_{A2}, r''_{B1}, r''_{B2}$;
- on calcule P''_1 (respectivement P''_2), ensemble des exécutions obtenues à partir des éléments de P'_1 et P'_2 par une application de la règle r'_{B2} (respectivement r'_{B1}) ;
- on calcule P en prenant la clôture de P''_1 et $P''2$ par les règles $r_{A2}, r''_{B1}, r''_{B2}$.

P représente alors les exécutions abstraites du protocole (plus exactement, chaque exécution abstraite du protocole correspond à un sous-ensemble d'un élément de P). Le résultat retourné par notre implémentation se trouve à la figure 6.3.4 (nous avons omis les listes de nonces utilisés, qui sont utiles au cours de la construction des exécutions, mais ne présentent plus d'intérêt par la suite). Dans ce cas précis, la version abstraite de notre propriété de secret

$$\{\mathrm{k}(a,a) \multimap n', \mathrm{k}(a,a) \multimap k, \multimap a, \multimap i, \multimap 0, \multimap n, \multimap \mathrm{k}(i,a), \mathrm{k}(a,a) \multimap n, \mathrm{k}(a,a) \multimap 0\}$$
$$\{\multimap n', \mathrm{k}(a,a) \multimap k, \multimap a, \multimap i, \multimap 0, \multimap n, \multimap \mathrm{k}(i,a), \mathrm{k}(a,a) \multimap n, \mathrm{k}(a,a) \multimap 0\}$$
$$\{\mathrm{k}(a,a) \multimap n', \multimap k, \multimap a, \multimap i, \multimap 0, \multimap n, \multimap \mathrm{k}(i,a), \mathrm{k}(a,a) \multimap n, \mathrm{k}(a,a) \multimap 0\}$$
$$\{\multimap n', \multimap k, \multimap a, \multimap i, \multimap 0, \multimap n, \multimap \mathrm{k}(i,a), \mathrm{k}(a,a) \multimap n, \mathrm{k}(a,a) \multimap 0\}$$

FIG. 6.3.4 – *Résultat retourné pour le protocole Andrew Secure RPC*

pour A, est :

$$\begin{aligned} \mathit{secret}_A \ \hat{=}\ \forall l, l'.\ \mathit{tr}_\oplus(l,l') \ \wedge\ l \Vdash_\oplus \langle a, \{n\}_{\mathrm{k}(a,a)}\rangle \ \wedge \\ l \Vdash_\oplus \{\langle \{n\}_0, n'\rangle\}_{\mathrm{k}(a,a)} \ \wedge\ l \Vdash_\oplus \{\langle k, 0\rangle\}_{\mathrm{k}(a,a)} \ \Rightarrow\ \neg(l \Vdash_\oplus k) \end{aligned}$$

Parmi les exécutions retournées par notre implémentation, celles vérifiant les prémisses de la propriété de secret sont celles qui contiennent $\mathrm{k}(a,a) \multimap n$, $\mathrm{k}(a,a) \multimap n'$ et $\mathrm{k}(a,a) \multimap k$. Il ne reste donc que :

$$\{\mathrm{k}(a,a) \multimap n', \mathrm{k}(a,a) \multimap k, \multimap a, \multimap i, \multimap 0, \multimap n, \multimap \mathrm{k}(i,a), \mathrm{k}(a,a) \multimap n, \mathrm{k}(a,a) \multimap 0\}$$

exécution dans laquelle k est secret.

Sur le protocole de Denning et Sacco (page 31), nous obtenons le résultat de la figure 6.3.5. Parmi ces exécutions, celles qui satisfont les prémisses de la propriété de secret sont celles qui

$$\begin{aligned} &\{\mathrm{k}(a,s) \multimap t, \mathrm{k}(a,s) \multimap k, \mathrm{k}(a,s) \multimap i, \mathrm{k}(a,s) \multimap a, \mathrm{k}(a,s) \multimap 0, \\ &\qquad \multimap a, \multimap s, \multimap i, \multimap 0, \multimap \mathrm{k}(i,a), \multimap \mathrm{k}(i,s)\} \\ &\{\multimap t, \mathrm{k}(a,s) \multimap k, \mathrm{k}(a,s) \multimap i, \mathrm{k}(a,s) \multimap a, \mathrm{k}(a,s) \multimap 0, \\ &\qquad \multimap a, \multimap s, \multimap i, \multimap 0, \multimap \mathrm{k}(i,a), \multimap \mathrm{k}(i,s)\} \\ &\{\mathrm{k}(a,s) \multimap t, \multimap k, \mathrm{k}(a,s) \multimap i, \mathrm{k}(a,s) \multimap a, \mathrm{k}(a,s) \multimap 0, \\ &\qquad \multimap a, \multimap s, \multimap i, \multimap 0, \multimap \mathrm{k}(i,a), \multimap \mathrm{k}(i,s)\} \\ &\{\multimap t, \multimap k, \mathrm{k}(a,s) \multimap i, \mathrm{k}(a,s) \multimap a, \mathrm{k}(a,s) \multimap 0, \multimap a, \multimap s, \multimap i, \multimap 0, \multimap \mathrm{k}(i,a), \multimap \mathrm{k}(i,s)\} \end{aligned}$$

FIG. 6.3.5 – *Résultat retourné pour le protocole de Denning et Sacco*

contiennent $\mathrm{k}(a,s) \multimap k$ et $\mathrm{k}(a,s) \multimap t$, il ne reste donc que :

$$\{\mathrm{k}(a,s) \multimap t, \mathrm{k}(a,s) \multimap k, \mathrm{k}(a,s) \multimap i, \mathrm{k}(a,s) \multimap a, \mathrm{k}(a,s) \multimap 0,$$
$$\multimap a, \multimap s, \multimap i, \multimap 0, \multimap \mathrm{k}(i,a), \multimap \mathrm{k}(i,s)\}$$

Dans cette exécution, k est secret.

Explorons maintenant les limites de cette abstraction. Le protocole suivant est extrait de [BAN89] et est attribué par les auteurs à Yahalom :

<u>Yahalom</u> :

$$\begin{array}{llcll}
1. & A & \rightarrow & B & : \langle A, N_A \rangle \\
2. & B & \rightarrow & S & : \langle B, \{\langle A, N_A, N_B \rangle\}_{K_{BS}} \rangle \\
3. & S & \rightarrow & A & : \langle \{\langle B, K_{AB}, N_A, N_B \rangle\}_{K_{AS}}, \{\langle A, K_{AB} \rangle\}_{K_{BS}} \rangle \\
4. & A & \rightarrow & B & : \langle \{\langle A, K_{AB} \rangle\}_{K_{BS}}, \{N_B\}_{K_{AB}} \rangle
\end{array}$$

Sur ce protocole, notre implémentation nous fournit le résultat qui se trouve à la figure 6.3.6. Comme on peut le constater k n'est pas secret dans cette exécution. Regardons où

$$\{\mathrm{k}(a,s) \multimap n', \mathrm{k}(a,s) \multimap k, \multimap k, \mathrm{k}(a,s) \multimap a, \mathrm{k}(a,s) \multimap s,$$
$$\mathrm{k}(a,s) \multimap i, \mathrm{k}(a,s) \multimap \mathrm{k}(i,a), \mathrm{k}(a,s) \multimap \mathrm{k}(i,s), \mathrm{k}(a,s) \multimap n, \mathrm{k}(a,s) \multimap 0,$$
$$\multimap a, \multimap s, \multimap i, \multimap \mathrm{k}(i,a), \multimap \mathrm{k}(i,s), \multimap n, \multimap 0\}$$

FIG. 6.3.6 – *Résultat retourné pour le protocole de Yahalom*

l'abstraction a pu perdre de l'information. À l'étape 3 du protocole, a reçoit le message $\langle \{\langle a, \langle k, \langle n, n' \rangle\rangle\rangle\}_{\mathrm{k}(a,s)}, x \rangle$ et renvoie $\langle x, \{n'\}_k \rangle$ (rappelons que, dans notre abstraction, il n'y a plus qu'un seul participant a). Comme on a :

$$\alpha(\langle \{\langle a, \langle k, \langle n, n' \rangle\rangle\rangle\}_{\mathrm{k}(a,s)}, x \rangle) \;=\; \alpha(\langle \{\langle a, \langle n, \langle n', k \rangle\rangle\rangle\}_{\mathrm{k}(a,s)}, x \rangle)$$

a doit renvoyer, dans les exécutions abstraites du protocole, le message $\{k\}_n$. Or n est connu de l'intrus, et par conséquent k n'est plus secret (dans notre abstraction).

6.4 Que montre-t-on réellement ?

Comme nous avons pu le constater dans les sections précédentes, nos abstractions ne permettent pas toujours de montrer certaines propriétés des protocoles, bien qu'elles soient visiblement vraies. Ceci n'est pas surprenant : le problème de départ étant indécidable alors que le problème abstrait ne l'est pas, il est nécessaire de perdre de l'information lors du processus d'abstraction. Dans les travaux consacrés à l'étude de protocoles cryptographiques au moyen d'abstractions, les deux points suivants, concernant une abstraction donnée, sont d'ailleurs toujours énoncés, d'une manière où d'une autre :

– on peut trouver des protocoles P, pour lesquels on sait par ailleurs qu'il satisfont une propriété φ, mais tels que ce résultat ne puisse pas être prouvé au moyen de l'abstraction considérée ;

- les attaques obtenues comme contre-exemples, lorsque la preuve par abstraction d'une propriété φ sur un protocole P échoue, ne correspondent pas toujours à de réelles attaques sur le protocole P.

Pour positiver ces considérations, nous préférons écrire :
- lorsqu'il est possible de prouver une propriété φ d'un protocole P au moyen de l'abstraction, c'est en fait un résultat plus fort que φ qui est prouvé ;
- les attaques obtenues comme contre-exemples, lorsque la preuve par abstraction d'une propriété φ sur un protocole P échoue, sont de réelles attaques, mais dans une modélisation différente de P.

Voyons maintenant ce que nous apportent ces nouvelles considérations dans le cas de nos abstractions. Si on dispose d'un protocole P modélisé par une Σ_P-algèbre $\mathcal{C}_P$ ainsi que d'une $\Sigma_P^\bullet$-algèbre $\mathcal{A}$, correcte relativement à $\mathcal{C}_P$ et un morphisme surjectif $\alpha : [\mathcal{C}_P] \to [\mathcal{A}]$, on peut considérer la congruence induite par α :

$$E_\alpha \;\hat{=}\; \{(c_1, c_2) \in \mathcal{C}_P^2 \mid \alpha(c_1) = \alpha(c_2)\}$$

$[\mathcal{A}]$ est alors le quotient de $[\mathcal{C}_P]$ par E_α et α est la surjection canonique de $[\mathcal{C}_P]$ sur $[\mathcal{C}_P]/E_\alpha$. Pour une propriété de secret, les seuls prédicats utilisés dans $\mathcal{A}$ sont $tr_\oplus$ et $\Vdash_\oplus$. Or, ces prédicats sont définis dans $\mathcal{A}$ exactement de la même manière que tr et $\Vdash$ sont définis dans $\mathcal{C}_P$. Nous aboutissons donc à la première remarque fondamentale suivante : vérifier $\varphi_\ominus$ dans $\mathcal{A}$, c'est vérifier φ dans un monde où les équations de E_α sont satisfaites.

Allons encore un peu plus loin, et considérons une congruence E sur $\mathcal{C}_P$ telle que $E \subseteq E_\alpha$ (par exemple, E pourrait être la congruence sur $\mathcal{C}_P$ engendrée par des équations entre termes de $\mathrm{T}_\Sigma(X)$, à condition que ces équations soient compatibles avec α). On peut écrire α comme composée des surjections canoniques :

$$\alpha_1 : [\mathcal{C}_P] \to [\mathcal{C}_P]/E \quad \text{et} \quad \alpha_2 : [\mathcal{C}_P]/E \to [\mathcal{C}_P]/E_\alpha$$

Définissons $\Vdash$ et tr sur $[\mathcal{C}_P]/E$ exactement de la même manière qu'ils sont définis dans $\mathcal{C}_P$. On obtient alors une Σ_P-structure, $\mathcal{C}_P^E$, et on constate que $\mathcal{A}$ est correcte relativement à $\mathcal{C}_P^E$ et α_2. Notre deuxième remarque fondamentale est donc la suivante : si $\varphi_\ominus$ est vraie sur $\mathcal{A}$, alors φ est vraie sur $\mathcal{C}_P^E$ et ce, quel que soit E contenu dans E_α.

À titre d'exemple, considérons la congruence E engendrée par l'équation suivante :

$$\{\langle m_1, m_2\rangle\}_k \;=\; \langle \{m_1\}_k, \{m_2\}_k\rangle \tag{6.1}$$

Un certain nombre de travaux récents [CS03, CT03, CKRT03] s'intéressent au problème de la vérification de propriétés de protocoles cryptographiques en présence de théories équationnelles sur les messages. L'équation 6.1 apparaît comme un cas particulier d'application des résultats obtenus par les auteurs de [CT03]. Or, dans la structure $\mathcal{A}$ réalisant l'abstraction par linéarisation (section précédente), cette équation est naturellement satisfaite. Au moyen de cette abstraction, nous avons prouvé la correction des protocoles de Denning Sacco et Andrew Secure RPC (pour une propriété de secret). Nous pouvons donc en déduire que le secret est également vérifié en présence de l'équation 6.1. On pourrait faire de même avec les équations :

$$\{\{m\}_{k_1}\}_{k_2} \;=\; \{\{m\}_{k_2}\}_{k_1} \tag{6.2}$$
$$\langle m_1, \langle m_2, m_3\rangle\rangle \;=\; \langle\langle m_1, m_2\rangle, m_3\rangle \tag{6.3}$$
$$\langle m_1, m_2\rangle \;=\; \langle m_2, m_1\rangle \tag{6.4}$$

qui sont également satisfaites dans $\mathcal{A}$.

Par ailleurs, les deux dernières équations sont celles qui nous ont empêché de prouver la propriété de secret dans le cas du protocole de Yahalom (voir plus haut). Il existe donc une attaque dans ce protocole (pour la propriété de secret) en présence de messages pour lesquels le constructeur de couples est associatif et commutatif.

En résumé, nous avons expliqué ici comment mettre en pratique la théorie générale présentée au chapitre précédent, sur trois types d'abstractions. L'abstraction par projection des noms est simple, mais ne permet pas toujours d'obtenir un ensemble d'exécutions finies. L'abstraction par filtrage est également simple et intuitive et permet toujours d'obtenir un ensemble d'exécutions finies. Cependant, elle ne retourne des résultats intéressants que lorsqu'il est possible de faire l'hypothèse que les participants du protocole ont la possibilité de vérifier que les messages qu'ils reçoivent sont de la bonne forme. L'abstraction par linéarisation est moins intuitive, mais conduit toujours à un ensemble fini d'exécutions. Toutefois, elle ne permet de prouver des propriétés que si le protocole considéré est suffisamment robuste vis-à-vis des propriétés algébriques des messages. Par conséquent, lorsque cette abstraction échoue, on obtient en général une attaque sur le protocole qui est valable modulo une certaine théorie équationnelle des messages (pertinente ou non).

Nous revenons maintenant sur un point plus théorique en montrant, dans le chapitre suivant, que la traduction choisie pour obtenir la contrepartie abstraite des énoncés est, en un certain sens, optimale.

Chapitre 7

Optimalité

Nous avons déjà montré, au moyen de la proposition 5.1.1, que la structure $\mathcal{C}^\alpha$ était la plus précise possible qui soit correcte relativement à $\mathcal{C}$ et α. Dans le même ordre d'idées, nous aimerions montrer que $(\varphi_\ominus, \varphi_\oplus)$ constitue le meilleur choix possible d'énoncés tel que :

$$\mathcal{A} \models \varphi_\ominus \ \Rightarrow\ \mathcal{C} \models \varphi \quad \text{et} \quad \mathcal{C} \models \varphi \ \Rightarrow\ \mathcal{A} \models \varphi_\oplus$$

L'exemple suivant montre que, en général, ce n'est pas le cas.

Exemple : Considérons à nouveau la spécification des listes d'entiers, ainsi que l'abstraction qui en a été donnée dans le chapitre 5. L'énoncé $\varphi \mathrel{\hat{=}} \forall l : List.\ succ(0) \in l \vee \neg(succ(0) \in l)$ (qui est bien entendu satisfait par tout modèle de la spécification des entiers) est remplacé au niveau abstrait par les deux énoncés :

$$\begin{aligned} \varphi_\ominus &= \forall l : List.\ succ(0) \in_\ominus l \vee \neg(succ(0) \in_\oplus l) \\ \varphi_\oplus &= \forall l : List.\ succ(0) \in_\oplus l \vee \neg(succ(0) \in_\ominus l) \end{aligned}$$

On avait déjà remarqué que pour $\in_\oplus$ on peut utiliser le prédicat d'appartenance standard, alors qu'il fallait définir le prédicat $\in_\ominus$ au moyen de l'équivalence :

$$x \in_\ominus l \quad \Leftrightarrow \quad x \in l \wedge x = 0$$

Les deux énoncés précédents sont dont équivalents (sur la spécification abstraite des entiers) à :

$$\begin{aligned} &\forall l : List.\ (succ(0) \in l \wedge succ(0) = 0) \vee \neg(succ(0) \in l) \\ &\forall l : List.\ succ(0) \in l \vee \neg(succ(0) \in l \wedge succ(0) = 0) \end{aligned}$$

On en déduit que $\varphi_\ominus$ est faux (toujours sur la spécification abstraite des entiers), alors que φ est vrai (sur la spécification des entiers). Par conséquent, le couple d'énoncés $(\varphi_\ominus, \varphi_\oplus)$ n'est pas optimal vis-à-vis de φ et de l'abstraction considérés, puisque n'importe quel couple d'énoncés valides sur la spécification abstraite des entiers constituerait un meilleur choix. $\square$

Cet exemple montre qu'il existe des cas particuliers où un couple d'énoncés choisi de manière *ad hoc* par rapport au problème d'abstraction et à l'énoncé φ de départ peut constituer un meilleur choix que $(\varphi_\ominus, \varphi_\oplus)$. Cependant, nous allons montrer qu'il n'existe pas de meilleur choix possible parmi ceux qui se calculent de manière purement syntaxique et indépendamment de la signature à partir de l'énoncé de départ.

On voit apparaître ici une notion d'indépendance vis-à-vis de la signature. Or jusqu'à présent on avait toujours considéré des signatures, parfois quelconques, mais toujours fixées. Pour formaliser le concept de « système logique », nous allons, après avoir fait quelques rappels sur la théorie des catégories, présenter les institutions, qui ont été introduites par J. Goguen et R. Burstall dans ce but. Dans ce cadre, notre procédé d'abstractions ainsi que l'hypothèse d'indépendance vis-à-vis de la signature se formulent aisément. Nous pourrons alors énoncer et démontrer notre résultat d'optimalité.

7.1 Catégories et institutions

Nous allons tout d'abord présenter succinctement les catégories, qui formalisent la notion d' « ensemble d'objets muni de morphismes » (on pourra consulter [Mac98, Oos95] pour une présentation complète).

Définition – 7.1.1

Une catégorie $\mathbb{C}$ *est constituée :*

- *d'une classe* $|\mathbb{C}|$ *dont les éléments sont appelés les objets de* $\mathbb{C}$ *;*
- *d'une classe* $\mathrm{Hom}_{\mathbb{C}}$ *dont les éléments sont appelés les morphismes de* $\mathbb{C}$*,*

ces deux classes étant munies des opérations suivantes :

- *le domaine et le codomaine d'un morphisme* $f \in \mathrm{Hom}_{\mathbb{C}}$ *sont des objets de* $\mathbb{C}$*, notés respectivement* $\mathrm{dom}(f)$ *et* $\mathrm{cod}(f)$ *;*
- *la composée de deux morphismes* $f, g \in \mathrm{Hom}_{\mathbb{C}}$ *tels que* $\mathrm{cod}(f) = \mathrm{dom}(g)$ *est un morphisme, noté* $g \circ f$ *dont le domaine est* $\mathrm{dom}(f)$ *et le codomaine est* $\mathrm{cod}(g)$ *;*
- *si* $f, g, h \in \mathrm{Hom}_{\mathbb{C}}$ *sont tels que* $\mathrm{cod}(f) = \mathrm{dom}(g)$ *et* $\mathrm{cod}(g) = \mathrm{dom}(h)$*, alors :*

$$(h \circ g) \circ f \quad = \quad h \circ (g \circ f)$$

- *pour chaque objet* $C \in \mathbb{C}$*, il existe un morphisme* $\mathrm{id}_C \in \mathrm{Hom}_{\mathbb{C}}$ *dont le domaine et le codomaine sont* C *et tel que, quel que soient* $f, g \in \mathrm{Hom}_{\mathbb{C}}$ *avec* $\mathrm{dom}(f) = \mathrm{cod}(g) = C$*, on ait :*

$$f \circ \mathrm{id}_C \ = \ f \quad \textit{et} \quad \mathrm{id}_C \circ g \ = \ g$$

Dans la suite on identifiera $|\mathbb{C}|$ et $\mathbb{C}$ et, C et C' étant deux objets de $\mathbb{C}$, la notation $f : C \to C' \in \mathrm{Hom}_{\mathbb{C}}$ (ou bien $f \in \mathrm{Hom}_{\mathbb{C}}(C, C')$) désignera un morphisme $f \in \mathrm{Hom}_{\mathbb{C}}$ tel que $\mathrm{dom}(f) = C$ et $\mathrm{cod}(f) = C'$.

Exemple : On a déjà rencontré la catégorie $\mathfrak{Alg}(\Theta)$ dont les objets sont les Θ-algèbres et les morphismes sont les morphismes de Θ-algèbres (Θ étant une signature algébrique fixée). Une autre catégorie très utilisée est $\mathbb{S}\mathrm{et}$ dont les objets sont les ensembles et les morphismes sont les fonctions. Définissons maintenant une notion de morphisme entre signatures algébriques afin d'obtenir une catégorie. Soient $\Theta = (S, \Omega)$ et $\Theta' = (S', \Omega')$ deux signatures algébriques. Un morphisme θ de Θ vers Θ' est un couple d'applications $(\theta : S \to S', \theta : \Omega \to \Omega')$ telles que, si $f \in \Omega$ est un symbole de fonction d'arité $((s_1, \ldots, s_n), s_0)$, alors $\theta(f) \in \Omega'$ est un symbole de fonction d'arité $((\theta(s_1), \ldots, \theta(s_n)), \theta(s_0))$. Les signatures algébriques, munies des morphismes ainsi définis, forment une catégorie. On peut faire de même avec les signatures du premier ordre en convenant qu'un morphisme σ entre deux signatures du premier ordre $\Sigma = (S, \Omega, \Pi)$ et $\Sigma' = (S', \Omega', \Pi')$ est un couple composé d'un morphisme $[\sigma]$ entre les signatures algébriques sous-jacentes [Σ] et [Σ′] et d'une application $\sigma : \Pi \to \Pi'$ telle que, si $P \in \Pi$ est un symbole de prédicat d'arité $(s_1, \ldots, s_n)$, alors $\theta(P) \in \Pi'$ est un symbole de prédicat d'arité $(\theta(s_1), \ldots, \theta(s_n))$. La catégorie des signatures du premier ordre ainsi obtenue sera notée Sig. Nous l'utiliserons beaucoup par la suite. □

Remarque : Sig constitue le premier pas vers une présentation de la logique du premier ordre considérant les signatures comme des objets à part entière. Les deux ingrédients suivants sont les structures et les énoncés. Dans le chapitre 4, on a construit sur chaque signature Σ une classe de structures $\mathfrak{Str}(\Sigma)$ ainsi qu'un ensemble d'énoncés $\mathrm{Sen}(\Sigma)$. Afin de s'affranchir de la donnée d'une signature particulière, nous devons présenter ces constructions comme des fonctions qui, à une signature Σ donnée, associent les objets $\mathfrak{Str}(\Sigma)$ et $\mathrm{Sen}(\Sigma)$ qui conviennent. Avant d'aller plus loin, constatons qu'il s'agit de fonctions bien particulières. Soient $\Sigma, \Sigma' \in \mathrm{Sig}$ et soit σ un morphisme de signatures, de Σ vers Σ'. Si $\varphi \in \mathrm{Sen}(\Sigma)$, il est facile d'appliquer σ à chaque sorte et chaque symbole de fonction et de prédicat qui apparaît dans φ afin d'obtenir un énoncé $\varphi' \in \mathrm{Sen}(\Sigma')$. Inversement, si $\mathcal{M}' \in \mathfrak{Str}(\Sigma')$, σ nous permet de définir un modèle $\mathcal{M} \in \mathfrak{Str}(\Sigma)$ de la manière suivante (on note $\Sigma = (S, \Omega, \Pi)$ et $\Sigma' = (S', \Omega', \Pi')$) :

- pour $s \in S$, on pose $\mathcal{M}_s \mathrel{\hat{=}} \mathcal{M}'_{\sigma(s)}$;
- pour $f \in \Omega$, on pose $f^{\mathcal{M}} \mathrel{\hat{=}} \sigma(f)^{\mathcal{M}'}$;
- pour $P \in \Pi$, on pose $P^{\mathcal{M}} \mathrel{\hat{=}} \sigma(P)^{\mathcal{M}'}$.

Nous ne sommes donc pas seulement en présence de fonctions qui, à une signature Σ, associent une classe de structure $\mathfrak{Str}(\Sigma)$ et une classe d'énoncés $\mathrm{Sen}(\Sigma)$, mais nous disposons également de fonctions qui, à un morphisme de signature $\sigma : \Sigma \to \Sigma'$, associent une fonction de $\mathrm{Sen}(\Sigma)$ dans $\mathrm{Sen}(\Sigma')$ et une fonction de $\mathfrak{Str}(\Sigma')$ vers $\mathfrak{Str}(\Sigma)$. Tout ceci nous conduit à la notion de foncteur. □

Définition – 7.1.2

> *Soient $\mathbb{C}$ et $\mathbb{D}$ deux catégories. Un foncteur (respectivement un foncteur contravariant) de $\mathbb{C}$ vers $\mathbb{D}$ est constitué de deux fonctions :*
>
> $$F : |\mathbb{C}| \to |\mathbb{D}| \quad \textit{et} \quad F : \mathrm{Hom}_{\mathbb{C}} \to \mathrm{Hom}_{\mathbb{D}}$$
>
> *telles que :*
> - *quel que soit $f : C \to C' \in \mathrm{Hom}_{\mathbb{C}}$, on a $F(f) : F(C) \to F(C') \in \mathrm{Hom}_{\mathbb{D}}$;*
> - *quels que soient $f, g \in \mathrm{Hom}_{\mathbb{C}}$ avec $\mathrm{cod}(f) = \mathrm{dom}(g)$, on a $F(g \circ f) = F(g) \circ F(f)$ (respectivement $F(g \circ f) = F(f) \circ F(g)$) ;*
> - *quel que soit $C \in \mathbb{C}$, $F(\mathrm{id}_C) = \mathrm{id}_{F(C)}$.*

Exemple : La construction des énoncés et des structures sur des signatures du premier ordre peut être interprétée en termes de foncteurs. En ce qui concerne les énoncés tout d'abord, on dispose d'un foncteur qui :

- à toute signature Σ associe un ensemble $\mathrm{Sen}(\Sigma)$;
- à tout morphisme de signatures $\sigma : \Sigma \to \Sigma'$ associe une fonction de $\mathrm{Sen}(\Sigma)$ vers $\mathrm{Sen}(\Sigma')$, notée (provisoirement) $\mathrm{Sen}(\sigma)$.

On obtient ainsi un foncteur Sen de Sig vers $\mathbb{S}\mathrm{et}$. Pour les structures maintenant, on dispose d'un foncteur qui :

- à toute signature Σ associe $\mathfrak{Str}(\Sigma)$ (qui est une catégorie) ;
- à tout morphisme de signatures $\sigma : \Sigma \to \Sigma'$ associe une fonction de $\mathfrak{Str}(\Sigma')$ vers $\mathfrak{Str}(\Sigma)$, notée (provisoirement là encore) $\mathfrak{Str}(\sigma)$.

Cette fonction $\mathfrak{Str}(\sigma)$ induit un foncteur entre les catégories $\mathfrak{Str}(\Sigma')$ et $\mathfrak{Str}(\Sigma)$ qui associe à un morphisme $\alpha' : \mathcal{M}' \to \mathcal{N}' \in \mathrm{Hom}_{\mathfrak{Str}(\Sigma')}$ un morphisme $\alpha : \mathfrak{Str}(\sigma)(\mathcal{M}') \to \mathfrak{Str}(\sigma)(\mathcal{N}')$ défini de la manière suivante : si s est une sorte de Σ et $m \in \mathfrak{Str}(\sigma)(\mathcal{M}')_s$, alors par définition $m \in \mathcal{M}'_{\sigma(s)}$ et $\alpha'(m) \in \mathcal{N}'_{\sigma(s)}$. Il suffit dont de poser $\alpha(m) = \alpha'(m) \in \mathfrak{Str}(\sigma)(\mathcal{N}')_s$. Par conséquent, $\mathfrak{Str}$ est un foncteur contravariant de Sig vers $\mathbb{C}$at (la catégorie des catégories, dont les objets sont les catégories[1] et les morphismes sont les foncteurs). Dans la suite, pour $\sigma : \Sigma \to \Sigma'$ morphisme de signatures, $\mathcal{M}' \in \mathfrak{Str}(\Sigma')$ et $\varphi \in \mathrm{Sen}(\Sigma)$, $\mathrm{Sen}(\sigma)(\varphi)$ sera noté plus simplement $\sigma(\varphi)$ et $\mathfrak{Str}(\sigma)(\mathcal{M}')$ sera noté $\mathcal{M}'|_\sigma$. □

Remarque : Le dernier ingrédient à faire entrer en jeu pour définir un système logique complet est la relation de satisfaction. Il s'agira plus précisément d'une famille de relation $(\models_\Sigma)_{\Sigma \in \mathrm{Sig}}$. Les relations de cette famille ne sont cependant pas indépendantes. Considérons deux signatures Σ et Σ', ainsi que $\varphi \in \mathrm{Sen}(\Sigma)$ et $\mathcal{M}' \in \mathfrak{Str}(\Sigma')$. Alors :

- si Σ est incluse dans Σ', alors regarder si φ est valide dans $\mathcal{M}'$ doit donner le même résultat que regarder si φ est valide dans $\mathcal{M}$ obtenu à partir de $\mathcal{M}'$ en retirant tous les symboles qui ne sont pas dans Σ. Autrement dit, si on note σ le morphisme de signatures injectif qui représente l'inclusion de Σ dans Σ', on doit avoir équivalence entre $\mathcal{M}'|_\sigma \models'_\Sigma \varphi$ et $\mathcal{M}' \models_\Sigma \sigma(\varphi)$;
- si Σ' est obtenue à partir de Σ en identifiant certains symboles, alors regarder si φ est valide dans $\mathcal{M}'$ modulo identification des symboles en question doit donner le même résultat que regarder si φ est valide dans la structure $\mathcal{M}$ obtenue à partir de $\mathcal{M}'$ en interprétant chaque symbole par l'interprétation qui est donnée à sa classe d'équivalence dans $\mathcal{M}'$. Autrement dit, si on note σ le morphisme de signatures surjectif qui représente cette identification, on doit avoir équivalence entre $\mathcal{M}'|_\sigma \models_{\Sigma'} \varphi$ et $\mathcal{M}' \models_\Sigma \sigma(\varphi)$.

Ayant obtenu une condition reliant $\models_\Sigma$ et $\models_{\Sigma'}$ dans le cas de morphismes injectifs (respectivement surjectifs) entre Σ et Σ', on en déduit que cette condition doit être vérifiée pour tout morphisme de signatures σ. On aboutit ainsi à la notion d'institution [GB92]. □

DÉFINITION – 7.1.3

Une institution est un quadruplet $\mathfrak{I} = (\mathrm{Sig}, \mathrm{Sen}, \mathfrak{Str}, \models)$ *où :*

- Sig *est un catégorie appelée catégorie des signatures de* $\mathfrak{I}$ *;*
- Sen *est un foncteur de* Sig *vers* $\mathbb{S}$et *(pour* $\Sigma \in \mathrm{Sig}$*, les éléments de* $\mathrm{Sen}(\Sigma)$ *sont appelés les énoncés de* $\mathfrak{I}$ *construits sur* Σ*) ;*
- $\mathfrak{Str}$ *est un foncteur contravariant de* Sig *vers* $\mathbb{C}$at *(pour* $\Sigma \in \mathrm{Sig}$*, les objets de* $\mathfrak{Str}(\Sigma)$ *sont appelés les* Σ*-structures de* $\mathfrak{I}$*) ;*
- $\models$ *est une famille* $(\models_\Sigma)_{\Sigma \in \mathrm{Sig}}$ *où pour chaque* $\Sigma \in \mathrm{Sig}$*,* $\models_\Sigma$ *est une relation entre* $\mathfrak{Str}(\Sigma)$ *et* $\mathrm{Sen}(\Sigma)$*, telle que quel que soit* $\sigma : \Sigma \to \Sigma'$ *morphisme de signatures,* $\varphi \in \mathrm{Sen}(\Sigma)$ *et* $\mathcal{M}' \in \mathfrak{Str}(\Sigma')$ *on ait :*

$$\mathcal{M}' \models_{\Sigma'} \sigma(\varphi) \quad \Leftrightarrow \quad \mathcal{M}'|_\sigma \models_\Sigma \varphi$$

(cette dernière condition est appelée condition de satisfaction).

Nous considérerons uniquement, ici et dans la suite, l'institution de la logique de premier ordre. D'autres exemples se trouvent dans [AKKB99].

[1] Plus exactement, pour éviter des problèmes logiques, les petites catégories (c'est à dire, celles dont la classe des objets est un ensemble, ainsi que chaque classe constituée des morphismes entre deux objets.)

7.2 Abstractions et traductions

Appliquons le formalisme des catégories et des institutions aux abstractions du chapitre 5. Nous avions défini une fonction de Sig vers Sig qui à Σ associe $\Sigma^\bullet$. On peut compléter cette définition en la prolongeant aux morphismes de signatures. Soient donc deux signatures $\Sigma = (S, \Omega, \Pi)$ et $\Sigma' = (S', \Omega', \Pi')$, ainsi qu'un morphisme de signatures $\sigma : \Sigma \to \Sigma'$. On définit un morphisme de signatures $\sigma^\bullet : \Sigma^\bullet \to \Sigma'^\bullet$ en posant :

- pour $s \in S$, $\sigma^\bullet(s) \hat{=} \sigma(s)$;
- pour $f \in \Omega$, $\sigma^\bullet(f) \hat{=} \sigma(f)$;
- pour $P \in \Pi$, $\sigma^\bullet(P_\ominus) \hat{=} \sigma(P)_\ominus$ et $\sigma^\bullet(P_\oplus) \hat{=} \sigma(P)_\oplus$.

On obtient ainsi un foncteur noté $(\_)^\bullet$ de Sig vers Sig. Pour chaque signature $\Sigma = (S, \Omega, \Pi)$ on peut également définir un morphisme de signatures de $\Sigma^\bullet$ vers Σ, noté $|\_|_\Sigma$ en posant :

- pour $s \in S$, $|s|_\Sigma \hat{=} s$;
- pour $f \in \Omega$, $|f|_\Sigma \hat{=} f$;
- pour $P \in \Pi$, $|P_\ominus|_\Sigma = |P_\oplus|_\Sigma \hat{=} P$.

Remarquons que, si $\mathcal{C} \in \mathfrak{Str}(\Sigma)$, alors $\mathcal{C}|_{|\_|_\Sigma} \in \mathfrak{Str}(\Sigma^\bullet)$ est la structure $\mathcal{C}^\iota$ (où ι est l'identité de $[\mathcal{C}]$).

Pour un morphisme de signatures $\sigma : \Sigma \to \Sigma'$, le diagramme suivant est alors commutatif :

$$\begin{tikzcd} \Sigma \arrow[r, "\sigma"] & \Sigma' \\ \Sigma^\bullet \arrow[u, "|\_|_\Sigma"] \arrow[r, "\sigma^\bullet"'] & \Sigma'^\bullet \arrow[u, "|\_|_{\Sigma'}"'] \end{tikzcd}$$

Considérons maintenant les signatures $\Sigma^\bullet$ et $\Sigma^{\bullet\bullet}$ obtenues à partir d'une signature $\Sigma = (S, \Omega, \Pi) \in$ Sig. Les constructions précédentes permettent d'exhiber deux morphismes de $\Sigma^{\bullet\bullet}$ vers $\Sigma^\bullet$:

- le premier est simplement $|\_|_{\Sigma^\bullet}$ qui agit sur les symboles de prédicats de $\Sigma^{\bullet\bullet}$ de la manière suivante :

$$\begin{array}{ll} |(P_\ominus)_\ominus|_{\Sigma^\bullet} = P_\ominus & \qquad |(P_\ominus)_\oplus|_{\Sigma^\bullet} = P_\ominus \\ |(P_\oplus)_\ominus|_{\Sigma^\bullet} = P_\oplus & \qquad |(P_\oplus)_\oplus|_{\Sigma^\bullet} = P_\oplus \end{array}$$

  (pour $P \in \Pi$) ;
- le second est obtenu en appliquant le foncteur $(\_)^\bullet$ à $|\_|_\Sigma$ et agit sur un symbole de prédicat de $\Sigma^{\bullet\bullet}$ de la manière suivante :

$$\begin{array}{ll} |(P_\ominus)_\ominus|^\bullet_\Sigma = P_\ominus & \qquad |(P_\ominus)_\oplus|^\bullet_\Sigma = P_\oplus \\ |(P_\oplus)_\ominus|^\bullet_\Sigma = P_\ominus & \qquad |(P_\oplus)_\oplus|^\bullet_\Sigma = P_\oplus \end{array}$$

  (pour $P \in \Pi$).

Si $\mathcal{A} \in \mathfrak{Str}(\Sigma^\bullet)$, alors $\mathcal{A}|_{|\_|_{\Sigma^\bullet}} \in \mathfrak{Str}(\Sigma^{\bullet\bullet})$ est la structure $\mathcal{A}^\iota$ si ι est l'identité de $[\mathcal{A}]$. On sait que cette structure est correcte relativement à $\mathcal{A}$ (nous parlons ici d'abstractions de $\Sigma^\bullet$-structures par des $\Sigma^{\bullet\bullet}$-structures, et non plus d'abstractions de Σ-structures par des $\Sigma^\bullet$-structures). Comme nous aurons besoin, dans la preuve d'optimalité, de réduire $\mathcal{A}$ suivant le morphisme $|\_|^\bullet_\Sigma$, nous allons montrer que cette structure aussi est correcte, relativement à $\mathcal{A}$ et à l'identité (au moins lorsque $\mathcal{A}$ est elle-même une structure correcte relativement à une structure $\mathcal{C}$ et un morphisme α). C'est l'objet du lemme suivant.

Lemme – 7.2.1

Soient $\Sigma = (S, \Omega, \Pi)$ une signature, $\mathcal{A}$ une $\Sigma^\bullet$-structure telle que, pour $P : s_1 \times \ldots \times s_n \in \Pi$ on ait $P_\ominus^\mathcal{A} \subseteq P_\oplus^\mathcal{A}$. On note $|\_|_\Sigma^\bullet$ le morphisme de signature de $\Sigma^{\bullet\bullet}$ vers $\Sigma^\bullet$ obtenu en appliquant le foncteur $(\_)^\bullet$ au morphisme $|\_|_\Sigma$. Si ι est l'identité de $[\mathcal{A}]$, alors $\mathcal{A} \sqsubseteq_\iota \mathcal{A}|_{|\_|_\Sigma^\bullet}$.

Démonstration :

Posons $\mathcal{A}^\bullet = \mathcal{A}|_{|\_|_\Sigma^\bullet}$. Chaque symbole de prédicat P de Σ correspond à deux symboles $P_\ominus$ et $P_\oplus$ dans $\Sigma^\bullet$ et quatre symboles $(P_\ominus)_\ominus$, $(P_\ominus)_\oplus$, $(P_\oplus)_\ominus$ et $(P_\oplus)_\oplus$ dans $\Sigma^{\bullet\bullet}$. Il suffit donc de montrer que :

$$(P_\ominus)_\ominus^{\mathcal{A}^\bullet} \subseteq P_\ominus^\mathcal{A} \subseteq (P_\ominus)_\oplus^{\mathcal{A}^\bullet} \quad \text{et} \quad (P_\oplus)_\ominus^{\mathcal{A}^\bullet} \subseteq P_\oplus^\mathcal{A} \subseteq (P_\oplus)_\oplus^{\mathcal{A}^\bullet}$$

Utilisant la définition du foncteur $(\_)^\bullet$ puis celle de $|\_|_\Sigma$ il vient :

$$\begin{array}{rcccl} |(P_\ominus)_\ominus|_\Sigma^\bullet &=& (|P_\ominus|_\Sigma)_\ominus &=& P_\ominus \\ |(P_\ominus)_\oplus|_\Sigma^\bullet &=& (|P_\ominus|_\Sigma)_\oplus &=& P_\oplus \\ |(P_\oplus)_\ominus|_\Sigma^\bullet &=& (|P_\oplus|_\Sigma)_\ominus &=& P_\ominus \\ |(P_\oplus)_\oplus|_\Sigma^\bullet &=& (|P_\oplus|_\Sigma)_\oplus &=& P_\oplus \end{array}$$

Les interprétations des symboles de prédicats dans $\mathcal{A}^\bullet$ sont données par :

$$\begin{array}{rcccl} (P_\ominus)_\ominus^{\mathcal{A}^\bullet} &=& |(P_\ominus)_\ominus|_\Sigma^{\bullet\,\mathcal{A}} &=& P_\ominus^\mathcal{A} \\ (P_\ominus)_\oplus^{\mathcal{A}^\bullet} &=& |(P_\ominus)_\oplus|_\Sigma^{\bullet\,\mathcal{A}} &=& P_\oplus^\mathcal{A} \\ (P_\oplus)_\ominus^{\mathcal{A}^\bullet} &=& |(P_\oplus)_\ominus|_\Sigma^{\bullet\,\mathcal{A}} &=& P_\ominus^\mathcal{A} \\ (P_\oplus)_\oplus^{\mathcal{A}^\bullet} &=& |(P_\oplus)_\oplus|_\Sigma^{\bullet\,\mathcal{A}} &=& P_\oplus^\mathcal{A} \end{array}$$

Comme on a par hypothèse $P_\ominus^\mathcal{A} \subseteq P_\oplus^\mathcal{A}$, on en déduit que :

$$(P_\ominus)_\ominus^{\mathcal{A}^\bullet} \subseteq (P_\ominus)_\oplus^{\mathcal{A}^\bullet} \quad \text{et} \quad (P_\oplus)_\ominus^{\mathcal{A}^\bullet} \subseteq (P_\oplus)_\oplus^{\mathcal{A}^\bullet}$$

d'où le résultat. ■

7.3 Traductions et optimalité

Commençons par définir les traductions.

Définition – 7.3.1

On appelle traduction une famille $\tau = (\tau_\Sigma)_{\Sigma \in \mathrm{Sig}}$ telle que, pour chaque $\Sigma \in \mathrm{Sig}$, τ_Σ est une application de $\mathrm{Sen}(\Sigma)$ vers $\mathrm{Sen}(\Sigma^\bullet) \times \mathrm{Sen}(\Sigma^\bullet)$. Pour $\varphi \in \mathrm{Sen}(\Sigma)$, on posera $\tau_\Sigma(\varphi) = (\tau_\Sigma^1(\varphi), \tau_\Sigma^2(\varphi))$.

Seules les traductions satisfaisant une certaine condition de correction peuvent être utilisées dans des raisonnements par abstraction.

Définition – 7.3.2

Une traduction τ est dite correcte si, pour tout $\Sigma \in \mathrm{Sig}$, pour toute structure $\mathcal{A} \in \mathfrak{Str}(\Sigma^\bullet)$ correcte relativement à $\mathcal{C} \in \mathfrak{Str}(\Sigma)$ et un morphisme surjectif α et pour tout $\varphi \in \mathrm{Sen}(\Sigma)$, on a :

$$\mathcal{A} \models \tau^1_\Sigma(\varphi) \;\Rightarrow\; \mathcal{C} \models \varphi \quad \textit{et} \quad \mathcal{C} \models \varphi \;\Rightarrow\; \mathcal{A} \models \tau^2_\Sigma(\varphi)$$

Finalement, l'hypothèse informelle de traduction indépendante vis-à-vis de la signature est capturée par la définition suivante.

Définition – 7.3.3

Une traduction τ est dite naturelle si, quels que soient $\Sigma, \Sigma' \in \mathrm{Sig}$, $\sigma : \Sigma \to \Sigma'$ morphisme de signatures et $\varphi \in \mathrm{Sen}(\Sigma)$ on a :

$$\tau_{\Sigma'}(\sigma(\varphi)) \quad = \quad \sigma^\bullet(\tau_\Sigma(\varphi))$$

c'est à dire si, pour chaque $\sigma : \Sigma \to \Sigma'$ morphisme de signatures, le diagramme suivant est commutatif :

$$\begin{tikzcd}
\Sigma \arrow[d, "\sigma"'] & \mathrm{Sen}(\Sigma) \arrow[r, "\tau_\Sigma"] \arrow[d, "\sigma"'] & \mathrm{Sen}(\Sigma^\bullet)^2 \arrow[d, "\sigma^\bullet \times \sigma^\bullet"] \\
\Sigma' & \mathrm{Sen}(\Sigma') \arrow[r, "\tau_{\Sigma'}"'] & \mathrm{Sen}(\Sigma'^\bullet)^2
\end{tikzcd}$$

où $\sigma^\bullet \times \sigma^\bullet$ désigne l'application qui à $(\varphi, \psi) \in \mathrm{Sen}(\Sigma^\bullet)^2$ associe $(\sigma^\bullet(\varphi), \sigma^\bullet(\psi))$.

Remarque : Soient F et G deux foncteurs, tous deux d'une catégorie $\mathbb{C}$ vers une catégorie $\mathbb{D}$. Une transformation naturelle τ de F vers G associe à chaque $C \in \mathbb{C}$ un morphisme $\tau_C : F(C) \to G(C) \in \mathrm{Hom}_\mathbb{D}$ tel que, quels que soient $C, C \in \mathbb{C}$, $f : C \to C' \in \mathrm{Hom}_\mathbb{C}$ le diagramme :

$$\begin{tikzcd}
F(C) \arrow[r, "\tau_C"] \arrow[d, "F(f)"'] & G(C) \arrow[d, "G(f)"] \\
F(C') \arrow[r, "\tau_{C'}"'] & G(C')
\end{tikzcd}$$

soit commutatif. Une traduction τ est alors naturelle si, et seulement si, elle définit une transformation naturelle du foncteur Sen vers le foncteur $\mathrm{Sen}^\bullet$ (ce dernier agit sur Sig de la manière suivante : à $\Sigma \in \mathrm{Sig}$, il associe $\mathrm{Sen}(\Sigma^\bullet)^2$ et à σ, morphisme de signatures, il associe $\sigma^\bullet \times \sigma^\bullet$). □

Nous sommes maintenant prêts pour énoncer et démontrer notre résultat d'optimalité.

Théorème – 7.3.1

Soit τ une traduction correcte et naturelle. Soient $\Sigma = (S, \Omega, \Pi) \in \mathrm{Sig}$ et $\mathcal{A} \in \mathfrak{Str}(\Sigma^\bullet)$ telle que, pour $P : s_1 \times \ldots \times s_n \in \Pi$ on ait $P^\mathcal{A}_\ominus \subseteq P^\mathcal{A}_\oplus$. Quel que soit $\varphi \in \mathrm{Sen}(\Sigma)$ on a :

$$\mathcal{A} \models \tau^1_\Sigma(\varphi) \;\Rightarrow\; \mathcal{A} \models \varphi_\ominus \quad \textit{et} \quad \mathcal{A} \models \varphi_\oplus \;\Rightarrow\; \mathcal{A} \models \tau^2_\Sigma(\varphi)$$

(ce résultat s'applique en particulier lorsque $\mathcal{A}$ est correcte relativement à $\mathcal{C} \in \mathfrak{Str}(\Sigma)$ et un morphisme surjectif α).

Démonstration :

On note $|\_|$ le morphisme de $\Sigma^\bullet$ vers Σ et $|\_|^\bullet$ le morphisme obtenu en lui appliquant le foncteur $(\_)^\bullet$. Comme $\varphi = |\varphi_\ominus|$, on a :

$$\mathcal{A} \models \tau^1_\Sigma(\varphi) \quad\Leftrightarrow\quad \mathcal{A} \models \tau^1_\Sigma(|\varphi_\ominus|)$$

Maintenant, appliquant le fait que τ est naturelle sur le morphisme $|\_|$ il vient :

$$\mathcal{A} \models \tau^1_\Sigma(\varphi) \quad\Leftrightarrow\quad \mathcal{A} \models |\tau^1_{\Sigma^\bullet}(\varphi_\ominus)|^\bullet$$

La condition de satisfaction de la définition 7.1.3 nous donne :

$$\mathcal{A} \models \tau^1_\Sigma(\varphi) \quad\Leftrightarrow\quad \mathcal{A}|_{|\_|^\bullet} \models \tau^1_{\Sigma^\bullet}(\varphi_\ominus)$$

Le lemme 7.2.1 nous assurant que $\mathcal{A}|_{|\_|^\bullet}$ est correcte relativement à $\mathcal{A}$ et l'identité et τ étant correcte, on a :

$$\mathcal{A}|_{|\_|^\bullet} \models \tau^1_{\Sigma^\bullet}(\varphi_\ominus) \quad\Rightarrow\quad \mathcal{A} \models \varphi_\ominus$$

ce qui donne finalement :

$$\mathcal{A} \models \tau^1_\Sigma(\varphi) \quad\Rightarrow\quad \mathcal{A} \models \varphi_\ominus$$

On raisonne de la même manière pour montrer que :

$$\mathcal{A} \models \varphi_\oplus \quad\Rightarrow\quad \mathcal{A} \models \tau^2_\Sigma(\varphi)$$

■

7.4 Conclusion

Nous avons considéré, dans cette partie, les propriétés des protocoles cryptographiques qui peuvent s'exprimer sur les traces des exécutions de ces protocoles. Vis-à-vis de ce choix, une modélisation des protocoles et des propriétés dans le cadre de la logique du premier ordre nous a semblé pertinente. Nous avons donc associé à un protocole cryptographique de la classe considérée une signature du premier ordre, ainsi qu'une structure sur cette signature. Les propriétés de ces protocoles ont alors pu être énoncées au moyen d'énoncés de la logique du premier ordre. Nous avons ensuite mis en place un cadre général fournissant des abstractions correctes pour les structures et les énoncés du premier ordre et nous avons instancié cette approche générale en donnant des abstractions pertinentes pour l'étude des protocoles cryptographiques. Nous avons discuté ces abstractions au moyen d'exemples et des résultats fournis par l'implémentation. Nous avons enfin, dans ce chapitre, énoncé et démontré un résultat d'optimalité sur l'abstraction des énoncés. Nous suivons, dans les parties suivantes, la même approche, mais appliquée à des propriétés (et par conséquent des modélisations) différentes des protocoles. Signalons quelques points qui mériteraient, à notre avis, d'être étudiés. Le premier concerne ce qui vient d'être fait ici sur l'optimalité et les deux autres concernent l'étude pratique des protocoles cryptographiques au moyen d'abstractions.

Sur l'optimalité. Nous avons réalisé nos abstractions et prouvé leur optimalité dans le cadre de la logique du premier ordre. Autrement dit, utilisant le langage introduit dans ce chapitre, nous avons travaillé dans l'institution de la logique du premier ordre. On peut alors naturellement se demander si ce travail peut être étendu à d'autres institutions. Il semble cependant difficile d'élaborer une théorie non triviale des abstractions dans une institution quelconque, sans plus d'hypothèses. Cependant, il existe des formalismes dont le but est de décrire des institutions [MTP97, Paw01]. Ces formalismes donnent d'ailleurs un rôle central à une variante de la logique du premier ordre que nous avons utilisée dans cette partie, ce qui pourrait conférer une portée plus générale à nos résultats.

Produit d'abstractions. Il existe une notion très simple de produit sur les abstractions, permettant d'obtenir de meilleurs résultats que chaque abstraction prise séparément. Pour définir ce produit, considérons une Σ-structure $\mathcal{C}$ ainsi que deux $\Sigma^\bullet$-structures $\mathcal{A}_1$ et $\mathcal{A}_2$, correctes relativement à $\mathcal{C}$ et à des morphismes surjectifs α_1 et α_2, respectivement. La fonction :

$$\begin{array}{rccl} \alpha : & [\mathcal{C}] & \to & [\mathcal{A}_1] \times [\mathcal{A}_2] \\ & c & \mapsto & (\alpha_1(c), \alpha_2(c)) \end{array}$$

est alors un morphisme de $[\Sigma]$-algèbres, qui induit un morphisme surjectif entre $[\mathcal{C}]$ et $\mathcal{B}$ (où $\mathcal{B}$ désigne l'image de α). Pour faire de $\mathcal{B}$ une Σ-structure $\mathcal{A}$, il suffit de définir l'interprétation des symboles de prédicat $P_\ominus$ et $P_\oplus$ pour chaque symbole P apparaissant dans Σ. Faisons-le simplement dans le cas où $P : s$ est un prédicat unaire. On pose, pour $(a_1, a_2) \in [\mathcal{A}]_s = \mathcal{B}_s \subseteq (\mathcal{A}_1)_s \times (\mathcal{A}_2)_s$:

$$\begin{array}{rcl} P_\ominus^{\mathcal{A}}(a_1, a_2) & \hat{=} & P_\ominus^{\mathcal{A}_1}(a_1) \vee P_\ominus^{\mathcal{A}_2}(a_2) \\ P_\oplus^{\mathcal{A}}(a_1, a_2) & \hat{=} & P_\oplus^{\mathcal{A}_1}(a_1) \wedge P_\oplus^{\mathcal{A}_2}(a_2) \end{array}$$

On vérifie alors sans peine que $\mathcal{A}$ est correcte relativement à $\mathcal{C}$ et α et qu'elle permet de prouver (respectivement de réfuter) tout ce qui était prouvable (respectivement réfutable) dans $\mathcal{C}$ au moyen de $\mathcal{A}_1$ ou $\mathcal{A}_2$. Pour revenir à une application concrète, faire le produit des abstractions par filtrage et par linéarisation permettrait de traiter tous les protocoles (et les propriétés) qui peuvent être traités par au moins une des deux abstractions. Nous conjecturons que cette abstraction donnerait de bons résultats sur une large classe de protocoles, mais nous ne l'avons pas encore implémentée.

Abstractions et classes décidables. La question que nous allons évoquer ici trouve son origine dans une discussion avec Y. Lakhnech. Les instances du problème de la vérification de propriétés de protocoles cryptographiques sont, dans cette partie, les couples (P, φ) où P est un processus, élément de $\mathrm{Proc}_{\mathrm{Tr}}$, et φ est un énoncé du premier ordre sur la signature Σ_P. On pose :

$$\mathit{Inst} \quad \hat{=} \quad \{(P, \varphi) \mid P \in \mathrm{Proc}_{\mathrm{Tr}} \text{ et } \varphi \in \mathrm{Sen}(\Sigma_P)\}$$

Le problème considéré est de savoir, pour $(P, \varphi) \in \mathit{Inst}$ donné, si $\mathcal{C}_P \models \varphi$ et nous avons choisi de répondre à cette question au moyen d'abstractions. Les abstractions que nous avons choisies (par exemple l'abstraction par linéarisation) se présentent, pour chaque processus P, sous la forme d'une $\Sigma_P^\bullet$-structure $\mathcal{A}_P$, correcte relativement à $\mathcal{C}_P$ et un morphisme surjectif α_P. Relativement à un tel choix d'abstraction, on pose :

$$\mathit{Inst}^\alpha \quad \hat{=} \quad \{(P, \varphi) \in \mathit{Inst} \mid (\mathcal{C}_P \models \varphi \text{ et } \mathcal{A}_P \models \varphi_\ominus) \text{ ou } \mathcal{C}_P \not\models \varphi\}$$

Comme les abstractions que nous avons considérées conduisent à des problèmes décidables, $Inst^\alpha$ apparaît alors comme une sous-classe décidable de $Inst$, ce qui est surtout d'un intérêt théorique, puisqu'il semble difficile de donner une définition explicite de cette classe sans faire référence à l'abstraction choisie. On peut par contre être tenté d'inverser le raisonnement : on part d'une sous-classe décidable $Inst' \subseteq Inst$ (par exemple celle induite par les résultats de [CCM01]), est-il possible d'en déduire une abstraction pour laquelle on aurait $Inst' \subseteq Inst^\alpha$?

Deuxième partie

Propriétés de jeux

Chapitre 8

Introduction

8.1 Un exemple

Supposons que deux participants A et B désirent échanger leurs signatures sur un texte de contrat C (autrement dit, A et B se sont mis d'accord sur un message C et désirent s'échanger les messages $\mathrm{sig}_A(C)$ et $\mathrm{sig}_B(C)$ qui désignent les messages obtenus en appliquant à C les signatures de A et B). La manière la plus simple de procéder est probablement la suivante :

$$\begin{array}{lcccl} 1. & A & \rightarrow & B & : \mathrm{sig}_A(C) \\ 2. & B & \rightarrow & A & : \mathrm{sig}_B(C) \end{array}$$

Le problème est le suivant : après la première étape, B possède un avantage sur A. Il peut par exemple montrer à une troisième personne que A est prêt à conclure un contrat avec lui, sans pour autant s'engager lui-même. En fait, pour réaliser électroniquement une signature de contrat, il est nécessaire d'introduire une dissymétrie, qui a pour effet de désavantager l'un des participants (typiquement, le premier à signer le contrat). Ce type de problème ne concerne pas seulement les protocoles de signature de contrats, mais concerne plus généralement tous les protocoles d'échange électronique. Citons par exemple :

- le commerce électronique ;
- les protocoles de non-répudiation : il s'agit pour A d'envoyer à B une donnée de nature électronique tout en assurant que, dans le futur, A ne pourra pas nier avoir envoyé la donnée de même que B ne pourra pas nier l'avoir reçue ;
- le courrier électronique avec preuve de réception ;
- ...

De tels protocoles doivent permettre d'échanger des données tout en assurant une propriété d'équité (qui doit permettre d'éviter qu'un des participants ne soit désavantagé par rapport à l'autre). En 1980, S. Even et Y. Yacobi ont montré qu'il n'existait pas de protocole, déterministe et ne faisant pas appel à un tiers de confiance, pour résoudre ce problème [EY80]. Les protocoles d'échange équitable sont donc de deux sortes. D'une part les protocoles non-déterministes qui sont soit de nature probabiliste, soit basé sur un échange graduel de données. D'autre part les protocoles faisant appel à un tiers de confiance. Nous n'étudierons dans la suite que ces derniers, et nous nous concentrerons même sur un sous-ensemble des protocoles d'échange avec tiers de confiance constitué des protocoles dits optimistes. Constatons en effet que, si A et B se sont mis d'accord sur un contrat C, ainsi que sur l'identité T d'un tiers de confiance (que nous appellerons TTP dans la suite), le protocole suivant permet d'assurer un échange équitable :

$$\begin{array}{llllll} 1. & A & \rightarrow & T & : & \{\mathrm{sig}_A(C)\}_{\mathrm{k}^{+1}(T)} \\ 2. & B & \rightarrow & T & : & \{\mathrm{sig}_B(C)\}_{\mathrm{k}^{+1}(T)} \\ 3. & T & \rightarrow & A & : & \mathrm{sig}_B(C) \\ 4. & T & \rightarrow & B & : & \mathrm{sig}_A(C) \end{array}$$

Ce protocole a cependant le défaut de toujours faire intervenir la TTP , même dans le cas où A et B sont tous deux honnêtes. Il se forme alors un goulot d'étranglement au niveau de la TTP qui diminue l'efficacité de cette approche. Les protocoles optimistes ont pour but de permettre aux participants de n'avoir recours à la TTP qu'en cas de litige. C'est relativement délicat puisqu'il faut par conséquent que A et B s'échangent suffisamment d'informations pour pouvoir convaincre T qu'ils sont impliqués dans le protocole, mais pas suffisamment pour qu'un observateur extérieur puisse être convaincu qu'ils sont impliqués. Dans [GJM99], J. Garay, M. Jakobsson et P. MacKenzie ont proposé une construction particulière, appelée *private contract signature* (nous utiliserons le terme de présignature) et possédant les propriétés suivantes :

- la présignature d'un contrat C par un participant A se fait relativement à l'autre participant, B, et à la TTP , T. On la note $\mathrm{pcs}_A(C,B,T)$;
- il est possible, pour B, de produire un message indistinguable de $\mathrm{pcs}_A(C,B,T)$ pour un observateur extérieur (*i.e.* quelqu'un qui n'est ni A, ni B, ni T). Ce message sera noté $\mathrm{pcs}'_A(C,B,T)$;
- $\mathrm{pcs}_A(C,B,T)$ peut être converti par T en une signature classique $\mathrm{sig}_A^T(C)$.

La distinction entre $\mathrm{sig}_A(C)$ et $\mathrm{sig}_A^T(C)$ n'est importante que pour étudier des propriétés particulières de la TTP . Nous supposerons dans la suite, pour plus de simplicité, que l'action de la TTP est transparente, *i.e.* $\mathrm{sig}_A(C) = \mathrm{sig}_A^T(C)$. Notre approche s'étendrait sans difficultés au cas où ces constructions sont distinctes. Le protocole proposé dans [GJM99] commence de la manière suivante :

<u>GJM</u> :

$$\begin{array}{llllll} 1. & A & \rightarrow & B & : & \mathrm{pcs}_A(C,B,T) \\ 2. & B & \rightarrow & A & : & \mathrm{pcs}_B(C,A,T) \\ 3. & A & \rightarrow & B & : & \mathrm{sig}_A(C) \\ 4. & B & \rightarrow & A & : & \mathrm{sig}_B(C) \end{array}$$

Voyons comment interviennent dans ce protocole les propriétés des présignatures. Tout d'abord, après la première étape, B ne peut pas prouver à un observateur extérieur que A est prêt à signer le contrat C avec lui. En effet, pour l'observateur extérieur, le message $\mathrm{pcs}_A(C,B,T)$ envoyé par A à B est indistinguable de $\mathrm{pcs}'_A(C,B,T)$ que B aurait pu produire lui-même. Après la troisième étape, B est en mesure de montrer $\mathrm{sig}_A(C)$ à l'observateur, ce qui prouve que A est impliqué dans le contrat. Mais comme A possède $\mathrm{pcs}_B(C,A,T)$, il peut faire appel à T pour obtenir $\mathrm{sig}_B(C)$, autrement dit B est également impliqué dans le contrat. Pour faire appel à T, A utilise le protocole suivant :

<u>Recouvrement (pour A)</u> :

$$\begin{array}{llllll} 1. & A & \rightarrow & T & : & \langle \mathrm{pcs}_B(C,A,T), \mathrm{sig}_A(C)\rangle \\ 2. & T & \rightarrow & A & : & \text{if } (\mathit{aborted}) \\ & & & & & \text{then } \mathrm{sig}_T(\mathrm{sig}_A(\langle C,A,B,\mathit{abort}\rangle)) \\ & & & & & \text{else } \mathrm{sig}_B(C) \end{array}$$

Dans ce sous-protocole, A demande à T de convertir $\mathrm{pcs}_B(C,A,T)$ en $\mathrm{sig}_B(C)$. T répond à cette requête, mais uniquement dans le cas où il n'a pas déjà reçu une demande d'abandon.

L'abandon permet au participant A de s'assurer que T ne convertira plus des présignatures en signatures concernant le contrat C. Il est utilisé par A dans le cas où il ne reçoit pas de B le message prévu à la deuxième étape du protocole :

Abandon (pour A) :

$$\begin{array}{llllll} 1. & A & \rightarrow & T & : & \mathrm{sig}_A(\langle C, A, B, abort\rangle) \\ 2. & T & \rightarrow & A & : & \text{if } (resolved) \\ & & & & & \text{then } \mathrm{sig}_B(C) \\ & & & & & \text{else } \mathrm{sig}_T(\mathrm{sig}_A(\langle C, A, B, abort\rangle)) \end{array}$$

Le message $\mathrm{sig}_T(\mathrm{sig}_A(\langle C, A, B, abort\rangle))$ est la marque d'abandon envoyée par T, c'est en fait une promesse de la part de T de ne plus convertir les présignatures en signatures. Remarquons que B n'a pas besoin d'un tel protocole, puisqu'il lui suffit de ne pas envoyer le deuxième message. Par contre, il lui faut un protocole pour demander à T de convertir une présignature en signature :

Recouvrement (pour B) :

$$\begin{array}{llllll} 1. & B & \rightarrow & T & : & \langle \mathrm{pcs}_A(C, B, T), \mathrm{sig}_B(C)\rangle \\ 2. & T & \rightarrow & B & : & \text{if } (aborted) \\ & & & & & \text{then } \mathrm{sig}_T(\mathrm{sig}_A(\langle C, A, B, abort\rangle)) \\ & & & & & \text{else } \mathrm{sig}_A(C) \end{array}$$

Décrivons maintenant informellement quelques propriétés que l'on peut attendre d'un tel protocole :

- *(fairness)* le protocole est dit équitable s'il est impossible à un participant corrompu d'obtenir un contrat signé valide sans permettre aux autres de faire de même ;
- *(timeliness)* le protocole est dit convergeant si à tout moment tout participant peut atteindre un point où le protocole est (de son point de vue) terminé, tout en respectant l'équité ;
- *(viability)* le protocole est dit viable s'il est possible à des participants honnêtes de conclure le protocole sans intervention de la TTP ;
- *(abuse-freeness)* le protocole ne permet pas les abus de confiance si aucun participant n'a la possibilité de prouver à un observateur extérieur qu'il est, avec les autres participants, impliqué dans un protocole de signature de contrat dont il peut déterminer l'issue.

Pour mettre l'accent sur les différences qui existent entre ces protocoles et ceux de la première partie, nous dirons donc que :

- ils font intervenir de nouvelles constructions dans les messages (il nous faudra donc étendre la grammaire définissant l'ensemble Msg des messages) ;
- ces protocoles ne sont pas linéaires mais branchants : les participants peuvent à certains moments choisir de lancer des sous-protocoles, le processus qui les décrit n'est donc plus une séquence d'envois et réceptions de messages, mais un arbre ;
- la notion d'intrus actif est à remplacer par celle de participant potentiellement malhonnête, les propriétés attendues montrent d'ailleurs clairement qu'une des notions importantes à prendre en compte est ce que peuvent obtenir ou ne pas obtenir les différents participants.

8.2 Travaux existants

On peut citer trois types d'approches concernant la vérification formelle de protocoles optimistes d'échange. Dans chaque approche, le protocole est modélisé par un système de transitions. Dans les travaux de V. Shmatikov et J. Mitchell [SM00a, SM00b, SM02], il s'agit de systèmes de transitions ordinaires, finis, qui sont étudiés à l'aide de l'outil de vérification MURφ. R. Chadha, M. Kanovich et A. Scedrov utilisent un formalisme de réécriture [CKS01], ce qui leur permet d'utiliser des systèmes de transitions infinis, en contrepartie les preuves doivent être faites à la main. Les propriétés qu'ils énoncent s'écrivent en termes de jeux, alors qu'avec MURφ il est simplement possible de vérifier que tout état atteignable satisfait une certaine spécification. La troisième approche est celle de S. Kremer et J.-F. Raskin [KR01, KR02]. Elle utilise comme modèles des systèmes de transitions alternés et les propriétés attendues sont énoncées dans le langage de la logique temporelle alternée [AHK98]. Ce formalisme étend les systèmes de transitions classiques avec des concepts de théorie des jeux. Il en résulte que le modèle est simple à comprendre (c'est un système de transitions dans lequel le comportement des différents participants est représenté) et la vérification automatique d'un protocole donné possible, en utilisant l'outil MOCHA. Un des inconvénients majeur était que l'étape de transformation d'un protocole en un système de transitions alterné était effectuée manuellement et incluait des simplifications qui demandaient une justification à part. Notre travail, en collaboration avec S. Kremer et J.-F. Raskin, avait essentiellement pour but de réaliser et justifier automatiquement ces simplifications, au moyen de techniques d'interprétation abstraite. Les concepts mis en jeu étaient déjà présents dans la littérature, mais il a fallu les adapter à cette approche, que nous détaillons dans la section suivante. La plupart des travaux associés à ce type de protocole doivent prendre en compte le fait que les participants au protocole, ainsi que les canaux de communication, peuvent fonctionner suivant plusieurs niveaux d'honnêteté. Ainsi, un participant peut être :

- honnête : il suit le protocole ;
- faiblement malhonnête : il ne suit pas le protocole et peut utiliser les messages reçus pour en construire de nouveaux et les envoyer ;
- fortement malhonnête : il peut de plus espionner tous les canaux de communication.

De même, un canal de communication peut être :

- opérationnel : il délivre immédiatement les messages envoyés ;
- résistant : il délivre les messages envoyés en un temps fini (mais non borné) ;
- non-fiable : il ne délivre pas nécessairement les messages.

Vu la forme des propriétés attendues, il semble évident que si tous les canaux sont non-fiables, on ne pourra rien prouver. L'hypothèse minimale typique à faire sur les canaux de communication est la suivante : des canaux non-fiables entre A et B et des canaux résistants entre A et T et B et T. Le fait de distinguer plusieurs types de comportements malhonnêtes pour les participants permet de faire des analyses plus fines. Par exemple, l'attaque sur le protocole précédent découverte par V. Shmatikov et J. Mitchell existe dans notre modèle uniquement en présence d'un participant fortement malhonnête.

8.3 Notre approche

Comme signalé plus haut, nous avons, en collaboration avec S. Kremer et J.-F. Raskin, développé une approche permettant de justifier et d'automatiser la méthode utilisée dans

[KR01, KR02], à base d'interprétation abstraite. Cette approche fait appel à un certain nombre de concepts qui étaient présents dans la littérature mais que nous avons adapté et combiné. Citons-les brièvement :

- les systèmes de transitions alternés et la logique temporelle alternée ont été décrits pour la première fois dans [AHK98]. Pour les utiliser dans le cadre des protocoles cryptographiques, il faut utiliser une variante de ces systèmes qui autorise la présence de contraintes d'équité (cette variante était déjà présente dans [AHK98] et utilisée dans [KR01, KR02]). Les systèmes de transitions alternés et la logique temporelle alternée que nous décrirons au chapitre 9 incluront directement ces contraintes d'équité ;
- la modélisation de protocoles optimistes d'échange a été proposée pour la première fois dans [KR01, KR02]. Nous avons supprimé de cette modélisation toutes les simplifications manuelles qu'elle comportait, afin d'obtenir un système de transitions alterné, certes infini, mais fournissant une sémantique convaincante pour les protocoles ;
- la théorie de l'interprétation abstraite des systèmes de transitions alternés a été présentée dans [HMMR00] mais cette présentation ne tenait pas compte des contraintes d'équité. Nous avons donc étendu cette théorie au cas où les systèmes de transitions alternés possèdent des contraintes d'équité et nous présentons la théorie obtenue au chapitre 11. Notons que les preuves de corrections que nous présentons sont différentes de celles de [HMMR00]. Dans cet article, les auteurs travaillaient sur l'algorithme de *model-checking* et en donnaient une version abstraite dont ils justifiaient la correction. Nous avons préféré travailler directement sur la sémantique formelle de la logique temporelle alternée sans faire référence au *model-checking*. Une dernière différence est que la théorie présentée dans [HMMR00] vaut pour le μ-calcul alternant, qui contient strictement la logique temporelle alternée. Nous avons, pour notre part, présentés les résultats concernant les abstractions dans le cadre, plus simple, de cette logique ;
- en ce qui concerne la vérification effective, nous utilisons l'outil de *model-checking* [Moc] adapté par S. Kremer et J.-F. Raskin pour prendre en compte les contraintes d'équité.

Signalons aussi que, comme dans les autres approches formelles traitant ce type de protocoles, nous ne modéliserons que les exécutions d'une seule session. Les noms de participant et le texte du contrat sont donc des constantes et on les désignera par conséquent par des lettres minuscules.

8.4 Syntaxe pour les protocoles d'échange

Tout d'abord, il faut étendre la grammaire définissant les messages pour prendre en compte les signatures et présignatures. On ajoute donc les règles suivantes :

$$\begin{array}{rcl} m, m', \ldots \in \mathrm{Msg} & ::= & \ldots \\ & | & \mathrm{sig}_n(m) \\ & | & \mathrm{pcs}_{n_1}(m, n_2, n_3) \\ & | & \mathrm{pcs'}_{n_1}(m, n_2, n_3) \end{array}$$

Noter la présence de deux constructions distinctes, $\mathrm{pcs}_{n_1}(m, n_2, n_3)$ et $\mathrm{pcs'}_{n_1}(m, n_2, n_3)$ pour les présignatures. La première correspond à la présignature par n_1, réellement faite par n_1 alors que la seconde est la présignature de n_1 réalisée par n_2. Pour un observateur extérieur (*i.e.* ni n_1, ni n_2, ni n_3) on a $\mathrm{pcs}_{n_1}(m, n_2, n_3) = \mathrm{pcs'}_{n_1}(m, n_2, n_3)$. Bien entendu, seul le message $\mathrm{pcs}_{n_1}(m, n_2, n_3)$ peut être converti en $\mathrm{sig}_{n_1}(m)$ par n_3. Ayant étendu l'ensemble des messages, il faut adapter la définition de la relation $\Vdash$ entre ensembles de messages et messages (chapitre 1). Nous ajoutons donc les règles suivantes à celles qui se trouvent page 19 :

$$\frac{s \Vdash \mathrm{sig}_n(m)}{s \Vdash n} \qquad \frac{s \Vdash \mathrm{sig}_n(m)}{s \Vdash m}$$

$$\frac{s \Vdash \mathrm{pcs}_{n_1}(m,n_2,n_3)}{s \Vdash n_1} \qquad \frac{s \Vdash \mathrm{pcs}_{n_1}(m,n_2,n_3)}{s \Vdash n_2} \qquad \frac{s \Vdash \mathrm{pcs}_{n_1}(m,n_2,n_3)}{s \Vdash n_3} \qquad \frac{s \Vdash \mathrm{pcs}_{n_1}(m,n_2,n_3)}{s \Vdash m}$$

$$\frac{s \Vdash \mathrm{pcs'}_{n_1}(m,n_2,n_3)}{s \Vdash n_1} \qquad \frac{s \Vdash \mathrm{pcs'}_{n_1}(m,n_2,n_3)}{s \Vdash n_2} \qquad \frac{s \Vdash \mathrm{pcs'}_{n_1}(m,n_2,n_3)}{s \Vdash n_3} \qquad \frac{s \Vdash \mathrm{pcs'}_{n_1}(m,n_2,n_3)}{s \Vdash m}$$

$$\frac{s \Vdash m \quad s \Vdash \mathrm{k}^{-1}(n)}{s \Vdash \mathrm{sig}_n(m)} \qquad \frac{s \Vdash \mathrm{pcs}_{n_1}(m,n_2,n_3) \quad s \Vdash \mathrm{k}^{-1}(n_3)}{s \Vdash \mathrm{sig}_{n_1}(m)}$$

$$\frac{s \Vdash m \quad s \Vdash \mathrm{k}^{-1}(n_1) \quad s \Vdash n_2 \quad s \Vdash n_3}{s \Vdash \mathrm{pcs}_{n_1}(m,n_2,n_3)} \qquad \frac{s \Vdash m \quad s \Vdash n_1 \quad s \Vdash \mathrm{k}^{-1}(n_2) \quad s \Vdash n_3}{s \Vdash \mathrm{pcs'}_{n_1}(m,n_2,n_3)}$$

Commentons ces nouvelles règles. Elles se séparent en deux groupes : les règles d'analyse (trois premières lignes) et de synthèse (deux dernières). Les deux règles de la première ligne expriment qu'il est possible de retrouver le signataire et le texte du contrat à partir du texte signé. La deuxième et la troisième ligne expriment qu'il en est de même avec les présignatures (y compris les fausses présignatures). La quatrième ligne traduit le fait que, pour signer un message sous le nom de n_1, il faut connaître la clé privée de n_1. Toutefois, si on dispose d'une présignature et de la clé privée du tiers de confiance à qui il est fait référence dans cette celle-ci, on peut la convertir en une signature conventionnelle. La dernière ligne explique qu'il faut connaître la clé privée de n_1 pour faire une présignature en son nom, où à défaut celle de n_2 pour faire une fausse présignature, à destination de n_2.

Comme dans la première partie, un participant peut recevoir des messages, ou en envoyer, après avoir créé un certain nombre de nouveaux nonces. Nous considérons maintenant qu'il peut avoir le choix entre plusieurs processus pour poursuivre son exécution, ce qui conduit à la grammaire suivante :

$$\begin{array}{rcl} R, R', \ldots \in \mathrm{Proc}_{\mathrm{Player}} & ::= & (\nu n_1).\ldots.(\nu n_k).\overline{c_{i\to j}}\langle m\rangle.R \\ & | & c_{i\to j}(x).\mathrm{case}\ x\ \mathrm{of}\ m.[m' = m''].R \\ & | & R + R' \\ & | & 0 \end{array}$$

Exemple : Le processus décrivant le rôle du participant A dans le protocole de J. Garay, M. Jakobsson et P. MacKenzie page 90 est le suivant :

$$\begin{array}{rcl} R_A & \hat{=} & \overline{c_{a\to b}}\langle \mathrm{pcs}_a(c,b,t)\rangle. \\ & & (\ \ c_{b\to a}(x).\mathrm{case}\ x\ \mathrm{of}\ \mathrm{pcs}_b(c,a,t). \\ & & \quad \overline{c_{a\to b}}\langle \mathrm{sig}_a(c)\rangle. \\ & & \quad (\ \ c_{b\to a}(y).\mathrm{case}\ y\ \mathrm{of}\ \mathrm{sig}_b(c).0 \\ & & \quad +\ R_A^{recovery}\ ) \\ & & +R_A^{abort}\ ) \end{array}$$

où on a posé :

$$
\begin{array}{rcl}
R_A^{abort} & \hat{=} & \overline{c_{a\to t}}\langle \mathrm{sig}_a(\langle c, \langle a, \langle b, abort\rangle\rangle\rangle)\rangle. \\
& & (\ \ c_{t\to a}(u).\mathrm{case}\ u\ \mathrm{of}\ \mathrm{sig}_b(c).0 \\
& & +\ c_{t\to a}(u).\mathrm{case}\ u\ \mathrm{of}\ \mathrm{sig}_T(\mathrm{sig}_a(\langle c, \langle a, \langle b, abort\rangle\rangle\rangle)).0\ ) \\
R_A^{recovery} & \hat{=} & \overline{c_{a\to t}}\langle\langle \mathrm{pcs}_b(c,a,t), \mathrm{sig}_a(c)\rangle\rangle. \\
& & (\ \ c_{t\to a}(v).\mathrm{case}\ v\ \mathrm{of}\ \mathrm{sig}_t(\mathrm{sig}_a(\langle c, \langle a, \langle b, abort\rangle\rangle\rangle)).0 \\
& & +\ c_{t\to a}(v).\mathrm{case}\ v\ \mathrm{of}\ \mathrm{sig}_b(c).0\ )
\end{array}
$$

□

Pour la TTP , c'est un peu différent. En effet, elle ne suit pas à proprement parler un rôle, mais doit se tenir prête à répondre à toute requête. La réponse qu'elle donne dépend des réponses qu'elle a pu envoyer auparavant. Plus précisément, lorsqu'elle reçoit un message d'une certaine forme m, elle commence par regarder si elle a auparavant envoyé un certain message m'. Si c'est le cas, elle envoie un message m_1, sinon elle envoie un message m_2. Les processus décrivant la TTP sont donc définis au moyen de la grammaire suivante :

$$
\begin{array}{rcl}
T, \ldots \in \mathrm{Proc}_{\mathrm{TTP}} & ::= & c_{i\to 0}(x).\mathrm{case}\ x\ \mathrm{of}\ m.\ (\ \ [\mathrm{sent}(m')].\overline{c_{0\to i}}\langle m_1\rangle.0 \\
& & \qquad\qquad\qquad\qquad +\ [\neg\mathrm{sent}(m')].\overline{c_{0\to i}}\langle m_2\rangle.0\ )\ [\!]\ T \\
& | & 0
\end{array}
$$

où le processus $[\mathrm{sent}(m')].\overline{c_{0\to i}}\langle m_1\rangle.0$ (respectivement $[\neg\mathrm{sent}(m')].\overline{c_{0\to i}}\langle m_2\rangle.0$) signifie : si le message m' a été envoyé auparavant (respectivement n'a pas été envoyé auparavant), alors envoyer le message m_1 (respectivement m_2).

Exemple : Toujours dans le cadre du protocole page 90, on a le processus suivant pour décrire la TTP :

$$
\begin{array}{rcl}
T & \hat{=} & \quad c_{a\to t}(x).\mathrm{case}\ x\ \mathrm{of}\ \langle \mathrm{pcs}_b(c,a,t), \mathrm{sig}_a(c)\rangle. \\
& & \qquad (\ \ [\mathrm{sent}(aborted)].\overline{c_{t\to i}}\langle aborted\rangle.0 \\
& & \qquad +\ [\neg\mathrm{sent}(aborted)].\overline{c_{t\to i}}\langle \mathrm{sig}_b(c)\rangle.0) \\
& & [\!]\ \ c_{a\to t}(x).\mathrm{case}\ x\ \mathrm{of}\ \mathrm{sig}_a(\langle c,a,b,abort\rangle). \\
& & \qquad (\ \ [\mathrm{sent}(\mathrm{sig}_a(c))].\overline{c_{t\to i}}\langle \mathrm{sig}_b(c)\rangle.0 \\
& & \qquad +\ [\neg\mathrm{sent}(\mathrm{sig}_a(c))].\overline{c_{t\to i}}\langle aborted\rangle.0) \\
& & [\!]\ \ c_{b\to t}(x).\mathrm{case}\ x\ \mathrm{of}\ \langle \mathrm{pcs}_a(c,b,t), \mathrm{sig}_b(c)\rangle.\ \ 0 \\
& & \qquad (\ \ [\mathrm{sent}(aborted)].\overline{c_{t\to i}}\langle aborted\rangle.0 \\
& & \qquad +\ [\neg\mathrm{sent}(aborted)].\overline{c_{t\to i}}\langle \mathrm{sig}_a(c)\rangle.0)
\end{array}
$$

où on a posé :

$$
aborted\ \ \hat{=}\ \ \mathrm{sig}_t(\mathrm{sig}_a(\langle c,a,b,abort\rangle))
$$

□

Finalement, un protocole d'échange optimiste sera décrit par un processus donné par la grammaire suivante :

$$
\begin{array}{rcl}
P, \ldots \in \mathrm{Proc}_{\mathrm{Game}} & ::= & P\ [\!]\ R \\
& | & T
\end{array}
$$

respectant les conventions suivantes :

- $(c_{i\to j})_{i,j\geq 0, i\neq j}$ est une famille de noms distincts ;
- dans R_i n'interviennent que les $c_{i\to j}$ avec $j \in \{0, 1, \ldots, n\}$, $j \neq i$ pour les émissions et $c_{j\to i}$ avec $j \in \{0, 1, \ldots, n\}$, $j \neq i$ pour les réceptions ;
- dans une construction $(\nu n_1). \ldots (\nu n_k).\overline{c_{i\to j}}\langle m\rangle.R$, $n_1, \ldots, n_k$ doivent apparaître dans m ;
- dans une construction $c_{i\to j}(x)$.case x of $m.[m' = m''].R$, x ne doit pas apparaître dans $[m' = m''].R$ (pour faire référence au message x dans $[m' = m''].R$, il faut utiliser m) ;
- 0 correspond à la TTP .

Le reste de cette partie est organisé de la manière suivante : nous allons tout d'abord présenter les systèmes de transitions alternés, ainsi que la logique temporelle alternée. Nous montrerons ensuite comment utiliser ce formalisme pour donner une sémantique aux protocoles optimistes d'échange et comment exprimer les propriétés de ces protocoles. Nous expliquerons ensuite comment raisonner par abstraction dans ces modèles et nous appliquerons ces techniques à la sémantique des protocoles. Nous terminerons par des considérations et des résultats pratiques. Cette approche s'inscrit dans le cadre décrit dans le chapitre 2. La figure 2.2.1 page 28 se retrouve, dans ce contexte, instanciée de la manière suivante :

- le langage L de description des systèmes étudiés est $\mathrm{Proc}_{\mathrm{Game}}$ (défini plus haut) ;
- la classe $\mathfrak{M}$ des modèles (concrets) correspond aux systèmes de transitions alternés, l'ensemble M utilisé pour décrire ces systèmes de transitions est adapté d'un langage de commandes gardées et l'ensemble E des énoncés (concrets) est l'ensemble des énoncés de la logique temporelle alternée (chapitre 9) ;
- la fonction de traduction $[] : L \to M$ sera décrite dans le chapitre 10 ;
- la classe $\mathfrak{M}^\alpha$ des modèles abstraits est une variante des systèmes de transitions alternés, l'ensemble M^α utilisé pour décrire ces systèmes de transitions est une variante du langage de commandes gardées utilisé dans le cas concret et l'ensemble des énoncés abstraits E^α est l'ensemble des énoncés de la logique temporelle alternée, dont on précisera la sémantique dans l'ensemble des systèmes de transitions alternés abstraits (chapitre 11) ;
- la fonction d'abstraction $\alpha : M \to M^\alpha$ sera décrite dans le chapitre 12.

Chapitre 9

Systèmes de transitions alternés, logique temporelle alternée

Nous allons ici décrire les systèmes de transitions alternés (appelés ATS pour abréger) ainsi que la logique temporelle alternée (ATL). Ces concepts ont été introduits par R. Alur, T. Henzinger et O. Kupferman [AHK98], afin de modéliser des systèmes ouverts. Dans de tels systèmes, une transition ne traduit pas une évolution du système, mais plutôt un déplacement dans un jeu entre les différentes composantes du système. Par rapport à une structure de Kripke, un ATS fait donc intervenir explicitement une notion de composantes (appelées agents du système). De la même manière, par rapport aux logiques temporelles classiques, ATL fait intervenir explicitement dans les énoncés quels agents tentent de coopérer pour obtenir un certain résultat. Dans le cas d'un système ouvert, par exemple, on pourra alors aisément exprimer que, quelles que soient les actions effectuées par l'environnement, le système ne se bloquera pas. Dans le cas des protocoles cryptographiques, nous exprimerons plutôt des propriétés du type : quelles que soient les actions effectuées par le premier participant, le second pourra toujours agir de son côté afin de ne pas être lésé. Nous allons maintenant donner les définitions précises des ATS (sous plusieurs formes : ATS locaux, contraintes d'équité) et de ATL. La description de la sémantique de protocoles cryptographiques au moyen de ce formalisme sera donnée dans un chapitre ultérieur.

Dans tout le reste du chapitre, on suppose donnés deux ensembles : Ω (appelé ensemble des agents, on le supposera fini) et Π (ensemble des propositions atomiques, non nécessairement fini).

9.1 Systèmes de transitions alternés

Au lieu de donner directement la définition d'un ATS, nous allons commencer par le cas plus simple des ATS locaux (*lock-step synchronous ATS* dans [AHK98]), que nous étendrons pour obtenir les ATS libres (qui sont les ATS de [AHK98]), puis les ATS contraints (les *fair ATS* de [AHK98], obtenus à partir des ATS libres en ajoutant des contraintes d'équité) et pour finir, en ajoutant un état initial, nous obtiendrons les ATS qui nous permettront de modéliser la classe des protocoles cryptographiques qui nous intéresse. Dans un ATS local, chaque agent $\omega \in \Omega$ est caractérisé par son état local, qui est un élément d'un ensemble Q_ω. Un état (global) du système est alors déterminé par la donnée de l'état local de chaque agent. L'ensemble des états locaux est donc $Q = \prod_{\omega\in\Omega} Q_\omega$. Lorsqu'il doit effectuer une transition, chaque agent consulte l'état (global) courant du système et met à jour son état local, suivant sa fonction de transition δ_ω, qui est en fait une relation entre Q et Q_ω. Lorsque chaque agent a choisi son nouvel état local, on obtient le nouvel état (global) du système. Ceci nous conduit à poser la définition suivante.

Définition – 9.1.1

Un ATS local (sur Ω et Π) est un triplet :

$$((Q_\omega)_{\omega\in\Omega}, (\delta_\omega)_{\omega\in\Omega}, \pi)$$

où :

- *$(Q_\omega)_{\omega\in\Omega}$ est une famille d'ensemble. Pour $\omega \in \Omega$, on dit que Q_ω est l'ensemble des états locaux de ω et l'ensemble :*

$$Q \quad \hat{=} \quad \prod_{\omega\in\Omega}$$

  est appelé ensemble des états (globaux) du système ;
- *$(\delta_\omega)_{\omega\in\Omega}$ est une famille de relations. Pour $\omega \in \Omega$, $\delta_\omega \subseteq Q \times Q_\omega$ est appelée relation de transition local de ω et la relation :*

$$\delta \quad \hat{=} \quad \{(q, (q'_\omega)) \in Q \times Q \mid \forall \omega \in \Omega.\ \delta_\omega(q, q'_\omega)\}$$

  est appelée relation de transition (globale) du système ;
- *π est une fonction de Q vers 2^Π qui indique, pour chaque état $q \in Q$ quelles sont les propositions atomiques qui sont vraies en q.*

Nous supposerons de plus que les ATS locaux vérifient la condition de réactivité (ou non-bloquage) suivante : quels que soient $\omega \in \Omega$ et $q \in Q$, il existe $q'_\omega \in Q_\omega$ tel que $\delta_\omega(q, q'_\omega)$.

Remarque : Avec les notations de la définition précédente, on notera souvent $q_\omega \in \delta_\omega(q)$ à la place de $\delta_\omega(q, q_\omega)$. Plus généralement, si R est une relation binaire entre deux ensembles A et B, on notera indifféremment $R(a, b)$ ou $b \in R(a)$ pour exprimer le fait que $(a, b) \in R$. □

Exemple : Imaginons par exemple que $\Omega = \{a, b\}$ et $\Pi = \{x, y, z\}$ et posons $Q_a = 2^{\{x,y\}}$ et $Q_b = 2^{\{z\}}$. L'idée est que les variables x et y appartiennent à a, alors que z appartient à b. Supposons que a puisse mettre à vrai la valeur de x sans condition, mais qu'il ne puisse la mettre à faux que si y elle-même est fausse. On obtient ainsi pour a la relation de transition locale $\delta_a \subseteq Q \times Q_a$ où $Q = 2^{\{x,y,z\}}$, définie de la manière suivante : $\delta_a(q, q_a)$ est vraie si, et seulement si :

$$y \in q_a \ \Leftrightarrow \ y \in q \quad \text{et} \quad x \notin q_a \ \Rightarrow \ y \notin q$$

Imaginons que b puisse mettre à vrai (respectivement à faux) la valeur de z si, et seulement si, x est fausse (respectivement vraie). La relation de transition locale pour b, $\delta_b \subseteq Q \times Q_b$, est alors définie de la manière suivante : $\delta_b(q, q_b)$ est vraie si, et seulement si :

$$z \in q_b \ \Leftrightarrow \ x \notin q$$

□

Remarque : Pour $\omega_0 \in \Omega$, on notera proj_{ω_0} la fonction :

$$\begin{array}{lcl} Q & \to & Q_{\omega_0} \\ q = (q_\omega)_{\omega \in \Omega} & \mapsto & q_{\omega_0} \end{array}$$

□

Pour passer, de cette définition, à celle des ATS libres, observons comment les agents permettent au système d'évoluer. Lorsqu'un agent $\omega \in \Omega$ choisit son nouvel état local q'_ω , il impose en fait une contrainte sur le nouvel état global du système (précisément : il impose que la composante suivant ω du nouvel état global soit égale à q'_ω). Pour obtenir les ATS libres, on oublie donc cette distinction entre états locaux des agents, mais on garde un fonctionnement sous forme de contraintes. Ainsi pour faire évoluer le système, chaque agent $\omega \in \Omega$ choisit un sous ensemble Q'_ω de l'ensemble des états du système (suivant ce que lui permettent l'état courant et sa fonction de transition) et le nouvel état du système est l'unique élément qui se trouve dans l'intersection $\bigcap_{\omega \in \Omega} Q'_\omega$ (on imposera que cette intersection soit un singleton).

Définition – 9.1.2

Un ATS libre (sur Ω et Π) est un triplet (Q, Δ, π) où :
- *Q est l'ensemble des états,*
- *Δ est une relation, $\Delta \subseteq Q \times \Omega \times 2^Q$. Pour $q \in Q$ fixé, les ensembles $Q'_\omega \in \Delta(q, \omega)$ sont les ensembles d'états parmi lesquels ω peut imposer que se trouve l'état suivant. On supposera que, si $q \in Q$ et si, pour tout $\omega \in \Omega$, on a $Q'_\omega \in \Delta(q, \omega)$, alors l'intersection $\bigcap_{\omega \in \Omega} Q'_\omega$ est un singleton q' (et on notera simplement $\bigcap_{\omega \in \Omega} Q'_\omega = q'$) ;*
- *π est une fonction de Q vers 2^Π.*

Nous supposerons de plus que les ATS libres vérifient la condition de réactivité (ou non-bloquage) suivante : quels que soient $\omega \in \Omega$ et $q \in Q$, il existe $Q'_\omega \in \Delta(q, \omega)$.

Remarque : À un ATS local $((Q_\omega)_{\omega \in \Omega}, (\delta_\omega)_{\omega \in \Omega}, \pi)$ correspond bien entendu l'ATS libre (Q, Δ, π) où on a posé :

$$\begin{array}{lcl} Q & \hat{=} & \prod_{\omega \in \Omega} Q_\omega \\ \Delta & \hat{=} & \{(q, \omega, Q'_\omega) \in Q \times \Omega \times 2^Q \mid \exists q'_\omega \in \delta_\omega(q).\ Q_\omega = \mathrm{proj}_\omega^{-1}(q'_\omega)\} \end{array}$$

□

Exemple : Si on réécrit l'ATS de l'exemple précédent comme un ATS libre, on obtient les relations de transition suivantes :

$$\begin{array}{lcl} \Delta(q, a) & \hat{=} & \{q' \in Q \mid y \in q' \Leftrightarrow y \in q \text{ et } x \notin q' \Rightarrow y \notin q\} \\ \Delta(q, b) & \hat{=} & \{q' \in Q \mid z \in q' \Leftrightarrow x \notin q\} \end{array}$$

□

Dans certains protocoles cryptographiques, particulièrement les protocoles de signature de contrats, il arrive qu'un participant ait, à un point du protocole, la possibilité d'attendre un message en provenance d'un autre participant, ou bien de lancer une requête à la TTP si le message en question n'est pas reçu en un temps raisonnable. Nous modéliserons cette situation en laissant, à ce point du protocole, la possibilité au participant de ne rien faire, recevoir le message (à condition qu'il ait effectivement été envoyé) ou envoyer une requête à la TTP . Dans un ATS libre, cette modélisation permettrait à un participant, même honnête, d'attendre indéfiniment. Or ce n'est pas le cas en réalité : les protocoles sont implémentés de sorte que, au bout d'un certain temps, si aucun message n'est reçu, une requête est envoyée à la TTP . Nous allons donc utiliser des contraintes d'équité qui permettent de spécifier qu'un agent ne peut pas éviter indéfiniment de faire certaines actions. Une condition d'équité sur un ATS libre est un ensemble de contraintes d'équité, chacune de ces contraintes étant constituée d'un sous-ensemble des actions que les agents peuvent effectuer.

Définition – 9.1.3

> *Soit (Q, Δ, π) un ATS libre. Une contrainte d'équité γ sur cet ATS est une relation $\gamma \subseteq \Delta \subseteq Q \times \Omega \times 2^Q$. Une condition d'équité sur cet ATS est un ensemble de contraintes d'équité. Un ATS contraint (sur Ω et Π) est un quadruplet (Q, Δ, Γ, π) où (Q, Δ, π) est un ATS libre et Γ est une condition d'équité sur cet ATS. Un ATS (sur Ω et Π) est un quintuplet $(Q, q_0, \Delta, \Gamma, \pi)$ où (Q, Δ, Γ, π) est un ATS contraint et $q_0 \in Q$. q_0 sera appelé état initial de l'ATS. La classe des ATS sur Ω et Π sera notée $\mathfrak{Ats}(\Omega, \Pi)$.*

Exemple : Dans l'ATS qui nous sert d'exemple depuis le début, a peut mettre x à faux uniquement lorsque y est lui-même faux, mais il n'est pas obligé de le faire. La contrainte d'équité suivante :

$$\gamma(q, a) \quad \hat{=} \quad \left[\begin{array}{ll} \varnothing & \text{si } y \in q \\ \{q' \in Q \mid y \notin q \text{ et } x \notin q'\} & \text{sinon} \end{array} \right.$$

permet d'obliger a à faire ce choix, au bout d'un temps fini. □

Il est clair que tout ATS libre, muni d'une condition d'équité vide, constitue un ATS contraint. Les notions de successeur d'un état et d'exécution dans un ATS se définissent aisément comme dans les structures de Kripke.

Définition – 9.1.4

> *Soit $\mathcal{S} = (Q, q_0, \Delta, \pi, \Gamma)$ un ATS (sur Ω et Π). On dit que $q' \in Q$ est un successeur de $q \in Q$ dans $\mathcal{S}$ (et on note $q \xrightarrow{\mathcal{S}} q'$, ou $q \xrightarrow{\Delta} q'$, ou encore $q \to q'$ lorsqu'aucune confusion n'est à craindre) si quel que soit $\omega \in \Omega$, il existe $Q'_\omega \in \Delta(q, \omega)$ tel que $q' \in Q'_\omega$.*

Remarque : Avec les mêmes notations, on a :

$$\begin{aligned} \{q' \in Q \mid q \xrightarrow{\mathcal{S}} q'\} &= \bigcap_{\omega \in \Omega} \bigcup_{Q'_\omega \in \Delta(q,\omega)} Q'_\omega \\ &= \bigcup_{(Q'_\omega)_{\omega \in \Omega} \in \prod_{w \in \Omega} \Delta(q,\omega)} \bigcap_{\omega \in \Omega} Q'_\omega \end{aligned}$$

□

Définition – 9.1.5

> *Une exécution d'un ATS $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$ est une suite infinie d'états $(q_i)_{i \geq 0}$ telle que, quel que soit $i \geq 0$, on ait $q_i \xrightarrow{\mathcal{S}} q_{i+1}$.*

L'intérêt des conditions d'équité est de permettre d'évincer des exécutions qui ne seraient pas équitables, relativement à ces conditions.

Définition – 9.1.6

Soit $(q_i)_{i\geq 0}$ une exécution d'un ATS $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$. Une contrainte d'équité $\gamma \in \Gamma$ est dite activée, à la position $i \geq 0$ de l'exécution, et pour un agent $\omega \in \Omega$, si $\gamma(q_i, \omega)$ n'est pas vide. Elle est dite acceptée, à cette même position, et pour ce même agent, s'il existe $Q'_\omega \in \gamma(q_i, \omega)$ tel que $q_{i+1} \in Q'_\omega$. L'exécution est dite équitable, pour un ensemble d'agents $\Omega' \subseteq \Omega$, si, quel que soit $\omega \in \Omega'$ et quel que soit $\gamma \in \Gamma$:

- *soit le nombre de positions de l'exécution pour lesquelles γ n'est pas activée (pour ω) est infini ;*
- *soit le nombre de positions de l'exécution pour lesquelles γ est acceptée (pour ω) est infini.*

On notera $\mathrm{Fair}^{\mathcal{S}}(\Omega')$ *les exécutions de $\mathcal{S}$ qui sont équitables pour Ω'.*

Nous en venons maintenant au concept de stratégie, qui nous sera utile pour définir la sémantique des énoncés de ATL. Informellement, une stratégie est une fonction qui indique à un agent quel état suivant choisir, en se basant sur les états passés.

Définition – 9.1.7

Soient $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$ un ATS et $\omega \in \Omega$. On appelle stratégie pour ω toute fonction $f_\omega : Q^+ \to 2^Q$ (où Q^+ désigne l'ensemble des suites finies non vides d'éléments de Q) telle que quel que soit $q_0, \ldots, q_n \in Q^+$, on ait $f_\omega(q_0, \ldots, q_n) \in \Delta(q_n, \omega)$. Si $\Omega' \subseteq \Omega$, un ensemble de stratégies pour Ω' est une fonction f qui à $\omega \in \Omega'$ associe une stratégie pour ω, notée f_ω.

À une stratégie (ou un ensemble de stratégies), on peut associer l'ensemble des exécutions de l'ATS qui respectent cette stratégie (il s'agit de l'ensemble des exécutions qui peuvent se produire lorsque l'agent en question respecte les recommandations de sa stratégie).

Définition – 9.1.8

Soient $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$ un ATS, $\omega \in \Omega$ et f_ω une stratégie pour ω. On dit qu'une exécution $(q_i)_{i\geq 0}$ de $\mathcal{S}$ respecte f_ω si quel que soit $i > 0$ on a $q_{i+1} \in f_\omega(q_0, \ldots, q_i)$. L'ensemble de ces exécutions sera noté $\mathrm{Exec}^{\mathcal{S}}(f_\omega)$. *Si $\Omega' \subseteq \Omega$ et f est un ensemble de stratégies pour Ω', on note* $\mathrm{Exec}^{\mathcal{S}}(f)$ *l'ensemble des exécutions qui respectent chaque stratégie f_ω pour $\omega \in \Omega'$, précisément :*

$$\mathrm{Exec}^{\mathcal{S}}(f) \quad \hat{=} \quad \bigcap_{\omega \in \Omega'} \mathrm{Exec}^{\mathcal{S}}(f_\omega)$$

On peut voir $\mathrm{Exec}^{\mathcal{S}}(f)$ comme l'ensemble des exécutions que les agents $\omega \in \Omega'$ peuvent obtenir lorsqu'ils coopèrent et suivent chacun leur stratégie.

9.2 Logique temporelle alternée

La logique temporelle alternée est une variante de CTL adaptée aux systèmes de transitions alternés. La différence majeure par rapport à CTL consiste en la présence de deux modalités paramétrées par des ensembles d'agents. La présence de la première de ces modalités signifie que les agents concernés, s'il coopèrent, peuvent agir de sorte à rendre vrai l'énoncé qui se trouve sous la modalité. Quand à la seconde, son but est d'exprimer que les agent concernés ne peuvent pas éviter que l'énoncé sous la modalité soit vrai (*i.e.* ils ne peuvent pas coopérer de manière à le rendre faux). Ces deux modalités sont duales l'une de l'autre (*i.e.* on pourrait se passer de l'une et la définir en fonction de l'autre au moyen de négations) mais les considérer toutes deux permettra de limiter l'application des négations aux propositions atomiques, ce qui facilitera la mise en place des abstractions.

DÉFINITION – 9.2.1

Les énoncés de la logique temporelle alternée ATL (sur Ω et Π) sont définis au moyen de la grammaire suivante :

$$
\begin{array}{rcll}
\varphi, \psi, \ldots \in \mathrm{Atl}(\Omega, \Pi) & ::= & p & p \in \Pi \\
& | & \neg p & \\
& | & \varphi \vee \psi & \\
& | & \varphi \wedge \psi & \\
& | & \langle \Omega' \rangle \circ \varphi & \Omega' \subseteq \Omega \\
& | & \langle \Omega' \rangle \Box \varphi & \\
& | & \langle \Omega' \rangle \Diamond \varphi & \\
& | & [\Omega'] \circ \varphi & \\
& | & [\Omega'] \Box \varphi & \\
& | & [\Omega'] \Diamond \varphi &
\end{array}
$$

Remarque : Nous nous éloignons une fois de plus de [AHK98] en ne considérant pas la construction $\langle \Omega \rangle \varphi \, \mathcal{U} \, \psi$, d'une part parce qu'elle n'intervient pas dans l'expression des propriétés typiques des protocoles et, d'autre part, parce que son traitement est plus technique que celui des autres constructions. □

Dans ces conditions, la négation d'un énoncé n'est plus une construction syntaxique mais une involution (*i.e.* une bijection qui est son propre inverse) de $\mathrm{Atl}(\Omega, \Pi)$.

DÉFINITION – 9.2.2

La négation d'un énoncé $\varphi \in \mathrm{Atl}(\Omega, \Pi)$, notée $\neg(\varphi)$, est définie par induction sur φ de la manière suivante :

- *si $p \in \Pi$, alors :*

$$\neg(p) \;\hat{=}\; \neg p \quad et \quad \neg(\neg p) \;\hat{=}\; p$$

- *si $\varphi, \psi \in \mathrm{Atl}(\Omega, \Pi)$, alors :*

$$\neg(\varphi \vee \psi) \;\hat{=}\; \neg(\varphi) \wedge \neg(\psi) \quad et \quad \neg(\varphi \wedge \psi) \hat{=} \neg(\varphi) \vee \neg(\psi)$$

- *si $\varphi \in \mathrm{Atl}(\Omega, \Pi)$ et $\Omega' \subseteq \Omega$, alors :*

$$
\begin{array}{rclcrcl}
\neg(\langle \Omega' \rangle \circ \varphi) & \hat{=} & [\Omega'] \circ \neg(\varphi) & \qquad & \neg([\Omega'] \circ \varphi) & \hat{=} & \langle \Omega' \rangle \circ \neg(\varphi) \\
\neg(\langle \Omega' \rangle \Box \varphi) & \hat{=} & [\Omega'] \Diamond \neg(\varphi) & & \neg([\Omega'] \Box \varphi) & \hat{=} & \langle \Omega' \rangle \Diamond \neg(\varphi) \\
\neg(\langle \Omega' \rangle \Diamond \varphi) & \hat{=} & [\Omega'] \Box \neg(\varphi) & & \neg([\Omega'] \Diamond \varphi) & \hat{=} & \langle \Omega' \rangle \Box \neg(\varphi)
\end{array}
$$

Dans la suite, on notera $\neg\varphi$ à la place de $\neg(\varphi)$.

Remarque : À ce stade, on peut constater que ATS et ATL constituent une généralisation des structures de Kripke et de CTL :

– toute structure de Kripke peut être considérée comme un ATS sur l'ensemble d'agents $\Omega = \{1\}$ et dont la condition d'équité est vide ;
– les sous ensembles de $\Omega = \{1\}$ sont $\varnothing$ et Ω, or :
  – $\langle\varnothing\rangle \circ \varphi$ signifie que toute exécution satisfait $\circ\,\varphi$ (on écrirait $\mathtt{AX}\ \varphi$ dans CTL) ;
  – $\langle\Omega\rangle \circ \varphi$ signifie qu'il existe une exécution satisfaisant $\circ\,\varphi$ ($\mathtt{EX}\ \varphi$ dans CTL) ;

  et on pourrait de même établir des correspondances pour les autres constructions de la logique ATL.

□

Le moyen le plus simple de définir la sémantique des énoncés de ATL est de construire une fonction qui, à un ATS et un énoncé de ATL, associe l'ensemble des états de l'ATS qui satisfont l'énoncé. Cette fonction est définie par induction sur l'énoncé considéré.

Définition – 9.2.3

Soit $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$ un ATS. La sémantique d'un énoncé $\varphi \in \mathrm{Atl}(\Omega, \Pi)$ dans $\mathcal{S}$, notée $[\![\varphi]\!]_{\mathcal{S}}$, est définie par induction sur φ de la manière suivante :

- *si $p \in \Pi$, alors $q \in [\![p]\!]_{\mathcal{S}}$ (respectivement $q \in [\![\neg p]\!]_{\mathcal{S}}$) si $p \in \pi(q)$ (respectivement $p \notin \pi(q)$) ;*
- *si $\varphi, \psi \in \mathrm{Atl}(\Omega, \Pi)$, alors $q \in [\![\varphi \vee \psi]\!]_{\mathcal{S}}$ (respectivement $q \in [\![\varphi \wedge \psi]\!]_{\mathcal{S}}$) si $q \in [\![\varphi]\!]_{\mathcal{S}} \cup [\![\psi]\!]_{\mathcal{S}}$ (respectivement $q \in [\![\varphi]\!]_{\mathcal{S}} \cap [\![\psi]\!]_{\mathcal{S}}$) ;*
- *si $\varphi \in \mathrm{Atl}(\Omega, \Pi)$ et $\Omega' \subseteq \Omega$, alors $q \in [\![\langle\Omega'\rangle \circ \varphi]\!]_{\mathcal{S}}$ s'il existe f ensemble de stratégies pour Ω' tel que si $(q_i)_{i \geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$ et $q_0 = q$, alors $q_1 \in [\![\varphi]\!]_{\mathcal{S}}$;*
- *de manière duale (et avec les mêmes notations), $q \in [\![[\Omega'] \circ \varphi]\!]_{\mathcal{S}}$ si pour tout ensemble f de stratégies pour Ω', il existe $(q_i)_{i \geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$ tel que $q_0 = q$ et $q_1 \in [\![\varphi]\!]_{\mathcal{S}}$;*
- *$q \in [\![\langle\Omega'\rangle \square \varphi]\!]_{\mathcal{S}}$ s'il existe f ensemble de stratégies pour Ω' tel que $\mathrm{Exec}^{\mathcal{S}}(f) \subseteq \mathrm{Fair}^{\mathcal{S}}(\Omega')$ et si $(q_i)_{i \geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$ et $q_0 = q$, alors, quel que soit $i \geq 0$, $q_i \in [\![\varphi]\!]_{\mathcal{S}}$;*
- *de manière duale, $q \in [\![[\Omega'] \square \varphi]\!]_{\mathcal{S}}$ si pour tout ensemble f de stratégies pour Ω', il existe $(q_i)_{i \geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f) \cap \mathrm{Fair}^{\mathcal{S}}(\Omega \setminus \Omega')$ telle que $q_0 = q$ et, quel que soit $i \geq 0$, $q_i \in [\![\varphi]\!]_{\mathcal{S}}$;*
- *$q \in [\![\langle\Omega'\rangle \lozenge \varphi]\!]_{\mathcal{S}}$ s'il existe f ensemble de stratégies pour Ω' tel que si $(q_i)_{i \geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f) \cap \mathrm{Fair}^{\mathcal{S}}(\Omega \setminus \Omega')$ et si $q_0 = q$, alors il existe $i \geq 0$ tel que $q_i \in [\![\varphi]\!]_{\mathcal{S}}$;*
- *de manière duale, $q \in [\![[\Omega'] \lozenge \varphi]\!]_{\mathcal{S}}$ si pour tout ensemble f de stratégies pour Ω' tel que $\mathrm{Exec}^{\mathcal{S}}(f) \subseteq \mathrm{Fair}^{\mathcal{S}}(\Omega')$, il existe $(q_i)_{i \geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$ telle que $q_0 = q$ et, pour un certain $i \geq 0$, $q_i \in [\![\varphi]\!]_{\mathcal{S}}$.*

On dit que $\mathcal{S}$ satisfait un énoncé $\varphi \in \mathrm{Atl}(\Omega, \Pi)$, et on note $\mathcal{S} \models \varphi$, si $q_0 \in [\![\varphi]\!]_{\mathcal{S}}$.

Remarque : On pourra être surpris des choix qui sont faits de considérer des exécutions équitables parfois pour Ω', parfois pour $\Omega \setminus \Omega'$ et d'autres fois encore pour $\varnothing$. Donnons quelques justifications informelles :

– pour $\langle\Omega'\rangle \circ \varphi$, il suffit aux agents de Ω' de coopérer une fois seulement pour que φ soit vrai dans l'état qui suit. Les contraintes d'équité de ces agents, pas plus que celles de leurs adversaires qui vont essayer de rendre φ faux, n'interviennent donc pas ici ;
– pour $\langle\Omega'\rangle \square \varphi$, les agents de Ω' doivent coopérer de sorte à ce que φ soit vrai tout au long de l'exécution. Leur stratégie va donc être appliquée pour choisir toutes les transitions suivantes, et doit par conséquent tenir compte des contraintes d'équité. Inversement, pour les adversaires qui vont essayer de rendre φ faux au moins une fois, on peut oublier les contraintes d'équité ;

– pour $\langle\Omega'\rangle \Diamond \varphi$, c'est exactement le contraire.
L'idée générale est que les contraintes n'ont de sens que sur les exécutions infinies. Or, pour valider un énoncé de la forme $\langle\Omega'\rangle \circ \varphi$ ou $\langle\Omega'\rangle \Diamond \varphi$, il suffit de considérer les préfixes finis. Ainsi, il est inutile de soumettre les agent qui doivent agir de manière à satisfaire ces énoncés à des contraintes d'équité (au contraire de leurs opposants). Par contre, pour valider un énoncé de la forme $\langle\Omega'\rangle \Diamond \varphi$, les agents de Ω' doivent agir en permanence et on doit, par conséquent, vérifier qu'ils ne violent pas leur contraintes d'équité. $\square$

Pour $q \in Q$, le fait que $q \in [\![\varphi]\!]_{\mathcal{S}}$ sera également noté $\mathcal{S}, q \models \varphi$. On vérifiera sans peine que $q \in [\![\neg\varphi]\!]_{\mathcal{S}}$ est équivalent à $q \notin [\![\varphi]\!]_{\mathcal{S}}$.

9.3 Description d'ATS

Comme nous aurons besoin dans la suite de définir concrètement des ATS, et que de telles définitions peuvent s'avérer fastidieuses, nous allons présenter dans cette dernière section une notation plus pratique. Nous nous inspirons pour cela du langage de commandes gardées présenté dans [HMMR00], qui va nous permettre de décrire des ATS locaux, mais possédant des contraintes d'équité. Nous définirons tout d'abord un ensemble $\Omega = \{\omega_1, \ldots, \omega_n\}$ au moyen de la notation :

$$\textbf{agents} \quad : \quad \omega_1, \ldots, \omega_n$$

À chaque agent $\omega_i \in \Omega$, nous associerons un ensemble de variables $X_{\omega_i} = \{\omega_i.x_1, \ldots\}$ (éventuellement infini) au moyen de la déclaration suivante :

$$\textbf{vars}(\omega_i) \quad : \quad \omega_i.x_1, \ldots$$

(la notation $\omega_i.x_j$ permet de s'assurer que les ensembles de variables associés à chaque agent sont distincts). Ces variables seront interprétées dans le domaine $\mathbb{B} \triangleq \{\text{true}, \text{false}\}$ des booléens et ainsi, l'ensemble des états locaux de ω_i sera $Q_{\omega_i} \triangleq \mathbb{B}^{X_{\omega_i}}$. L'ensemble des états (globaux) de l'ATS est alors :

$$Q \;\triangleq\; \prod_{\omega\in\Omega} Q_\omega \;=\; \mathbb{B}^X$$

où on a posé $X \triangleq X_{\omega_1} \cup \ldots \cup X_{\omega_n}$. Nous prendrons $\Pi \triangleq X$ comme ensemble de propositions atomiques. La fonction d'évaluation de ces propositions sera naturellement définie comme :

$$\begin{array}{rccl} \pi : & Q & \rightarrow & 2^\Pi \\ & q & \mapsto & \{p \in \Pi \mid q(p) = \text{true}\} \end{array}$$

La relation de transition locale de chaque agent $\omega_i \in \Omega$ sera définie syntaxiquement sous la forme d'ensembles de commandes gardées, de la forme $[\!] \; g \rightarrow u$. Dans une telle commande, la garde g représente un prédicat sur Q qui indique quand la commande peut s'appliquer et la mise à jour u est une relation entre Q et Q_{ω_i} qui indique quel est l'effet (local) de la commande. g sera donc un énoncé booléen sur X et u un énoncé booléen sur $X \cup X'_{\omega_i}$ (nous autoriserons la présence de conjonctions et disjonctions infinies dans ces énoncés). La notation X'_{ω_i} désigne ici un ensemble disjoint de X_{ω_i} pour lequel il existe une bijection :

$$\begin{array}{ccc} X_{\omega_i} & \rightarrow & X'_{\omega_i} \\ \omega_i.x & \mapsto & \omega_i.x' \end{array}$$

On supposera de plus que $X' \hat{=} X'_{\omega_1} \cup \ldots \cup X'_{\omega_n}$ et X sont disjoints. Cet ensemble X'_{ω_i} nous permettra de définir les nouvelles valeurs des variables de ω_i tout en conservant les anciennes. Si on a associé à l'agent ω_i les commandes gardées :

$$[\![\ g_1 \to u_1, \ldots, [\![\ g_k \to u_k$$

alors la relation de transition locale de ω_i sera constituée des couples (q, q_{ω_i}) dans $Q \times Q_{\omega_i}$ pour lesquels il existe $j \in \{1, \ldots, k\}$ tel que :

- en tant que valuation sur X, q satisfait g_j ;
- si on note q'_{ω_i} la valuation sur X'_{ω_i} induite par q_{ω_i} et r la valuation sur $X \cup X'_{\omega_i}$ induite par q et q'_{ω_i}, alors la valuation r satisfait u_j.

Pour exprimer que les commandes gardées en question définissent la relation de transition locale de ω_i, nous écrirons :

$$\begin{array}{rcl} \textbf{commands}(\omega_i) & : & [\![\ g_1 \to u_1 \\ & & \vdots \\ & & [\![\ g_k \to u_k \end{array}$$

(ces commandes gardées pourront éventuellement être en nombre infini). Lorsque nous désignerons une mise à jour u sous la forme :

$$u \quad \hat{=} \quad \omega_i.x'_1 := b_1; \ldots ; \omega_i.x'_n := b_n$$

où $b_1, \ldots, b_n$ sont des booléens, cela signifiera que u est vraie si, dans le nouvel état, la valeur de chaque $\omega_i.x_j$ (pour $1 \leq j \leq n$) est b_j, les valeurs des autres variables restant inchangées. Dans le cas particulier où toutes les valeurs des variables sont inchangées, on notera :

$$u \quad \hat{=} \quad \text{idle}$$

À ce point, nous avons donc défini un ATS local, auquel correspond un ATS libre. Nous le munirons de conditions d'équité en écrivant certaines commandes gardées sous la forme $[\![_{\mathrm{f}}\ g \to u$. Les commandes gardées de cette forme induiront un sous ensemble de la relation de transition de l'agent ω_i concerné,qui induit *a fortiori* un sous ensemble $\gamma^{g,u}_{\omega_i}$ de la relation de transition globale du système, autrement dit une contrainte d'équité. L'ensemble Γ de ces contraintes d'équité constituera la condition d'équité associée à l'ATS. Pour finir, nous définirons un état initial au moyen de déclarations de la forme :

$$\textbf{init}(\omega_i) \quad : \quad u$$

où u est un énoncé booléen sur X'_{ω_i}. À cet énoncé correspondent les valuations sur X'_{ω_i} qui le satisfont. Ces valuations induisent elles-mêmes des valuations sur X_{ω_i} et la donnée d'une telle valuation pour chaque agent constitue un état initial (nous imposerons donc de plus que ces énoncés définissent un, et un seul, état initial). Avec les constructions qui précèdent, l'ATS donné en exemple depuis le début du chapitre s'écrirait :

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{ats}\ \textsc{Exemple} =} \\
\quad \textbf{agents} & : & a, b \\
\\
\quad \textbf{vars}(a) & : & a.x, a.y \\
\quad \textbf{vars}(b) & : & b.z \\
\\
\quad \textbf{init}(a) & : & a.x := \text{true}; a.y := \text{false} \\
\quad \textbf{init}(b) & : & b.z := \text{true} \\
\\
\quad \textbf{commands}(a) & : & [\!] \ \text{true} \rightarrow a.x := \text{true} \\
 & & [\!]_{\mathrm{f}} \ \neg a.y \rightarrow a.x := \text{false} \\
\\
\quad \textbf{commands}(b) & : & [\!] \ a.x \rightarrow b.z := \text{false} \\
 & & [\!] \ \neg a.x \rightarrow b.z := \text{true} \\
\multicolumn{3}{l}{\textbf{end}}
\end{array}
$$

Rappelons que les ATS que nous avons définis doivent satisfaire une condition de non-bloquage. Pour chaque description d'ATS donnée par la suite, nous devrons donc faire en sorte que l'ATS correspondant soit non-bloquant (d'ailleurs, le *model-checker* MOCHA, qui utilise une syntaxe semblable à celle que l'on vient de donner, complète lui-même les ATS qu'il reçoit en entrée afin qu'ils soient non-bloquants. Comme cette complétion modifie l'ATS, il est plus prudent de s'assurer que les ATS que l'on utilise sont déjà non-bloquants. Ainsi, dans l'exemple ci-dessus, on peut constater que, pour chaque agent, la disjonction des gardes qui apparaissent dans son ensemble de commandes gardées est une tautologie. Ainsi, dans chaque état, chaque agent peut appliquer au moins une des ses commandes gardées. L'ATS est par conséquent non-bloquant. Nous allons maintenant, dans le chapitre suivant, mettre en application les définitions qui ont été données ici afin de définir, en termes d'ATS, la sémantique de protocoles cryptographiques.

Chapitre 10

Modélisation des protocoles

Ce chapitre est consacré à la définition d'une sémantique, en termes d'ATS, pour les processus P décrivant des protocoles optimistes d'échange. Plus exactement, nous montrerons comment associer à un tel processus une famille d'ATS, chaque ATS de cette famille correspondant à une hypothèse d'honnêteté (respectivement de fiabilité) sur les participants au protocole (respectivement les canaux de communication). Cette modélisation des issue de [KR01, KR02]. Notre contribution dans ce chapitre (qui résulte d'un travail commun avec S. Kremer et J.-F. Raskin) a consisté à supprimer les simplifications qui se trouvaient dans les modélisations dont on s'est inspiré, afin d'obtenir une modélisation fidèle à la réalité, quitte à ce que le modèle obtenu soit infini (les simplifications étant maintenant réalisées au moyen de techniques d'interprétation abstraite, voir à ce sujet les chapitres qui suivent). Comme nous l'avons déjà expliqué, nous ne chercherons ici qu'à modéliser une seule session du protocole considéré. Cette hypothèse nous permet de réaliser quelques simplifications dans le processus P décrivant ce protocole. Ces simplifications sont détaillées dans la section suivante. Nous définirons ensuite quels agents interviendront dans l'ATS en question (ces agents représenteront bien entendu les participants au protocole, mais aussi les canaux de communication) ainsi que la composition de leur état local. Viendra alors la section la plus longue de ce chapitre qui détaillera, pour chaque agent et pour chaque niveau d'honnêteté (ou de fiabilité selon le cas), la relation de transition locale associée. Nous détaillerons ensuite ces constructions sur un exemple puis nous terminerons en donnant des exemples de modélisation de propriétés de sécurité typiques de ces protocoles. Dans tout le reste de ce chapitre, on suppose fixé un processus :

$$P \quad \hat{=} \quad T \parallel R_1 \parallel \ldots \parallel R_n \in \mathrm{Proc}_{\mathrm{Game}}$$

Afin d'illustrer les définitions de cette partie, relativement technique, nous utiliserons le protocole suivant :

<u>Jouet</u> :

$$\begin{array}{lcccl} 1. & A & \rightarrow & B & : \ \mathrm{sig}_A(N) \\ 2. & B & \rightarrow & A & : \ \mathrm{sig}_B(N) \end{array}$$

Dans ce protocole, complètement irréaliste, A crée un nonce N et le signe, avant de l'envoyer à B. En retour, B envoie à A le nonce N qu'il a lui-même signé. Comme A est visiblement désavantagé dans ce protocole, nous faisons appel à un tiers de confiance pour rétablir l'équilibre :

Recouvrement :

$$\begin{array}{lcccl} 1. & A & \rightarrow & T & : & \mathrm{sig}_A(N) \\ 2. & T & \rightarrow & A & : & \mathrm{sig}_B(N) \end{array}$$

Ne nous attardons pas sur le fait que T n'a pas les moyens de créer le message qu'il doit envoyer à A, ni sur le fait que c'est maintenant B qui est désavantagé : ce protocole nous servira juste pour illustrer les constructions qui seront données par la suite. En termes de processus, en notant t, a et b à la place de 0, 1 et 2, respectivement, on obtient la description suivante :

$$\begin{array}{rcl} P & \hat{=} & T \parallel R_a \parallel R_b \\ T & \hat{=} & c_{a\to t}(x).\mathrm{case}\ x\ \mathrm{of}\ \mathrm{sig}_a(y).\,(\ [\mathrm{sent}(\mathrm{sig}_b(y))].\overline{c_{a\to t}}\langle \mathrm{sig}_b(y)\rangle.0 \\ & & \qquad\qquad\qquad\qquad + [\neg\mathrm{sent}(\mathrm{sig}_b(y))].\overline{c_{a\to t}}\langle \mathrm{sig}_b(y)\rangle.0\ ) \\ R_a & \hat{=} & (\nu n).\overline{c_{a\to b}}\langle \mathrm{sig}_a(n)\rangle.\,(\ c_{b\to a}(x).\mathrm{case}\ x\ \mathrm{of}\ \mathrm{sig}_b(n).0 \\ & & \qquad\qquad\qquad\qquad + \overline{c_{a\to t}}\langle \mathrm{sig}_a(n)\rangle.c_{t\to a}(x).\mathrm{case}\ x\ \mathrm{of}\ \mathrm{sig}_b(n).0\ ) \\ R_b & \hat{=} & c_{a\to b}(x).\mathrm{case}\ x\ \mathrm{of}\ \mathrm{sig}_a(y).\overline{c_{b\to a}}\langle \mathrm{sig}_b(y)\rangle.0 \end{array}$$

10.1 Prétraitement

Nous appelons prétraitement la succession des différentes étapes qui, partant d'un protocole défini au moyen d'un processus $P \in \mathrm{Proc}_{\mathrm{Game}}$, fournit une description plus adaptée pour en donner ensuite la sémantique en termes d'ATS. Ce prétraitement est composé de trois étapes. Dans la première, nous supprimons les constructions dynamiques de nonces. En effet, comme nous n'allons modéliser qu'une seule session du protocole en question, il est possible de remplacer ces constructions dynamiques de nonces, en utilisant pour représenter ces nonces de nouveaux noms, non utilisés par ailleurs dans le processus. Ainsi, par exemple, le processus :

$$R \ \hat{=}\ (\nu n).\overline{c_{1\to 2}}\langle x\rangle.0$$

peut être remplacé par :

$$R' \ \hat{=}\ \overline{c_{1\to 2}}\langle n_0\rangle.0$$

où n_0 est un nouveau nom. À partir de ce moment, les processus ne contiennent plus que des émissions et réceptions de messages ainsi que des branchements. Il est donc facile, et plus visuel, de les représenter comme des arbres, dont les arêtes sont étiquetées par des actions (envoyer ou recevoir un message) et dont les nœuds correspondent aux différents choix qui se présentent. Cette transformation constitue la deuxième étape du prétraitement. La dernière étape vise à extraire des arbres obtenus la structure qui nous permettra de définir simplement la sémantique de ces protocoles. Cette structure est ce que nous appellerons un point : il s'agit d'un couple constitué des actions qui ont permis d'aboutir à ce point et des actions qui peuvent être exécutées à partir de celui-ci. Détaillons maintenant chacune de ces trois étapes.

Élimination des constructions dynamiques de nonces. Cette construction prend en entrée un processus et retourne un processus sans création dynamique de nonces, ainsi que les nouveaux nonces qui ont été utilisés. Elle est définie de la manière suivante :

$$\begin{array}{l}
\mathrm{elim}((\nu n_1).\ldots(\nu n_k).\overline{c_{i\to j}}\langle m\rangle.R) \;\hat{=}\; (\overline{c_{i\to j}}\langle m\rangle.R'[n'_1/n_1,\ldots,n'_k/n_k],\mathcal{N}'\cup\{n'_1,\ldots,n'_k\})\\
\mathrm{elim}(c_{i\to j}(x).\text{case } x \text{ of } m.[m'=m''].R) \;\hat{=}\; (c_{i\to j}(x).\text{case } x \text{ of } m.[m'=m''].R',\mathcal{N}')\\
\mathrm{elim}(R_1+R_2) \;\hat{=}\; (R'_1+R'_2,\mathcal{N}'_1\cup\mathcal{N}'_2)\\
\mathrm{elim}(0) \;\hat{=}\; (0,\varnothing)
\end{array}$$

où les $n'_1,\ldots,n'_k$ qui interviennent à la première ligne sont des noms frais (*i.e.* qui apparaissent pour la première fois) et où on a posé $(R',\mathcal{N}') \hat{=} \mathrm{elim}(R)$ (respectivement $(R'_1,\mathcal{N}'_1) \hat{=} \mathrm{elim}(R_1)$, $(R'_2,\mathcal{N}'_2) \hat{=} \mathrm{elim}(R_2)$). On peut ensuite définir l'opération d'élimination des constructions dynamiques de nonces dans un processus $P \in \mathrm{Proc}_{\mathrm{Game}}$. C'est l'objet de la définition suivante.

Définition – 10.1.1

Soit $P = T \parallel R_1 \parallel \ldots \parallel R_n \in \mathrm{Proc}_{\mathrm{Game}}$. L'élimination des constructions dynamiques de nonces dans P est définie par induction sur n de la manière suivante :

$$\begin{array}{rcl}
\mathrm{elim}(T) & \hat{=} & (T,())\\
\mathrm{elim}(T \parallel R_1 \parallel \ldots \parallel R_{n-1} \parallel R_n) & \hat{=} & (P' \parallel R'_n,(\mathcal{N}_1,\ldots,\mathcal{N}_{n-1},\mathcal{N}_n))
\end{array}$$

où on a posé :

$$\begin{array}{rcl}
(P',(\mathcal{N}_1,\ldots,\mathcal{N}_{n-1})) & \hat{=} & \mathrm{elim}(T \parallel R_1 \parallel \ldots \parallel R_{n-1})\\
(R'_n,\mathcal{N}_n) & \hat{=} & \mathrm{elim}(R_n)
\end{array}$$

Remarquons que cette définition permet de garder trace des nouveaux noms qui ont été utilisés à l'intérieur de chaque rôle.

Exemple : Sur le protocole jouet (page 107), l'élimination des constructions dynamiques de nonces fournit le résultat suivant :

$$(T \parallel R'_a \parallel R_b,(\{n_0\},\varnothing))$$

avec :

$$\begin{array}{rcl}
R'_a & \hat{=} & \overline{c_{a\to b}}\langle \mathrm{sig}_a(n_0)\rangle.(\; c_{b\to a}(x).\text{case } x \text{ of } \mathrm{sig}_b(n_0).0\\
& & \quad + \overline{c_{a\to t}}\langle \mathrm{sig}_a(n_0)\rangle.c_{t\to a}(x).\text{case } x \text{ of } \mathrm{sig}_b(n_0).0\;)
\end{array}$$

□

On pose maintenant :

$$(T \parallel R'_1 \parallel \ldots \parallel R'_n,(\mathcal{N}_1,\ldots,\mathcal{N}_n)) \;\hat{=}\; \mathrm{elim}(P)$$

où $P = T \parallel R_1 \parallel \ldots \parallel R_n$ est le processus considéré tout au long de ce chapitre. $\mathcal{N}_1,\ldots,\mathcal{N}_n$ représentent les nonces que doivent connaître, au début de l'exécution, les participants qui agissent suivant les processus $R_1,\ldots,R_n$. Ce ne sont pas les seules connaissances initiales des participants en question : ils doivent également connaître leur clé privée, les identités et clés publiques des autres participants, voire même d'autres nonces (qui pourront servir aux participants agissant de manière malhonnête). On notera $IK_1,\ldots,IK_n$ ces ensembles de connaissances initiales. Nous considérons donc à partir de maintenant le couple :

$$(T \parallel R'_1 \parallel \ldots \parallel R'_n,(IK_1,\ldots,IK_n))$$

Transformation des processus en arbres. Avant d'effectuer cette transformation, il faut définir les actions, qui sont soit des envois, soit des réceptions (sous conditions) de messages. Elles sont définies au moyen de la grammaire suivante :

$$\begin{array}{rcl} a,\ldots \in A & ::= & \mathrm{send}(i,m) \\ & | & \mathrm{recv}(m) \\ & | & \mathrm{recv}(m) \text{ when } m' = m'' \end{array}$$

L'action $\mathrm{recv}(m)$ when $m' = m''$ est une action de réception qui ne peut avoir lieu que si les messages m' et m'' sont égaux (généralement, il s'agit de tests effectués sur des nonces). À toute action $a \in A$, nous associons l'action $[a]$ obtenue en supprimant le test d'égalité éventuellement présent. Ainsi on a :

$$\begin{array}{rcl} [\mathrm{recv}(m)] \;=\; [\mathrm{recv}(m) \text{ when } m' = m''] & \hat{=} & \mathrm{recv}(m) \\ [\mathrm{send}(i,m)] & \hat{=} & \mathrm{send}(i,m) \end{array}$$

Nous pouvons maintenant associer à tout processus $R \in \mathrm{Proc}_{\mathrm{Player}}$ un arbre dont les arêtes sont étiquetées par des actions (*i.e.* une partie de A^* close par préfixe, l'étoile étant prise au sens des langages), au moyen de la définition suivante.

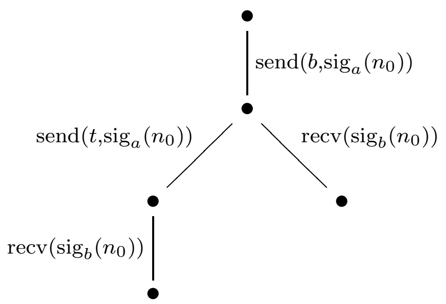
DÉFINITION – 10.1.2

L'arbre associé à un processus $R \in \mathrm{Proc}_{\mathrm{Player}}$, noté $\mathrm{tree}(R)$, est défini par induction sur R de la manière suivante :

- $\mathrm{tree}(0) \hat{=} \{\varepsilon\}$ *(où ε désigne le mot vide) ;*
- $\mathrm{tree}(\overline{c_{i\to j}}\langle m\rangle.R) \hat{=} \{\mathrm{send}(j,m).w \mid w \in \mathrm{tree}(R)\}$ *(rappelons que le traitement qui précède a permis d'éliminer les constructions de nouveaux nonces) ;*
- $\mathrm{tree}(c_{i\to j}(x).\mathrm{case}\ x\ \mathrm{of}\ m.[m' = m''].R) \hat{=} \{\mathrm{recv}(m) \text{ when } m' = m''.w \mid w \in \mathrm{tree}(R)\}$ *;*
- $\mathrm{tree}(R + R') \hat{=} \mathrm{tree}(R) \cup \mathrm{tree}(R')$.

Exemple : L'arbre associé au processus R'_a (obtenu par élimination des constructions dynamiques de nonces sur le processus R_a associé au rôle A du protocole jouet, 107) est le suivant (à gauche, sous forme de langage et, à droite, sous une forme plus visuelle) :

$\{\ \varepsilon,$
$\quad \mathrm{send}(b, \mathrm{sig}_a(n_0)),$
$\quad \mathrm{send}(b, \mathrm{sig}_a(n_0)).\mathrm{recv}(\mathrm{sig}_b(n_0)),$
$\quad \mathrm{send}(b, \mathrm{sig}_a(n_0)).\mathrm{send}(t, \mathrm{sig}_a(n_0)),$
$\quad \mathrm{send}(b, \mathrm{sig}_a(n_0)).\mathrm{send}(t, \mathrm{sig}_a(n_0)).\mathrm{recv}(\mathrm{sig}_b(n_0))\ \}$



□

Notons que nous n'avons pas associé d'arbre au processus définissant le comportement de la TTP , on pourra en effet effectuer la dernière étape directement sur le processus T, sans passer par une représentation arborescente, contrairement aux autres processus. L'élimination des constructions dynamiques de nonces nous avait permis d'obtenir, à partir du processus $P = T \parallel R_1 \parallel \ldots \parallel R_n$ considéré dans ce chapitre, le couple :

$$(T \parallel R'_1 \parallel \ldots \parallel R'_n, (IK_1, \ldots, IK_n))$$

où chaque R'_i est obtenu à partir de R_i en éliminant les constructions dynamiques de nonces et IK_i représente la connaissance initiale du participant jouant le rôle R_i (comprenant, entre autres, les nonces créés dans le processus d'élimination). Nous appliquons maintenant la transformation en arbres pour obtenir le triplet :

$$(T, (TR_1, \ldots, TR_n), (IK_1, \ldots, IK_n))$$

où on a posé $TR_i \mathrel{\hat{=}} \mathrm{tree}(R'_i)$.

Extraction des points d'un processus. Nous appellerons point d'un processus un couple de la forme $((a_1, \ldots, a_p), (a'_1, \ldots, a'_q))$ où $a_1, \ldots, a_p, a'_1, \ldots, a'_q$ sont des actions (on peut éventuellement avoir $p = 0$ ou $q = 0$). La signification d'un tel point est la suivante : si le participant qui agit selon le processus en question a déjà réalisé les actions $a_1, \ldots, a_p$, mais aucune des actions $a'_1, \ldots, a'_q$, alors, si les conditions s'y prêtent, il peut poursuivre le protocole en exécutant une de ces actions. D'un arbre $L \subseteq A^*$, il est facile d'extraire les points, comme le montre la définition suivante.

Définition – 10.1.3

*Soit $L \subseteq A^*$ un arbre (*i.e.* un langage clos par préfixe). L'ensemble des points de L est défini de la manière suivante : à chaque $w = a_1.\ldots.a_p \in L$ (y compris ε) dont les fils sont $w.a'_1, \ldots, w.a'_q$ correspond le point $(([a_1], \ldots, [a_p]), (a'_1, \ldots, a'_q))$ et à chaque $w = a_1.\ldots.a_p \in L$ ne possédant pas de fils dans L correspond le point $(([a_1], \ldots, [a_p]), ())$. L'ensemble de ces points est noté* points(L).

Exemple : À partir de l'arbre de l'exemple page 110, on trouve les points suivants :

$$\begin{aligned}\{ \; & ((), (\mathrm{send}(b, \mathrm{sig}_a(n_0)))),\\ & ((\mathrm{send}(b, \mathrm{sig}_a(n_0))), (\mathrm{send}(t, \mathrm{sig}_a(n_0)), \mathrm{recv}(\mathrm{sig}_b(n_0)))),\\ & ((\mathrm{send}(b, \mathrm{sig}_a(n_0)), \mathrm{recv}(\mathrm{sig}_b(n_0))), ()),\\ & ((\mathrm{send}(b, \mathrm{sig}_a(n_0)), \mathrm{send}(t, \mathrm{sig}_a(n_0))), (\mathrm{recv}(\mathrm{sig}_b(n_0)))),\\ & ((\mathrm{send}(b, \mathrm{sig}_a(n_0)), \mathrm{send}(t, \mathrm{sig}_a(n_0)), \mathrm{recv}(\mathrm{sig}_b(n_0))), ()) \; \}\end{aligned}$$

Noter la correspondance entre :
- le point dont la partie gauche (l'histoire) est vide et la racine de l'arbre associé à R'_a ;
- les points dont la partie droite (le futur) est vide et les nœuds terminaux de l'arbre.

□

Partant des arbres $TR_1, \ldots, TR_n$ obtenus lors de l'étape précédente, on pose :

$$PR_1 \;\hat{=}\; \mathrm{points}(TR_1) \quad \ldots \quad PR_n \;\hat{=}\; \mathrm{points}(TR_n)$$

Remarque : On peut faire de même avec le processus T, quoique de manière plus directe. Nous appellerons point de la TTP un triplet $(a, m, (a_1, a_2))$ où a, a_1, a_2 sont des actions et m un message. La signification d'un tel point sera la suivante : si la TTP a effectué l'action a et elle a déjà envoyé le message m, alors elle peut poursuivre en effectuant l'action a_1. Sinon, c'est à dire si elle a effectué a mais n'a pas envoyé le message m, alors elle peut poursuivre en effectuant a_2. À chaque motif de la forme :

$$
\begin{aligned}
c_{i\to 0}(x).\text{case } x \text{ of } m.\ (\ & [\text{sent}(m')].\overline{c_{0\to i}}\langle m_1\rangle.0 \\
+\ & [\neg\text{sent}(m')].\overline{c_{0\to i}}\langle m_2\rangle.0\ )
\end{aligned}
$$

apparaissant dans T, il suffit alors d'associer le point $(\text{recv}(m), m', (\text{send}(i, m_1), \text{send}(i, m_2)))$. L'ensemble de ces points sera noté PT. □

Exemple : Sur le protocole jouet (page 107), on obtient, pour la TTP , l'unique point :

$$(\text{recv}(\text{sig}_a(n_0)), \text{sig}_b(n_0), (\text{send}(a, \text{sig}_b(n_0)), \text{send}(a, \text{sig}_b(n_0))))$$

□

Nous pouvons maintenant définir l'ATS associé au processus P, en utilisant seulement pour cela les couples :

$$(PR_1, IK_1), \ldots, (PR_n, IK_n)$$

ainsi que l'ensemble de points PT, obtenus par les traitements qui précèdent.

10.2 ATS associé à un protocole

Nous allons dans cette partie associer un ATS au protocole P. Plus exactement, nous associerons un ATS à P une fois fixé le niveau d'honnêteté des participants ainsi que le niveau de fiabilité des canaux de communication. Cet ATS possédera un nombre fini d'agents, mais l'ensemble des états sera infini (plus précisément : chaque agent possédera un nombre infini de variables). Expliquons tout d'abord comment représenter les niveaux de fiabilité et d'honnêteté.

Niveau d'honnêteté et de fiabilité. Comme expliqué dans l'introduction, nous considérons plusieurs comportements possibles pour les participants (honnête, faiblement malhonnête et fortement malhonnête) et les canaux de communication (opérationnel, résistant et non-fiable). La sémantique d'un protocole ne peut donc être définie qu'après avoir fixé un niveau d'honnêteté pour les participants, ainsi qu'un niveau de fiabilité pour les canaux de communication. Posons :

$$
\begin{aligned}
\Omega_{\text{Part}} &\mathrel{\hat{=}} \{1, \ldots, n\} \\
\Omega_{\text{TTP}} &\mathrel{\hat{=}} \{0\} \\
\Omega_{\text{Com}} &\mathrel{\hat{=}} \{c_{i\to j} \mid i, j \in \{0, 1, \ldots, n\} \text{ et } i \neq j\} \\
\Omega &\mathrel{\hat{=}} \Omega_{\text{TTP}} \cup \Omega_{\text{Part}} \cup \Omega_{\text{Com}}
\end{aligned}
$$

de sorte que Ω_{Part} est l'ensemble des participants au protocole (à l'exception de la TTP), Ω_{TTP} ne contient que la TTP et Ω_{Com} est l'ensemble des canaux de communication. Le choix des niveaux d'honnêteté et fiabilité peut alors se faire au moyen de deux fonctions :

$$
\begin{aligned}
\ell : \ \Omega_{\text{Part}} &\to \{\text{honest}, \text{weakly}, \text{strongly}\} \\
\ell : \ \Omega_{\text{Com}} &\to \{\text{operational}, \text{resilient}, \text{unreliable}\}
\end{aligned}
$$

Par souci d'homogénéité, nous dirons que la TTP ne peut avoir qu'un seul niveau de fiabilité, noté ttp et nous poserons $\ell(0) \mathrel{\hat{=}} \text{ttp}$. Nous allons définir maintenant la sémantique du processus P relativement à ces deux fonctions, supposées définies une fois pour toutes. Avant cela, explicitons une notation dont nous aurons besoin par la suite.

Proposition $\mathrm{knows}_i(m)$**.** Il nous sera utile, parfois, de garder certaines commandes d'un agent $i \in \Omega_{\mathrm{Part}}$ par le fait que l'agent en question connaisse ou non un certain message clos $m \in \mathrm{Msg}$. Pour cela, nous utiliserons la proposition :

$$\mathrm{knows}_i(m)$$

avec la convention que cette proposition est vraie, dans l'état local q_i de i, si, et seulement si, m peut être déduit à partir de la connaissance initiale de i ainsi que des messages reçus par i. Cette proposition peut d'ailleurs être définie par l'énoncé suivant (rappelons que les conjonctions et disjonctions infinies sont autorisées) :

$$\mathrm{knows}_i(m) \quad \hat{=} \quad \bigvee_{\substack{s \subseteq \mathrm{Msg} \\ s \Vdash m}} \ \bigwedge_{m' \in s} (i.\mathrm{recv}(m') \vee m' \in IK_i)$$

Cette définition de knows() est équivalente à la suivante, où disjonctions et conjonctions infinies ont été remplacées par des quantifications :

$$\mathrm{knows}_i(m) \quad = \quad \exists s.\ s \Vdash m \wedge (\forall m_1 \in s.\ i.\mathrm{recv}(m_1) \vee m_1 \in IK_i)$$

Nous sommes maintenant prêts à associer un ATS au protocole P. Nous allons tout d'abord décrire quels sont les agents qui interviennent dans cet ATS et comment sont constitués les états locaux de ces agents. Nous donnerons ensuite les commandes gardées qui décrivent le comportement de chaque type d'agent, suivant son niveau de fiabilité.

Les agents, leur état local. Les agents de l'ATS donnant la sémantique du processus P sont d'une part les participants à ce protocole, incluant la TTP (*i.e.* $\Omega_{\mathrm{Part}} \cup \Omega_{\mathrm{TTP}}$) et d'autre part les canaux de communication entre ces agents, dont l'ensemble est noté Ω_{Com} (voir plus haut).

Considérons un agent $i \in \Omega_{\mathrm{Part}}$ représentant un participant du protocole (à l'exception de la TTP). L'état local d'un tel agent devra garder trace des messages qui ont été envoyés et reçus. Ainsi, pour chaque message $m \in \mathrm{Msg}$, clos, nous définirons une variable $i.\mathrm{recv}(m)$ qui sera vraie si, et seulement si, l'agent i a reçu le message m. Notons que l'émetteur (réel ou supposé) de ce message constituera une information inaccessible à l'agent qui reçoit le message en question. Pour chaque agent $j \in \Omega_{\mathrm{Part}} \cup \Omega_{\mathrm{TTP}}$, avec $j \neq i$, nous définirons également une variable $i.\mathrm{send}(j, m)$ qui indiquera si l'agent i a envoyé le message m à l'agent j. Finalement, nous aurons également besoin d'une variable notée $i.\mathrm{stopped}$ pour indiquer si l'agent i a terminé le protocole. Dans l'état initial de l'ATS, l'agent n'a reçu ni envoyé aucun message et il ne peut pas non plus avoir terminé le protocole. Toutes les variables locales de cet agent sont dont initialisées avec la valeur false.

L'état local de l'agent représentant la TTP est sensiblement défini de la même manière, à ceci près que, comme la TTP n'est pas supposée terminer le protocole, il n'y a pas besoin de variable stopped. La TTP n'a pas besoin non plus de garder trace des requêtes qu'elle a reçu, les réponses qu'elle a envoyé lui suffisent en effet pour lui permettre de continuer à jouer son rôle. La TTP ne possède donc que des variables de la forme $0.\mathrm{send}(j, m)$, où $j \in \Omega_{\mathrm{Part}}$ et $m \in \mathrm{Msg}$ est clos, toutes initialisées avec la valeur false.

Pour un agent $c_{i \to j} \in \Omega_{\mathrm{Com}}$, nous ne retiendrons dans son état local que les messages qui ont été transmis, sous la forme d'une variable $c_{i \to j}.\mathrm{fwd}(m)$ (pour chaque $m \in \mathrm{Msg}$ clos). Cette variable sera naturellement fausse dans l'état initial.

La figure 10.2.1 reprend ces constructions, dans le langage de description des ATS présenté à la fin du chapitre 9.

Déclaration des agents :

$$\textbf{agents} \quad : \quad i \in \Omega_{\text{Part}}, 0 \in \Omega_{\text{TTP}}, c_{i\to j} \in \Omega_{\text{Com}}$$

Pour la TTP (et pour tout $j \in \Omega_{\text{Part}}$ et tout $m \in$ Msg, clos) :

$$\begin{array}{lcl} \textbf{vars}(0) & : & 0.\text{send}(j,m) \\ \textbf{init}(0) & : & 0.\text{send}(j,m) := \text{false} \end{array}$$

Pour le participant $i \in \Omega_{\text{Part}}$ (et pour tout $j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$ avec $j \neq i$ et $m \in$ Msg, clos) :

$$\begin{array}{lcl} \textbf{vars}(i) & : & i.\text{stopped}, i.\text{recv}(m), i.\text{send}(j,m) \\ \textbf{init}(i) & : & i.\text{stopped} := \text{false}; i.\text{recv}(m) := \text{false}; i.\text{send}(j,m) := \text{false} \end{array}$$

Pour $i, j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$ avec $i \neq j$ (et pour tout $m \in$ Msg, clos) :

$$\begin{array}{lcl} \textbf{vars}(c_{i\to j}) & : & c_{i\to j}.\text{fwd}(m) \\ \textbf{init}(c_{i\to j}) & : & c_{i\to j}.\text{fwd}(m) := \text{false} \end{array}$$

FIG. 10.2.1 – *Les agents de l'ATS associé à P et leur état local*

Agent honnête $i \in \Omega_{\text{Part}}$. La première chose que peut faire un tel agent, c'est envoyer et recevoir des messages conformément au protocole. Ainsi, pour un point $((a_1, \ldots, a_p), (a'_1, \ldots, a'_q)) \in PR_i$ avec $q \geq 1$ et pour $k \in \{1, \ldots, q\}$, on a une commande gardée qui correspond au fait que, après avoir effectué les actions $a_1, \ldots, a_p$, et sous certaines conditions, le participant i peut effectuer l'action a'_k. Comme il peut y avoir des variables libres dans ces actions, il faut considérer toutes les instanciations possibles et cohérentes entre-elles de celles-ci. On considère pour cela une substitution σ. Dans la commande gardée en question, on commence par vérifier que le participant i n'a pas arrêté le protocole et qu'il a déjà effectué les actions $a_1\sigma, \ldots, a_p\sigma$. On vérifie ensuite qu'il n'a effectué aucune action parmi $a'_1\sigma, \ldots, a'_q\sigma$. Il est alors en mesure d'exécuter l'action $a'_k\sigma$, mais il reste pour cela des conditions à vérifier qui dépendent de la forme de $a'_k\sigma$:

- si $a'_k = \text{send}(j, m)$, alors l'agent i peut envoyer le message $m\sigma$ à j à condition qu'il soit capable de produire le message $m\sigma$;
- si $a'_k = \text{recv}(m)$ when $m' = m''$, alors l'agent i peut accepter le message $m\sigma$, à condition qu'il ait reçu $m\sigma$ d'un des canaux de communication $c_{j\to i}$ et que les messages $m'\sigma$ et $m''\sigma$ soient égaux.

Les commandes obtenues sont soumises à des contraintes d'équité car, si l'agent est honnête, il ne peut pas refuser indéfiniment d'avancer dans le protocole (sous réserve qu'il en ait la possibilité). Nous n'avons pas considéré, dans ce qui précède, les point de la forme $((a_1, \ldots, a_p), ()) \in PR_i$. Un tel point signifie que, après avoir effectué les actions $a_1\sigma, \ldots, a_p\sigma$ l'agent i peut décider d'arrêter le protocole (s'il ne l'a pas déjà fait). La commande gardée obtenue est également soumise à une contrainte d'équité, puisque l'agent est honnête. La dernière commande intervenant dans la description d'un agent honnête est une commande lui permettant d'attendre. Les commandes décrites ici sont détaillées dans la figure 10.2.2.

Pour $((a_1,\dots,a_p),(a'_1,\dots,a'_q)) \in PR_i$ et une substitution σ :
– si $q \geq 1$, $k \in \{1,\dots,q\}$ et $a'_k = \mathrm{send}(j,m)$:

$$\begin{array}{lll} []_{\mathrm{f}} & \neg i.\mathrm{stopped} & \to i.\mathrm{send}(j,m\sigma) := \mathrm{true} \\ & \wedge\ i.[a_1\sigma] \wedge \dots \wedge i.[a_p\sigma] & \\ & \wedge\ \neg i.[a'_1\sigma] \wedge \dots \wedge \neg i.[a'_q\sigma] & \\ & \wedge\ \mathrm{knows}_i(m\sigma) & \end{array}$$

– si $q \geq 1$, $k \in \{1,\dots,q\}$ et $a'_k = \mathrm{recv}(m)$ when $m' = m''$:

$$\begin{array}{lll} []_{\mathrm{f}} & \neg i.\mathrm{stopped} & \to i.\mathrm{recv}(m\sigma) := \mathrm{true} \\ & \wedge\ i.[a_1\sigma] \wedge \dots \wedge i.[a_p\sigma] & \\ & \wedge\ \neg i.[a'_1\sigma] \wedge \dots \wedge \neg i.[a'_q\sigma] & \\ & \wedge\ (\vee_{\substack{j\in\Omega_{\mathrm{Part}}\cup\Omega_{\mathrm{TTP}} \\ j\neq i}} c_{j\to i}.\mathrm{fwd}(m\sigma)) & \\ & \wedge\ m'\sigma = m''\sigma & \end{array}$$

– si $q = 0$:

$$[]_{\mathrm{f}}\ \neg i.\mathrm{stopped} \wedge i.[a_1\sigma] \wedge \dots \wedge i.[a_p\sigma] \to i.\mathrm{stopped} := \mathrm{true}$$

Pour attendre :

$$[]\ \mathrm{true} \to \mathrm{idle}$$

FIG. 10.2.2 – *Comportement honnête d'un agent $i \in \Omega_{\mathrm{Part}}$*

Agent faiblement malhonnête $i \in \Omega_{\mathrm{Part}}$. Un tel agent peut envoyer chaque message clos $m \in \mathrm{Msg}$ qu'il est capable de produire et accepter chaque message qui lui est transmis par un canal de communication $c_{j\to i}$, tout ceci sans se soucier de ce que lui permet le protocole. Il a également la possibilité de décréter qu'il a terminé le protocole ou encore de ne rien faire. Les commandes correspondantes sont détaillées dans la figure 10.2.3

Agent fortement malhonnête $i \in \Omega_{\mathrm{Part}}$. À la différence d'un agent faiblement malhonnête, un agent fortement malhonnête peut accepter de recevoir des messages de tous les canaux de communication. Les commandes ainsi obtenues sont décrites à la figure 10.2.4.

Agent $0 \in \Omega_{\mathrm{TTP}}$ représentant la TTP . Nous avons fait le choix de modéliser cet agent de sorte qu'il réponde immédiatement aux demandes qu'il reçoit. Ainsi, pour chaque point $(\mathrm{recv}(m), m', (\mathrm{send}(i,m_1), \mathrm{send}(i,m_2))) \in PT$ et chaque substitution σ, nous définissons deux commandes. La première, à la réception du message $m\sigma$, teste si le message $m'\sigma$ a déjà été envoyé et, dans ce cas, envoie $m_1\sigma$ à i. La seconde, à la réception du message $m\sigma$, teste si le message $m'\sigma$ n'a pas déjà été envoyé et, dans ce cas, envoie $m_2\sigma$ à i. La TTP a aussi la possibilité de ne rien faire, mais uniquement lorsqu'aucune des commandes précédentes ne s'applique. C'est pourquoi elle possède une commande, gardée par la conjonction des négations des gardes des commandes précédentes et dont l'action est idle. Ces commandes sont regroupées figure 10.2.5

Émission de $m \in \text{Msg}$, clos, vers $j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$, $j \neq i$:

$$[\!] \neg i.\text{stopped} \wedge \text{knows}_i(m) \wedge \neg i.\text{send}(j, m) \rightarrow i.\text{send}(j, m) := \text{true}$$

Réception de $m \in \text{Msg}$, clos :

$$[\!] \neg i.\text{stopped} \wedge (\vee_{\substack{j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}} \\ j \neq i}} c_{j \rightarrow i}.\text{fwd}(m)) \wedge \neg i.\text{recv}(m) \rightarrow i.\text{recv}(m) := \text{true}$$

Arrêt du protocole :

$$[\!] \neg i.\text{stopped} \rightarrow i.\text{stopped} := \text{true}$$

Pour attendre :

$$[\!] \text{ true} \rightarrow \text{idle}$$

FIG. 10.2.3 – *Comportement faiblement malhonnête d'un agent $i \in \Omega_{\text{Part}}$*

Émission de $m \in \text{Msg}$, clos, vers $j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$, $j \neq i$:

$$[\!] \neg i.\text{stopped} \wedge \text{knows}_i(m) \wedge \neg i.\text{send}(j, m) \rightarrow i.\text{send}(j, m) := \text{true}$$

Réception de $m \in \text{Msg}$, clos :

$$[\!] \neg i.\text{stopped} \wedge (\vee_{\substack{j,k \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}} \\ j \neq i, k \neq j}} c_{j \rightarrow i}.\text{fwd}(m)) \wedge \neg i.\text{recv}(m) \rightarrow i.\text{recv}(m) := \text{true}$$

Arrêt du protocole :

$$[\!] \neg i.\text{stopped} \rightarrow i.\text{stopped} := \text{true}$$

Pour attendre :

$$[\!] \text{ true} \rightarrow \text{idle}$$

FIG. 10.2.4 – *Comportement fortement malhonnête d'un agent $i \in \Omega_{\text{Part}}$*

Pour $(\text{recv}(m), m', (\text{send}(i, m_1), \text{send}(i, m_2))) \in PT$ et une substitution σ :

$$[\!] (\vee_{j \Omega_{\text{Part}}} c_{j \rightarrow 0}.\text{fwd}(m\sigma)) \wedge (\vee_{j \in \Omega_{\text{Part}}} 0.\text{send}(j, m'\sigma)) \wedge \neg 0.\text{send}(i, m_1\sigma) \rightarrow 0.\text{send}(i, m_1\sigma) := \text{true}$$

$$[\!] (\vee_{j \in \Omega_{\text{Part}}} c_{j \rightarrow 0}.\text{fwd}(m\sigma)) \wedge \neg(\vee_{j \in \Omega_{\text{Part}}} 0.\text{send}(j, m'\sigma)) \wedge \neg 0.\text{send}(i, m_2\sigma) \rightarrow 0.\text{send}(i, m_2\sigma) := \text{true}$$

Pour attendre :

$$[\!] \bigwedge_{p \in PT, \sigma} \begin{pmatrix} \neg(\vee_{j \in \Omega_{\text{Part}}} c_{j \rightarrow 0}.\text{fwd}(m\sigma)) \\ \vee \neg 0.\text{send}(m'\sigma) \\ \vee\ 0.\text{send}(i, m_1\sigma) \end{pmatrix} \wedge \bigwedge_{p \in PT, \sigma} \begin{pmatrix} \neg(\vee_{j \in \Omega_{\text{Part}}} c_{j \rightarrow 0}.\text{fwd}(m\sigma)) \\ \vee (\vee_{j \in \Omega_{\text{Part}}} 0.\text{send}(j, m'\sigma)) \\ \vee\ 0.\text{send}(i, m_2\sigma) \end{pmatrix} \rightarrow \text{idle}$$

(où $p = (\text{recv}(m), m', (\text{send}(i, m_1), \text{send}(i, m_2))) \in PT$).

FIG. 10.2.5 – *Comportement de la TTP*

Canal opérationnel $c_{i \to j} \in \Omega_{\text{Com}}$. Un tel canal doit immédiatement transmettre tout message envoyé par i à j. On a donc, pour tout message clos $m \in \text{Msg}$, une commande gardée qui, si m a été envoyé par i à j, mais pas transmis par $c_{i \to j}$, met la valeur true dans la variable $\text{fwd}(m)$ de $c_{i \to j}$. Un tel canal a la possibilité d'attendre, mais uniquement lorsqu'aucune des commandes précédentes ne peut être appliquée. Les commandes obtenues sont regroupées à la figure 10.2.6.

Pour délivrer un message $m \in \text{Msg}$, clos :

$$[\!]\ i.\text{send}(j, m) \wedge \neg c_{i \to j}.\text{fwd}(m) \to c_{i \to j}.\text{fwd}(m) := \text{true}$$

Pour attendre :

$$[\!]\ \wedge_{m \in \text{Msg}} (\neg i.\text{send}(j, m) \vee c_{i \to j}.\text{fwd}(m)) \to \text{idle}$$

FIG. 10.2.6 – *Canal opérationnel* $c_{i \to j} \in \Omega_{\text{Com}}$

Canal résistant $c_{i \to j} \in \Omega_{\text{Com}}$. À la différence d'un canal opérationnel, un canal résistant peut attendre arbitrairement longtemps (mais pas indéfiniment) avant de délivrer un message. Un tel canal a donc toujours la possibilité de ne rien faire et il n'y a par conséquent pas de restriction pour exécuter la commande idle. En contrepartie, on met des contraintes d'équité sur les commandes permettant de délivrer des messages, afin que ceux-ci soient envoyés en temps fini. On obtient ainsi les commandes de la figure 10.2.7

Pour délivrer un message $m \in \text{Msg}$, clos :

$$[\!]_{\text{f}}\ i.\text{send}(j, m) \wedge \neg c_{i \to j}.\text{fwd}(m) \to c_{i \to j}.\text{fwd}(m) := \text{true}$$

Pour attendre :

$$[\!]\ \text{true} \to \text{idle}$$

FIG. 10.2.7 – *Canal résistant* $c_{i \to j} \in \Omega_{\text{Com}}$

Canal non-fiable $c_{i \to j} \in \Omega_{\text{Com}}$. Un tel canal a la possibilité de ne jamais délivrer certains messages. On obtient donc sa modélisation à partir de celle d'un canal résistant en supprimant simplement les contraintes d'équité (figure 10.2.8).

Pour délivrer un message $m \in \text{Msg}$, clos :

$$[\!]\ i.\text{send}(j, m) \wedge \neg c_{i \to j}.\text{fwd}(m) \to c_{i \to j}.\text{fwd}(m) := \text{true}$$

Pour attendre :

$$[\!]\ \text{true} \to \text{idle}$$

FIG. 10.2.8 – *Canal non-fiable* $c_{i \to j} \in \Omega_{\text{Com}}$

Conclusion : sémantique de P. Les déclarations qui précèdent permettent de définir :

- un ensemble d'agents et un ensemble de propositions atomiques qui ne dépendent que du processus de départ $P \in \mathrm{Proc}_{\mathrm{Game}}$, on les notera Ω_P et Π_P ;
- un ATS sur Ω_P et Π_P qui ne dépend que du processus P de départ et des niveaux d'honnêteté et fiabilité ℓ, on le notera $\mathcal{S}_P^\ell$.

Ces objets constituent la sémantique de P. Techniquement parlant, la sémantique de P est composée des ensembles Ω_P et Π_P ainsi que de la famille d'ATS $(\mathcal{S}_P^\ell)_\ell$ (ℓ parcourant tous les niveaux d'honnêteté et de fiabilité possibles).

Définition – 10.2.1

> *Soit $P \in \mathrm{Proc}_{\mathrm{Game}}$ un processus. La sémantique, au sens des jeux, de P, notée $[\![P]\!]_{\mathrm{Game}}$, est le triplet :*
>
> $$(\Omega_P, \Pi_P, (\mathcal{S}_P^\ell)_\ell)$$
>
> *où ℓ parcourt les niveaux d'honnêteté possibles de ce processus.*

Nous avons donc défini la sémantique d'un protocole P sous forme d'une famille d'ATS. Avant de donner des exemples de propriétés de sécurité de ces protocoles sous forme d'énoncés de la logique ATL, nous allons détailler la sémantique obtenue sur un exemple, afin de fixer les idées.

10.3 Un exemple

Nous considérons le protocole jouet décrit page 107. Nous allons donner sa sémantique en termes d'ATS, pour un niveau d'honnêteté et de fiabilité fixé. Pour plus de simplicité, on note t, a et b à la place de 0, 1 et 2, respectivement. On choisit de prendre le participant a honnête, le participant b fortement malhonnête, les canaux de communication $c_{a\to b}$ et $c_{b\to a}$ non-fiables et les canaux $c_{a\to t}$, $c_{t\to a}$, $c_{b\to t}$ et $c_{t\to b}$ résistants. Détaillons maintenant la sémantique de ce protocole sous forme d'un ATS, en suivant la présentation de la section précédente.

Déclarations des agents, des variables et commandes d'initialisation. Pour le protocole jouet, la déclaration des agents (cf. figure 10.2.1, page 114) :

$$\textbf{agents} \quad : \quad a, b, t, c_{a\to b}, c_{a\to t}, c_{b\to a}, c_{b\to t}, c_{t\to a}, c_{t\to b}$$

En ce qui concerne les variables locales, on a la déclaration suivante (dans laquelle la notation m signifie « pour tout message $m \in \mathrm{Msg}$ ») :

$$\begin{array}{lcl}
\textbf{vars}(a) & : & a.\mathrm{stopped}, a.\mathrm{recv}(m), a.\mathrm{send}(t,m), a.\mathrm{send}(b,m) \\
\textbf{vars}(b) & : & b.\mathrm{stopped}, b.\mathrm{recv}(m), b.\mathrm{send}(t,m), b.\mathrm{send}(a,m) \\
\textbf{vars}(t) & : & t.\mathrm{send}(a,m), t.\mathrm{send}(b,m) \\
\textbf{vars}(c_{a\to b}) & : & c_{a\to b}.\mathrm{fwd}(m) \\
\textbf{vars}(c_{a\to t}) & : & c_{a\to t}.\mathrm{fwd}(m) \\
\textbf{vars}(c_{b\to a}) & : & c_{b\to a}.\mathrm{fwd}(m) \\
\textbf{vars}(c_{b\to t}) & : & c_{b\to t}.\mathrm{fwd}(m) \\
\textbf{vars}(c_{t\to a}) & : & c_{t\to a}.\mathrm{fwd}(m) \\
\textbf{vars}(c_{t\to b}) & : & c_{t\to b}.\mathrm{fwd}(m)
\end{array}$$

Comme expliqué dans la section précédente, toutes ces variables sont initialisées avec la valeur false :

$$
\begin{array}{lcl}
\textbf{init}(a) & : & a.\text{stopped}, a.\text{recv}(m), a.\text{send}(t,m), a.\text{send}(b,m) := \text{false} \\
\textbf{init}(b) & : & b.\text{stopped}, b.\text{recv}(m), b.\text{send}(t,m), b.\text{send}(a,m) := \text{false} \\
\textbf{init}(t) & : & t.\text{send}(a,m) := \text{false}; t.\text{send}(b,m) := \text{false} \\
\textbf{init}(c_{a\to b}) & : & c_{a\to b}.\text{fwd}(m) := \text{false} \\
\textbf{init}(c_{a\to t}) & : & c_{a\to t}.\text{fwd}(m) := \text{false} \\
\textbf{init}(c_{b\to a}) & : & c_{b\to a}.\text{fwd}(m) := \text{false} \\
\textbf{init}(c_{b\to t}) & : & c_{b\to t}.\text{fwd}(m) := \text{false} \\
\textbf{init}(c_{t\to a}) & : & c_{t\to a}.\text{fwd}(m) := \text{false} \\
\textbf{init}(c_{t\to b}) & : & c_{t\to b}.\text{fwd}(m) := \text{false}
\end{array}
$$

Agent honnête a. On applique les règles de la figure 10.2.2, page 115 en considérant les points de l'agent a dans l'ordre dans lequel ils apparaissent dans l'exemple page 111. Notons que, comme les messages apparaissant dans le rôle de a sont clos, il est inutile de faire intervenir ici une substitution.

$$
\begin{array}{ll}
\|_{\text{f}}\ \neg a.\text{stopped} \wedge \neg a.\text{send}(b, \text{sig}_a(n_0)) \wedge \text{knows}_i(\text{sig}_a(n_0)) & \to a.\text{send}(b, \text{sig}_a(n_0)) := \text{true} \\
\|_{\text{f}}\quad \neg a.\text{stopped} \wedge a.\text{send}(b, \text{sig}_a(n_0)) & \to a.\text{send}(t, \text{sig}_a(n_0)) := \text{true} \\
\quad \wedge\ \neg a.\text{send}(t, \text{sig}_a(n_0)) \wedge \neg a.\text{recv}(\text{sig}_b(n_0)) & \\
\quad \wedge\ \text{knows}_i(\text{sig}_a(n_0)) & \\
\|_{\text{f}}\quad \neg a.\text{stopped} \wedge a.\text{send}(b, \text{sig}_a(n_0)) & \to a.\text{recv}(\text{sig}_b(n_0)) := \text{true} \\
\quad \wedge\ \neg a.\text{send}(t, \text{sig}_a(n_0)) \wedge \neg a.\text{recv}(\text{sig}_b(n_0)) & \\
\quad \wedge\ (c_{t\to a}.\text{fwd}(\text{sig}_b(n_0)) \vee c_{b\to a}.\text{fwd}(\text{sig}_b(n_0))) & \\
\|_{\text{f}}\quad \neg a.\text{stopped} \wedge a.\text{send}(b, \text{sig}_a(n_0)) \wedge a.\text{send}(t, \text{sig}_a(n_0)) & \to a.\text{recv}(\text{sig}_b(n_0)) := \text{true} \\
\quad \wedge\ \neg a.\text{recv}(\text{sig}_b(n_0)) & \\
\quad \wedge\ (c_{t\to a}.\text{fwd}(\text{sig}_b(n_0)) \vee c_{b\to a}.\text{fwd}(\text{sig}_b(n_0))) & \\
\|_{\text{f}}\quad \neg a.\text{stopped} & \to a.\text{stopped} := \text{true} \\
\quad \wedge\ a.\text{send}(b, \text{sig}_a(n_0)) \wedge a.\text{send}(t, \text{sig}_a(n_0)) \wedge a.\text{recv}(\text{sig}_b(n_0)) & \\
\|_{\text{f}}\ \neg a.\text{stopped} \wedge a.\text{send}(b, \text{sig}_a(n_0)) \wedge a.\text{recv}(\text{sig}_b(n_0)) \to a.\text{stopped} := \text{true} & \\
\|\ \text{true} \to \text{idle} &
\end{array}
$$

Agent fortement malhonnête b. D'après la section précédente (figure 10.2.4, page 116), cet agent peut envoyer les messages qu'il peut produire à tous les autres (a et t ici) ainsi qu'espionner tous les canaux (lorsqu'il n'est pas l'émetteur). On obtient les commandes :

$$
\begin{array}{ll}
\|\ \neg b.\text{stopped} \wedge \text{knows}_b(m) \wedge \neg b.\text{send}(a,m) \to b.\text{send}(a,m) := \text{true} & \\
\|\ \neg b.\text{stopped} \wedge \text{knows}_b(m) \wedge \neg b.\text{send}(t,m) \to b.\text{send}(t,m) := \text{true} & \\
\|\quad \neg b.\text{stopped} \wedge \neg b.\text{recv}(m) & \to b.\text{recv}(m) := \text{true} \\
\quad \wedge\ (c_{a\to b}.\text{fwd}(m) \vee c_{a\to t}.\text{fwd}(m) \vee c_{t\to a}.\text{fwd}(m) \vee c_{t\to b}.\text{fwd}(m)) & \\
\|\ \neg b.\text{stopped} \to b.\text{stopped} := \text{true} & \\
\|\ \text{true} \to \text{idle} &
\end{array}
$$

TTP t. Il n'y a qu'un seul point pour la TTP du protocole jouet (exemple page 112). Les commandes décrites à la figure 10.2.5, page 116, s'écrivent alors :

$$
\begin{array}{ll}
[\!] \quad (c_{a\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma) \vee c_{b\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma)) & \to t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) := \mathrm{true} \\
\quad \wedge\ (t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) \vee t.\mathrm{send}(b, \mathrm{sig}_b(y)\sigma)) & \\
\quad \wedge\ \neg t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) & \\
[\!] \quad (c_{a\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma) \vee c_{b\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma)) & \to t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) := \mathrm{true} \\
\quad \wedge\ \neg(t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) \vee t.\mathrm{send}(b, \mathrm{sig}_b(y)\sigma)) & \\
\quad \wedge\ \neg t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) & \\
[\!] \bigwedge \quad (\neg c_{a\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma) \wedge \neg c_{b\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma)) & \to \mathrm{idle} \\
\qquad \vee\ (\neg t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) \wedge \neg t.\mathrm{send}(b, \mathrm{sig}_b(y)\sigma)) & \\
\qquad \vee\ t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) & \\
\quad \bigwedge \quad (\neg c_{a\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma) \wedge \neg c_{b\to t}.\mathrm{fwd}(\mathrm{sig}_a(y)\sigma)) & \\
\qquad \vee\ (t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) \vee t.\mathrm{send}(b, \mathrm{sig}_b(y)\sigma)) & \\
\qquad \vee\ t.\mathrm{send}(a, \mathrm{sig}_b(y)\sigma) &
\end{array}
$$

On remarque ici la présence de substitutions, puisqu'il y a des variables présentes dans le rôle de t. Dans la dernière commande, les conjonctions portent sur toutes les substitutions σ.

Canaux résistants $c_{a\to t}, c_{t\to a}, c_{b\to t}, c_{t\to b}$. On utilise les commandes décrites dans la figure 10.2.7, page 117 :

– pour $c_{a\to t}$:

$$
\begin{array}{l}
[\!]_{\mathrm{f}}\ a.\mathrm{send}(t, m) \wedge \neg c_{a\to t}.\mathrm{fwd}(m) \to c_{a\to t}.\mathrm{fwd}(m) := \mathrm{true} \\
[\!]\ \mathrm{true} \to \mathrm{idle}
\end{array}
$$

– pour $c_{t\to a}$:

$$
\begin{array}{l}
[\!]_{\mathrm{f}}\ t.\mathrm{send}(a, m) \wedge \neg c_{t\to a}.\mathrm{fwd}(m) \to c_{t\to a}.\mathrm{fwd}(m) := \mathrm{true} \\
[\!]\ \mathrm{true} \to \mathrm{idle}
\end{array}
$$

– pour $c_{b\to t}$:

$$
\begin{array}{l}
[\!]_{\mathrm{f}}\ b.\mathrm{send}(t, m) \wedge \neg c_{b\to t}.\mathrm{fwd}(m) \to c_{b\to t}.\mathrm{fwd}(m) := \mathrm{true} \\
[\!]\ \mathrm{true} \to \mathrm{idle}
\end{array}
$$

– pour $c_{t\to b}$:

$$
\begin{array}{l}
[\!]_{\mathrm{f}}\ t.\mathrm{send}(b, m) \wedge \neg c_{t\to b}.\mathrm{fwd}(m) \to c_{t\to b}.\mathrm{fwd}(m) := \mathrm{true} \\
[\!]\ \mathrm{true} \to \mathrm{idle}
\end{array}
$$

Canaux non-fiables $c_{a\to b}, c_{b\to a}$. On utilise les commandes décrites dans la figure 10.2.8, page 117 :

– pour $c_{a\to b}$:

$$
\begin{array}{l}
[\!]\ a.\mathrm{send}(b, m) \wedge \neg c_{a\to b}.\mathrm{fwd}(m) \to c_{a\to b}.\mathrm{fwd}(m) := \mathrm{true} \\
[\!]\ \mathrm{true} \to \mathrm{idle}
\end{array}
$$

– pour $c_{b\to a}$:

$$\begin{aligned}&\| \ b.\text{send}(a,m) \wedge \neg c_{b\to a}.\text{fwd}(m) \rightarrow c_{b\to a}.\text{fwd}(m) := \text{true}\\ &\| \ \text{true} \rightarrow \text{idle}\end{aligned}$$

Regroupant toutes les parties de description d'ATS données ici, nous obtenons un ATS, $\mathcal{S}_P^\ell$ associé au protocole décrit par P ainsi qu'aux niveaux d'honnêteté et fiabilité décrits plus haut. Insistons bien sur le fait que ceci ne constitue qu'une partie de la sémantique de P. La sémantique complète s'obtiendrait en considérant tous les niveaux d'honnêteté et fiabilité possibles.

10.4 Propriétés de sécurité

Comme pour les propriétés de traces, nous allons donner ici quelques exemples de propriétés de sécurité, sous forme d'énoncés de la logique ATL. Ces exemples sont extraits de [KR02]. Reprenons pour commencer le protocole de Garay, Jakobson et Mac Kenzie présenté dans l'introduction de cette partie. Supposons-le décrit par un processus $P \mathrel{\hat{=}} T \parallel R_a \parallel R_b$. Ce protocole est équitable, pour b, si a n'a pas de stratégie pour arriver à un point du protocole où, d'une part, il possède le contrat signé par b et, d'autre part, b n'a plus la possibilité d'obtenir le contrat signé par a. Utilisant la logique ATL, ceci s'écrit naturellement :

$$\mathit{fairness}_b \quad \hat{=} \quad \neg \langle \{a\} \cup \Omega_{\text{TTP}} \cup \Omega_{\text{Com}} \rangle \Diamond (\text{knows}_a(\text{sig}_b(c)) \wedge \neg \langle \{b\} \rangle \Diamond \text{knows}_b(\text{sig}_a(c)))$$

Cet énoncé, à première vue correct, soulève plusieurs interrogations, du domaine de la modélisation. Par exemple, il est clair que a doit apparaître dans la première modalité et b dans la seconde, mais comment choisit-on où placer les canaux de communication et la TTP ? Quels sont les niveaux d'honnêteté (et de fiabilité) pertinents pour vérifier une telle propriété ? Peut-on trouver d'autres énoncés pour cette même propriété ?

Où placer les canaux de communication et la TTP . Réécrivons l'énoncé précédant en calculant explicitement les négations et sans choisir *a priori* les ensembles d'agents paramétrant les modalités. On trouve :

$$\mathit{fairness}_b(A,B) \quad \hat{=} \quad [A]\,\Box(\neg\text{knows}_a(\text{sig}_b(c)) \vee \langle B \rangle \Diamond \text{knows}_b(\text{sig}_a(c)))$$

avec $A \subseteq \Omega$ et $B \subseteq \Omega$. Quelques constatations :
- si une propriété φ est inévitable pour un ensemble A d'agents, alors elle est également inévitable pour tout ensemble d'agents A' inclus dans A ;
- si une propriété φ est inévitable pour un ensemble A d'agents, alors toute conséquence ψ de φ est également inévitable pour cet ensemble d'agents ;
- si un ensemble B d'agents a une stratégie pour assurer une propriété φ, alors tout ensemble d'agents B' contenant B a une stratégie pour assurer φ.

Ces constatations nous indiquent que la propriété d'équité considérée sera d'autant plus forte que A est grand et B petit. Ceci justifie donc le choix :

$$A \ \hat{=} \ \{a\} \cup \Omega_{\text{Com}} \cup \Omega_{\text{TTP}} \quad \text{et} \quad B \ \hat{=} \ \{b\}$$

Intuitivement, cet énoncé signifie que le protocole est équitable pour b, même si les canaux de communication et la TTP sont du coté de a.

Niveaux d'honnêteté et fiabilité. Un protocole $P \in \mathrm{Proc}_{\mathrm{Game}}$ dont la sémantique est $[\![P]\!]_{\mathrm{Game}} = (\Omega_P, \Pi_P, (\mathcal{S}_P^\ell)_\ell)$ satisfait un énoncé $\varphi \in \mathrm{Atl}(\Omega_P, \Pi_P)$ dans un niveau d'honnêteté et de fiabilité ℓ si on a $q_P^\ell \in [\![\varphi]\!]_{\mathcal{S}_P^\ell}$. Cependant, certains niveaux sont parfois plus pertinents que d'autres quand on considère un énoncé particulier. Pour la propriété d'équité par exemple, le fait qu'elle soit vraie dans un modèle $(\mathcal{S}_P^\ell, q_P^\ell)$ implique qu'elle sera vraie dans tout modèle ou le niveau d'honnêteté et de fiabilité est tel qu'il donne plus de pouvoir (que ℓ) aux agents de B et moins de pouvoir aux agents de A. Il est donc judicieux de vérifier la propriété d'équité pour b dans un modèle où b est honnête, a est fortement malhonnête et les canaux de communication sont non-fiables (rappelons qu'il n'existe qu'un seul niveau d'honnêteté pour la TTP). Cependant, comme on l'a déjà fait remarquer, il y a de fortes chances pour que les propriétés attendues ne soient pas vraies dans les modèles où tous les canaux de communication sont non-fiables. La pire hypothèse que l'on puisse faire sur les canaux consiste donc à supposer que les canaux avec la TTP sont résistants et les autres non-fiables.

D'autres énoncés semblables. Pour un ensemble d'agents $\Omega' \subseteq \Omega$, on peut parfois hésiter, au moment d'écrire une propriété, entre la forme $\langle\Omega'\rangle\,\varphi$ et $[\Omega \setminus \Omega']\,\varphi$. Il faut savoir que, si $\langle\Omega'\rangle\,\varphi \Rightarrow [\Omega \setminus \Omega']\,\varphi$, la réciproque n'est, en général, pas vraie (ce n'est pas parce qu'un ensemble d'agents ne peut éviter qu'un énoncé soit vrai que les autres agent ont une stratégie pour rendre cet énoncé vrai). Écrire la propriété d'équité sous la forme :

$$\mathit{fairness}'_b \;\;\hat{=}\;\; [A]\,\square(\neg\mathrm{knows}_a(\mathrm{sig}_b(c)) \vee [A]\,\Diamond\,\mathrm{knows}_b(\mathrm{sig}_a(c)))$$

ne permettrait donc pas de s'assurer que b a un moyen d'obtenir le contrat signé par a. Par contre, la forme suivante est plus forte que $\mathit{fairness}_b$:

$$\mathit{fairness}''_b \;\;\hat{=}\;\; \langle B\rangle\,\square(\neg\mathrm{knows}_a(\mathrm{sig}_b(c)) \vee \langle B\rangle\,\Diamond\,\mathrm{knows}_b(\mathrm{sig}_a(c)))$$

Timeliness pour b : le protocole respecte la *timeliness* pour b si b peut terminer le protocole, tout en préservant l'équité.

$$\mathit{timeliness}_b \;\;\hat{=}\;\; \langle B\rangle\,\Diamond(b.\mathrm{stopped} \wedge (\neg\mathrm{knows}_b(\mathrm{sig}_a(c)) \;\Rightarrow\; \neg\,\langle A\rangle\,\Diamond\,\mathrm{knows}_a(\mathrm{sig}_b(c))))$$

Version plus forte :

$$\mathit{timeliness}'_b \;\;\hat{=}\;\; \langle B\rangle\,\Diamond(b.\mathrm{stopped} \wedge (\neg\mathrm{knows}_b(\mathrm{sig}_a(c)) \;\Rightarrow\; \langle B\rangle\,\square\,\neg\mathrm{knows}_a(\mathrm{sig}_b(c))))$$

Ici, la version forte n'a pas vraiment de sens. En effet, b ayant terminé le protocole, il n'est pas censé participer à une stratégie.

Abuse-freeness pour b : cette propriété est formulée dans [GJM99] sous la forme suivante : le protocole respecte l'*abuse-freeness* pour b s'il est impossible à a, à n'importe quel point du protocole, de prouver à un observateur extérieur qu'il a la possibilité d'abandonner le protocole ou de le terminer avec succès. Afin de représenter cette propriété par un énoncé de la logique ATL, les auteurs de [KR02] remarquent tout d'abord que, en un point du protocole où a possède à la fois une stratégie pour abandonner le protocole (sans que b puisse obtenir sa signature) et une stratégie pour conclure (en obtenant la signature de b), alors il peut prouver à un observateur extérieur qu'il a le pouvoir d'abandonner ou de terminer avec succès le protocole, simplement en lui montrant ces deux stratégies, ainsi qu'une preuve du fait que le protocole entre a et b est bien engagé. On arrive alors à la formalisation suivante de l'*abuse-freeness* pour b :

$$\mathit{abuse\_free}_b \;\;\hat{=}\;\; \neg\,\langle A\rangle\,\Diamond(\mathit{proof}_{a,b} \wedge \langle A\rangle\,\Diamond\,\mathit{conclude}_a \wedge \langle A\rangle\,\Diamond\,\mathit{abort}_a)$$

Les notations utilisées, inspirées de [KR02], ont la signification suivante :

- $proof_{a,b}$ représente le fait que a est capable de prouver, à un observateur extérieur, que le protocole entre a et b est engagé (dans le cas de l'exemple, le seul message qui puisse convenir est $\mathrm{sig}_b(c)$) ;
- $conclude_a$ signifie que a a conclu le protocole avec succès, en obtenant la signature de b, on pose donc $conclude_a \;\hat{=}\; \mathrm{knows}_a(\mathrm{sig}_b(c))$;
- $abort_a$ signifie que a a abandonné le protocole, sans possibilité pour b d'obtenir la signature de a, on pose $abort_a \;\hat{=}\; \neg \langle B \rangle \Diamond \mathrm{knows}_b(\mathrm{sig}_a(c))$.

Nous avons montré, dans ce chapitre, comment donner une sémantique aux protocoles optimistes d'échange, sous la forme d'une famille (finie) d'ATS à un nombre fini d'agents et un nombre infini d'états. Nous avons également expliqué comment utiliser la logique ATL pour décrire des propriétés de sécurité de ces protocoles. Savoir si le protocole satisfait une propriété donnée (modulo quelques hypothèses sur les niveaux d'honnêteté des participants et de fiabilité des canaux de communication) consiste donc à savoir si l'énoncé ATL représentant la propriété est vrai dans l'ATS correspondant au protocole et au niveau d'honnêteté et de fiabilité considérés. Comme l'ATS en question possède un nombre infini d'états, on ne peut espérer répondre à cette question au moyen de techniques de *model-checking* classiques. Nous avons donc choisi de nous ramener à un ATS fini, au moyen de techniques d'abstraction. Le chapitre suivant décrit la théorie permettant de faire des abstractions sur les systèmes de transitions alternés. L'application aux protocoles optimistes d'échange sera donnée dans le chapitre 12.

Chapitre 11

Abstractions

Nous avons vu dans le chapitre précédent comment donner la sémantique d'une certaine classe de protocoles cryptographiques sous forme d'ATS et, également, comment la logique ATL pouvait être utilisée pour exprimer des propriétés de sécurité de ces protocoles. Le principal obstacle à l'automatisation de la vérification de ces propriétés est que cet ATS est infini. Cet obstacle n'apparaissait pas dans [KR02] car la phase d'écriture de l'ATS correspondant au protocole était manuelle et comportait un certain nombre de simplifications qui, si elles permettaient d'obtenir un système fini, n'étaient pas justifiées formellement. Nous proposons donc de garder le principe de ces simplifications, mais en les justifiant formellement, au moyen de techniques d'interprétation abstraite. La théorie de l'interprétation abstraite pour les ATS est décrite dans [HMMR00]. La contribution que nous apportons, dans ce chapitre, à cette théorie est double. D'une part, dans un travail joint avec S. Kremer et J.-F. Raskin, nous l'avons étendue au cas où les ATS possèdent des contraintes d'équité. D'autre part, nous avons prouvé la correction des abstractions de manière directe (à la différence de [HMMR00], où la correction était prouvé *via* les algorithmes de *model-checking*). Le principe, quand on veut abstraire un ATS, consiste à utiliser, non plus une, mais deux relations de transition. La première est censée donner, intuitivement, moins de pouvoir aux agents, alors que la seconde doit leur en accorder plus. Ainsi, si on désire vérifier un énoncé de la forme $\langle\Omega'\rangle\,\varphi$, on utilise la première relation de transition pour les agents de Ω' et la seconde pour les autres. En effet, si les agents de Ω' ont une stratégie pour assurer φ, alors que leurs pouvoirs ont été diminués, contre des agents dont les pouvoirs ont été augmentés, alors *a fortiori* les agents de Ω' ont une stratégie pour assurer φ. L'abstraction des contraintes d'équité s'avère plus compliquée, bien que le principe reste le même. Elle consiste à donner pour chaque contrainte d'équité, deux contraintes abstraites, l'une étant vraie plus souvent et l'autre moins souvent que la contrainte de départ. Dans la première section, nous allons formaliser ces constructions et expliquer comment elles permettent de manipuler, au niveau abstrait, des objets comme les stratégies et les ensembles d'exécutions associés. Dans la deuxième section, nous expliquerons comment remplacer les énoncés de la logique ATL sur le modèle concret par des énoncés sur le domaine abstrait tout en assurant un résultat de correction. Nous expliquerons enfin comment, en pratique, calculer l'abstraction d'un ATS décrit au moyen de la syntaxe présentée au chapitre 9. Dans les sections qui suivent, on supposera donné un ATS $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$ sur Ω et Π appelé ATS concret. On note également $\Pi^\bullet$ la réunion de deux copies disjointes de Π :

$$\Pi^\bullet \quad \hat{=} \quad \{P_\oplus \mid P \in \Pi\} \cup \{P_\ominus \mid P \in \Pi\}$$

11.1 Abstraction des ATS

Définition – 11.1.1

Un ATS abstrait, sur Ω et $\Pi^\bullet$, est un septuplet :

$$\mathcal{S}' \quad = \quad (Q', q'_0, \Delta'_\oplus, \Delta'_\ominus, \Gamma'_\oplus, \Gamma'_\ominus, \pi')$$

où :
- *Q' est un ensemble et $q'_0 \in Q'$;*
- *$\Delta'_\oplus$ et $\Delta'_\ominus$ sont des relations (sous-ensembles de $Q' \times \Omega \times 2^{Q'}$) ;*
- *$\Gamma'_\oplus$ et $\Gamma'_\ominus$ sont des ensembles de couples. Les éléments de $\Gamma'_\oplus$ seront notés sous la forme $\gamma'_\oplus = (\gamma'_{\oplus?}, \gamma'_{\oplus!})$ et ceux de $\Gamma'_\ominus$ sous la forme $\gamma'_\ominus = (\gamma'_{\ominus?}, \gamma'_{\ominus!})$ où $\gamma'_{\oplus?}$ et $\gamma'_{\ominus?}$ (respectivement $\gamma'_{\oplus!}$ et $\gamma'_{\ominus!}$) sont des relations sur $Q' \times \Omega$ (respectivement $Q' \times \Omega \times Q'$) ;*
- *π' est une fonction de Q' vers $2^{\Pi^\bullet}$.*

Le but d'un ATS abstrait est de remplacer l'ATS de départ dans le processus de vérification automatique des propriétés. Il est évident que, pour pouvoir remplir ce rôle, l'ATS abstrait doit vérifier certaines conditions en rapport avec l'ATS de départ. C'est l'objet de la définition suivante.

Définition – 11.1.2

Un ATS abstrait $\mathcal{S}' = (Q', q'_0, \Delta'_\oplus, \Delta'_\ominus, \Gamma'_\oplus, \Gamma'_\ominus, \pi')$, sur Ω et $\Pi^\bullet$, est dit correct relativement à $\mathcal{S}$ et une surjection $\alpha : Q \to Q'$ si les conditions suivantes sont satisfaites :
- *$q'_0 = \alpha(q_0)$;*
- *quels que soient $\omega \in \Omega$ et $q \in Q$, on a :*
  - *si $Q_\omega \in \Delta(q, \omega)$ alors il existe $Q'_\omega \in \Delta'_\oplus(\alpha(q), \omega)$ tel que $\alpha(Q_\omega) \subseteq Q'_\omega$,*
  - *si $Q'_\omega \in \Delta'_\ominus(\alpha(q), \omega)$ alors il existe $Q_\omega \in \Delta(q, \omega)$ tel que $\alpha(Q_\omega) \subseteq Q'_\omega$;*
- *il existe deux bijections :*

$$\begin{array}{rcl} \Gamma & \to & \Gamma'_\ominus \\ \gamma & \mapsto & \gamma'_\ominus \end{array} \quad \textit{et} \quad \begin{array}{rcl} \Gamma & \to & \Gamma'_\oplus \\ \gamma & \mapsto & \gamma'_\oplus \end{array}$$

  telles que quels que soient $\omega \in \Omega$, $\gamma \in \Gamma$ et $q_1, q_2 \in Q$, on ait :
  - *si $\gamma'_{\oplus?}(\alpha(q_1), \omega)$, alors $\gamma(q_1, \omega)$ est non vide,*
  - *s'il existe $Q_\omega \in \gamma(q_1, \omega)$ tel que $q_2 \in Q_\omega$, alors $\gamma'_{\oplus!}(\alpha(q_1), \omega, \alpha(q_2))$,*
  - *si $\gamma(q_1, \omega)$ est non vide, alors $\gamma'_{\ominus?}(\alpha(q_1), \omega)$,*
  - *si $\gamma'_{\ominus!}(\alpha(q_1), \omega, \alpha(q_2))$, alors il existe $Q_\omega \in \gamma(q_1, \omega)$ tel que $q_2 \in Q_\omega$;*
- *quels que soient $q \in Q$ et $p \in \Pi$, si $p_\ominus \in \pi'(\alpha(q))$, alors $p \in \pi(q)$ et si $p \in \pi(q)$, alors $p_\oplus \in \pi'(\alpha(q))$.*

On note dans ce cas $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}'$.

Remarque : Les conditions imposées à $\Gamma'_\ominus$ et $\Gamma'_\oplus$ méritent une explication. À une contrainte d'équité $\gamma \in \Gamma$ correspondent deux contraintes abstraites $\gamma'_\ominus \in \Gamma'_\ominus$ et $\gamma'_\oplus \in \Gamma'_\ominus$. La première est sensée laisser moins de pouvoir à un agent et la seconde plus. Intuitivement, pour que $\gamma'_\ominus$ laisse moins de pouvoir à un agent ω, elle doit être plus souvent activée que la contrainte concrète correspondante γ et moins souvent acceptée. C'est pourquoi la définition précédente impose que, si γ est activée en q_1 pour ω, alors $\gamma'_{\ominus?}(\alpha(q_1), \omega)$ est vraie et si $\gamma'_{\ominus!}(\alpha(q_1), \omega, \alpha(q_2))$

est vraie, alors γ est acceptée entre q_1 et q_2 pour ω. On raisonne de même pour justifier les choix concernant $\gamma'_\oplus$. $\square$

Nous allons maintenant définir, dans les ATS abstraits, des notions de stratégies et d'exécutions satisfaisant ces stratégies. Nous commençons par les stratégies qui se déclinent en deux versions, suivant que l'on accorde plus ou moins de pouvoir aux agents concernés.

Définition – 11.1.3

Soient $\mathcal{S}' = (Q', q'_0, \Delta'_\oplus, \Delta'_\ominus, \Gamma'_\oplus, \Gamma'_\ominus, \pi')$ un ATS abstrait et $\omega \in \Omega$. Une $\ominus$-stratégie (respectivement $\oplus$-stratégie) pour ω est une fonction f'_ω de Q'^+ vers $2^{Q'}$ telle que, quels que soient $q'_0, \ldots, q'_n \in Q'^+$ on ait $f'_\omega(q'_0, \ldots, q'_n) \in \Delta'_\ominus(q'_n, \omega)$ (respectivement $f'_\omega(q'_0, \ldots, q'_n) \in \Delta'_\oplus(q'_n, \omega)$). Soit $\Omega' \subseteq \Omega$, un ensemble de $\ominus$-stratégies (respectivement $\oplus$-stratégies) pour Ω' est une fonction f' qui, à tout $\omega \in \Omega'$, associe une $\ominus$-stratégie (respectivement $\oplus$-stratégie) pour ω, notée f'_ω.

Nous considérons maintenant les exécutions associées aux stratégies abstraites.

Définition – 11.1.4

Soient $\mathcal{S}' = (Q', q'_0, \Delta'_\oplus, \Delta'_\ominus, \Gamma'_\oplus, \Gamma'_\ominus, \pi')$ un ATS abstrait et f' un ensemble de $\ominus$-stratégies (respectivement $\oplus$-stratégies) pour un ensemble d'agents $\Omega' \subseteq \Omega$. Une suite $(q'_i)_{i\geq 0}$ d'éléments de Q' satisfait f' si, quel que soit $i \geq 0$, on a :

- *pour $\omega \in \Omega'$, $q'_{i+1} \in f'_\omega(q'_0, \ldots, q'_i)$;*
- *pour $\omega \notin \Omega'$, il existe $Q'_\omega \in \Delta'_\oplus(q'_i, \omega)$ (respectivement $Q'_\omega \in \Delta'_\ominus(q'_i, \omega)$) tel que $q'_{i+1} \in Q'_\omega$.*

On notera $\mathrm{Exec}_\ominus^{\mathcal{S}'}(f')$ (respectivement $\mathrm{Exec}_\oplus^{\mathcal{S}'}(f')$) l'ensemble des suites satisfaisant f'.

Le lemme suivant nous sera utile pour remplacer le calcul de la sémantique d'un énoncé de la logique ATL dans $\mathcal{S}$ par un calcul dans $\mathcal{S}'$ (qui sera décrit plus loin). Précisément, ce lemme explique comment obtenir des stratégies à partir de $\ominus$-stratégies et, réciproquement, comment à partir de stratégies obtenir des $\oplus$-stratégies.

Lemme – 11.1.1

Soient $\mathcal{S}' = (Q', q'_0, \Delta'_\oplus, \Delta'_\ominus, \Gamma'_\oplus, \Gamma'_\ominus, \pi')$ un ATS abstrait, correct relativement à $\mathcal{S}$ et une surjection α. Soit Ω' un ensemble d'agents, on a les résultats suivants :

1. *Pour tout ensemble f' de $\ominus$-stratégies pour Ω', il existe un ensemble f de stratégies pour Ω' tel que, si $(q_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$, alors $(\alpha(q_i))_{i\geq 0} \in \mathrm{Exec}_\ominus^{\mathcal{S}'}(f')$;*
2. *Pour tout ensemble f de stratégies pour Ω', il existe un ensemble f' de $\oplus$-stratégies pour Ω' tel que pour tout $(q'_i)_{i\geq 0} \in \mathrm{Exec}_\oplus^{\mathcal{S}'}(f')$ et tout $q \in Q$ tel que $\alpha(q) = q'_0$, il existe $(q_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$ tel que $(q'_i)_{i\geq 0} = (\alpha(q_i))_{i\geq 0}$ et $q_0 = q$.*

Démonstration :

Considérons tout d'abord le premier point. Pour chaque $\omega \in \Omega'$, on définit une fonction f_ω de Q^+ vers 2^Q de la manière suivante : soit $q_0, \ldots, q_n \in Q^+$, alors $\alpha(q_0), \ldots, \alpha(q_n) \in Q'^+$ et $f'_\omega(\alpha(q_0), \ldots, \alpha(q_n)) \in Q'^+ \in \Delta'_\ominus(\alpha(q_n), \omega)$. On sait alors qu'il existe $Q_\omega \in \Delta(q_n, \omega)$ tel que $\alpha(Q_\omega) \subseteq f'_\omega(\alpha(q_0), \ldots, \alpha(q_n)) \in Q'^+$. On pose alors $f_\omega(q_0, \ldots, q_n) \mathrel{\hat{=}} Q_\omega$. Cette fonction f_ω est clairement une stratégie pour ω et on obtient ainsi un ensemble f de stratégies pour Ω'. Soit maintenant $(q_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$, on veut montrer que $(\alpha(q_i))_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_\ominus(f')$. Pour cela, considérons $i \geq 0$ et $\omega \in \Omega'$. On sait que $q_{i+1} \in f_\omega(q_0, \ldots, q_i)$ et, par conséquent, $\alpha(q_{i+1}) \in \alpha(f_\omega(q_0, \ldots, q_i)) \subseteq f'_\omega(\alpha(q_0), \ldots, \alpha(q_n))$. Si maintenant $\omega \notin \Omega'$, on a $q_{i+1} \in Q_\omega$ avec $Q_\omega \in \Delta(q_i, \omega)$. Il existe alors $Q'_\omega \in \Delta'_\oplus(\alpha(q_i), \omega)$ tel que $\alpha(Q_\omega) \subseteq Q'_\omega$. Par conséquent, $\alpha(q_{i+1}) \in Q'_\omega$ avec $Q'_\omega \in \Delta'_\oplus(\alpha(q_i), \omega)$. On en déduit que $(\alpha(q_i))_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_\ominus(f')$.

Passons maintenant au second point. Nous allons construire progressivement l'ensemble de $\oplus$-stratégies f', ainsi qu'un élément arbitraire de $\mathrm{Exec}^{\mathcal{S}'}_\oplus(f')$. On part de $q'_0 \in Q'$ quelconque, ainsi qu'un de ses antécédents par α, $q_0 \in Q$. Supposons maintenant $n \geq 0$, $q'_0, \ldots, q'_{n-1}$ et $q_0, \ldots, q_{n-1}$ construits et f' définie sur les suites de longueur strictement inférieure à n. Pour $\omega \in \Omega'$, on pose $Q_\omega = f_\omega(q_0, \ldots, q_{n-1}) \in \Delta(q_{n-1}, \omega)$. On sait qu'il existe $Q'_\omega \in \Delta'_\oplus(q'_{n-1}, \omega)$ tel que $\alpha(Q_\omega) \subseteq Q'_\omega$ et on pose $f'_\omega(q'_0, \ldots, q'_{n-1}) = Q'_\omega$. Pour $q''_0, \ldots, q''_{n-1} \in Q'^n$ différent de $q'_0, \ldots, q'_{n-1}$, on pose $f'_\omega(q''_0, \ldots, q''_{n-1})$ égal à un élément quelconque de $\Delta'_\oplus(q''_{n-1}, \omega)$. Un prolongement quelconque de $q'_0, \ldots, q'_{n-1}$ respectant f' s'obtient alors en posant :

$$\{q'_n\} \;=\; \bigcap_{\omega\in\Omega} Q'_\omega$$

où les Q'_ω pour $\omega \in \Omega'$ sont ceux définis précédemment et pour chaque $\omega \notin \Omega'$, Q'_ω est un élément quelconque de $\Delta'_\ominus(q'_{n-1}, \omega)$. On sait alors qu'il existe $Q_\omega \in \Delta(q_{n-1}, \omega)$ tel que $\alpha(Q_\omega) \subseteq Q'_\omega$. Posons :

$$\{q_n\} \;=\; \bigcap_{\omega\in\Omega} Q_\omega$$

on a alors :

$$\alpha(\{q_n\}) \;=\; \alpha(\bigcap_{\omega\in\Omega} Q_\omega) \;\subseteq\; \bigcap_{\omega\in\Omega} \alpha(Q_\omega) \;\subseteq\; \bigcap_{\omega\in\Omega} Q'_\omega \;=\; \{q'_n\}$$

d'où l'on déduit que $\alpha(q_n) = q'_n$. Cette construction nous permet d'obtenir toutes les suites de $\mathrm{Exec}^{\mathcal{S}'}_\oplus(f')$, ainsi qu'un de leurs antécédents dans $\mathrm{Exec}^{\mathcal{S}}(f)$. ■

Nous donnons maintenant la contrepartie abstraite de la notion d'équité. Comme c'était le cas pour les stratégies, l'équité abstraite se décline en deux versions.

DÉFINITION – 11.1.5

Soient $\mathcal{S}' = (Q', q'_0, \Delta'_\oplus, \Delta'_\ominus, \Gamma'_\oplus, \Gamma'_\ominus, \pi')$ un ATS abstrait et $\omega \in \Omega$. Une suite $(q'_i)_{i\geq 0}$ d'éléments de Q' est dite $\ominus$-équitable (respectivement $\oplus$-équitable) pour $\omega \in \Omega$ et $\gamma'_\ominus \in \Gamma'_\ominus$ (respectivement $\gamma'_\oplus \in \Gamma'_\oplus$) si :

- *soit le nombre de positions $i \geq 0$ pour lesquelles $\gamma'_{\ominus ?}(q'_i)$ (respectivement $\gamma'_{\oplus ?}(q'_i)$) est fausse est infini ;*
- *soit le nombre de positions $i \geq 0$ pour lesquelles $\gamma'_{\ominus !}(q'_i, \omega, q'_{i+1})$ (respectivement $\gamma'_{\oplus !}(q'_i, \omega, q'_{i+1})$) est vraie est infini.*

On notera $\mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega')$ (respectivement $\mathrm{Fair}^{\mathcal{S}'}_{\oplus}(\Omega')$) l'ensemble des suites qui sont $\ominus$-équitables (respectivement $\oplus$-équitables) pour chaque agent $\omega \in \Omega' \subseteq \Omega$ et chaque $\gamma'_{\ominus} \in \Gamma'_{\ominus}$ (respectivement $\gamma'_{\oplus} \in \Gamma'_{\oplus}$).

Lemme – 11.1.2

Soient $\mathcal{S}' = (Q', q'_0, \Delta'_{\oplus}, \Delta'_{\ominus}, \Gamma'_{\oplus}, \Gamma'_{\ominus}, \pi')$ un ATS abstrait, correct relativement à $\mathcal{S}$ et une surjection α. Soient Ω' un ensemble d'agents et $(q_i)_{i\geq 0}$ une suite d'éléments de Q. On a les résultats suivants :

1. *Si $(\alpha(q_i))_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega')$, alors $(q_i)_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}}(\Omega)$;*
2. *Si $(q_i)_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}}(\Omega')$, alors $(\alpha(q_i))_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}'}_{\oplus}(\Omega)$.*

Démonstration :

Supposons tout d'abord que $(\alpha(q_i))_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega')$. Par définition, pour tout $\omega \in \Omega'$ et tout $\gamma \in \Gamma$:

- soit le nombre de positions $i \geq 0$ pour lesquelles $\gamma'_{\ominus ?}(\alpha(q_i))$ est fausse est infini ;
- soit le nombre de positions $i \geq 0$ pour lesquelles $\gamma'_{\ominus !}(\alpha(q_i), \omega, \alpha(q_{i+1}))$ est vraie est infini.

Si on est dans le premier cas, alors le nombre de positions i pour lesquelles $\gamma(q_i, \omega)$ est non vide est infini et, si on est dans le deuxième cas, alors le nombre de positions i pour lesquelles il existe $Q_\omega \in \gamma(q_i, \omega)$ tel que $q_{i+1} \in Q_\omega$ est infini. On en déduit donc que $(q_i)_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}}(\Omega')$.

Supposons maintenant que $(q_i)_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}}(\Omega')$. Par définition, pour tout $\omega \in \Omega'$ et tout $\gamma \in \Gamma$:

- soit le nombre de positions i pour lesquelles γ n'est pas activée (pour ω) est infini ;
- soit le nombre de positions i pour lesquelles γ est acceptée (pour ω) est infini.

Si on est dans le premier cas, alors le nombre de positions i telles que $\gamma'_{\oplus ?}(\alpha q_i)$ est infini et, si on est dans le second cas, alors le nombre de positions i pour lesquelles $\gamma'_{\oplus !}(\alpha(q_i), \omega, \alpha(q_{i+1}))$ est infini. On en déduit que $(\alpha(q_i))_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}'}_{\oplus}(\Omega)$. ■

11.2 Sémantique abstraite des énoncés ATL

Nous allons décrire dans cette partie comment calculer la sémantique d'un énoncé de la logique ATL dans un ATS abstrait. Comme on peut s'y attendre, cette définition s'obtient, à partir de la définition de la sémantique de ATL, en donnant moins de pouvoir aux agents lorsque l'on veut vérifier qu'ils peuvent faire quelque chose et plus de pouvoir quand on veut s'assurer que, pour ces mêmes agents, quelque chose est inévitable.

Définition – 11.2.1

Soit $\mathcal{S}' = (Q', q'_0, \Delta'_{\oplus}, \Delta'_{\ominus}, \Gamma'_{\oplus}, \Gamma'_{\ominus}, \pi')$ un ATS abstrait. La sémantique abstraite d'un énoncé $\varphi \in \mathrm{Atl}(\Omega, \Pi)$ dans $\mathcal{S}'$, notée $[\![\varphi]\!]'_{\mathcal{S}'}$, est définie par induction sur φ de la manière suivante :

- *si $p \in \Pi$, alors $q' \in [\![p]\!]'_{\mathcal{S}'}$ (respectivement $q' \in [\![\neg p]\!]'_{\mathcal{S}'}$) si $p_{\ominus} \in \pi'(q')$ (respectivement $p_{\oplus} \notin \pi'(q')$) ;*
- *si $\varphi, \psi \in \mathrm{Atl}(\Omega, \Pi)$, alors $q' \in [\![\varphi \vee \psi]\!]'_{\mathcal{S}'}$ (respectivement $q' \in [\![\varphi \wedge \psi]\!]'_{\mathcal{S}'}$) si $q' \in [\![\varphi]\!]'_{\mathcal{S}'} \cup [\![\psi]\!]'_{\mathcal{S}'}$ (respectivement $q' \in [\![\varphi]\!]'_{\mathcal{S}'} \cap [\![\psi]\!]'_{\mathcal{S}'}$) ;*

– *si* $\varphi \in \mathrm{Atl}(\Omega, \Pi)$ *et* $\Omega' \subseteq \Omega$, *alors* $q' \in [\![\langle\Omega'\rangle \circ \varphi]\!]'_{\mathcal{S}'}$ *s'il existe* f' *ensemble de* $\ominus$*-stratégies pour* Ω' *tel que si* $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f')$ *et* $q'_0 = q'$, *alors* $q'_1 \in [\![\varphi]\!]'_{\mathcal{S}'}$ *;*
– *de manière duale (et avec les mêmes notations),* $q' \in [\![[\Omega'] \circ \varphi]\!]'_{\mathcal{S}'}$ *si pour tout ensemble* f' *de* $\oplus$*-stratégies pour* Ω', *il existe* $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\oplus}(f')$ *telle que* $q'_0 = q'$ *et* $q'_1 \in [\![\varphi]\!]'_{\mathcal{S}'}$ *;*
– $q' \in [\![\langle\Omega'\rangle \square \varphi]\!]'_{\mathcal{S}'}$ *s'il existe* f' *ensemble de* $\ominus$*-stratégies pour* Ω' *tel que* $\mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f') \subseteq \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega')$ *et si* $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f')$ *et* $q'_0 = q'$, *alors, quel que soit* $i \geq 0$, $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$ *;*
– *de manière duale,* $q' \in [\![[\Omega'] \square \varphi]\!]'_{\mathcal{S}'}$ *si pour tout ensemble* f' *de* $\oplus$*-stratégies pour* Ω', *il existe* $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\oplus}(f') \cap \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega \setminus \Omega')$ *telle que* $q'_0 = q'$ *et, quel que soit* $i \geq 0$, $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$ *;*
– $q' \in [\![\langle\Omega'\rangle \lozenge \varphi]\!]'_{\mathcal{S}'}$ *s'il existe* f' *ensemble de* $\ominus$*-stratégies pour* Ω' *tel que si* $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f') \cap \mathrm{Fair}^{\mathcal{S}'}_{\oplus}(\Omega \setminus \Omega')$ *et si* $q'_0 = q'$, *alors il existe* $i \geq 0$ *tel que* $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$ *;*
– *de manière duale,* $q' \in [\![[\Omega'] \lozenge \varphi]\!]'_{\mathcal{S}'}$ *si pour tout ensemble* f' *de* $\oplus$*-stratégies pour* Ω' *tel que* $\mathrm{Exec}^{\mathcal{S}'}_{\oplus}(f') \subseteq \mathrm{Fair}^{\mathcal{S}'}_{\oplus}(\Omega')$, *il existe* $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\oplus}(f)$ *telle que* $q'_0 = q'$ *et, pour un certain* $i \geq 0$, $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$.

Comme d'habitude, on dit que $\mathcal{S}'$ satisfait un énoncé $\varphi \in \mathrm{Atl}(\Omega, \Pi)$, et on note $\mathcal{S}' \models \varphi$, si $q'_0 \in [\![\varphi]\!]'_{\mathcal{S}'}$. Le lemme suivant explique que la sémantique abstraite d'un énoncé constitue (modulo la surjection α) une sous-approximation correcte de la sémantique (concrète).

Lemme – 11.2.1

Soient $\mathcal{S}'$ *un ATS abstrait, correct relativement à* $\mathcal{S}$ *et une surjection* α *et* φ *un énoncé ATL. Si* $q \in Q$ *est tel que* $\alpha(q) \in [\![\varphi]\!]'_{\mathcal{S}'}$, *alors* $q \in [\![\varphi]\!]_{\mathcal{S}}$.

Démonstration :

On procède par induction sur φ. Dans le cas des propositions atomiques, négation et opérations booléennes, la preuve est triviale. Contentons-nous donc de traiter deux cas typiques d'énoncés contenant une modalité :

– si $\alpha(q) \in [\![\langle\Omega'\rangle \square \varphi]\!]'_{\mathcal{S}'}$, il existe f' ensemble de $\ominus$-stratégies pour Ω' tel que $\mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f') \subseteq \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega')$ et si $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f')$ et $q'_0 = \alpha(q)$, alors, quel que soit $i \geq 0$, $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$. D'après le lemme 11.1.1, il existe alors f ensemble de stratégies pour Ω' tel que si $(q_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$, alors $(\alpha(q_i))_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f')$. Comme $\mathrm{Exec}^{\mathcal{S}'}_{\ominus}(f') \subseteq \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega')$, on en déduit (en appliquant le lemme 11.1.2) que $\mathrm{Exec}^{\mathcal{S}}(f) \subseteq \mathrm{Fair}^{\mathcal{S}}(\Omega')$. Si on suppose de plus que $q_0 = q$, on aura alors $\alpha(q_0) = \alpha(q)$ et par conséquent, quel que soit $i \geq 0$, $\alpha(q_i) \in [\![\varphi]\!]'_{\mathcal{S}'}$. Par hypothèse d'induction, il vient $q_i \in [\![\varphi]\!]_{\mathcal{S}}$, quel que soit $i \geq 0$;
– de manière duale, si $\alpha(q) \in [\![[\Omega'] \square \varphi]\!]'_{\mathcal{S}'}$, alors pour tout ensemble f' de $\oplus$-stratégies pour Ω', il existe $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\oplus}(f') \cap \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega \setminus \Omega')$ telle que $q'_0 = \alpha(q)$ et, quel que soit $i \geq 0$, $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$. Soit donc f un ensemble de stratégies pour Ω' est soit f' l'ensemble de $\oplus$-stratégies associé par le lemme 11.1.1. D'après ce même lemme, on sait que, si $(q'_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}'}_{\oplus}(f')$ est telle que $q'_0 = \alpha(q)$, il existe $(q_i)_{i\geq 0} \in \mathrm{Exec}^{\mathcal{S}}(f)$ telle que $(q'_i)_{i\geq 0} = (\alpha(q_i))_{i\geq 0}$ et $q_0 = q$. Par hypothèse d'induction, si quel que soit $i \geq 0$ on a $q'_i \in [\![\varphi]\!]'_{\mathcal{S}'}$, alors quel que soit $i \geq 0$, on a $q_i \in [\![\varphi]\!]_{\mathcal{S}}$. Si de plus $(q'_i)_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}'}_{\ominus}(\Omega \setminus \Omega')$, alors par application du lemme 11.1.2 on aura $(q_i)_{i\geq 0} \in \mathrm{Fair}^{\mathcal{S}}(\Omega \setminus \Omega')$.

■

Nous terminons cette partie par le théorème de correction (qui est une conséquence immédiate du lemme précédent).

Théorème – 11.2.1

Soient $\mathcal{S}'$ un ATS abstrait et $\varphi \in \mathrm{Atl}(\Omega, \Pi)$. Si $\mathcal{S} \sqsubseteq_\alpha \mathcal{S}'$ et $\mathcal{S}' \models \varphi$, alors $\mathcal{S} \models \varphi$.

Nous avons donc, à ce point, décrit des modèles constituant des abstractions correctes des ATS. Nous expliquons, dans la section suivante, comment obtenir la description d'un ATS abstrait, correct par rapport à un ATS donné, à partir de la description de ce dernier.

11.3 Guide pour les abstractions

Les sections précédentes ont permis de définir des abstractions pour les ATS et la logique ATL. Néanmoins, il pourrait s'avérer fastidieux de partir du système décrivant un protocole cryptographique, de définir le système abstrait associé puis de montrer que ce système est correct par rapport à l'ATS de départ. Comme expliqué dans le chapitre 2, nous proposons ici un guide permettant de définir de manière quasiment syntaxique un ATS abstrait associé à un ATS décrit au moyen de la syntaxe présentée au chapitre 9.

La description de l'ATS de départ doit commencer par la déclaration des agents, de la forme :

$$\textbf{agents} \quad : \quad \omega_1, \ldots, \omega_n$$

On reprend cette déclaration telle quelle dans l'ATS abstrait (puisque l'ensemble des agents de l'ATS abstrait doit être le même que pour l'ATS de départ). Vient ensuite, pour chaque agent $\omega_i \in \Omega$, la déclaration des variables locales de cet agent :

$$\textbf{vars}(\omega_i) \quad : \quad \omega_i.x_1, \ldots, \omega_i.x_k$$

Dans le système abstrait, l'agent ω_i possédera également des variables locales :

$$\textbf{vars}(\omega_i) \quad : \quad \omega_i.y_1, \ldots, \omega_i.y_{k'}$$

L'ensemble de ces variables sera noté Y_{ω_i}. Le choix de cet ensemble de variables dépendra de l'abstraction, mais il faudra définir, à part, une surjection $\alpha_i : \mathbb{B}^{X_{\omega_i}} \to \mathbb{B}^{Y_{\omega_i}}$. La donnée d'une telle surjection, pour chaque agent $\omega_i \in \Omega$, induira une surjection $\alpha : \mathbb{B}^X \to \mathbb{B}^Y$ (où Y est la réunion disjointe des ensembles de variables locales des agents dans le système abstrait). Il faudra également définir à part une fonction $\pi' : \mathbb{B}^Y \to 2^{\Pi^\bullet}$ telle que, quels que soient $p \in \Pi$ et $q \in \mathbb{B}^X$, on ait :

- si $p_\ominus \in \pi'(\alpha(q))$, alors $p \in \pi(q)$;
- si $p \in \pi(q)$, alors $p_\oplus \in \pi'(\alpha(q))$.

Une manière simple de définir syntaxiquement π' consiste à associer à chaque $p \in \Pi$ deux énoncés booléens sur Y, noté $(p)_\ominus$ et $(p)_\oplus$ et tels que, quel que soit $q \in \mathbb{B}^X$ on ait :

- si $\alpha(q)$ satisfait $(p)_\ominus$, alors $p \in \pi(q)$;
- si $p \in \pi(q)$, alors $\alpha(q)$ satisfait $(p)_\oplus$.

(remarquer le lien, au moins intuitif, si ce n'est formel, avec les abstractions présentées dans la première partie de cette thèse). Considérons maintenant une commande gardée $\Vert\, g \to u$ appartenant à l'agent ω_i. On va décliner cette commande en deux versions dans l'ATS abstrait, notées $\ominus\Vert\, g_\ominus \to u'$ et $\oplus\Vert\, g_\oplus \to u'$, dans lesquelles $g_\ominus$ et $g_\oplus$ seront des énoncés booléens sur Y et u' un énoncé booléen sur $Y \cup Y'_{\omega_i}$. Il faudra s'assurer que, quel que soient $q \in \mathbb{B}^X$ et $q_{\omega_i} \in \mathbb{B}^{X_{\omega_i}}$ on a :

– si $\alpha(q)$ satisfait $g_\ominus$, alors q satisfait g ;
– si q satisfait g, alors $\alpha(q)$ satisfait $g_\oplus$;
– si la valuation sur $X \cup X'_{\omega_i}$ induite par q et q_{ω_i} satisfait u, alors la valuation sur $Y \cup Y'_{\omega_i}$ induite par $\alpha(q)$ et $\alpha_i(q_{\omega_i})$ satisfait u'.

En ce qui concerne $g_\ominus$ (respectivement $g_\oplus$), il suffit de remplacer chaque occurrence positive de chaque $p \in \Pi$ par l'énoncé $p_\ominus$ (respectivement $p_\oplus$) et chaque occurrence négative par $p_\oplus$ (respectivement $p_\ominus$). De même, si la déclaration :

$$\mathbf{init}(\omega_i) \quad : \quad u$$

est utilisée dans l'ATS de départ pour définir l'état local initial de ω_i, on écrira dans la définition de l'ATS abstrait :

$$\mathbf{init}(\omega_i) \quad : \quad u'$$

où u' est un énoncé sur Y'_{ω_i} tel que, quel que soit $q_{\omega_i} \in \mathbb{B}^{X_{\omega_i}}$, si la valuation sur X'_{ω_i} induite par q_{ω_i} satisfait u, alors la valuation sur Y'_{ω_i} induite par $\alpha_i(q_{\omega_i})$ satisfait u'. Il reste maintenant à traiter le cas des contraintes d'équité. Considérons donc une telle contrainte $\gamma \mathrel{\hat{=}} \gamma^{g,u}_{\omega i}$, induite par la présence d'une commande $\|_{\mathrm{f}}\ g \rightarrow u$. On définit alors $\gamma'_\ominus$ et $\gamma'_\oplus$ en convenant que, quels que soient $q'_1, q'_2 \in \mathbb{B}^Y$:
– $\gamma'_{\oplus ?}(q'_1, \omega_i)$ est vraie si q'_1 satisfait $g_\ominus$;
– $\gamma'_{\oplus !}(q'_1, \omega_i, q'_2)$ est vraie si la valuation sur $Y \cup Y'_{\omega_i}$ induite par q'_1 et q'_2 satisfait $u_\oplus$;
– $\gamma'_{\ominus ?}(q'_1, \omega_i)$ est vraie si q'_1 satisfait $g_\oplus$;
– $\gamma'_{\ominus !}(q'_1, \omega_i, q'_2)$ est vraie si la valuation sur $Y \cup Y'_{\omega_i}$ induite par q'_1 et q'_2 satisfait $u_\ominus$.

$u_\ominus$ et $u_\oplus$ s'obtiennent à partir de u de la manière habituelle. En pratique, $\gamma'_\ominus$ et $\gamma'_\oplus$ seront définies au moyen d'énoncés booléens sur Y (et éventuellement Y'), notés $\gamma'_{\ominus ?}(\omega_i)$, $\gamma'_{\ominus !}(\omega_i)$, $\gamma'_{\oplus ?}(\omega_i)$ et $\gamma'_{\oplus !}(\omega_i)$. Ces définitions des contraintes d'équité abstraites devront être définies à part, mais, pour rappeler qu'une contrainte est associée aux commandes abstraites associées à $\|\ g \rightarrow u$, on écrira dans le système abstrait :

$$\begin{array}{l} \ominus\|_{\mathrm{f}}\ g_\ominus \rightarrow u' \\ \oplus\|_{\mathrm{f}}\ g_\oplus \rightarrow u' \end{array}$$

Exemple : Considérons l'exemple d'ATS qui avait été utilisé tout au long du chapitre 9 et, plus particulièrement, la description qui en avait été donnée page 106. Nous décidons de réaliser une abstraction de cet ATS en supprimant la variable $a.y$. L'ensemble des variables de l'ATS abstrait est alors $Y \mathrel{\hat{=}} \{a.x, b.z\}$. Rappelons que l'ensemble des variables de l'ATS concret est $X \mathrel{\hat{=}} \{a.x, a.y, b.z\}$. On a donc une surjection :

$$\begin{array}{rccc} \alpha: & \mathbb{B}^X & \rightarrow & \mathbb{B}^Y \\ & q & \mapsto & q|_Y \end{array}$$

$q|_Y$ désignant la restriction de q (qui est une fonction de X vers $\mathbb{B}$) à Y. On pose ensuite :

$$\begin{array}{rcccl} (a.x)_\oplus & = & (a.x)_\ominus & \hat{=} & a.x \\ (b.z)_\oplus & = & (b.z)_\ominus & \hat{=} & b.z \end{array}$$

puisque ces variables ne sont pas abstraites, puis :

$$\begin{array}{rcl} (a.y)_\ominus & \hat{=} & \text{false} \\ (a.y)_\oplus & \hat{=} & \text{true} \end{array}$$

Appliquant le guide d'abstraction qui précède, on obtient la description suivante pour l'ATS abstrait :

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{ats}\ \textsc{Exemple'} =} \\
\quad \textbf{agents} & : & a, b \\
 & & \\
\quad \textbf{vars}(a) & : & a.x \\
\quad \textbf{vars}(b) & : & b.z \\
 & & \\
\quad \textbf{init}(a) & : & a.x := \text{true} \\
\quad \textbf{init}(b) & : & b.z := \text{true} \\
 & & \\
\quad \textbf{commands}(a) & : & \ominus\| \ \text{true} \rightarrow a.x := \text{true} \\
 & & \oplus\| \ \text{true} \rightarrow a.x := \text{true} \\
 & & \ominus\|_{\text{f}} \ \neg(a.y)_{\oplus} \rightarrow a.x := \text{false} \\
 & & \oplus\|_{\text{f}} \ \neg(a.y)_{\ominus} \rightarrow a.x := \text{false} \\
 & & \\
\quad \textbf{commands}(b) & : & \ominus\| \ a.x \rightarrow b.z := \text{false} \\
 & & \oplus\| \ a.x \rightarrow b.z := \text{false} \\
 & & \ominus\| \ \neg a.x \rightarrow b.z := \text{true} \\
 & & \oplus\| \ \neg a.x \rightarrow b.z := \text{true} \\
\textbf{end} & &
\end{array}
$$

Les commandes ne faisaient, pour la plupart, pas référence à $a.y$. Elles ne sont pas réellement abstraites, à l'exception des deux commandes :

$$\ominus\|_{\text{f}} \ \neg(a.y)_{\oplus} \rightarrow a.x := \text{false}$$

$$\oplus\|_{\text{f}} \ \neg(a.y)_{\ominus} \rightarrow a.x := \text{false}$$

qui peuvent, avec les définitions qui ont été données de $(a.y)_{\oplus}$ et $(a.y)_{\ominus}$, se réécrire :

$$\ominus\|_{\text{f}} \ \text{false} \rightarrow a.x := \text{false}$$

$$\oplus\|_{\text{f}} \ \text{true} \rightarrow a.x := \text{false}$$

Le joueur a, dans sa version la plus faible, ne peut donc jamais mettre $a.x$ à false, alors qu'il le peut dans sa version la plus forte. Considérons la commande du système concret qui a donné les deux commandes qui précèdent :

$$\|_{\text{f}} \ \neg a.y \rightarrow a.x := \text{false}$$

Cette commande était associé à une contrainte d'équité. Les versions abstraites de ces contraintes sont :

$$
\begin{array}{rcl}
\gamma'_{\oplus ?}(q'_1, a) & \hat{=} & \text{false} \\
\gamma'_{\oplus !}(q'_1, a, q'_2) & \hat{=} & \neg a.y' \\
\gamma'_{\ominus ?}(q'_1, a) & \hat{=} & \text{true} \\
\gamma'_{\ominus !}(q'_1, a, q'_2) & \hat{=} & \neg a.y'
\end{array}
$$

Remarquons qu'il sera impossible au joueur a, dans sa version la plus faible, de satisfaire cette contrainte d'équité. $\square$

Arrêtons-nous là en ce qui concerne la théorie des abstractions pour les ATS et la logique temporelle alternée. Nous allons, dans le chapitre qui suit, l'appliquer aux ATS qui donnent la sémantique des protocoles cryptographiques. Pour ce faire, nous utiliserons le guide d'abstraction et nous appliquerons les principes qu'il contient à la description des ATS modélisant les protocoles cryptographiques, telle qu'elle a été donnée au chapitre 10.

Chapitre 12

Sémantique abstraite des protocoles

Nous allons ici appliquer les techniques du chapitre précédent à la sémantique des protocoles cryptographiques telle qu'elle a été définie dans le chapitre 10. Ceci nous fournira une abstraction correcte de la modélisation de ces protocoles cryptographiques qui, de plus, sera finie. Nous pourrons alors lui appliquer des techniques de *model-checking*, mais nous réservons ceci au chapitre suivant. Pour l'instant, nous reprenons les notations du chapitre 10 : il s'agissait d'un protocole $P \hat{=} T \parallel R_1 \parallel \ldots \parallel R_n \in \mathrm{Proc}_{\mathrm{Game}}$ dont on avait extrait, après plusieurs traitements, des ensembles de points $(PT, PR_1, \ldots, PR_n)$ et des connaissances initiales $(IK_1, \ldots, IK_n)$. La sémantique d'un tel protocole était un triplet :

$$[\![P]\!]_{\mathrm{Game}} \quad = \quad (\Omega_P, \Pi_P, (\mathcal{S}_P^\ell)_\ell)$$

où, pour chaque niveau d'honnêteté et de fiabilité ℓ, $\mathcal{S}_P^\ell$ est un ATS sur Ω_P et Π_P. Nous allons suivre pas à pas la construction de cet ATS (que l'on notera $\mathcal{S}$ pour simplifier) pour définir un ATS abstrait $\mathcal{S}'$ qui sera correct relativement à $\mathcal{S}$ et une surjection α. Nous construirons cet ATS abstrait en fonction d'un sous-ensemble fini de Msg, noté M, arbitraire mais fixé. Considérant un élément $\perp_{\mathrm{Msg}} \notin M$, on peut déjà définir une surjection α sur les messages :

$$\begin{array}{rlcl} \alpha : & \mathrm{Msg} & \rightarrow & \mathrm{Msg}' \\ & m \in M & \mapsto & m \\ & m \notin M & \mapsto & \perp_{\mathrm{Msg}} \end{array}$$

où on a posé $\mathrm{Msg}' \hat{=} M \cup \{\perp_{\mathrm{Msg}}\}$. Le plan de cette partie suit la construction de l'ATS associé à P et ℓ telle qu'elle a été présentée au chapitre 10. Nous commencerons donc par décrire les agents du système abstrait, ainsi que la composition de leur état local. Nous expliquerons ensuite comment donner des versions abstraites des commandes gardées qui définissent le comportement de chaque agent du système. Cette étape peut se faire de manière quasiment syntaxique. Nous insisterons néanmoins sur deux cas particuliers : le cas où des contraintes d'équité sont présentes et celui où (du fait de l'abstraction) des bloquages peuvent apparaître dans l'ATS abstrait.

12.1 Les agents, leur état local

Comme dans le chapitre 10, on désigne par Ω_{Part} l'ensemble des participants au protocole (à l'exception de la TTP), par Ω_{Com} l'ensemble des canaux de communication et Ω_{TTP} désigne l'ensemble d'agents réduit à la TTP . L'ensemble des agents de l'ATS donnant la sémantique de P était $\Omega \hat{=} \Omega_{\text{Part}} \cup \Omega_{\text{TTP}} \cup \Omega_{\text{Com}}$ et il en est de même pour l'ATS abstrait. La description de cet ATS abstrait commence donc par la déclaration :

$$\textbf{agents} \quad : \quad i \in \Omega_{\text{Part}}, 0 \in \Omega_{\text{TTP}}, c_{i\to j} \in \Omega_{\text{Com}}$$

Les variables des agents de l'ATS concret pouvaient prendre quatre formes différentes :

$$i.\text{stopped}, i.\text{send}(j, m), i.\text{recv}(m) \text{ et } c_{i\to j}.\text{fwd}(m)$$

Nous définissons une fonction d'abstraction sur ces variables, notée également α, en posant :

$$\begin{array}{rcl}
\alpha(i.\text{stopped}) & \hat{=} & i.\text{stopped} \\
\alpha(i.\text{send}(j, m)) & \hat{=} & \left[\begin{array}{ll} i.\text{send}(j, m) & \text{si } m \in M \\ i.\text{send}(j, \bot_{\text{Msg}}) & \text{sinon} \end{array} \right. \\
\alpha(i.\text{recv}(m)) & \hat{=} & \left[\begin{array}{ll} i.\text{recv}(m) & \text{si } m \in M \\ i.\text{recv}(\bot_{\text{Msg}}) & \text{sinon} \end{array} \right. \\
\alpha(c_{i\to j}.\text{fwd}(m)) & \hat{=} & \left[\begin{array}{ll} c_{i\to j}.\text{fwd}(m) & \text{si } m \in M \\ c_{i\to j}.\text{fwd}(\bot_{\text{Msg}}) & \text{sinon} \end{array} \right.
\end{array}$$

Autrement dit, l'abstraction d'une variable s'obtient en remplaçant le message m qui s'y trouve par $\alpha(m)$. Notons X l'ensemble des variables de l'ATS de départ et Y l'ensemble de leurs abstractions (*via* α). On obtient alors une surjection α de X sur Y et nous utiliserons les éléments de Y comme variables de l'ATS abstrait. On pose donc, tout d'abord pour $i \in \Omega_{\text{Part}}$ (et pour chaque $m \in M$ et $j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$, $j \neq i$) :

$$\begin{array}{lll}
\textbf{vars}(i) & : & i.\text{stopped}, i.\text{recv}(m), i.\text{recv}(\bot_{\text{Msg}}), i.\text{send}(j, m), i.\text{send}(j, \bot_{\text{Msg}}) \\
\textbf{init}(i) & : & i.\text{stopped} := \text{false}; \\
 & & i.\text{recv}(m) := \text{false}; i.\text{recv}(\bot_{\text{Msg}}) := \text{false}; \\
 & & i.\text{send}(j, m) := \text{false}; i.\text{send}(j, \bot_{\text{Msg}}) := \text{false}
\end{array}$$

Pour l'agent $0 \in \Omega_{\text{TTP}}$ désignant la TTP on pose (pour $m \in M$ et $j \in \Omega_{\text{Part}}$) :

$$\begin{array}{lll}
\textbf{vars}(0) & : & 0.\text{send}(j, m), 0.\text{send}(j, \bot_{\text{Msg}}) \\
\textbf{init}(0) & : & 0.\text{send}(j, m) := \text{false}; 0.\text{send}(j, \bot_{\text{Msg}}) := \text{false}
\end{array}$$

Pour un canal de communication $c_{i\to j} \in \Omega_{\text{Com}}$ (et pour $m \in M$) :

$$\begin{array}{lll}
\textbf{vars}(c_{i\to j}) & : & c_{i\to j}.\text{fwd}(m), c_{i\to j}.\text{fwd}(\bot_{\text{Msg}}) \\
\textbf{init}(c_{i\to j}) & : & c_{i\to j}.\text{fwd}(m) := \text{false}; c_{i\to j}.\text{fwd}(\bot_{\text{Msg}}) := \text{false}
\end{array}$$

Remarquons que, pour une variable $x \in X$, on a $\alpha(x) = x$ si, et seulement si, $x \in X \cap Y$ et $\alpha(x) \in Y \setminus X$ si, et seulement si, $x \in X \setminus Y$. Il nous faut maintenant définir une surjection α de l'ensemble des états de l'ATS de départ (les valuations sur X, *i.e.* $\mathbb{B}^X$) dans l'ensemble des états de l'ATS abstrait (les valuations sur Y, *i.e.* $\mathbb{B}^Y$). Pour ce faire, étant donnée une valuation $q : X \to \mathbb{B}$ et une variable $y \in Y$, nous poserons que y est vraie dans $\alpha(q)$ si, et seulement si, il existe une variable $x \in X$ telle que $\alpha(x) = y$ et x est vraie dans q. Formellement, ceci s'écrit :

$$\begin{array}{rccc}
\alpha : & \mathbb{B}^X & \to & \mathbb{B}^Y \\
 & q & \mapsto & \left(\begin{array}{ccc} Y & \to & \mathbb{B} \\ y & \mapsto & \bigvee_{x \in \alpha^{-1}(y)} q(x) \end{array} \right)
\end{array}$$

Nous disposons maintenant de deux ensembles d'états $\mathbb{B}^X$ et $\mathbb{B}^Y$, reliés au moyen d'une surjection α. L'ATS de départ est muni d'une fonction π qui à un état $q \in \mathbb{B}^X$ associe le sous-ensemble de X constitué des variables qui sont vraies en q. On peut naturellement faire de même au niveau abstrait, à savoir définir une fonction π qui à un état abstrait $q' \in \mathbb{B}^Y$ associe le sous-ensemble de Y constitué des variables qui sont vraies en q'. Comme il est précisé dans la définition des ATS abstraits, il nous faut définir une fonction $\pi' : \mathbb{B}^Y \to 2^{X^\bullet}$ satisfaisant les conditions suivantes :

- si $x_\ominus \in \pi'(\alpha(q))$, alors $x \in \pi(q)$;
- si $x \in \pi(q)$, alors $x_\oplus \in \pi'(\alpha(q))$.

Prenons un exemple pour fixer les idées.

Exemple : Considérons la variable $i.\mathrm{recv}(m)$ pour un certain $i \in \Omega_{\mathrm{Part}}$ et $m \in \mathrm{Msg}$. Dans le cas où $m \in M$, on a $\alpha(i.\mathrm{recv}(m)) = i.\mathrm{recv}(m)$, donc $\alpha^{-1}(i.\mathrm{recv}(m)) = \{i.\mathrm{recv}(m)\}$ et, par conséquent :

$$\forall q \in \alpha^{-1}(q').\ i.\mathrm{recv}(m) \in \pi(q) \quad \Leftrightarrow \quad \exists q \in \alpha^{-1}(q').\ i.\mathrm{recv}(m) \in \pi(q)$$

et on conviendra donc, dans ce cas, que $i.\mathrm{recv}(m)_\ominus, i.\mathrm{recv}(m)_\oplus \in \pi'(q')$ si, et seulement si, $i.\mathrm{recv}(m) \in \pi(q')$. Dans le cas contraire (*i.e.* $m \notin M$), et par construction de $\alpha : \mathbb{B}^X \to \mathbb{B}^Y$, on a $\alpha(i.\mathrm{recv}(m)) = i.\mathrm{recv}(\bot_{\mathrm{Msg}})$ et :

$$i.\mathrm{recv}(\bot_{\mathrm{Msg}}) \in \pi(q') \quad \Leftrightarrow \quad \exists q \in \alpha^{-1}(q').\ i.\mathrm{recv}(m) \in \pi(q)$$

On conviendra donc que $\pi'(q')$ ne contient jamais $i.\mathrm{recv}(m)_\ominus$ et qu'il contient $i.\mathrm{recv}(m)_\oplus$ si, et seulement si, $i.\mathrm{recv}(\bot_{\mathrm{Msg}}) \in \pi(q')$. □

Formellement, pour $x \in X$ et $q \in Q = \mathbb{B}^X$:

- si $x \in X \cap Y$ (*i.e.* $\alpha(x) = x$), alors $x_\ominus, x_\oplus \in \pi'(x)$ si, et seulement si, $\alpha(x)$ est vrai en q' ;
- si $x \in X \setminus Y$ (*i.e.* $\alpha(x) \in Y \setminus X$), alors $x_\ominus \notin \pi'(x)$, d'une part, et $x_\oplus \in \pi'(x)$ si, et seulement si, $\alpha(x)$ est vrai en q'.

Comme expliqué dans le chapitre précédent, il est possible de formuler la définition qui précède en associant à chaque variable $x \in X$ deux énoncés propositionnels sur Y, notés $(x)_\ominus$ et $(x)_\oplus$, et définis par :

- si $x \in X \cap Y$, alors $(x)_\ominus = (x)_\oplus \mathrel{\hat{=}} \alpha(x)$;
- si $x \in X \setminus Y$, alors $(x)_\ominus \mathrel{\hat{=}} \mathrm{false}$ et $(x)_\oplus \mathrel{\hat{=}} \alpha(x)$.

Ce qui importe, c'est que l'on dispose d'une surjection α entre les états de l'ATS concrets et ceux de l'ATS abstrait telle que, quels que soient $q \in \mathbb{B}^X$ et $x \in X$, on ait :

- si $x_\ominus \in \pi'(\alpha(q))$, alors $x \in \pi(q)$;
- si $x \in \pi(q)$, alors $x_\oplus \in \pi'(\alpha(q))$,

autrement dit, en utilisant les énoncés propositionnels $(x)_\ominus$ et $(x)_\oplus$:

- si $(x)_\ominus$ est vrai en $\alpha(q)$, alors x est vraie en q ;
- si x est vraie en q, alors $(x)_\oplus$ est vraie en $\alpha(q)$.

12.2 Relations de transition locales

Nous avions défini, dans la section 10.2, une proposition $\text{knows}_i(m)$ (pour chaque agent $i \in \Omega_{\text{Part}}$ et chaque message $m \in \text{Msg}$) en posant :

$$\text{knows}_i(m) \quad \hat{=} \quad \bigvee_{\substack{s \subseteq \text{Msg} \\ s \Vdash m}} \bigwedge_{m_1 \in s} (i.\text{recv}(m_1) \vee m_1 \in IK_i)$$

Nous allons associer à cette proposition deux contreparties abstraites, notées respectivement $\text{knows}_{\ominus i}(m)$ et $\text{knows}_{\oplus i}(m)$ (pour $m \in \text{Msg}'$). Commençons par le cas où $m \in M$. Il suffit alors de placer des $\ominus$ et $\oplus$ sur les variables propositionnelles qui interviennent dans l'énoncé définissant $\text{knows}_i(m)$, comme on a l'habitude de le faire depuis la première partie de cette thèse. On trouve, pour $i \in \Omega_{\text{Part}}$ et $m \in M$:

$$\begin{aligned}
\text{knows}_{\ominus i}(m) &\quad \hat{=} \quad \bigvee_{\substack{s \subseteq \text{Msg} \\ s \Vdash m}} \bigwedge_{m_1 \in s} ((i.\text{recv}(m_1))_\ominus \vee m_1 \in IK_i) \\
\text{knows}_{\oplus i}(m) &\quad \hat{=} \quad \bigvee_{\substack{s \subseteq \text{Msg} \\ s \Vdash m}} \bigwedge_{m_1 \in s} ((i.\text{recv}(m_1))_\oplus \vee m_1 \in IK_i)
\end{aligned}$$

Posons également :

$$\begin{aligned}
\text{knows}_{\ominus i}(\bot_{\text{Msg}}) &\quad \hat{=} \quad \text{false} \\
\text{knows}_{\oplus i}(\bot_{\text{Msg}}) &\quad \hat{=} \quad \text{true}
\end{aligned}$$

Avec de telles définitions, il est aisé de constater que, si $\text{knows}_{\ominus i}(\alpha(m))$ est vraie dans un état abstrait $\alpha(q)$, alors $\text{knows}_i(m)$ est vraie en q et, si $\text{knows}_i(m)$ est vraie en q, alors $\text{knows}_{\oplus i}(\alpha(m))$ est vraie en $\alpha(q)$. Nous donnerons, dans le chapitre suivant, des formulations plus explicites et effectives pour ces propositions mais, pour l'instant, nous pourrons nous contenter des formulations qui précèdent.

Agent honnête $i \in \Omega_{\text{Part}}$. Les commandes gardées décrivant les comportements abstraits des agents se déduisent de manière quasi-automatique des commandes apparaissant dans le système concret. Il serait néanmoins fastidieux d'expliquer point par point comment on abstrait chaque occurrence de chaque variable pour obtenir les versions abstraites. Nous allons donc détailler la construction sur des cas particuliers de commandes simples, puis généraliser pour obtenir la construction générale.

Exemple : Dans cet esprit, considérons un point de la forme $((a), (a')) \in PR_i$ et une substitution σ. Supposons que a est de la forme $\text{recv}(m)$ et que a' est de la forme $\text{send}(j, m')$. La commande gardée qui décrivait, dans l'ATS concret, l'envoi de message spécifié par a' avait pour garde :

$$g \quad \hat{=} \quad \neg i.\text{stopped} \wedge i.[\text{recv}(m\sigma)] \wedge \neg i.[\text{send}(j, m'\sigma)] \wedge \text{knows}_i(m'\sigma)$$

Les deux commandes décrivant l'abstraction associée auront pour gardes :

$$\begin{aligned}
g_\ominus &\quad \hat{=} \quad \neg(i.\text{stopped})_\oplus \wedge (i.[\text{recv}(m\sigma)])_\ominus \wedge \neg(i.[\text{send}(j, m'\sigma)])_\oplus \wedge \text{knows}_{\ominus i}(\alpha(m'\sigma)) \\
g_\oplus &\quad \hat{=} \quad \neg(i.\text{stopped})_\ominus \wedge (i.[\text{recv}(m\sigma)])_\oplus \wedge \neg(i.[\text{send}(j, m'\sigma)])_\ominus \wedge \text{knows}_{\oplus i}(\alpha(m'\sigma))
\end{aligned}$$

Par définition, on a :

$$
\begin{array}{rcl}
(i.\text{stopped})_\ominus & = & (i.\text{stopped})_\oplus \;=\; i.\text{stopped} \\
(i.[\text{recv}(m\sigma)])_\ominus & = & \left[\begin{array}{ll} i.[\text{recv}(m\sigma)] & \text{si } m\sigma \in M \\ \text{false} & \text{sinon} \end{array}\right. \\
(i.[\text{send}(j, m'\sigma)])_\oplus & = & i.[\text{send}(j, \alpha(m'\sigma))] \\
(i.[\text{recv}(m\sigma)])_\oplus & = & i.[\text{recv}(\alpha(m\sigma))] \\
(i.[\text{send}(j, m'\sigma)])_\ominus & = & \left[\begin{array}{ll} i.[\text{send}(j, m'\sigma)] & \text{si } m'\sigma \in M \\ \text{false} & \text{sinon} \end{array}\right.
\end{array}
$$

Par conséquent, la garde $g_\ominus$ ne peut être activée que si $m\sigma, m'\sigma \in M$ et, dans ce cas, s'écrit :

$$g_\ominus \quad = \quad \neg i.\text{stopped} \wedge i.[\text{recv}(m\sigma)] \wedge \neg i.[\text{send}(j, m'\sigma)] \wedge \text{knows}_{\ominus i}(m'\sigma)$$

$g_\oplus$ quant à elle peut être activée même si $m\sigma \notin M$ ou $m'\sigma \notin M$, mais, suivant les cas, elle peut s'écrire sous plusieurs formes. Tout d'abord, si $m\sigma, m'\sigma \in M$, on a :

$$g_\oplus \quad = \quad \neg i.\text{stopped} \wedge i.[\text{recv}(m\sigma)] \wedge \neg i.[\text{send}(j, m'\sigma)] \wedge \text{knows}_{\oplus i}(m'\sigma)$$

Si par contre on a $m\sigma \in M$ et $m'\sigma \notin M$, alors :

$$g_\oplus \quad = \quad \neg i.\text{stopped} \wedge i.[\text{recv}(m\sigma)] \wedge \text{true} \wedge \text{true}$$

Dans le cas où $m\sigma \notin M$ et $m'\sigma \in M$, on a :

$$g_\oplus \quad = \quad \neg i.\text{stopped} \wedge i.[\text{recv}(\bot_{\text{Msg}})] \wedge \neg i.[\text{send}(j, m'\sigma)] \wedge \text{knows}_{\oplus i}(m'\sigma)$$

Et finalement, lorsque $m\sigma \notin M$ et $m'\sigma \notin M$:

$$g_\oplus \quad = \quad \neg i.\text{stopped} \wedge i.[\text{recv}(\bot_{\text{Msg}})] \wedge \text{true} \wedge \text{true}$$

La mise à jour peut prendre deux formes :

$$u' \;\hat{=}\; i.\text{send}(j, m'\sigma) := \text{true} \quad \text{et} \quad u' \;\hat{=}\; i.\text{send}(j, \bot_{\text{Msg}}) := \text{true}$$

suivant que $m'\sigma \in M$ ou non. On obtient dans tous les cas deux commandes gardées :

$$\ominus\|_{\text{f}}\ g_\ominus \to u' \quad \text{et} \quad \oplus\|_{\text{f}}\ g_\oplus \to u'$$

En ce qui concerne les contraintes d'équité abstraites $\gamma'_\ominus$ et $\gamma'_\oplus$ associées à ces commandes, on pose :

$$
\begin{array}{rcl}
\gamma'_{\ominus ?}(i) & \hat{=} & g_\oplus \\
\gamma'_{\ominus !}(i) & \hat{=} & \alpha(m'\sigma) \neq \bot_{\text{Msg}} \wedge i.\text{send}(j, m'\sigma)' \\
\gamma'_{\oplus ?}(i) & \hat{=} & g_\ominus \\
\gamma'_{\oplus !}(i) & \hat{=} & i.\text{send}(j, \alpha(m'\sigma))'
\end{array}
$$

□

Dans le cas général, considérons un point $((a_1,\ldots,a_p),(a'_1,\ldots,a'_q)) \in PR_i$ avec $q \geq$, une substitution σ et $k \in \{1,\ldots,q\}$ tel que $a'_k\sigma$ soit de la forme $\mathrm{send}(j,m)$. Les commandes abstraites correspondantes sont :

$$\begin{array}{lll}
\ominus\|_{\mathrm{f}} & \neg(i.\mathrm{stopped})_\oplus & \rightarrow i.\mathrm{send}(j,\alpha(m\sigma)) := \mathrm{true} \\
& \wedge\ (i.[a_1\sigma])_\ominus \wedge (\ldots \wedge i.[a_p\sigma])_\ominus & \\
& \wedge\ \neg(i.[a'_1\sigma])_\oplus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\oplus & \\
& \wedge\ \mathrm{knows}_{\ominus i}(\alpha(m\sigma)) & \\
\oplus\|_{\mathrm{f}} & \neg(i.\mathrm{stopped})_\ominus & \rightarrow i.\mathrm{send}(j,\alpha(m\sigma)) := \mathrm{true} \\
& \wedge\ (i.[a_1\sigma])_\oplus \wedge (\ldots \wedge i.[a_p\sigma])_\oplus & \\
& \wedge\ \neg(i.[a'_1\sigma])_\ominus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\ominus & \\
& \wedge\ \mathrm{knows}_{\oplus i}(\alpha(m\sigma)) &
\end{array}$$

et les contraintes d'équité associées, $\gamma'_\ominus$ et $\gamma'_\oplus$, sont définies par :

$$\begin{array}{rcl}
\gamma'_{\ominus ?}(i) & \hat{=} & \neg(i.\mathrm{stopped})_\ominus \wedge (i.[a_1\sigma])_\oplus \wedge (\ldots \wedge i.[a_p\sigma])_\oplus \\
& & \qquad \wedge \neg(i.[a'_1\sigma])_\ominus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\ominus \wedge \mathrm{knows}_{\oplus i}(\alpha(m\sigma)) \\
\gamma'_{\ominus !}(i) & \hat{=} & \alpha(m\sigma) \neq \bot_{\mathrm{Msg}} \wedge i.\mathrm{send}(j,m\sigma)' \\
\gamma'_{\oplus ?}(i) & \hat{=} & \neg(i.\mathrm{stopped})_\oplus \wedge (i.[a_1\sigma])_\ominus \wedge (\ldots \wedge i.[a_p\sigma])_\ominus \\
& & \qquad \wedge \neg(i.[a'_1\sigma])_\oplus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\oplus \wedge \mathrm{knows}_{\ominus i}(\alpha(m\sigma)) \\
\gamma'_{\oplus !}(i) & \hat{=} & i.\mathrm{send}(j,\alpha(m\sigma))'
\end{array}$$

Si maintenant a'_k est de la forme $\mathrm{recv}(m)$ when $m' = m''$, alors les commandes abstraites correspondantes sont :

$$\begin{array}{lll}
\ominus\|_{\mathrm{f}} & \neg(i.\mathrm{stopped})_\oplus & \rightarrow i.\mathrm{recv}(\alpha(m\sigma)) := \mathrm{true} \\
& \wedge\ (i.[a_1\sigma])_\ominus \wedge \ldots \wedge (i.[a_p\sigma])_\ominus & \\
& \wedge\ \neg(i.[a'_1\sigma])_\oplus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\oplus & \\
& \wedge\ (\vee_{\substack{0\leq j\leq n \\ j\neq i}}(c_{j\rightarrow i}.\mathrm{fwd}(m\sigma))_\ominus) & \\
& \wedge\ \alpha(m'\sigma) \neq \bot_{\mathrm{Msg}} \wedge \alpha(m'\sigma) = \alpha(m''\sigma) & \\
\oplus\|_{\mathrm{f}} & \neg(i.\mathrm{stopped})_\ominus & \rightarrow i.\mathrm{recv}(\alpha(m\sigma)) := \mathrm{true} \\
& \wedge\ (i.[a_1\sigma])_\oplus \wedge \ldots \wedge (i.[a_p\sigma])_\oplus & \\
& \wedge\ \neg(i.[a'_1\sigma])_\ominus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\ominus & \\
& \wedge\ (\vee_{\substack{0\leq j\leq n \\ j\neq i}}(c_{j\rightarrow i}.\mathrm{fwd}(m\sigma))_\oplus) & \\
& \wedge\ \alpha(m'\sigma) = \alpha(m''\sigma) &
\end{array}$$

et les contraintes d'équité associées, $\gamma'_\ominus$ et $\gamma'_\oplus$, sont définies par :

$$\begin{array}{rcl}
\gamma'_{\ominus ?}(i) & \hat{=} & \neg(i.\mathrm{stopped})_\ominus \wedge (i.[a_1\sigma])_\oplus \wedge \ldots \wedge (i.[a_p\sigma])_\oplus \\
& & \qquad \wedge \neg(i.[a'_1\sigma])_\ominus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\ominus \wedge (\vee_{\substack{0\leq j\leq n \\ j\neq i}}(c_{j\rightarrow i}.\mathrm{fwd}(m\sigma))_\oplus) \\
\gamma'_{\ominus !}(i) & \hat{=} & \alpha(m\sigma) \neq \bot_{\mathrm{Msg}} \wedge i.\mathrm{recv}(m\sigma)' \\
\gamma'_{\oplus ?}(i) & \hat{=} & \neg(i.\mathrm{stopped})_\oplus \wedge (i.[a_1\sigma])_\ominus \wedge \ldots \wedge (i.[a_p\sigma])_\ominus \\
& & \qquad \wedge \neg(i.[a'_1\sigma])_\oplus \wedge \ldots \wedge \neg(i.[a'_q\sigma])_\oplus \wedge (\vee_{\substack{0\leq j\leq n \\ j\neq i}}(c_{j\rightarrow i}.\mathrm{fwd}(m\sigma))_\ominus)
\end{array}$$

$$\gamma'_{\oplus !}(i) \ \hat{=}\ i.\mathrm{recv}(\alpha(m\sigma))'$$

Pour un point de la forme $((a_1,\ldots,a_p),()) \in PR_i$ et une substitution σ, on a les commandes gardées :

$$\ominus\|_{\mathrm{f}}\ \neg(i.\mathrm{stopped})_{\oplus} \wedge (i.[a_1\sigma])_{\ominus} \wedge \ldots \wedge (i.[a_p\sigma])_{\ominus} \rightarrow i.\mathrm{stopped} := \mathrm{true}$$

$$\oplus\|_{\mathrm{f}}\ \neg(i.\mathrm{stopped})_{\ominus} \wedge (i.[a_1\sigma])_{\oplus} \wedge \ldots \wedge (i.[a_p\sigma])_{\oplus} \rightarrow i.\mathrm{stopped} := \mathrm{true}$$

munies des contraintes d'équité $\gamma'_{\ominus}$ et $\gamma'_{\oplus}$ définies par :

$$\begin{array}{rcl}
\gamma'_{\ominus ?}(i) & \hat{=} & \neg(i.\mathrm{stopped})_{\ominus} \wedge (i.[a_1\sigma])_{\oplus} \wedge \ldots \wedge (i.[a_p\sigma])_{\oplus} \\
\gamma'_{\ominus !}(i) & \hat{=} & i.\mathrm{stopped}' \\
\gamma'_{\oplus ?}(i) & \hat{=} & \neg(i.\mathrm{stopped})_{\oplus} \wedge (i.[a_1\sigma])_{\ominus} \wedge \ldots \wedge (i.[a_p\sigma])_{\ominus} \\
\gamma'_{\oplus !}(i) & \hat{=} & i.\mathrm{stopped}'
\end{array}$$

Finalement, il reste les commandes :

$$\begin{array}{l}
\ominus\|\ \mathrm{true} \rightarrow \mathrm{idle} \\
\oplus\|\ \mathrm{true} \rightarrow \mathrm{idle}
\end{array}$$

Agent faiblement malhonnête $i \in \Omega_{\mathrm{Part}}$. Les commandes gardées décrivant le comportement d'un tel agent ont été décrites à la figure 10.2.3, page 116. Comme elles ne présentent pas de contraintes d'équité et sont dans l'impossibilité de créer un bloquage au niveau abstrait (grâce à la présence de la commande $\|$ true $\rightarrow$ idle) nous nous contenterons de renvoyer le lecteur à la figure 12.2.1 où sont données les abstractions de ces commandes, sans plus de détails.

Agent fortement malhonnête $i \in \Omega_{\mathrm{Part}}$. Pour les mêmes raisons que celles évoquées dans le cas du comportement faiblement malhonnête, nous nous contentons de renvoyer le lecteur à la figure 10.2.4, page 116, où était défini le comportement fortement malhonnête, ainsi qu'à la figure 12.2.2 où se trouve son abstraction.

Agent $0 \in \Omega_{\mathrm{TTP}}$ représentant la TTP . À partir des commandes gardées définissant le comportement de l'agent représentant la TTP (figure 10.2.5, page 116), on dérive les commandes qui suivent, décrivant les comportement de l'agent correspondant dans le système abstrait. Pour chaque point $(\mathrm{recv}(m), m', (\mathrm{send}(i,m_1), \mathrm{send}(i,m_2))) \in PT$ et chaque substitution σ, on a les commandes :

$$\begin{array}{l}
\ominus\ \|\ (\vee_{j\in\Omega_{\mathrm{Part}}}(c_{j\rightarrow 0}.\mathrm{fwd}(m\sigma))_{\ominus}) \wedge (\vee_{j\in\Omega_{\mathrm{Part}}}(0.\mathrm{send}(j,m'\sigma))_{\ominus}) \wedge \neg(0.\mathrm{send}(i,m_1\sigma))_{\oplus} \rightarrow \\
\hfill 0.\mathrm{send}(i,\alpha(m_1\sigma)) := \mathrm{true} \\
\oplus\ \|\ (\vee_{j\in\Omega_{\mathrm{Part}}}(c_{j\rightarrow 0}.\mathrm{fwd}(m\sigma))_{\oplus}) \wedge (\vee_{j\in\Omega_{\mathrm{Part}}}(0.\mathrm{send}(j,m'\sigma))_{\oplus}) \wedge \neg(0.\mathrm{send}(i,m_1\sigma))_{\ominus} \rightarrow \\
\hfill 0.\mathrm{send}(i,\alpha(m_1\sigma)) := \mathrm{true} \\
\ominus\ \|\ (\vee_{j\in\Omega_{\mathrm{Part}}}(c_{j\rightarrow 0}.\mathrm{fwd}(m\sigma))_{\ominus}) \wedge \neg(\vee_{j\in\Omega_{\mathrm{Part}}}(0.\mathrm{send}(j,m'\sigma))_{\oplus}) \wedge \neg(0.\mathrm{send}(i,m_2\sigma))_{\oplus} \rightarrow \\
\hfill 0.\mathrm{send}(i,\alpha(m_2\sigma)) := \mathrm{true} \\
\oplus\ \|\ (\vee_{j\in\Omega_{\mathrm{Part}}}(c_{j\rightarrow 0}.\mathrm{fwd}(m\sigma))_{\oplus}) \wedge \neg(\vee_{j\in\Omega_{\mathrm{Part}}}(0.\mathrm{send}(j,m'\sigma))_{\ominus}) \wedge \neg(0.\mathrm{send}(i,m_2\sigma))_{\ominus} \rightarrow \\
\hfill 0.\mathrm{send}(i,\alpha(m_2\sigma)) := \mathrm{true}
\end{array}$$

Émission de $m \in \text{Msg}$ vers $j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$, $j \neq i$:

$$\ominus \| \ \neg(i.\text{stopped})_{\oplus} \wedge \text{knows}_{\ominus i}(\alpha(m)) \wedge \neg(i.\text{send}(j,m))_{\oplus} \rightarrow i.\text{send}(j,\alpha(m)) := \text{true}$$
$$\oplus \| \ \neg(i.\text{stopped})_{\ominus} \wedge \text{knows}_{\oplus i}(\alpha(m)) \wedge \neg(i.\text{send}(j,m))_{\ominus} \rightarrow i.\text{send}(j,\alpha(m)) := \text{true}$$

Réception de $m \in \text{Msg}$:

$$\ominus \| \ \neg(i.\text{stopped})_{\oplus} \wedge (\vee_{\substack{0 \leq j \leq n \\ j \neq i}} (c_{j \rightarrow i}.\text{fwd}(m))_{\ominus}) \wedge \neg(i.\text{recv}(m))_{\oplus} \rightarrow i.\text{recv}(\alpha(m)) := \text{true}$$
$$\oplus \| \ \neg(i.\text{stopped})_{\ominus} \wedge (\vee_{\substack{0 \leq j \leq n \\ j \neq i}} (c_{j \rightarrow i}.\text{fwd}(m))_{\oplus}) \wedge \neg(i.\text{recv}(m))_{\ominus} \rightarrow i.\text{recv}(\alpha(m)) := \text{true}$$

Arrêt du protocole :

$$\ominus \| \ \neg(i.\text{stopped})_{\oplus} \rightarrow i.\text{stopped} := \text{true}$$
$$\oplus \| \ \neg(i.\text{stopped})_{\ominus} \rightarrow i.\text{stopped} := \text{true}$$

Pour attendre :

$$\ominus \| \ \text{true} \rightarrow \text{idle}$$
$$\oplus \| \ \text{true} \rightarrow \text{idle}$$

FIG. 12.2.1 – *Abstraction du comportement de $i \in \Omega_{\text{Part}}$, faiblement malhonnête*

Émission de $m \in \text{Msg}$ vers $j \in \Omega_{\text{Part}} \cup \Omega_{\text{TTP}}$, $j \neq i$:

$$\ominus \| \ \neg(i.\text{stopped})_{\oplus} \wedge \text{knows}_{\ominus i}(\alpha(m)) \wedge \neg(i.\text{send}(j,m))_{\oplus} \rightarrow i.\text{send}(j,\alpha(m)) := \text{true}$$
$$\oplus \| \ \neg(i.\text{stopped})_{\ominus} \wedge \text{knows}_{\oplus i}(\alpha(m)) \wedge \neg(i.\text{send}(j,m))_{\ominus} \rightarrow i.\text{send}(j,\alpha(m)) := \text{true}$$

Réception de $m \in \text{Msg}$:

$$\ominus \| \ \neg(i.\text{stopped})_{\oplus} \wedge (\vee_{\substack{0 \leq j,k \leq n \\ j \neq i, k \neq j}} (c_{j \rightarrow i}.\text{fwd}(m))_{\ominus}) \wedge \neg(i.\text{recv}(m))_{\oplus} \rightarrow i.\text{recv}(\alpha(m)) := \text{true}$$
$$\oplus \| \ \neg(i.\text{stopped})_{\ominus} \wedge (\vee_{\substack{0 \leq j,k \leq n \\ j \neq i, k \neq j}} (c_{j \rightarrow i}.\text{fwd}(m))_{\oplus}) \wedge \neg(i.\text{recv}(m))_{\ominus} \rightarrow i.\text{recv}(\alpha(m)) := \text{true}$$

Arrêt du protocole :

$$\ominus \| \ \neg(i.\text{stopped})_{\oplus} \rightarrow i.\text{stopped} := \text{true}$$
$$\oplus \| \ \neg(i.\text{stopped})_{\ominus} \rightarrow i.\text{stopped} := \text{true}$$

Pour attendre :

$$\ominus \| \ \text{true} \rightarrow \text{idle}$$
$$\oplus \| \ \text{true} \rightarrow \text{idle}$$

FIG. 12.2.2 – *Abstraction du comportement de $i \in \Omega_{\text{Part}}$, fortement malhonnête*

ainsi que les commandes suivantes, permettant à la TTP d'attendre, lorsqu'aucune requête n'est en attente :

$$\ominus \llbracket \bigwedge_{p \in PT, \sigma} \begin{pmatrix} \neg(\vee_{j \in \Omega_{\text{Part}}}(c_{j \to 0}.\text{fwd}(m\sigma))_{\oplus}) \\ \vee\ \neg(0.\text{send}(m'\sigma))_{\oplus} \\ \vee\ (0.\text{send}(i, m_1\sigma))_{\ominus} \end{pmatrix} \wedge \bigwedge_{p \in PT, \sigma} \begin{pmatrix} \neg(\vee_{j \in \Omega_{\text{Part}}}(c_{j \to 0}.\text{fwd}(m\sigma))_{\oplus}) \\ \vee\ (\vee_{j \in \Omega_{\text{Part}}}(0.\text{send}(j, m'\sigma))_{\ominus}) \\ \vee\ (0.\text{send}(i, m_2\sigma))_{\ominus} \end{pmatrix} \to \text{idle}$$

$$\oplus \llbracket \bigwedge_{p \in PT, \sigma} \begin{pmatrix} \neg(\vee_{j \in \Omega_{\text{Part}}}(c_{j \to 0}.\text{fwd}(m\sigma))_{\ominus}) \\ \vee\ \neg(0.\text{send}(m'\sigma))_{\ominus} \\ \vee\ (0.\text{send}(i, m_1\sigma))_{\oplus} \end{pmatrix} \wedge \bigwedge_{p \in PT, \sigma} \begin{pmatrix} \neg(\vee_{j \in \Omega_{\text{Part}}}(c_{j \to 0}.\text{fwd}(m\sigma))_{\ominus}) \\ \vee\ (\vee_{j \in \Omega_{\text{Part}}}(0.\text{send}(j, m'\sigma))_{\oplus}) \\ \vee\ (0.\text{send}(i, m_2\sigma))_{\oplus} \end{pmatrix} \to \text{idle}$$

(où $p = (\text{recv}(m), m', (\text{send}(i, m_1), \text{send}(i, m_2))) \in PT$).

Remarque : Écrivons les commandes décrivant la relation Δ pour la TTP , dans le système concret, sous la forme :

$$\llbracket\ g_1 \to u_1, \ldots, \llbracket\ g_p \to u_p$$

Par construction, en chaque état q du système concret, une (et une seule) de ces commandes s'applique ce qui assure qu'il n'y a pas, dans l'ATS concret, de bloquage dû à cet agent. Autrement dit :

$$g_1 \vee \ldots \vee g_p \quad \Leftrightarrow \quad \text{true}$$

Nous venons de fournir les contreparties abstraites de ces commandes sous la forme :

$$\ominus\llbracket\ g_{1\ominus} \to u'_1, \ldots, \ominus\llbracket\ g_{p\ominus} \to u'_p \oplus \llbracket\ g_{1\oplus} \to u'_1, \ldots, \oplus\llbracket\ g_{p\oplus} \to u'_p$$

Par construction, on sait que :

$$\begin{aligned} g_{1\ominus} \vee \ldots \vee g_{p\ominus} &\quad \Rightarrow \quad g_1 \vee \ldots \vee g_p \\ g_1 \vee \ldots \vee g_p &\quad \Rightarrow \quad g_{1\oplus} \vee \ldots \vee g_{p\oplus} \end{aligned}$$

On peut donc être sur que, en chaque état abstrait, au moins une commande parmi $\oplus\llbracket\ g_{1\oplus} \to u'_1, \ldots, \oplus\llbracket\ g_{p\oplus} \to u'_p$ s'applique. Ce n'est par contre pas le cas avec les commandes $\ominus\llbracket\ g_{1\ominus} \to u'_1, \ldots, \ominus\llbracket\ g_{p\ominus} \to u'_p$. Si on fournit cette description à MOCHA, une commande :

$$\ominus\llbracket\ \neg g_{1\ominus} \wedge \ldots \neg g_{p\ominus} \to \text{idle}$$

sera automatiquement ajoutée, perturbant notre modélisation. Nous préférons par conséquent associer à la TTP une nouvelle variable, 0.blocked, dont la valeur de départ est false, et qui pourra être mise à true au moyen de la commande :

$$\ominus\llbracket\ \neg g_{1\ominus} \wedge \ldots \neg g_{p\ominus} \to 0.\text{blocked}' := \text{true}$$

On obtiendra ainsi un système non-bloquant (qui restera donc tel quel lorsque sa description sera fournie à MOCHA). On pourra par ailleurs facilement s'assurer que la commande qui vient d'être ajoutée n'est jamais appliquée, puis faire ensuite la vérification standard des propriétés qui nous intéressent. □

Canal opérationnel $c_{i\to j} \in \Omega_{\mathrm{Com}}$. Pour chaque message clos $m \in \mathrm{Msg}$, on a les commandes :

$$\begin{array}{l}
\ominus\llbracket\ (i.\mathrm{send}(j,m))_\ominus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\oplus \to c_{i\to j}.\mathrm{fwd}(\alpha(m)) := \mathrm{true}\\
\oplus\llbracket\ (i.\mathrm{send}(j,m))_\oplus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\ominus \to c_{i\to j}.\mathrm{fwd}(\alpha(m)) := \mathrm{true}\\
\ominus\llbracket\ \wedge_{m\in\mathrm{Msg}} (\neg(i.\mathrm{send}(j,m))_\oplus \vee (c_{i\to j}.\mathrm{fwd}(m))_\ominus) \to \mathrm{idle}\\
\oplus\llbracket\ \wedge_{m\in\mathrm{Msg}} (\neg(i.\mathrm{send}(j,m))_\ominus \vee (c_{i\to j}.\mathrm{fwd}(m))_\oplus) \to \mathrm{idle}
\end{array}$$

(comparer avec la définition du comportement de ces canaux dans le système concret, donnée à la figure 10.2.6, page 117).

Remarque : Comme dans le cas de la TTP , il est possible ici que, parmi les commandes :

$$\llbracket\ g_{1\ominus} \to u'_1, \ldots, \llbracket\ g_{p\ominus} \to u'_p$$

définissant la relation $\Delta'_\ominus$ pour l'agent $c_{i\to j}$, aucune ne s'applique en un certain état abstrait q'. Nous définissons donc une nouvelle variable, $c_{i\to j}.\mathrm{blocked}$, dont la valeur de départ est false, et qui pourra être mise à true au moyen de la commande :

$$\llbracket\ \neg g_{1\ominus} \wedge \ldots \neg g_{p\ominus} \to c_{i\to j}.\mathrm{blocked}' := \mathrm{true}$$

ce qui nous permettra de vérifier que cette commande n'est jamais appliquée. □

Canal résistant $c_{i\to j} \in \Omega_{\mathrm{Com}}$. Pour chaque message clos $m \in \mathrm{Msg}$, on a les commandes :

$$\begin{array}{l}
\ominus\llbracket_{\mathrm{f}}\ (i.\mathrm{send}(j,m))_\ominus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\oplus \to c_{i\to j}.\mathrm{fwd}(\alpha(m)) := \mathrm{true}\\
\oplus\llbracket_{\mathrm{f}}\ (i.\mathrm{send}(j,m))_\oplus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\ominus \to c_{i\to j}.\mathrm{fwd}(\alpha(m)) := \mathrm{true}
\end{array}$$

Les contraintes d'équité associées sont définies par :

$$\begin{array}{rcl}
\gamma'_{\ominus ?}(c_{i\to j}) & \hat{=} & (i.\mathrm{send}(j,m))_\oplus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\ominus\\
\gamma'_{\ominus !}(c_{i\to j}) & \hat{=} & \alpha(m) \neq \bot_{\mathrm{Msg}} \wedge c_{i\to j}.\mathrm{fwd}(\alpha(m))'\\
\gamma'_{\oplus ?}(c_{i\to j}) & \hat{=} & (i.\mathrm{send}(j,m))_\ominus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\oplus\\
\gamma'_{\oplus !}(c_{i\to j}) & \hat{=} & c_{i\to j}.\mathrm{fwd}(\alpha(m))'
\end{array}$$

On a également les commandes :

$$\begin{array}{l}
\ominus\llbracket\ \mathrm{true} \to \mathrm{idle}\\
\oplus\llbracket\ \mathrm{true} \to \mathrm{idle}
\end{array}$$

(voir la définition du comportement de ces canaux dans le système concret, figure 10.2.7, page 117).

Canal non-fiable $c_{i\to j} \in \Omega_{\mathrm{Com}}$. Comme la définition du comportement d'un canal non-fiable dans le système concret (figure 10.2.8, page 117) ne fait pas intervenir de contraintes d'équité, et ne crée pas de bloquage dans le système abstrait, nous ne détaillons pas les commandes abstraites qui correspondent et qui se trouvent à la figure 12.2.3.

Pour $m \in \mathrm{Msg}$, clos :

$$\begin{aligned}
&\ominus\| \ (i.\mathrm{send}(j,m))_\ominus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\oplus \to c_{i\to j}.\mathrm{fwd}(\alpha(m)) := \mathrm{true}\\
&\oplus\| \ (i.\mathrm{send}(j,m))_\oplus \wedge \neg(c_{i\to j}.\mathrm{fwd}(m))_\ominus \to c_{i\to j}.\mathrm{fwd}(\alpha(m)) := \mathrm{true}\\
&\ominus\| \ \mathrm{true} \to \mathrm{idle}\\
&\oplus\| \ \mathrm{true} \to \mathrm{idle}
\end{aligned}$$

FIG. 12.2.3 – *Abstraction du comportement de $c_{i\to j} \in \Omega_{\mathrm{Com}}$, non-fiable*

À la description d'ATS abstrait que nous avons détaillée dans ce chapitre correspond, bien entendu, un ATS abstrait $\mathcal{S}'^{\ell}_P$, correct relativement à $\mathcal{S}^{\ell}_P$ et à la surjection α définie dans la première section. Insistons également sur le fait que cette construction ne dépend que de l'ensemble fini $M \subseteq \mathrm{Msg}$ (qui doit être fourni par l'utilisateur, généralement il s'agit de l'ensemble des messages pouvant être acceptés par un participant honnête). Les deux caractéristiques fondamentales de l'ATS abstrait obtenu sont, d'une part, le fait qu'il soit fini et, d'autre part, le fait qu'il soit correct relativement à $\mathcal{S}^{\ell}_P$. Ainsi, si φ est un énoncé ATL qui représente une propriété de sécurité du protocole P, et si $\mathcal{S}'^{\ell}_P \models \varphi$, on peut en déduire que $\mathcal{S}^{\ell}_P \models \varphi$. Comme $\mathcal{S}'^{\ell}_P$ est fini, on peut de plus espérer pouvoir utiliser des techniques de *model-checking* afin de savoir si $\mathcal{S}'^{\ell}_P \models \varphi$. Nous avons de plus montré, dans ce chapitre, comment calculer l'ATS abstrait directement à partir du protocole. On peut cependant remarquer que certaines des commandes fournies sont paramétrées par un ensemble infini d'objets (des substitutions, des messages). Autrement dit, cette construction n'est pas effective. Dans le chapitre suivant, nous expliquerons comment calculer effectivement l'ATS abstrait, nous parlerons aussi du *model-checking* de ces ATS et des résultats obtenus.

Chapitre 13

En pratique

Dans le chapitre précédent, nous avons expliqué comment abstraire les ATS donnant la sémantique des protocoles cryptographiques en des ATS abstraits finis. Afin d'automatiser le processus de vérification des protocoles dans ce modèle, il faut donc, d'une part, produire effectivement les ATS abstraits à partir de la description du protocole et, d'autre part, adapter les outils existants de *model-checking* pour qu'ils acceptent des ATS abstraits. Nous traiterons successivement ces deux points dans les sections qui suivent puis nous terminerons en donnant quelques résultats.

13.1 Calcul des ATS abstraits

Dans le chapitre qui précède, nous avons associé aux protocoles cryptographiques des ATS abstraits finis (*i.e.* sur un ensemble fini de variables), mais ces ATS sont toujours décrits par une infinité de commandes gardées (plus précisément : les commandes gardées décrivant ces ATS sont, en général, paramétrées par des ensembles infinis, de substitutions ou de messages) et, qui plus est, certaines de ces commandes contiennent des énoncés utilisant des conjonctions ou disjonctions infinies.

Exemple : Dans la description de l'abstraction du comportement honnête d'un agent, page 138, on trouve la commande gardée suivante :

$$\begin{array}{lll} \ominus\|_{\mathrm{f}} & \neg(i.\mathrm{stopped})_{\oplus} & \rightarrow i.\mathrm{send}(j,\alpha(m\sigma)) := \mathrm{true} \\ & \wedge\ (i.[a_1\sigma])_{\ominus} \wedge (\ldots \wedge i.[a_p\sigma])_{\ominus} & \\ & \wedge\ \neg(i.[a'_1\sigma])_{\oplus} \wedge \ldots \wedge \neg(i.[a'_q\sigma])_{\oplus} & \\ & \wedge\ \mathrm{knows}_{\ominus i}(\alpha(m\sigma)) & \end{array}$$

qui est donnée pour tout point $((a_1,\ldots,a_p),(a'_1,\ldots,a'_q)) \in PR_i$ avec $q \geq$, toute substitution σ et tout $k \in \{1,\ldots,q\}$ tel que $a'_k\sigma$ soit de la forme $\mathrm{send}(j,m)$. En ce qui concerne le point considéré et l'entier k, les choix possibles sont en nombre fini. Par contre, l'ensemble des substitutions est infini. Mais on constate que, pour une substitution σ fixée, les propositions :

$$(i.[a_1\sigma])_{\ominus},\ldots,(i.[a_p\sigma])_{\ominus},(i.[a'_1\sigma])_{\oplus},\ldots,(i.[a'_q\sigma])_{\oplus}$$

se réécrivent en utilisant :

$$i.[\alpha(a_1\sigma)],\ldots,i.[\alpha(a_p\sigma)],i.[\alpha(a'_1\sigma)],\ldots,i.[\alpha(a'_q\sigma)]$$

et, éventuellement, false. Il suffit donc de connaître l'ensemble des valeurs possibles de :

$$(\alpha(a_1\sigma), \ldots, \alpha(a_p\sigma), \alpha(a'_1\sigma), \ldots, \alpha(a'_q\sigma))$$

lorsque σ parcourt l'ensemble des substitutions (et cet ensemble est, lui, fini). Le deuxième problème avec cette commande gardée est que l'énoncé définissant $\text{knows}_{\ominus i}(\alpha(m\sigma))$ contient des conjonctions ou disjonctions infinies. $\square$

Au vu de cet exemple, nous constatons que pour produire les ATS abstraits associés à un protocole, il suffit :

- de savoir calculer, pour $m_1, \ldots, m_p \in \text{Msg}$, l'ensemble (fini) des valeurs prises par l'application :
$$\begin{array}{rccl} [m_1, \ldots, m_p] : & \text{Subst} & \rightarrow & \text{Msg}'^p \\ & \sigma & \mapsto & (m_1\sigma, \ldots, m_n\sigma) \end{array}$$
- de connaître, pour chaque $i \in \Omega_{\text{Part}}$ et chaque $m' \in \text{Msg}'$, des énoncés propositionnels finis équivalents à $\text{knows}_{\ominus i}(m')$ et $\text{knows}_{\oplus i}(m')$.

Nous allons successivement nous occuper de chacun de ces deux problèmes.

Calcul de l'image de $[m_1, \ldots, m_p]$. Théoriquement, ce problème est équivalent au suivant : étant donnés $m'_1, \ldots, m'_p \in \text{Msg}'$, existe-t-il une substitution σ telle que $\alpha(m_1\sigma) = m'_1, \ldots, \alpha(m_p\sigma) = m'_p$? Pour répondre à cette question, posons :

$$I \ \hat{=}\ \{i \in \{1, \ldots, p\} \mid m'_i \in M\} \quad \text{et} \quad J \ \hat{=}\ \{1, \ldots, p\} \setminus I$$

On a donc $m'_i \in M = \text{Msg} \cap \text{Msg}'$ pour chaque $i \in I$ et $m'_j = \perp_{\text{Msg}}$ pour $j \in J$. Dans ces conditions, une substitution σ répondant à la question doit être un unificateur de m_i et m'_i pour chaque $i \in I$. On commence donc par calculer une substitution σ', définie sur les variables apparaissant dans les m_i (pour $i \in I$) et telle que $m_i\sigma' = m'_i$ (pour $i \in I$). Une telle substitution, si elle existe, est unique (car les messages m'_i (pour $i \in I$) sont clos). On cherche ensuite s'il est possible d'étendre σ' en une substitution σ telle que, quel que soit $j \in J$, on ait $\sigma(m_j) \notin M$. Pour cela, notons $m''_j \ \hat{=}\ m_j\sigma'$ (l'instanciation peut n'être que partielle, dans le cas où m_j contient des variables qui ne sont pas dans le domaine de σ', le message $m''_j \in \text{Msg}$ n'est donc pas nécessairement clos), on cherche alors s'il existe une substitution σ'' telle que $m''_j\sigma'' \notin M$ (pour $j \in J$). On est donc ramené au problème suivant : étant donnés $m_1, \ldots, m_p \in \text{Msg}$, existe-t-il une substitution σ telle que $m_1\sigma \notin M, \ldots, m_p\sigma \notin M$? Pour répondre à cette question, considérons, pour chaque variable x apparaissant dans l'un des messages $m_1, \ldots, m_p$, l'ensemble M_x des messages clos qui peuvent instancier x. Cet ensemble peut être fini (pour des raisons de typage) ou infini. Dans le cas où M_x est infini (ce qui peut être décidé syntaxiquement sur $m_1, \ldots, m_p$), il est sur qu'il existe au moins un message clos $m \in M_x$ tel que, quelle que soit la substitution σ telle que $\sigma(x) = m$ et quel que soit $i \in \{1, \ldots, p\}$ tel que x apparaisse dans m_i, on ait $m_i\sigma \notin M$. On est ainsi ramené au cas où, pour chaque variable x apparaissant dans $m_1, \ldots, m_p$, M_x est fini. On calcule alors explicitement chaque ensemble M_x, puis toutes les substitutions σ possibles et on regarde si, pour l'une d'elles, on a $m_1\sigma \notin M, \ldots, m_p\sigma \notin M$.

Signalons pour finir une optimisation possible. Au lieu de considérer tous les p-uplets de messages abstraits $(m'_1, \ldots, m'_p) \in \text{Msg}'^p$ possibles et les ensembles I et J associés, on peut considérer tous les ensembles $I \subseteq \{1, \ldots, p\}$ possibles, puis considérer $J \ \hat{=}\ \{1, \ldots, p\} \setminus I$, $m'_j \ \hat{=}\ \perp_{\text{Msg}}$ pour $j \in J$ et, pour $i \in I$, $m'_i \in M$ parmi les instances de m_i.

Calcul de $\text{knows}_{\ominus i}(m)$ **et** $\text{knows}_{\oplus i}(m)$ **pour** $m \in M$. Commençons par $\text{knows}_{\ominus i}(m)$ et rappelons la définition donnée au chapitre précédent :

$$\text{knows}_{\ominus i}(m) \quad \hat{=} \quad \bigvee_{\substack{s \subseteq \text{Msg} \\ s \Vdash m}} \; \bigwedge_{m_1 \in s} ((i.\text{recv}(m_1))_{\ominus} \vee m_1 \in IK_i)$$

Comme $(i.\text{recv}(m_1))_{\ominus} = \text{false}$ dès que $m_1 \notin M$, on a en fait :

$$\text{knows}_{\ominus i}(m) \quad \hat{=} \quad \bigvee_{\substack{s \subseteq M \cup IK_i \\ s \Vdash m}} \; \bigwedge_{m_1 \in s \setminus IK_i} i.\text{recv}(m_1)$$

et les conjonctions et disjonctions qui interviennent ici sont finies, ce qui répond à la question posée. Notons tout de même que, pour plus d'efficacité, on déterminera les sous-ensembles $s \subseteq M \cup IK_i$ tels que $s \Vdash m$ en utilisant les techniques d'analyse et synthèse de L. Paulson (chapitre 1).

Passons maintenant au calcul de $\text{knows}_{\oplus i}(m)$. La définition est la suivante :

$$\text{knows}_{\oplus i}(m) \quad \hat{=} \quad \bigvee_{\substack{s \subseteq \text{Msg} \\ s \Vdash m}} \; \bigwedge_{m_1 \in s} ((i.\text{recv}(m_1))_{\oplus} \vee m_1 \in IK_i)$$

Il est clair qu'il existe des sous ensembles $s \subseteq \text{Msg} \setminus M$ tels que $s \Vdash m$ (puisque M est fini). Ainsi, si $i.\text{recv}(\bot_{\text{Msg}})$ est vrai, $\text{knows}_{\oplus i}(m)$ est vrai, quel que soit $m \in M$. Si maintenant $i.\text{recv}(\bot_{\text{Msg}})$ est faux, alors on peut se contenter, comme dans le cas de $\text{knows}_{\ominus i}(m)$, de considérer les sous-ensembles $s \subseteq M \cup IK_i$ tels que $s \Vdash m$. On a donc :

$$\text{knows}_{\ominus i}(m) \quad \hat{=} \quad i.\text{recv}(\bot_{\text{Msg}}) \;\vee \bigvee_{\substack{s \subseteq M \cup IK_i \\ s \Vdash m}} \; \bigwedge_{m_1 \in s \setminus IK_i} i.\text{recv}(m_1)$$

Avec ces considérations, nous sommes à même de produire un ATS abstrait en fonction du protocole cryptographique et du sous-ensemble fini $M \subseteq \text{Msg}$ fournis en entrée. Nous allons voir dans la section suivante comment vérifier, de manière effective, qu'un énoncé de la logique ATL est vrai dans un tel ATS abstrait.

13.2 Vérification effective

Vérification effective dans un ATS libre $\mathcal{S} = (Q, \Delta, \pi)$. La définition de la sémantique d'un énoncé $\varphi \in \text{Atl}(\Omega, \Pi)$ dans un ATS $\mathcal{S}$ fait intervenir les notions d'existence et de non-existence d'une stratégie obéissant à certaines conditions et n'est, par conséquent, *a priori* pas effective. Les auteurs de [AHK98] proposent un algorithme de *model-checking* inspiré des algorithmes de *model-checking* symbolique des logiques temporelles classiques. Les ingrédients intervenant, en général, dans la construction de tels algorithmes sont :

- une méthode de représentation symbolique des ensembles d'états (par exemple avec des diagrammes de décision binaires) ;
- des calculs de points fixes (dont on est assuré de la terminaison, car l'ensemble des états du système est supposé fini) ;
- un opérateur calculant les prédécesseurs d'un ensemble d'états.

Comme expliqué dans [AHK98], pour faire du *model-checking* sur des ATS libres, il suffit d'adapter la définition du calcul de prédécesseur au cas des ATS, comme dans la définition qui suit.

DÉFINITION – 13.2.1

Soit $\mathcal{Q} \subseteq Q$. L'ensemble des prédécesseurs de $\mathcal{Q}$, sous le contrôle de $\Omega' \subseteq \Omega$, dans $\mathcal{S}$, noté $\mathrm{CPre}_{\mathcal{S}}(\Omega', \mathcal{Q})$*, est défini par :*

$$\begin{aligned} \mathrm{CPre}_{\mathcal{S}}(\Omega', \mathcal{Q}) \ \hat{=}\ \{q \in Q \mid & \\ & \exists (Q_\omega)_{\omega \in \Omega'} \in \textstyle\prod_{\omega \in \Omega'} \Delta(q, \omega).\ \forall (Q_\omega)_{\omega \notin \Omega'} \in \textstyle\prod_{\omega \notin \Omega'} \Delta(q, \omega). \\ & \hfill \textstyle\bigcap_{\omega \in \Omega} Q_\omega \subseteq \mathcal{Q}\} \end{aligned}$$

De manière duale, l'ensemble des prédécesseurs de $\mathcal{Q}$, hors du contrôle de Ω', dans $\mathcal{S}$, noté $\mathrm{UPre}_{\mathcal{S}}(\Omega', \mathcal{Q})$*, est défini par :*

$$\begin{aligned} \mathrm{UPre}_{\mathcal{S}}(\Omega', \mathcal{Q}) \ \hat{=}\ \{q \in Q \mid & \\ & \forall (Q_\omega)_{\omega \in \Omega'} \in \textstyle\prod_{\omega \in \Omega'} \Delta(q, \omega).\ \exists (Q_\omega)_{\omega \notin \Omega'} \in \textstyle\prod_{\omega \notin \Omega'} \Delta(q, \omega). \\ & \hfill \textstyle\bigcap_{\omega \in \Omega} Q_\omega \subseteq \mathcal{Q}\} \end{aligned}$$

De manière informelle, $q \in \mathrm{CPre}_{\mathcal{S}}(\Omega', \mathcal{Q})$ si les agents de Ω' peuvent coopérer de sorte que l'état suivant soit dans $\mathcal{Q}$, quels que soient les choix faits par les agents hors de Ω'. Inversement, $q \in \mathrm{UPre}_{\mathcal{S}}(\Omega', \mathcal{Q})$ si, quels que soient les choix faits par les agents de Ω', les agents hors de Ω' peuvent coopérer de sorte que l'état suivant soit dans $\mathcal{Q}$. Nous pouvons maintenant donner la seconde définition de la sémantique d'un énoncé. On peut ensuite donner la sémantique des énoncés ATL en utilisant ces opérateurs. À titre d'exemple, on a (pour $\varphi \in \mathrm{Atl}(\Omega, \Pi)$ et $\Omega' \subseteq \Omega$) :

$$[\![\langle \Omega' \rangle \Diamond \varphi]\!]_{\mathcal{S}} \ = \ \mathrm{lfp}(C_{\Diamond, \Omega', \varphi}) \quad \text{et} \quad [\![[\Omega'] \Diamond \varphi]\!]_{\mathcal{S}} \ = \ \mathrm{lfp}(U_{\Diamond, \Omega', \varphi})$$

où lfp désigne le plus petit point fixe et $C_{\Diamond, \Omega', \varphi}$ et $U_{\Diamond, \Omega', \varphi}$ sont les applications :

$$\begin{array}{rccl} C_{\Diamond, \Omega', \varphi}: & 2^Q & \rightarrow & 2^Q \\ & \mathcal{Q} & \mapsto & \mathrm{CPre}_{\mathcal{S}}(\Omega', \mathcal{Q}) \cup [\![\varphi]\!]_{\mathcal{S}} \\ U_{\Diamond, \Omega', \varphi}: & 2^Q & \rightarrow & 2^Q \\ & \mathcal{Q} & \mapsto & \mathrm{UPre}_{\mathcal{S}}(\Omega', \mathcal{Q}) \cup [\![\varphi]\!]_{\mathcal{S}} \end{array}$$

Nous renvoyons le lecteur à [AHK98] pour une présentation complète de l'algorithme de *model-checking*. Expliquons maintenant comment sont gérées les contraintes d'équité.

Vérification effective dans un ATS $\mathcal{S} = (Q, q_0, \Delta, \Gamma, \pi)$. Le problème est d'inclure les contraintes d'équité dans le processus de vérification, mais comme expliqué dans [AHK98], il est toujours possible d'éliminer ces contraintes et de se ramener par conséquent à un ATS libre.

Vérification effective dans un ATS abstrait $\mathcal{S}' = (Q', q'_0, \Delta'_\ominus, \Delta'_\oplus, \Gamma'_\ominus, \Gamma'_\oplus, \pi')$. Comme expliqué dans [HMMR00], le *model-checking* d'un ATS abstrait se fait de la même manière que celui d'un ATS concret, à ceci près que les opérateurs de calcul des prédécesseurs doivent être remplacés par les opérateurs (abstraits) de la définition suivante.

Définition – 13.2.2

Soit $\mathcal{Q}' \subseteq Q'$*. L'ensemble des prédécesseurs de* $\mathcal{Q}'$*, sous le contrôle de* $\Omega' \subseteq \Omega$*, dans* $\mathcal{S}'$*, noté* $\mathrm{CPre'}_{\mathcal{S}'}(\Omega', \mathcal{Q}')$*, est défini par :*

$$\begin{aligned}\mathrm{CPre'}_{\mathcal{S}'}(\Omega', \mathcal{Q}') \ \hat{=}\ \{q' \in Q' \mid \\ \exists (Q'_\omega)_{\omega \in \Omega'} \in \textstyle\prod_{\omega \in \Omega'} \Delta'_{\ominus}(q', \omega).\ \forall (Q'_\omega)_{\omega \notin \Omega'} \in \textstyle\prod_{\omega \notin \Omega'} \Delta'_{\oplus}(q', \omega). \\ \textstyle\bigcap_{\omega \in \Omega} Q'_\omega \subseteq \mathcal{Q}'\}\end{aligned}$$

De manière duale, l'ensemble des prédécesseurs de $\mathcal{Q}'$*, hors du contrôle de* Ω'*, dans* $\mathcal{S}'$*, noté* $\mathrm{UPre'}_{\mathcal{S}'}(\Omega', \mathcal{Q}')$*, est défini par :*

$$\begin{aligned}\mathrm{UPre'}_{\mathcal{S}'}(\Omega', \mathcal{Q}') \ \hat{=}\ \{q' \in Q' \mid \\ \forall (Q'_\omega)_{\omega \in \Omega'} \in \textstyle\prod_{\omega \in \Omega'} \Delta'_{\oplus}(q', \omega).\ \exists (Q'_\omega)_{\omega \notin \Omega'} \in \textstyle\prod_{\omega \notin \Omega'} \Delta'_{\ominus}(q', \omega). \\ \textstyle\bigcap_{\omega \in \Omega} Q'_\omega \subseteq \mathcal{Q}'\}\end{aligned}$$

Autrement dit, pour vérifier que les agents de Ω' peuvent coopérer de sorte que l'état suivant soit dans $\mathcal{Q}$, quels que soient les choix faits par les agents hors de Ω', on vérifie que les agents de Ω', lorsque on leur donne moins de pouvoir, tout en augmentant celui des agents hors de Ω', peuvent malgré tout coopérer de sorte que l'état suivant soit dans $\alpha(\mathcal{Q})$. En ce qui concerne les contraintes d'équité, on procède de la même manière, en considérant tantôt les versions plus fortes, tantôt les versions plus faibles, suivant que l'on cherche à montrer que les agents peuvent, ou ne peuvent pas, obtenir quelque chose. Ces techniques ont été implémentées par S. Kremer en utilisant Slang, le langage de script de Mocha, ce qui a permis d'adapter ce *model-checker* à nos besoins.

13.3 Résultats

L'implémentation dont nous disposons correspond à une modélisation plus ancienne que celle présentée au chapitre 10 (mais néanmoins très proche). Pour simplifier la présentation, nous emploierons systématiquement les termes « ancienne modélisation » et « modélisation actuelle » dans cette section. Nous allons tout d'abord présenter les différences qui existent entre ces deux modélisations, puis décrire le fonctionnement de l'implémentation ainsi que les résultats obtenus.

La première différence correspond à la manière dont les différents participants gardent trace de la position à laquelle ils se trouvent à l'intérieur du protocole. Dans l'ancienne modélisation, nous avions associé à chaque nœud de l'arbre correspondant au rôle du participant en question une variable booléenne. Ces variables étaient mise à jour au fur et à mesure que le participant en question effectuait des actions (envoi et réception de messages) et permettaient donc de savoir à quel point, dans le protocole, il se trouvait. Rappelons que dans la modélisation actuelle, le point du protocole où se trouve un participant est déduit de l'ensemble des actions qu'il a effectuées. L'expérience montre que, dans le cas des protocoles de la littérature, cette identification entre point du protocole et actions effectuées n'est pas ambiguë. Chaque modélisation gère également les nonces différemment. Dans l'ancienne modélisation, les nonces étaient gérés un peu comme des identificateurs de session : nous les fixions une fois pour toutes. Si, par exemple, le protocole considéré commençait de la manière suivante :

$$1. \quad A \quad \rightarrow \quad B \quad : \quad \langle N_A, \mathrm{sig}_A(C)\rangle$$

et si la valeur de N_A était fixée à n_a, alors B ne pouvait accepter que $\langle n_a, \mathrm{sig}_a(c)\rangle$ et aucun autre message. Comme on l'a vu, la modélisation actuelle laisse la possibilité aux participants d'accepter n'importe quel message, du moment qu'il possède la forme d'un message attendu. D'autre part, chaque participant qui doit envoyer un message doit pouvoir construire ce message à partir de sa connaissance initiale et de ceux qu'il a reçu. En reprenant l'exemple précédent, si on met dans la connaissance initiale de A deux nonces n_a et n'_a, il pourra alors envoyer à B les messages $\langle n_a, \mathrm{sig}_a(c)\rangle$ et $\langle n'_a, \mathrm{sig}_a(c)\rangle$ et B devra accepter chacun de ces deux messages. Notons que, pour pouvoir arriver à prouver quelque chose au moyen de l'abstraction, il faut, bien entendu, conserver ces deux messages dans M. Ce changement dans la gestion des nonces entre les deux modélisations a conduit à faire une troisième modification, dans le comportement de la TTP . En effet, on peut imaginer que A utilise n_a lors de sa communication avec B, puis n'_a avec T. Si B communique ensuite avec T, il va utiliser n_a et la TTP sera alors en présence de deux jeux de messages : ceux qui font référence à n_a et ceux qui font référence à n'_a, ce qui correspond en fait pour elle à deux sessions distinctes. Il nous est donc apparu que, avec une gestion correcte des nonces, le comportement de la TTP devait nécessairement être multi-session, ce qui n'était pas le cas dans l'ancienne modélisation (la TTP était alors modélisée comme un participant normal). La modélisation actuelle permet par contre à la TTP de répondre à des requêtes qui appartiennent à des sessions différentes. Notons que le travail nécessaire pour adapter l'implémentation dont nous disposons à la modélisation actuelle serait quelque peu fastidieux, mais en aucun cas difficile (techniquement parlant).

La vérification d'un protocole optimiste d'échange au moyen de cette implémentation comporte deux phases. Dans la première, on utilise un programme écrit en OCAML qui prend une description du protocole (sous une forme proche des processus qui ont été utilisés ici), de l'abstraction considérée (sous forme d'une liste finie de messages) ainsi que des niveaux d'honnêteté et de fiabilité suivant lesquels on désire effectuer la vérification. On lui fournit également les propriétés (de la logique ATL) que l'on désire vérifier. Il retourne alors une description du modèle abstrait correspondant au protocole et au niveau d'honnêteté et de fiabilité souhaité, sous forme de modules pour le *model-checker* MOCHA. Chaque propriété est, quant à elle, transformée en un script de vérification pour MOCHA (dans le langage SLANG). On peut ensuite effectuer la vérification au moyen de MOCHA.

Nous avons effectué la vérification des propriétés usuelles sur un certain nombre de protocoles optimistes d'échange équitable, qui se trouvent dans la littérature : le protocole de non répudiation de Kremer et Markowitch [KM00], les protocoles de signature de contrats de Asokan, Schunter et Waidner [ASW98b] et de Garay, Jakobsson et MacKenzie [GJM99], en trouvant à chaque fois des résultats conformes à la littérature [KR01, KR02, SM02]. En ce qui concerne le protocole de courrier électronique certifié Asokan, Schunter et Waidner [ASW98a], nous avons même noté que l'attaque de V. Shmatikov et J. Mitchell (sur le protocole de Garay, Jakobsson et MacKenzie) s'appliquait là aussi. Ce point mérite d'être noté. En effet, notre méthode de vérification est construite de manière à prouver des propriétés. Cependant, nous avons remarqué que, lorsqu'une propriété φ n'est pas satisfaite dans le modèle abstrait, il arrive parfois que $\neg\varphi$ soit satisfaite (dans le modèle abstrait). On est alors sûr que φ est fausse sur le modèle concret. C'était précisément le cas ici.

13.4 Conclusion

Nous avons présenté dans cette partie les fruits d'un travail joint avec S. Kremer et J.-F. Raskin qui avait pour but d'étudier des protocoles optimistes d'échange. Ces protocoles et leur propriétés ont tout d'abord été modélisés au moyen des systèmes de transitions alternés et de la logique temporelle alternée (nous nous sommes limités à l'étude d'une seule session). Les modèles proposés ont ensuite été simplifiés au moyen de techniques d'interprétation abstraite. Les systèmes finis obtenus de cette manière constituent des approximations correctes du modèle de départ et il est possible de les vérifier automatiquement au moyen de techniques de *model-checking*. Évoquons maintenant quelques points qui nous paraissent susceptibles d'être développés ultérieurement.

Abstraction par prédicats (travail en cours). Nous avons évoqué plus haut le cas d'un protocole commençant par :

$$1. \quad A \quad \to \quad B \quad : \quad \langle N_A, \mathrm{sig}_A(C)\rangle$$

et signalé que, dans le cas où on disposait dans la connaissance initiale de A deux nonces n_a et n'_a, il fallait garder dans l'abstraction les messages $\langle n_a, \mathrm{sig}_a(c)\rangle$ et $\langle n'_a, \mathrm{sig}_a(c)\rangle$. Mais il est fastidieux (et peu économique, du point de vue du nombre de variables) de garder chaque instanciation de chaque message par les différents nonces possibles. Nous préférerions, par exemple, retenir qu'un message de la forme $\langle N_A, \mathrm{sig}_A(C)\rangle$ a été envoyé ou reçu avec des contraintes qui captureraient l'information nécessaire sur la valeur du nonce instanciant N_A. Nous sommes en train (toujours dans un travail joint avec S. Kremer et J.-F. Raskin) d'utiliser la notion d'abstraction par prédicats [HMMR00] dans ce but.

Observations partielles. Si on regarde attentivement la définition de la notion de stratégie (définition 9.1.7, page 101), on s'aperçoit que la stratégie d'un joueur se base sur la connaissance de l'état courant (global) du système (et des états précédents) pour lui permettre de choisir le prochain état. On peut argumenter en disant que, après tout, observer l'état global du système, c'est savoir quels messages ont été reçus et envoyés, ce qui semble raisonnable (en tout cas, faisable dans la réalité), mais il serait dommage qu'une stratégie impose à un joueur honnête d'aller espionner les émissions et réceptions des autres joueurs. Une méthode consisterait à utiliser les observations partielles qui sont présentées dans [AHK98]. Le *model-checking* de la logique ATL est alors indécidable, sauf si on se restreint au fragment de ATL ne contenant que des modalités de la forme $\langle\omega\rangle\,\varphi$, où ω est un agent. Clairement, ce fragment ne suffit pas pour nos propriétés de sécurité, mais on doit pouvoir sans difficulté se ramener au fragment de ATL ne contenant que des modalités de la forme $\langle\Omega_i\rangle\,\varphi$ pour $i \in \{1,2\}$ où (Ω_1, Ω_2) est une partition de l'ensemble Ω des agents, fixée une fois pour toutes. Autrement dit, on doit fixer deux groupes disjoints d'agents et n'utiliser que ceux-là dans nos propriétés, ce qui semble faisable. Par contre, la mise en œuvre demanderait de vérifier si les abstractions et les contraintes d'équité peuvent être gérées de la même manière que ce qui a été présenté ici.

Questions de décidabilité. À notre connaissance, il n'existe pas, dans la littérature concernant les protocoles d'échange équitable, d'étude des questions de décidabilité (sous des hypothèses telles que nombre fini de sessions, pas de création de nonces, etc.) comme c'est le cas pour, par exemple, les propriétés de secret. Il est d'ailleurs tout à fait possible que des résultats analogues puissent-être obtenus, mais la modélisation complètement différente (sous forme de jeux et de stratégies) empêche de partir de ces résultats. Une étude systématique des propriétés de décidabilité, dans le contexte des protocoles d'échange équitable, serait ainsi un prolongement pertinent de notre travail.

Troisième partie

Propriétés d'opacité

Chapitre 14

Introduction

14.1 Deux exemples

Dans [Cha88], D. Chaum imagine la situation suivante : trois cryptographes sont en train de dîner, lorsque le serveur les informe qu'un arrangement a été pris avec le maître d'hôtel et que le repas a été payé. Les trois cryptographes aimeraient s'assurer que c'est bien l'un d'eux (et non pas une personne extérieure) qui a payé le repas, tout en préservant son anonymat. D. Chaum propose de procéder de la manière suivante : chaque cryptographe va lancer une pièce de monnaie, visible uniquement de lui et de son voisin de droite. Chacun dit ensuite si les deux pièces qu'il voit sont les mêmes ou pas, sauf celui qui a payé le repas, qui dit le contraire. Si le nombre de cryptographes annonçant des pièces différentes est impair, alors l'un d'eux a fait un inversion et, par conséquent, l'un d'eux a payé le repas. Pour comprendre pourquoi ce protocole fonctionne, commençons par le formaliser. On note A_1, A_2 et A_3 les cryptographes participant au repas, Q_i le résultat du tirage effectué par le cryptographe A_i (on note par exemple $Q_i = 0$ si le résultat est « pile » et $Q_i = 1$ si c'est « face ») et $P_i \in \{0, 1\}$ indique si A_i a payé le repas (au plus un élément parmi P_1, P_2, P_3 est donc non nul). Le fait que les tirages Q_i et Q_j soient différents peut-être représenté au moyen de l'opérateur « ou exclusif » que nous noterons $\oplus$. Ainsi $Q_i \oplus Q_j = 1$ si $Q_i \neq Q_j$ et $Q_i \oplus Q_j = 0$ sinon. L'annonce faite par le cryptographe A_i correspond donc à $Q_i \oplus Q_j$ si A_i n'a pas payé le repas et A_j est son voisin de gauche. Si A_i a payé le repas, il annonce alors $1 \oplus (Q_i \oplus Q_j)$. Dans tous les cas, A_i annonce donc $P_i \oplus (Q_i \oplus Q_j)$. Nous obtenons donc le protocole suivant, dans lequel les trois premières lignes servent à simuler le fait que chaque cryptographe communique le résultat de son tirage à son voisin de droite et les trois dernières correspondent aux annonces faites par les cryptographes.

<u>Dîner des cryptographes (D. Chaum)</u> :

$$
\begin{array}{llcll}
1. & A_1 & \rightarrow & A_2 & : \{Q_1\}_{\mathrm{k}(A_1,A_2)} \\
2. & A_2 & \rightarrow & A_3 & : \{Q_2\}_{\mathrm{k}(A_2,A_3)} \\
3. & A_3 & \rightarrow & A_1 & : \{Q_3\}_{\mathrm{k}(A_3,A_1)} \\
4. & A_1 & \rightarrow & & : R_1 \mathrel{\hat=} P_1 \oplus Q_1 \oplus Q_3 \\
5. & A_2 & \rightarrow & & : R_2 \mathrel{\hat=} P_2 \oplus Q_2 \oplus Q_1 \\
6. & A_3 & \rightarrow & & : R_3 \mathrel{\hat=} P_3 \oplus Q_3 \oplus Q_2
\end{array}
$$

Après l'exécution de ce protocole, chacun des participants peut alors calculer :

$$\begin{aligned} R &\hat{=} R_1 \oplus R_2 \oplus R_3 \\ &= P_1 \oplus Q_1 \oplus Q_3 \oplus P_2 \oplus Q_2 \oplus Q_1 \oplus P_3 \oplus Q_3 \oplus Q_2 \\ &= P_1 \oplus P_2 \oplus P_3 \end{aligned}$$

Compte-tenu du fait qu'au plus un élément parmi P_1, P_2, P_3 est non nul, on a $R = 1$ si, et seulement si, l'un des P_i est non nul, bien qu'il ne soit *a priori* pas possible de savoir lequel. Notons ici que les différentes données qui apparaissent dans le protocole (les identités, les valeurs des Q_i et P_i) ne sont pas secrètes au sens de la cryptographie (en particulier, au sens de la partie I) mais que la fonction qui, à chaque participant A_i, associe P_i doit rester (en partie) inconnue. Nous expliquerons comment formaliser de telles propriétés. Donnons pour l'instant un deuxième exemple, toujours du même auteur, mais cette fois extrait de [Cha81]. Il s'agit, dans cet article, d'expliquer comment on peut permettre d'envoyer des messages de manière anonyme. Le principe consiste à utiliser un intermédiaire, appelé *MIX*, à qui les émetteurs envoient les messages (chiffrés avec la clé publique de *MIX*) qui sont ensuite redistribués aux receveurs. Imaginons par exemple que A_1, A_2 et A_3 veuillent respectivement envoyer les messages M_1, M_2 et M_3 à B_{j_1}, B_{j_2} et B_{j_3}. On a alors le protocole suivant :

<u>*MIX*</u> :

$$\begin{array}{llcll} 1. & A_1 & \rightarrow & MIX & : \{\langle \{M_1\}_{k^{+1}(B_{j_1})}, B_{j_1}\rangle\}_{k^{+1}(MIX)} \\ 2. & A_2 & \rightarrow & MIX & : \{\langle \{M_2\}_{k^{+1}(B_{j_2})}, B_{j_2}\rangle\}_{k^{+1}(MIX)} \\ 3. & A_3 & \rightarrow & MIX & : \{\langle \{M_3\}_{k^{+1}(B_{j_3})}, B_{j_3}\rangle\}_{k^{+1}(MIX)} \\ 4. & MIX & \rightarrow & B_{j_1} & : \langle \{M_1\}_{k^{+1}(B_{j_1})}, B_{j_1}\rangle \\ 5. & MIX & \rightarrow & B_{j_2} & : \langle \{M_2\}_{k^{+1}(B_{j_2})}, B_{j_2}\rangle \\ 6. & MIX & \rightarrow & B_{j_3} & : \langle \{M_3\}_{k^{+1}(B_{j_3})}, B_{j_3}\rangle \end{array}$$

Comme c'est l'intermédiaire *MIX* qui assure la redistribution des messages, il n'est *a priori* pas possible de mettre en relation chaque A_i avec le B_{j_i} qui correspond. Expliquons informellement quelles sont les propriétés auxquelles on s'intéresse dans cette partie au moyen de quelques exemples, extraits de [PK00] (nous utiliserons le terme générique opacité pour désigner ces propriétés) :

- *(anonymity)* on dit qu'une personne est anonyme dans un ensemble $\{A_1, \dots, A_n\}$ s'il n'est pas possible de distinguer son identité parmi $A_1, \dots, A_n$;
- *(unlinkability)* on dit que deux objets (messages, actions,...) sont indépendants si ces deux objets n'apparaissent pas plus reliés dans le système considéré qu'ils ne le seraient en toute généralité ;
- *(unobservability)* un objet est non observable s'il n'est pas possible de le distinguer d'un autre du même type.

On pourra trouver dans [PK00] d'autres types de propriétés, ainsi que les liens qui existent entre-elles.

14.2 Approches existantes

On peut distinguer essentiellement deux types d'approches concernant ce type de problèmes. La première est l'approche logique qui consiste à mettre en place une logique de la connaissance adaptée à la formalisation du concept d'opacité et à faire ensuite des preuves dans cette logique [SS99]. L'inconvénient majeur de ces approches est que les preuves de sécurité ne peuvent, en général, pas être construites automatiquement. Par contre, la définition d'une telle logique et la représentation des concepts d'opacité peut se faire indépendamment du modèle choisi. Ceci permet de donner des définitions convaincantes des différentes propriétés d'opacité. Au contraire, dans [SS96], les propriétés d'opacité sont définies à l'intérieur de CSP qui est un modèle classique pour analyser la sécurité des protocoles de communication [Sch96]. L'avantage de cette approche est que, dans le cas de processus finis, ces propriétés peuvent être décidées au moyen d'outils de *model-checking* adaptés. Notons que, dans cette dernière approche, la définition des propriétés d'opacité fait intervenir une certaine notion d'équivalence entre processus (l'équivalence de traces). Finalement, D. Hughes et V. Shmatikov ont proposé dans [HS03] un cadre relativement général pour formaliser les propriétés d'opacité, mais sans algorithme de décision. Nous allons voir ceci plus en détail dans la section suivante, puisque c'est cette approche que nous avons choisi de suivre, en étudiant comment vérifier automatiquement les propriétés décrites dans ce formalisme.

14.3 Notre approche

Replaçons-nous dans le cas du dîner des cryptographes, juste avant qu'ils exécutent le protocole leur permettant de savoir si l'un d'entre eux a payé ou pas, et imaginons un espion, situé non loin de là. Lorsque le protocole se déroule, l'espion voie les cryptographes lancer chacun une pièce et montrer le résultat à son voisin de droite (nous supposerons que le protocole est suffisamment bien exécuté pour que le résultat des tirages soit invisible). Il entend ensuite chaque cryptographe énoncer « les tirages sont les mêmes ! » ou « les tirages sont différents ! » En regardant la parité du nombre de tirages annoncés comme différents, l'espion peut, comme nous l'avons expliqué, savoir si l'un des cryptographes ment, autrement dit, savoir si l'un des cryptographes a payé le repas. Nous aimerions montrer qu'il ne peut, par contre, pas savoir de qui il s'agit. Une manière équivalente d'énoncer ceci consiste à dire que l'identité du payeur ne doit pas être déductible de l'exécution du protocole que l'espion vient d'observer. Nous dirons que le payeur est anonyme, dans cette exécution et du point de vue de l'espion, s'il existe une autre exécution du protocole, éventuellement avec des tirages aléatoires différents, dans laquelle le payeur n'est pas le même, mais identique à la première exécution, du point de vue de l'espion.

Si on cherche à formaliser ce qui vient d'être dit, on distingue trois sortes d'objets. Tout d'abord, il y a les différentes exécutions possibles du protocole. En effet, bien que l'on cherche à montrer l'opacité de quelque chose dans une exécution, on vient de voir que la définition même d'opacité fait intervenir les autres exécutions. Nous avons aussi parlé d'exécutions indistinguables, du point de vue de l'espion. Ceci sera formalisé au moyen d'une relation d'équivalence sur l'ensemble des exécutions, que nous appellerons équivalence observationnelle. Le troisième ingrédient est ce dont on veut préserver l'opacité (ici l'identité du payeur) : nous le modéliserons au moyen d'une fonction (appelée attribut) de l'ensemble des exécutions vers un ensemble adéquat. Nous dirons alors que la valeur de cet attribut dans une session donnée est opaque s'il existe une session équivalente dans laquelle la valeur de l'attribut est différente.

Ceci serait suffisant pour une théorie « basique » de l'opacité, mais il y a des cas où on veut montrer plus que l'opacité de la valeur de l'attribut. Par exemple, dans le cadre théorique proposé dans [HS03], la valeur de l'attribut est elle-même une fonction dont on veut montrer que la valeur en un point, le contenu de l'image, etc. sont opaques. Pour prendre en compte ces notions dans notre approche, nous ajouterons aux trois ingrédients précédents une correspondance de Galois, qui permettra de représenter la connaissance de la valeur de l'attribut que possède l'observateur. Pour montrer que l'objet correspondant à une exécution donnée est opaque pour l'observateur, on considérera alors la connaissance la plus précise que peut en avoir l'observateur, compte-tenu du fait que cette connaissance doit être compatible avec toutes les valeurs que peut prendre l'objet en question pour les exécutions qui sont équivalentes à l'exécution donnée. La propriété d'opacité sera alors satisfaite si la connaissance de l'intrus est suffisamment imprécise.

Notre contribution consiste donc, d'une part, à généraliser l'approche de [HS03] en fournissant un cadre plus large dans lequel choisir les objets qui doivent rester opaques ainsi que la connaissance de l'observateur et, d'autre part, à instancier tout ceci en représentant les exécutions du protocole et l'équivalence observationnelle au moyen d'un sous-ensemble du spi-calcul, le spi-calcul fini, et d'une notion d'équivalence décidable [Hüt02]. Nous montrerons comment utiliser l'algorithme de décision de cette notion d'équivalence afin de vérifier automatiquement que des propriétés d'opacité sont satisfaites. Comme dans les parties précédentes, cette méthode sera sûre, mais pas nécessairement complète.

Les processus du spi-calcul fini sont définis au moyen de la grammaire suivante :

$$
\begin{array}{rcl}
P, Q, \ldots \in \mathrm{Proc}_{\mathrm{Fin}} & ::= & \overline{n}\langle m\rangle.P \\
& | & n(x).P \\
& | & (\nu n).P \\
& | & P \parallel Q \\
& | & P + Q \\
& | & \text{case } m \text{ of } \langle x, y\rangle.P \\
& | & \text{case } m \text{ of } \{x\}_n.P \\
& | & [m = m'].P \\
& | & 0
\end{array}
$$

Nous n'avons donc, cette fois-ci, pas fait de restriction par rapport à l'ensemble Proc de processus défini dans le chapitre 1. Par contre, pour simplifier, et comme c'est l'usage en spi-calcul, nous nous limiterons à l'utilisation de clés symétriques (en fait, nous considérerons uniquement les messages chiffrés avec des éléments de Name $\subseteq$ Key). Nous n'utiliserons pas non plus le constructeur de hachage, ce qui fait que, dans cette partie, les messages considérés seront définis par la grammaire suivante :

$$
\begin{array}{rcl}
m, m', \ldots \in \mathrm{Msg} & ::= & n \\
& | & \langle m, m'\rangle \\
& | & \{m\}_n
\end{array}
$$

Le message $\{0\}_0$ sera noté 1.

Exemple : Voyons comment représenter dans ce langage le dîner des cryptographes (page 157). Dans ce protocole, le comportement du premier participant est particulier, puisqu'il communique le résultat du tirage qu'il a effectué à son voisin de droite avant que son voisin de gauche lui ait communiqué le sien. On pose donc :

$$\begin{array}{rcl} P_{A_1} & \hat{=} & \overline{c}\langle\{Q_1\}_{\mathrm{k}(A_1,A_2)}\rangle. \\ & & c(x_1).\text{case } x_1 \text{ of } \{q_k\}_{\mathrm{k}(A_k,A_1)}. \\ & & \quad [P_1=0].[q_k=0].[Q_1=0].\overline{c}\langle 0\rangle.0 \\ & & + [P_1=0].[q_k=0].[Q_1=1].\overline{c}\langle 1\rangle.0 \\ & & + [P_1=0].[q_k=1].[Q_1=0].\overline{c}\langle 1\rangle.0 \\ & & + [P_1=0].[q_k=1].[Q_1=1].\overline{c}\langle 0\rangle.0 \\ & & + [P_1=1].[q_k=0].[Q_1=0].\overline{c}\langle 1\rangle.0 \\ & & + [P_1=1].[q_k=0].[Q_1=1].\overline{c}\langle 0\rangle.0 \\ & & + [P_1=1].[q_k=1].[Q_1=0].\overline{c}\langle 0\rangle.0 \\ & & + [P_1=1].[q_k=1].[Q_1=1].\overline{c}\langle 1\rangle.0 \end{array}$$

(remarquons que l'opérateur « ou exclusif » a été ici encodé à l'aide de tests). Pour les autres participants (*i.e.* $i \in \{2, \ldots, k\}$, dans le cas où il y a k participants) on pose :

$$\begin{array}{rcl} P_{A_i} & \hat{=} & c(x_i).\text{case } x_i \text{ of } \{q_{i-1}\}_{\mathrm{k}(A_{i-1},A_i)}. \\ & & \overline{c}\langle\{Q_i\}_{\mathrm{k}(A_i,A_{i+1})}\rangle. \\ & & \quad [P_i=0].[q_{i-1}=0].[Q_i=0].\overline{c}\langle 0\rangle.0 \\ & & + [P_i=0].[q_{i-1}=0].[Q_i=1].\overline{c}\langle 1\rangle.0 \\ & & + [P_i=0].[q_{i-1}=1].[Q_i=0].\overline{c}\langle 1\rangle.0 \\ & & + [P_i=0].[q_{i-1}=1].[Q_i=1].\overline{c}\langle 0\rangle.0 \\ & & + [P_i=1].[q_{i-1}=0].[Q_i=0].\overline{c}\langle 1\rangle.0 \\ & & + [P_i=1].[q_{i-1}=0].[Q_i=1].\overline{c}\langle 0\rangle.0 \\ & & + [P_i=1].[q_{i-1}=1].[Q_i=0].\overline{c}\langle 0\rangle.0 \\ & & + [P_i=1].[q_{i-1}=1].[Q_i=1].\overline{c}\langle 1\rangle.0 \end{array}$$

et le processus complet est alors :

$$P \quad \hat{=} \quad P_{A_1} \parallel \ldots \parallel P_{A_k}$$

Pour obtenir une exécution possible de ce protocole on doit tout d'abord instancier P avec les identités $a_1, \ldots, a_k$ des participants, les tirages $q_1, \ldots, q_k \in \{0, 1\}$ qu'ils sont censés effectuer et le choix d'un participant distingué $p_1, \ldots, p_k \in \{0, 1\}$ (parmi lesquels au plus un est égal à 1). On obtient alors :

$$P((a_1, \ldots, a_k), (q_1, \ldots, q_k), (p_1, \ldots, p_k)) \;\hat{=}\; P[a_1/A_1, \ldots, a_k/A_k, q_1/Q_1, \ldots, q_k/Q_k, p_1/P_1, \ldots, p_k/P_k]$$

L'anonymat du participant distingué a_i sera formalisé en énonçant qu'il doit exister des instanciations de ce protocole équivalentes à celle-ci dans lesquelles ce n'est pas a_i qui est distingué. □

Le chapitre qui vient est consacré à la description de cette modélisation et en particulier à la définition de l'équivalence observationnelle considérée. Il existe beaucoup de notions d'équivalence observationnelles dans le spi-calcul [BN02] et nous consacrerons le chapitre suivant à en détailler certaines ainsi que les liens qui existent entre-elles. Nous obtiendrons de cette manière une notion d'équivalence, plus précise que la notion de départ (qui nous fournira donc une approximation sûre), mais décidable. Nous montrerons ensuite comment utiliser l'algorithme de décision pour qu'il retourne des instances du protocole, équivalentes à une instance donnée, qui pourront être utilisées afin de vérifier la propriété d'opacité considérée. Nous terminerons en expliquant comment ceci se comporte en pratique, en particulier en décrivant quelles parties de ce travail ont été implémentées.

Chapitre 15

Modélisation

L'exemple du chapitre précédent a mis en évidence un protocole générique, pouvant être instancié de diverses manières au moyen d'une famille de paramètres. C'est sur ces paramètres que l'on cherche à prouver des propriétés d'opacité. Nous sommes aidés dans cette tâche par le fait qu'un observateur, voyant se dérouler une session du protocole, n'a pas accès aux paramètres qui permettent, par instanciation, d'obtenir cette session à partir du protocole générique. Formellement, on dispose donc d'un ensemble W de paramètres, ainsi que d'une fonction P qui, à un paramètre $w \in W$, associe un processus clos $P(w)$. Ce processus $P(w)$ représente l'exécution du protocole instancié par le paramètre w, et c'est à ce processus (et pas à w) qu'a accès l'observateur extérieur. La première chose que nous allons faire est donc de définir une relation d'équivalence entre processus. Précisément, il s'agira de l'équivalence par tests (*testing equivalence*), proposée par R. De Nicola et M. Hennessy [NH84] dans le cadre du π-calcul, puis adaptée au spi-calcul par M. Abadi et A. Gordon [AG99]. Deux processus en relation seront alors dits indistinguables (pour l'observateur extérieur). Au moyen de cette relation d'équivalence, nous pourrons affirmer que, quel que soit ce que l'observateur peut connaître lorsqu'il observe une session $P(w)$, cette connaissance (notons-la $k(w)$) doit être compatible avec ce qu'il observerait du processus $P(w')$ (avec $w' \in W$), pourvu que $P(w)$ et $P(w')$ soient équivalents. Pour modéliser la connaissance de l'observateur, nous aurons donc besoin d'une fonction $k : W \to K$ ainsi que d'une notion de compatibilité entre W et K et nous devrons postuler que, si $P(w)$ et $P(w')$ sont équivalents alors w' est compatible avec $k(w)$ (autrement dit, nous devrons postuler que la relation d'équivalence sur W induite par k contient la relation d'équivalence induite par l'indistinguabilité des processus). Cette approche correspond à celle D. Hugues et V. Shmatikov [HS03], adaptée pour donner un rôle central aux correspondances de Galois et à l'interprétation abstraite [CC92a, CC92b]. Ainsi, et à la différence des parties précédentes, l'interprétation abstraite n'est plus seulement utilisée dans le but de simplifier le modèle, mais directement dans le modèle lui-même. Nous expliquerons ensuite comment représenter une propriété d'opacité dans ce modèle et comment en obtenir une approximation correcte (au moyen d'une abstraction simple).

15.1 L'équivalence par tests du spi-calcul

M. Abadi et A. Gordon ont introduit l'équivalence par tests dans le cadre du spi-calcul. Notons qu'il existe différentes notions d'équivalence observationnelle dans ce langage [BN02], mais toutes constituent des approximations sûres de l'équivalence par tests (et, vu l'approche que nous avons choisie, remplacer l'équivalence par tests par une relation plus stricte serait tout à fait correct). Nous définissons ici l'équivalence par tests pour les processus finis en calquant la définition donnée dans [AG99] pour le spi-calcul. Pour cela, la première chose à faire est de définir une relation de transition (*commitment relation*) sur les processus finis. Cette construction comporte elle-même plusieurs étapes, la première étant de définir la relation de réduction en un pas (*reduction relation*).

Définition – 15.1.1

> *Soient $P, Q \in \mathrm{Proc_{Fin}}$. On dit que P se réduit sur Q en un pas, et on note $P \triangleright Q$ si on est dans un des cas suivants :*
> - $P = \mathrm{case}\ \langle m_1, m_2 \rangle\ \mathrm{of}\ \langle x, y \rangle.P'$ *et* $Q = P'[m_1/x, m_2/y]$;
> - $P = \mathrm{case}\ \{m\}_n\ \mathrm{of}\ \{x\}_n.P'$ *et* $Q = P'[m/x]$;
> - $P = [m = m].P'$ *et* $Q = P'$.

Cette relation représente une étape purement calculatoire. Les autres évènements qui peuvent se produire sont des envois et réceptions de messages. Pour les prendre en compte, nous définissons deux nouveaux types d'objets : les abstractions (qui sont, intuitivement, des processus en attente de réception d'un message) et les concrétions (qui représentent, au contraire, des processus ayant émis un message). Formellement, une abstraction s'écrira $(x).P$ (où x est une variable et P un processus fini) et se transformera, à la réception d'un message m, en $P[m/x]$. Une concrétion sera de son coté notée $(\nu n_1, \ldots, n_k)\langle m \rangle.P$ et provient d'un processus qui a envoyé le message m (dans lequel $n_1, \ldots, n_k$ désignent de nouveaux noms) et qui se comporte maintenant comme le processus fini P. Un agent désignera soit un processus, soit une abstraction, soit une concrétion. Pour pouvoir manipuler plus simplement les agents, nous étendrons à leur ensemble les notions de composition en parallèle et de création de nouveaux noms. Tout ceci est contenu dans la définition suivante.

Définition – 15.1.2

> *Les ensembles* Abstr, Concr *et* Agent *(respectivement ensemble des abstractions, des concrétions et des agents) sont définis au moyen des grammaires suivantes :*
>
> $$\begin{array}{rcll} F, F', \ldots \in \mathrm{Abstr} & ::= & (x).P & (P \in \mathrm{Proc_{Fin}}) \\ C, C', \ldots \in \mathrm{Concr} & ::= & (\nu n_1, \ldots, n_k)\langle m \rangle.P & (P \in \mathrm{Proc_{Fin}}) \\ A, A', \ldots \in \mathrm{Agent} & ::= & P & (P \in \mathrm{Proc_{Fin}}) \\ & | & F & \\ & | & C & \end{array}$$

Afin d'alléger les notations (et conformément à la littérature), on utilise les notations suivantes :
- le k-uplet $n_1, \ldots, n_k$ est noté $\vec{n}$ et on pose $|\vec{n}| \mathrel{\hat{=}} k$;
- si $\vec{n} = (n_1, \ldots, n_k)$ et $\vec{n}' = (n'_1, \ldots, n'_{k'})$, alors $\vec{n}, \vec{n}'$ désigne le $(k + k')$-uplet $(n_1, \ldots, n_k, n'_1, \ldots, n'_{k'})$;
- si $\vec{n} = (n_1, \ldots, n_k)$, alors $\{\vec{n}\}$ désigne l'ensemble $\{n_1, \ldots, n_k\}$.

La concrétion $C = (\nu n_1, \ldots, n_k)\langle m \rangle.Q$ peut alors être notée $(\nu \vec{n})\langle m \rangle.Q$ avec $\vec{n} = (n_1, \ldots, n_k)$. Signalons que le cas $k = 0$ est tout à fait possible, on note alors $C = (\nu)\langle m \rangle.Q$.

Définition – 15.1.3

Soient $F = (x).P \in \mathrm{Abstr}$, $C = (\nu\vec{n})\langle m\rangle.Q \in \mathrm{Concr}$, $n' \in \mathrm{Name}$ *et* $R \in \mathrm{Proc}_{\mathrm{Fin}}$ *et supposons :*

$$x \notin \mathrm{fv}(R), n' \notin \{\vec{n}\}, \{\vec{n}\} \cap \mathrm{fn}(R) = \varnothing$$

(on peut toujours se ramener à ce cas par α*-renommage,* x *étant liée dans* $(x).P$ *et chaque* $n_i \in \{\vec{n}\}$ *étant lié dans* $(\nu\vec{n})\langle m\rangle.Q$*). On pose alors :*

$$\begin{array}{rcccl}
(\nu n').F & = & (\nu n').((x).P) & \hat{=} & (x).((\nu n').P) \\
R \parallel F & = & R \parallel ((x).P) & \hat{=} & (x).(R \parallel P) \\
F \parallel R & = & ((x).P) \parallel R & \hat{=} & (x).(P \parallel R) \\
(\nu n').C & = & (\nu n').((\nu\vec{n})\langle m\rangle.Q) & \hat{=} & \left[\begin{array}{l} (\nu n', \vec{n})\langle m\rangle.Q \text{ si } n' \in \mathrm{fn}(m) \\ (\nu\vec{n})\langle m\rangle.((\nu n').Q) \text{ sinon} \end{array}\right. \\
R \parallel C & = & R \parallel ((\nu\vec{n})\langle m\rangle.Q) & \hat{=} & (\nu\vec{n})\langle m\rangle.(R \parallel Q) \\
C \parallel R & = & ((\nu\vec{n})\langle m\rangle.Q) \parallel R & \hat{=} & (\nu\vec{n})\langle m\rangle.(Q \parallel R)
\end{array}$$

Dans le cas où $\{\vec{n}\} \cap \mathrm{fn}(P) = \varnothing$*, on définit également deux processus, notés* $F @ C$ *et* $C @ F$ *(et appelés interactions de* F *et* C *et de* C *et* F*, respectivement) par :*

$$\begin{array}{rcccl}
F @ C & = & (x).P @ (\nu\vec{n})\langle m\rangle.Q & \hat{=} & (\nu\vec{n}).(P[m/x] \parallel Q) \\
C @ F & = & (\nu\vec{n})\langle m\rangle.Q @ (x).P & \hat{=} & (\nu\vec{n}).(Q \parallel P[m/x])
\end{array}$$

Un crochet (*barb*) sera un nom n ou son dual $\bar{n}$ et une action α sera soit un crochet, soit l'action silencieuse notée τ. On notera Act l'ensemble des actions. Nous pouvons maintenant définir la relation de transition, notée $\rightarrow$, qui est une relation entre processus finis et agents, étiquetée par des actions.

Définition – 15.1.4

La relation de transition est définie au moyen des règles suivantes :

$$\frac{}{n(x).P \xrightarrow{n} (x).P} \qquad \frac{}{\overline{n}\langle m\rangle.P \xrightarrow{\bar{n}} (\nu)\langle m\rangle.P}$$

$$\frac{P \xrightarrow{n} F \quad Q \xrightarrow{\bar{n}} C}{P \parallel Q \xrightarrow{\tau} F @ C} \qquad \frac{P \xrightarrow{\bar{n}} C \quad Q \xrightarrow{n} F}{P \parallel Q \xrightarrow{\tau} C @ F}$$

$$\frac{P \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A} \qquad \frac{P \xrightarrow{\alpha} A}{Q + P \xrightarrow{\alpha} A} \qquad \frac{P \xrightarrow{\alpha} A}{P \parallel Q \xrightarrow{\alpha} A \parallel Q} \qquad \frac{P \xrightarrow{\alpha} A}{Q \parallel P \xrightarrow{\alpha} Q \parallel A}$$

$$\frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{n, \bar{n}\}}{(\nu n).P \xrightarrow{\alpha} (\nu n).A} \qquad \frac{P \rhd Q \quad Q \xrightarrow{\alpha} A}{P \xrightarrow{\alpha} A}$$

P étant un processus fini et A un agent, on notera $P \xrightarrow{\tau^*\beta} A$ si, et seulement si, P se transforme en un processus Q par application d'un nombre fini (éventuellement nul) de transitions silencieuses et $Q \xrightarrow{\beta} A$. Un tel crochet β représente un test que l'on peut effectuer sur chaque processus fini P, à savoir : le processus P peut-il (après avoir, éventuellement, réalisé un nombre fini d'actions silencieuses) réaliser une action étiquetée par β ? Deux processus seront équivalents si un observateur extérieur interagissant avec eux (représenté par le même processus fini composé en parallèle avec chacun d'eux) ne parvient à discriminer ces deux processus sur aucun crochet β.

DÉFINITION – 15.1.5

> *Soit β un crochet. On dit que deux processus finis P et Q sont équivalents pour le test β si, pour tout processus fini R et tout agent A tels que $P \parallel R \xrightarrow{\tau^*\beta} A$, il existe un agent B tel que $Q \parallel R \xrightarrow{\tau^*\beta} B$, et réciproquement. On dit que deux processus finis P et Q sont équivalents par tests, et on note $P \underset{\text{tst}}{\sim} Q$, si, pour tout crochet β, P et Q sont équivalents pour le test β.*

Muni de cette notion d'équivalence sur les processus, nous pouvons maintenant décrire l'espace des sessions d'un protocole cryptographique donné.

15.2 Les systèmes observables

L'ensemble des sessions d'un protocole s'inscrira dans le cadre de la définition suivante.

DÉFINITION – 15.2.1

> *Un système observable $\mathcal{W}$ est un triplet $(W, \sim, c)$ où W est un ensemble (dont les éléments sont appelés des mondes), $\sim$ est une relation d'équivalence sur W et c est une fonction de W vers un ensemble C (c est appelé un attribut de W).*

Exemple : Dans le chapitre précédent, le protocole cryptographique correspondant au dîner des cryptographes avec k participants (page 157) a été modélisé par un processus fini P dont les variables libres sont $A_1, \ldots, A_k, Q_1, \ldots, Q_k, P_1, \ldots, P_k$. Une session de ce protocole est paramétrée par un élément de l'ensemble :

$$W \quad \hat{=} \quad \{((a_1, \ldots, a_k), (q_1, \ldots, q_k), (p_1 \ldots, q_k)) \in \text{Name}^k \times \{0,1\}^{2k} \mid p_1 + \ldots + p_k \leq 1\}$$

et, si pour $w \in W$ on note $P(w)$ le processus P instancié avec les valeurs contenues dans w, on dira que w et w' sont équivalents si $P(w) \underset{\text{tst}}{\sim} P(w')$.On notera alors $w \sim w'$. □

Cet exemple met en valeur le fait que, pour nous et dans la suite, les ensembles de mondes que nous considérerons seront toujours décrits au moyen d'un ensemble W sur lequel est définie une fonction $P : W \to \text{Proc}_{\text{Fin}}$. Ceci nous permettra de définir une relation d'équivalence $\sim$ sur W en posant $w \sim w'$ si, et seulement si, $P(w) \underset{\text{tst}}{\sim} P(w')$. Plus précisément encore, cette fonction P sera définie au moyen d'un processus $P \in \text{Proc}_{\text{Fin}}$ avec des variables libres $x_1, \ldots, x_n$ et un élément $w \in W$ sera un n-uplet $v_1, \ldots, v_n$, $P(w)$ étant obtenu en instanciant, dans P, chaque x_i par v_i. Les attributs, quant à eux, nous permettront de nous intéresser, non pas à l'opacité des valeurs contenues dans w au travers d'une exécution $P(w)$, mais plutôt à l'opacité de certains objets, calculés à partir de w. Ces attributs nous autorisent ainsi à décrire précisément ce dont on désire préserver l'opacité dans une exécution du protocole.

Exemple : Considérons à nouveau le protocole page 157. L'ensemble des participants pourra être connu d'un observateur extérieur, mais on ne veut pas qu'il sache qui est le participant distingué, dans le cas où il y en a un. Il faudra donc prouver l'opacité de l'attribut qui, à chaque paramètre $w \in W$, associe l'ensemble des participants distingués dans la session correspondante du protocole :

$$\begin{array}{rrcl} c_1 : & W & \rightarrow & 2^{\text{Name}} \\ & w & \mapsto & \{a_i \mid 1 \leq i \leq k \text{ et } p_i = 1\} \end{array}$$

(où chaque $w \in W$ est noté sous la forme $w = ((a_1, \ldots, a_k), (q_1, \ldots, q_k), (p_1, \ldots, p_k)))$. Cette connaissance sera bien entendu partielle. En effet, un des buts du protocole est de s'assurer qu'il y a exactement un participant distingué. Ainsi un observateur extérieur saura si $c_1(w)$ est vide ou pas. Pour montrer l'opacité de c_1 dans une session donnée, nous montrerons donc qu'il existe une autre session w' telle que $w \sim w'$ et telle que $c_1(w) \neq c_1(w')$. L'ensemble des tirages aléatoires, représenté par l'attribut :

$$\begin{array}{rrcl} c_2 : & W & \rightarrow & \{0,1\}^k \\ & w & \mapsto & (q_1, \ldots, q_k) \end{array}$$

sera également en partie inconnu. □

Le dernier ingrédient à faire entrer en jeu dans notre modèle est la description de l'observateur.

15.3 L'observateur et les propriétés d'opacité

Considérons un système observable $\mathcal{W} = (W, \sim, c)$. En un monde $w \in W$, notons $\alpha(c(w))$ ce qu'un observateur peut voir de $c(w)$. Comme on l'a fait remarquer auparavant, cette quantité $\alpha(c(w))$ doit, en un certain sens, être compatible avec chaque $c(w')$ lorsque $w' \sim w$. Si nous notons A l'ensemble dans lequel la connaissance de l'observateur prend ses valeurs, cette notion de compatibilité peut être formalisée au moyen d'une relation binaire $\sqsubseteq$ entre C et A. La fonction $\alpha : C \rightarrow A$ devra donc satisfaire la condition suivante :

$$\forall w, w' \in W.\ w \sim w' \ \Rightarrow\ c(w') \sqsubseteq \alpha(c(w))$$

Il est également raisonnable de supposer que l'ensemble des connaissances possibles de l'observateur est ordonné au moyen d'une relation d'ordre $\sqsubseteq$ traduisant la notion de précision (si $a, a' \in K$ et $a \sqsubseteq a'$, on dira que a est plus précis que a'). La première hypothèse que nous ferons est la suivante :

$$\forall c \in C.\ \forall a, a' \in A.\ (c \sqsubseteq a \text{ et } a \sqsubseteq a') \ \Rightarrow\ c \sqsubseteq a' \tag{1}$$

qui traduit le fait que si on remplace une quantité compatible avec une valeur de l'attribut par une quantité moins précise, on obtient encore une quantité compatible. Afin de nous placer dans l'hypothèse la plus sûre possible, nous aimerions pouvoir choisir l'élément de A le plus petit possible, tout en étant compatible avec chaque élément d'un sous-ensemble de C donné. Nous supposerons pour cela que tout sous-ensemble de A possède une borne inférieure (ce qui est toujours possible, quitte à plonger A dans un ensemble ordonné plus grand) et on posera alors pour $w \in W$:

$$\begin{array}{rcl} \mathcal{A}_{c(w)} & \hat{=} & \{a \in A \mid \forall w' \in W.\ w \sim w' \ \Rightarrow\ c(w') \sqsubseteq a\} \\ \alpha(c(w)) & \hat{=} & \sqcap \mathcal{A}_{c(w)} \end{array}$$

$\mathcal{A}_{c(w)}$ représente les éléments de A qui sont compatibles avec chaque $c(w')$ pour $w' \in W$ tel que $w \sim w'$. Si $\sqcap$ désigne la borne inférieure dans A, $\alpha(c(w))$ est alors la connaissance la plus précise qui soit compatible avec chaque $c(w')$ pour $w' \sim w$. Pour que cette définition soit correcte, nous devons supposer que :

$$\forall c \in C.\ \forall \mathcal{A} \subseteq A.\ (\forall a \in \mathcal{A}.\ c \sqsubseteq a) \ \Rightarrow\ c \sqsubseteq \sqcap \mathcal{A} \tag{2}$$

cette hypothèse traduisant la continuité de la relation de compatibilité par rapport à la relation de précision. Nous utiliserons toujours par la suite des observateurs décrits de cette manière. Nous allons toutefois généraliser un peu cette construction. Définissons pour cela trois applications :

$$\begin{array}{rclcl}
\overline{c} & : & W & \to & 2^C \\
 & & w & \mapsto & \{c(w') \mid w' \in W \text{ et } w \sim w'\} \\
\sharp & : & 2^C & \to & A \\
 & & \mathcal{C} & \mapsto & \mathcal{C}^\sharp \mathrel{\hat{=}} \sqcap\{a \in A \mid \forall c \in \mathcal{C}.\ c \sqsubseteq a\} \\
\flat & : & A & \to & 2^C \\
 & & a & \mapsto & a^\flat \mathrel{\hat{=}} \{c \in C \mid c \sqsubseteq a\}
\end{array}$$

Avec ces notations, on a, tout d'abord, pour chaque $w \in W$:

$$\alpha(c(w)) \quad = \quad (\overline{c}(w))^\sharp$$

On a également la proposition suivante.

Proposition – 15.3.1

> *$\sharp$ et $\flat$ induisent une correspondance de Galois entre 2^C (muni de l'inclusion) et A (muni de la relation de précision $\sqsubseteq$).*

Démonstration :

Donnons une démonstration détaillée de ce résultat, afin de mieux appréhender les relations entre C, A, $\sharp$ et $\flat$. Suivant [CC92a], il suffit de montrer que :

- $\sharp$ et $\flat$ sont monotones (croissantes) ;
- quel que soit $\mathcal{C} \subseteq C$, on a $\mathcal{C} \subseteq (\mathcal{C}^\sharp)^\flat$;
- quel que soit $a \in A$, on a $(a^\flat)^\sharp \sqsubseteq a$.

Pour le premier point, considérons $\mathcal{C} \subseteq \mathcal{C}' \subseteq C$ et supposons que $a \in A$ est tel que, quel que soit $c \in \mathcal{C}'$, on a $c \sqsubseteq a$. Il s'ensuit que, quel que soit $c \in \mathcal{C}$, on a $c \sqsubseteq a$. Par conséquent :

$$\mathcal{A}' \ \hat{=}\ \{a \in A \mid \forall c \in \mathcal{C}'.\ c \sqsubseteq a\} \subseteq \{a \in A \mid \forall c \in \mathcal{C}.\ c \sqsubseteq a\} \ \hat{=}\ \mathcal{A}$$

et donc que :

$$\mathcal{C}^\sharp = \sqcap \mathcal{A} \sqsubseteq \sqcap \mathcal{A}' = \mathcal{C}'^\sharp$$

Supposons maintenant que $a, a' \in A$ sont tels que $a \sqsubseteq a'$. Pour $c \in C$ tel que $c \sqsubseteq a$, on a alors $c \sqsubseteq a'$ (d'après (1)), donc :

$$a^\flat \ = \ \{c \in C \mid c \sqsubseteq a\} \subseteq \{c \in C \mid c \sqsubseteq a'\} \ = \ a'^\flat$$

On en déduit que $\sharp$ et $\flat$ sont croissantes.

Soit maintenant $\mathcal{C} \subseteq C$ et posons $a = \mathcal{C}^\sharp$. On veut montrer que $\mathcal{C} \subseteq a^\flat$. Soit donc $c \in \mathcal{C}$, on sait (d'après (2)) que $c \sqsubseteq a$ et par conséquent $c \in a^\flat$, d'où le résultat.

Considérons finalement $a \in A$ et posons $\mathcal{C} = a^\flat$. On veut montrer que $\mathcal{C}^\sharp \sqsubseteq a$. Il suffit pour cela de montrer que, quel que soit $c \in \mathcal{C}$, on a $c \sqsubseteq a$, or c'est évident vu la définition de $\mathcal{C}$. ■

Une telle correspondance de Galois sera notée $(2^C, \subseteq) \underset{\flat}{\overset{\sharp}{\rightleftharpoons}} (A, \sqsubseteq)$. Nous allons l'utiliser pour définir la notion d'observateur sur un système observable.

Définition – 15.3.1

Un système observé est un quadruplet $\mathcal{W} = (W, \sim, c, (2^C, \subseteq) \underset{\flat}{\overset{\sharp}{\rightleftharpoons}} (A, \sqsubseteq))$ où $(W, \sim, c)$ est un système observable (l'attribut c étant une fonction de W dans C) et $(2^C, \subseteq) \underset{\flat}{\overset{\sharp}{\rightleftharpoons}} (A, \sqsubseteq)$ est une correspondance de Galois.

Une propriété d'opacité sera tout simplement une propriété de la connaissance de l'observateur, autrement dit un sous-ensemble de A. Comme il est naturel de supposer qu'une telle propriété est d'autant plus vraie que la connaissance de l'observateur est moins précise, nous supposerons que les sous-ensembles en question sont clos vers le haut, ce qui nous conduit à la définition suivante.

Définition – 15.3.2

Soit $\mathcal{W} = (W, \sim, c, (2^C, \subseteq) \underset{\flat}{\overset{\sharp}{\rightleftharpoons}} (A, \sqsubseteq))$ un système observé. Une propriété d'opacité sur $\mathcal{W}$ est une partie φ de A telle que, si $a \in \varphi$ et $a \sqsubseteq a'$, alors $a' \in \varphi$. On dit que $\mathcal{W}$ satisfait φ en $w \in W$, et on note $\mathcal{W}, w \models \varphi$, si $\overline{c}(w)^\sharp \in \varphi$. On dit que $\mathcal{W}$ satisfait φ, et on note $\mathcal{W} \models \varphi$, si on a $\mathcal{W}, w \models \varphi$, et ce, quel que soit $w \in W$.

Remarque : Le principal défaut de cette notion est que les propriétés ainsi considérées ne sont pas des objets syntaxiques. Un moyen simple de définir une telle propriété est de choisir un élément $a \in A$ et de lui associer la propriété :

$$\varphi_a \ \hat{=}\ \{a' \in A \mid a \sqsubseteq a'\}$$

mais, la plus part du temps, nous écrirons explicitement les propriétés en question sous forme d'un ensemble. □

Exemple : Reprenons les attributs c_1 et c_2 définis pour le dîner des cryptographes. On avait $c_1 : W \to C_1$ et $c_2 : W \to C_2$ avec :

$$C_1 = 2^{\text{Name}} \quad \text{et} \quad C_2 = \{0,1\}^k$$

Nous posons alors simplement :

$$A_1 = 2^{C_1} \quad \text{et} \quad A_2 = 2^{C_2}$$

munis des relations d'inclusion. Un élément c de C_1 (respectivement C_2) sera dit compatible avec a dans A_1 (respectivement A_2) si $c \in a$. On vérifie sans peine que toutes les hypothèses sont satisfaites. Ces abstractions consistent en fait à considérer que, sur $w \in W$, l'observateur observe $\overline{c}(w)$ sur un attribut c. En ce sens, elles sont les plus précises possibles, et par conséquent les plus sûres. Donnons également deux exemples de propriétés. La première énonce que l'observateur ne sait pas quel participant est distingué :

$$\varphi_1 \ \hat{=}\ \{a \in A_1 \mid \text{card}(a) \geq k\}$$

Une condition nécessaire pour que cette propriété soit vraie est que, dans chaque session, les participants sont deux à deux distincts. Nous chercherons donc à montrer que $\mathcal{W}' \models \varphi_1$ où $\mathcal{W}'$ est obtenu par restriction de $\mathcal{W}$ à l'ensemble :

$$W' \quad \hat{=} \quad \{((a_1,\ldots,a_k),(q_1,\ldots,q_k),(p_1,\ldots,p_k) \in W \mid \mathrm{card}(\{a_1,\ldots,a_k\}) = k\}$$

La seconde propriété énonce que, pour $1 \leq i \leq k$, l'observateur ne peut pas connaître le tirage aléatoire associé au participant a_i. On l'écrit :

$$\varphi_2 \quad \hat{=} \quad \{a \in A_2 \mid \forall i \in \{1,\ldots,k\}.\ \exists (q_1,\ldots,q_k),(q'_1,\ldots,q'_k) \in a.\ q_i \neq q'_i\}$$

$\square$

15.4 Abstractions

Supposons donnés un système observé $\mathcal{W} = (W, \sim, c, 2^C \overset{\sharp}{\underset{\flat}{\rightleftharpoons}} A)$, une propriété d'opacité $\varphi \subseteq A$ et $w \in W$. On cherche à savoir si $\mathcal{W}, w \models \varphi$, autrement dit si $\overline{c}(w)^\sharp \in \varphi$. Il n'est cependant pas envisageable, pour y parvenir, de calculer explicitement $\overline{c}(w)$, ni de calculer $\overline{c}(w)^\sharp$ (en tout cas pas directement). On peut par contre envisager deux types d'abstractions, que nous utiliserons conjointement, dans la suite. La première consiste à utiliser, à la place de la relation d'équivalence $\sim$, une relation plus forte $\dot{\sim} \subseteq \sim$. On pose alors :

$$\dot{c}(w) \quad \hat{=} \quad \{c(w') \mid w' \in W \text{ et } w \mathrel{\dot{\sim}} w'\}$$

On a ainsi $\dot{c}(w) \subseteq \overline{c}(w)$, donc $\dot{c}(w)^\sharp \sqsubseteq \overline{c}(w)^\sharp$, et on en déduit que $\dot{c}(w)^\sharp \in \varphi$ implique $\overline{c}(w)^\sharp \in \varphi$. Autrement dit :

$$\dot{\mathcal{W}}, w \models \varphi \quad \Rightarrow \quad \mathcal{W}, w \models \varphi$$

où on a posé $\dot{\mathcal{W}} \mathrel{\hat{=}} (W, \dot{\sim}, c, 2^C \overset{\sharp}{\underset{\flat}{\rightleftharpoons}} A)$. Nous mettrons en place cette abstraction en utilisant, à la place de l'équivalence par tests, une relation de bisimilarité pour laquelle il existe, dans le cas des processus finis, un algorithme de décision. La seconde abstraction présuppose l'existence, pour chaque $w \in W$, d'une énumération (non nécessairement exhaustive) $w^1, w^2, \ldots$ d'éléments de W équivalents à w. On pose alors, pour $i \geq 0$:

$$a^i(w) \quad \hat{=} \quad \{c(w^1), \ldots, c(w^i)\}^\sharp$$

La suite $(a^i(w))_{i \geq 0}$ est croissante et majorée par $\overline{c}(w)^\sharp$, de sorte que :

$$\exists i \geq 0.\ a^i(w) \in \varphi \quad \Rightarrow \quad \mathcal{W}, w \models \varphi$$

Notons que, même dans le cas où cette énumération est exhaustive, il est tout à fait possible que $\mathcal{W}, w \models \varphi$ sans pour autant qu'il existe $i \geq 0$ tel que $a^i(w) \in \varphi$.[1] En général, on a donc ici une approximation stricte.

Afin d'obtenir une telle énumération $w^1, w^2, \ldots$, nous allons transformer l'algorithme de décision de la relation de bisimilarité en une fonction $f : W \to \mathrm{Cstr}$ (où Cstr est un ensemble de contraintes sur W) telle que, si w' satisfait la contrainte $f(w)$, alors $w \sim w'$. Nous choisirons l'ensemble de contraintes de sorte que, une contrainte étant donnée, il soit possible d'énumérer les éléments qui la satisfont.

[1] En toute généralité, il est possible que la borne inférieure d'un sous-ensemble $\mathcal{A}$ de A soit dans φ sans qu'aucune borne inférieure d'un sous-ensemble fini de $\mathcal{A}$ soit dans φ. Nous reviendrons plus tard sur ce point.

15.5 Un autre exemple : un protocole de vote

Examinons le cas d'un protocole de vote avec k votants $A_1, \ldots, A_k$ et un serveur S centralisant les votes. Chaque votant A_i suit le processus :

$$P_{A_i} \;\hat{=}\; \overline{c}\langle \{V_i\}_{\mathrm{k}(A_i,S)}\rangle.0$$

$\mathrm{k}(A_i, S)$ désigne une clé symétrique partagée entre le votant A_i et le serveur S. V_i désigne le texte du vote de A_i et nous supposerons, pour simplifier, qu'il s'agit d'un booléen. Le serveur S contient un processus chargé de prendre en compte le vote de A_i :

$$\begin{array}{rcl} S_{A_i} & \hat{=} & c(x).\mathrm{case}\ x\ \mathrm{of}\ \{y\}_{\mathrm{k}(A_i,S)}.\overline{c}\langle y\rangle.0 \\ S & \hat{=} & S_{A_1} \parallel \ldots \parallel S_{A_k} \end{array}$$

Le protocole est décrit par le processus P suivant :

$$P \;\hat{=}\; P_{A_1} \parallel \ldots \parallel P_{A_k} \parallel S$$

Une session de ce protocole est obtenue en instanciant $A_1, \ldots, A_k, V_1, \ldots, V_k$ au moyen d'un $2k$-uplet pris dans l'ensemble :

$$W \;\hat{=}\; \mathrm{Name}^k \times \{0,1\}^k$$

Les votes ne sont pas secrets au sens de la cryptographie : le vote de chaque A_i est soit 0, soit 1, chacun de ces messages étant connu de l'intrus. Ce qui doit rester inaccessible à un observateur, c'est la correspondance entre participants et votes. Autrement dit, on veut s'assurer de l'opacité de l'attribut :

$$\begin{array}{rrcl} c: & W & \to & 2^{\mathrm{Name}\times\{0,1\}} \\ & w & \mapsto & \{(a_i, v_i) \mid 1 \leq i \leq k\} \end{array}$$

(en notant chaque $w \in W$ sous la forme $w = ((a_1, \ldots, a_k), (v_1, \ldots, v_k))$). Posons :

$$C \;\hat{=}\; 2^{\mathrm{Name}\times\{0,1\}} \quad \text{et} \quad A \;\hat{=}\; (2^{\{0,1\}})^{\mathrm{Name}}$$

Un élément de A est une fonction qui un $a \in \mathrm{Name}$ associe un sous-ensemble de $\{0,1\}$ qui contient les valeurs possibles pour le vote de a, du point de vue de l'observateur. $c \in C$ est compatible avec $a \in A$ si, quel que soit $(a_i, v_i) \in C$, on a $v_i \in a(a_i)$. On dit que $a \in A$ est plus précis que a' si, quel que soit a_i, on a $a(a_i) \subseteq a'(a_i)$. La propriété d'opacité que l'on désire vérifier est alors la suivante :

$$\varphi \;\hat{=}\; \{a \in A \mid \forall a_i.\ a(a_i) = \varnothing \text{ ou } a(a_i) = \{0,1\}\}$$

autrement dit, un observateur peut savoir si un participant a voté ou pas, mais pas ce qu'il a voté.

Notons que cette propriété d'opacité ne pourrait pas être satisfaite si chaque votant, au lieu d'utiliser la clé symétrique qu'il partage avec le serveur, utilisait la clé publique du serveur, autrement dit si le processus décrivant le comportement de A_i était donné par :

$$P_{A_i} \;\hat{=}\; \overline{c}\langle \{V_i\}_{\mathrm{k}^{+1}(S)}\rangle.0$$

En effet, pour savoir si A_i a voté 0 ou 1, il suffirait à l'observateur de chiffrer ces deux valeurs avec la clé publique de S et de comparer le résultat obtenu avec le message transitant entre A_i et S. Lorsque l'on désire utiliser des clés publiques pour réaliser un protocole de vote, on a coutume de combiner chaque vote avec un nonce, de la manière suivante :

$$P_{A_i} \quad \hat{=} \quad (\nu n).\overline{c}\langle\{\langle n, V_i\rangle\}_{\mathrm{k}^{+1}(S)}\rangle.0$$

ce qui empêche l'observateur d'agir de la manière précédente.

15.6 En conclusion

Il nous reste donc à décrire la relation de bisimilarité, ainsi que l'algorithme de décision associé, puis expliquer comment transformer un tel algorithme pour qu'il retourne des contraintes utilisables dans le cadre de la technique d'abstraction proposée à la section 15.4.

Chapitre 16

Relations de bisimilarité

M. Abadi et A. Gordon ont introduit dans [AG98] la notion de bisimilarité contextuelle (*framed bisimilarity*) et montré qu'il s'agissait d'une relation plus forte que l'équivalence par tests du spi-calcul. Le but était d'obtenir une notion d'équivalence pour les processus qui soit susceptible d'être associée à une technique de preuve effective, au moins dans le cas des processus à états finis (*i.e.* ceux dont le graphe de contrôle est fini). H. Hüttel a montré (dans un travail joint avec J. Kleist, U. Nestmann, B. Victor) que le fragment du spi-calcul constitué des processus à états finis avait la puissance des machines du Turing, avec pour conséquence que toute relation d'équivalence observationnelle non-triviale sur les processus du spi-calcul fini est indécidable (par un recours au théorème de Rice). [Hüt02], après avoir présenté ce résultat négatif, donne une preuve de décidabilité pour la bisimilarité contextuelle dans le cas du spi-calcul fini (les processus de $\mathrm{Proc}_{\mathrm{Fin}}$). On trouvera dans [BN02] une présentation des différentes notions de bisimilarité qui ont été définies pour étudier le spi-calcul, ainsi que des preuves d'inclusions (et, le cas échéant, du fait qu'elles soient strictes).Nous allons dans ce chapitre définir tout d'abord deux notions de bisimilarité : la bisimilarité contextuelle puis la h-bisimilarité contextuelle (*h-framed bisimilarity*). Pour un h bien choisi, ces deux relations sont égales et la deuxième est décidable, ce qui fournit un algorithme pour tester la bisimilarité contextuelle [Hüt02]. Afin d'obtenir un algorithme plus facile à transformer selon nos besoins, nous définirons une troisième notion de bisimilarité, plus forte, la h-bisimilarité gardée (qui combine la h-bisimilarité contextuelle avec la bisimilarité gardée (*fenced bisimilarity*) [EHHO99]). Auparavant, puisque la bisimilarité contextuelle relie entre eux des processus dans le cadre d'un contexte, nous définissons les environnements (*frame-theory pair*).

16.1 Environnements

Les notions de bisimilarité du spi-calcul sont dites sensibles au contexte, car elles ne mettent pas en relation des couples (P_1, P_2) de processus mais des triplets (e, P_1, P_2) où P_1 et P_2 sont des processus et e est appelé un environnement. Un environnement décrit la connaissance de l'observateur à un instant donné. Dans le cas de la bisimilarité contextuelle, cette connaissance présente deux aspects :

- les messages que l'observateur est capable de produire (ils se déduisent, de la manière habituelle, de l'ensemble des messages que l'observateur a pu accumuler au cours de ses observations précédentes) ;
- les couples de messages qui doivent être considérés comme indistinguables du point de

vue de l'observateur (par exemple deux messages chiffrés avec des clés inconnues).
Dans le cas de la bisimilarité contextuelle, un environnement est constitué de l'ensemble fini des noms connus de l'environnement (*frame*) et d'un ensemble fini de couples (m_1, m_2) de messages (*theory*), un tel couple signifiant que l'observateur ne peut distinguer m_1 issu du premier processus de m_2 issu du second.

DÉFINITION – 16.1.1

Un environnement est un couple (fr, th) *où* $fr \subseteq \mathrm{Name}$ *est un ensemble fini de noms et* $th \subseteq \mathrm{Msg} \times \mathrm{Msg}$ *est un ensemble fini de couples de messages clos. L'ensemble des environnements sera noté* Env. *Un environnement* $e = (fr, th) \in \mathrm{Env}$ *induit sur les messages une relation binaire notée* $e \vdash \_ = \_$ *et définie au moyen des règles suivantes :*

$$\frac{n \in fr}{(fr, th) \vdash n = n} \qquad \frac{(m_1, m_2) \in th}{(fr, th) \vdash m_1 = m_2} \qquad \frac{}{(fr, th) \vdash 0 = 0}$$

$$\frac{(fr, th) \vdash m_1 = m_2 \quad (fr, th) \vdash m_1' = m_2'}{(fr, th) \vdash \langle m_1, m_1' \rangle = \langle m_2, m_2' \rangle}$$

$$\frac{(fr, th) \vdash m_1 = m_2 \quad (fr, th) \vdash n_1 = n_2}{(fr, th) \vdash \{m_1\}_{n_1} = \{m_2\}_{n_2}}$$

Lorsque $e \vdash m_1 = m_2$, on dira que m_1 est indistinguable de m_2 dans e. Ceci signifie deux choses :

- tout d'abord, que l'observateur ne peux pas distinguer m_1 (relativement au premier processus) de m_2 (relativement au second) ;
- ensuite, qu'en appliquant les mêmes règles sur des messages obtenus du premier processus, d'une part, et des messages obtenus du second processus, d'autre part, e peut produire les messages m_1 et m_2.

Pour un environnement $e = (fr, th)$ et $i \in \{1, 2\}$ on notera :

$$\begin{array}{rcl} \mathrm{fn}(fr) & \hat{=} & fr \\ \mathrm{fn}(th_i) & \hat{=} & \displaystyle\bigcup_{(m_1, m_2) \in th} \mathrm{fn}(m_i) \\ \mathrm{fn}(e_i) & \hat{=} & \mathrm{fn}(fr) \cup \mathrm{fn}(th_i) \end{array}$$

DÉFINITION – 16.1.2

Un environnement $e = (fr, th)$ *est dit cohérent si, quel que soit* $(m_1, m_2) \in th$, *les conditions suivantes sont satisfaites :*

- *il existe* m_1' *et* n_1 *tels que* $m_1 = \{m_1'\}_{n_1}$ *et il n'existe pas* $n_2 \in fr$ *tel que* $e \vdash n_1 = n_2$ *;*
- *il existe* m_2' *et* n_2 *tels que* $m_2 = \{m_2'\}_{n_2}$ *et il n'existe pas* $n_1 \in fr$ *tel que* $e \vdash n_1 = n_2$ *;*
- *quel que soit* $(m_1', m_2') \in th$, *on a* $m_1 = m_1'$ *si, et seulement si,* $m_2 = m_2'$.

On note dans ce cas $\mathrm{cons}(e)$.

Le fait qu'un environnement soit cohérent signifie que, d'une part, il ne contient que des couples de messages chiffrés que l'observateur ne peut déchiffrer et, d'autre part, un message n'est jamais indistinguable de deux messages différents.

Exemple : Posons $e = (\{n\}, \{(\{0\}_k, \{n\}_k)$, on a alors :

$$\begin{array}{l} \mathrm{cons}(e) \\ e \vdash n = n \\ e \vdash \{0\}_k = \{n\}_k \\ e \vdash \langle n, \{0\}_k\rangle = \langle n, \{n\}_k\rangle \\ e \vdash \{\{0\}_k\}_n = \{\{n\}_k\}_n \\ e \nvdash k = k \\ e \nvdash \{n\}_k = \{0\}_k \end{array}$$

($e \nvdash m_1 = m_2$ signifiant que $e \vdash m_1 = m_2$ est faux). Par ailleurs, aucun des environnements suivant n'est cohérent :

$$\begin{array}{l} (\{a, b\}, \{(\{a\}_k, k)\}) \\ (\{a, b\}, \{(\{a\}_k, \{a\}_b)\}) \end{array}$$

$\square$

Si e et e' sont deux environnements, on désignera par $e \leq e'$ le fait que, quels que soient m_1, m_2 avec $e \vdash m_1 = m_2$, on a $e' \vdash m_1 = m_2$. On dit dans ce cas que e' étend e. On pose :

$$\mathrm{Ext}(e) \quad \hat{=} \quad \{e' \in \mathrm{Env} \mid e \leq e' \text{ et } \mathrm{cons}(e')\}$$

Souvent, partant d'un environnement e et d'un couple de messages (m_1, m_2), on cherchera à étendre e en un environnement e' cohérent tel que $e' \vdash m_1 = m_2$. Autrement dit, notant e sous la forme (fr, th), on s'intéressera à l'ensemble $\mathrm{Ext}((fr, th \cup \{(m_1, m_2)\}))$. Il se peut très bien que cet ensemble soit vide, il suffit par exemple de considérer $fr = \{n\}$, $th = \varnothing$, $m_1 = \{0\}_n$ et $m_2 = \{n\}_n$. La proposition décrit quelques uns des liens qui existent entre extensions et consistance.

Proposition – 16.1.1

Soit $e = (fr, th)$ un environnement, alors :

1. *Si $\{\vec{n}\} \subseteq \mathrm{Name}$ est tel que $\{\vec{n}\} \cap (\mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing$ et si e est cohérent, alors $e' = (fr \cup \{\vec{n}\}, th)$ est cohérent et $e \leq e'$;*
2. *Si $e' \in \mathrm{Env}$, on a équivalence entre :*
   - *$e \leq e'$,*
   - *quels que soient $n \in fr$ et $(m_1, m_2) \in th$, on a $e' \vdash n = n$ et $e' \vdash m_1 = m_2$;*
3. *Si $e' = (fr', th')$ est cohérent, on a équivalence entre :*
   - *$e \leq e'$,*
   - *$fr \subseteq fr'$ et, quel que soit $(m_1, m_2) \in th$, on a $e' \vdash m_1 = m_2$.*

Démonstration :

Le premier point est évident. En ce qui concerne le deuxième, le sens direct est évident. La réciproque se fait en considérant m_1, m_2 tels que $e \vdash m_1 = m_2$ et en prouvant, par induction sur la dérivation de cette proposition, que $e' \vdash m_1 = m_2$. Le cas de base de cette preuve par induction concerne bien sûr les $n \in \mathit{fr}$ et $(m'_1, m'_2) \in \mathit{th}$. Le troisième et dernier point est un peu plus délicat. Pour le sens direct, il suffit de montrer que $\mathit{fr} \subseteq \mathit{fr}'$. Soit donc $n \in \mathit{fr}$, on a $e \vdash n = n$ et par conséquent $e' \vdash n = n$. Comme e' est cohérent, nécessairement $n \in \mathit{fr}'$. Reste maintenant à faire la réciproque. Considérons pour cela m_1, m_2 tels que $e \vdash m_1 = m_2$ et montrons, par induction sur la dérivation de cette proposition, que $e' \vdash m_1 = m_2$:

- si $m_1 = m_2 = n \in \mathit{fr}$, alors, par hypothèse, $e' \vdash m_1 = m_2$;
- si $(m_1, m_2) \in \mathit{th}$, alors, par hypothèse, $e' \vdash m_1 = m_2$;
- si $m_1 = \langle m'_1, m''_1 \rangle$ et $m_2 = \langle m'_2, m''_2 \rangle$ avec $e \vdash m'_1 = m'_2$ et $e \vdash m''_1 = m''_2$, alors, par hypothèse d'induction, $e' \vdash m'_1 = m'_2$ et $e' \vdash m''_1 = m''_2$, d'où $e' \vdash m_1 = m_2$;
- si $m_1 = \{m'_1\}_{n_1}$ et $m_2 = \{m'_2\}_{n_2}$ avec $e \vdash m'_1 = m'_2$ et $e \vdash n_1 = n_2$, alors, par hypothèse d'induction, $e' \vdash m'_1 = m'_2$ et $e' \vdash n_1 = n_2$, d'où $e' \vdash m_1 = m_2$.

■

Un problème classique concernant les environnements consiste à partir d'un environnement cohérent $e = (\mathit{fr}, \mathit{th})$ et d'un couple de messages clos (m_1, m_2) et à chercher les extensions cohérentes et minimales de $e' \mathrel{\hat{=}} (\mathit{fr}, \mathit{th} \cup \{(m_1, m_2)\})$. Nous traiterons ce problème dans la section 16.4.

16.2 Bisimilarité contextuelle

La bisimilarité contextuelle relie entre eux des processus relativement à un environnement. Les relations ternaires de cette forme (mettant en relation un environnement et deux processus) seront appelées des relations contextuelles.

Définition – 16.2.1

Une relation contextuelle est une relation ternaire $\mathcal{S} \subseteq \mathrm{Env} \times \mathrm{Proc}_{\mathrm{Fin}} \times \mathrm{Proc}_{\mathrm{Fin}}$ telle que, quel que soit $(e, P_1, P_2) \in \mathcal{S}$, e est cohérent. Si $\mathcal{S}$ est une telle relation, on appelle relation contextuelle réciproque de $\mathcal{S}$, et on note $\mathcal{S}^{-1}$, la relation contextuelle :

$$\mathcal{S}^{-1} \quad \hat{=} \quad \{(e^{-1}, P_2, P_1) \mid (e, P_1, P_2) \in \mathcal{S}\}$$

où, pour $e = (\mathit{fr}, \mathit{th}) \in \mathrm{Env}$, e^{-1} désigne l'environnement (cohérent si e l'est) :

$$e^{-1} \quad \hat{=} \quad (\mathit{fr}, \{(m_2, m_1) \mid (m_1, m_2) \in \mathit{th}\}$$

Le fait que $(e, P_1, P_2) \in \mathcal{S}$ sera noté $e \vdash P_1 \mathrel{\mathcal{S}} P_2$ et, par conséquent :

$$e \vdash P_1 \mathrel{\mathcal{S}^{-1}} P_2 \quad \Leftrightarrow \quad e^{-1} \vdash P_2 \mathrel{\mathcal{S}} P_1$$

Intuitivement, deux processus sont bisimilaires s'il existe un moyen pour que le premier simule le second et réciproquement, de sorte qu'un observateur ne puisse pas les distinguer. Un tel moyen est représenté par une relation contextuelle appelée une bisimulation. Nous commençons par définir les simulations, les bisimulations n'étant rien de plus que des simulations dont la réciproque est également une simulation.

DÉFINITION – 16.2.2

Une simulation contextuelle est une relation contextuelle $\mathcal{S}$ telle que, si $e \vdash P_1 \; \mathcal{S} \; P_2$ avec $e = (fr, th)$, les conditions suivantes sont satisfaites :

1. *Si $P_1 \xrightarrow{\tau} P_1'$, alors il existe un processus P_2' tel que $P_2 \xrightarrow{\tau} P_2'$ et $e \vdash P_1' \; \mathcal{S} \; P_2'$;*
2. *Si $P_1 \xrightarrow{n} (x).P_1'$ avec $n \in fr$, alors il existe une abstraction $(x).P_2'$ telle que $P_2 \xrightarrow{n} (x).P_2'$ et, quels que soient $\{\vec{n}\} \subseteq \mathrm{Name}$ et $m_1, m_2 \in \mathrm{Msg}$ satisfaisant les conditions suivantes :*
   – *m_1 et m_2 sont clos,*
   – *$\{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing$,*
   – *$(fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2$;*
   on a $(fr \cup \{\vec{n}\}, th) \vdash P_1'[m_1/x] \; \mathcal{S} \; P_2'[m_2/x]$;
3. *si $P_1 \xrightarrow{\bar{n}} (\nu\vec{n_1})\langle m_1\rangle.P_1'$ avec $n \in fr$ et $\{\vec{n_1}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(e_1)) = \varnothing$, alors il existe une concrétion $(\nu\vec{n_2})\langle m_2\rangle.P_2'$ telle que $P_2 \xrightarrow{\bar{n}} (\nu\vec{n_2})\langle m_2\rangle.P_2'$ et $\{\vec{n_2}\} \cap (\mathrm{fn}(P_2) \cup \mathrm{fn}(e_2)) = \varnothing$ et il existe $e' \in \mathrm{Env}$ tel que $e \leq e'$, $e' \vdash m_1 = m_2$ et $e' \vdash P_1' \; \mathcal{S} \; P_2'$.*

Une bisimulation contextuelle est une relation contextuelle $\mathcal{S}$ telle que $\mathcal{S}$ et $\mathcal{S}^{-1}$ sont des simulations contextuelles. La réunion de toutes les bisimulations contextuelles est encore une bisimulation contextuelle, on l'appelle relation de bisimilarité contextuelle et on la note $\underset{\mathrm{ctx}}{\sim}$.

Remarque : Expliquons chacun des trois points caractérisant une simulation :

1. Si P_1 peut exécuter une action silencieuse et se transformer en P_1', alors P_2 peut effectuer une action silencieuse et se transformer en P_2' tel que P_1' soit en relation avec P_2'. Cette condition semble naturelle, mais il faut se rendre compte qu'elle accorde beaucoup de pouvoir à l'observateur : il peut ainsi discriminer deux processus totalement silencieux mais qui ne peuvent pas effectuer le même nombre d'actions ;
2. Si P_1 peut se transformer en un agent $(x).P_1'$, en attente de réception sur un canal n, connu de l'observateur, alors P_2 doit pouvoir faire de même, *i.e.* se transformer en $(x).P_2'$ en attente de réception sur le même canal et, de plus, lorsque l'observateur fournit à ces abstractions des messages m_1 et m_2, indistinguables de son point de vue, les processus obtenus doivent être en relation. Cette condition n'est pas testée dans l'environnement $e = (fr, th)$ de départ, mais dans un environnement $e' \mathrel{\hat{=}} (fr \cup \{\vec{n}\}, th)$ étendu avec de nouveaux noms (afin de permettre à l'observateur de créer, lui aussi, de nouveaux noms) ;
3. Si P_1 peut émettre un message m_1, sur un canal connu de l'observateur, après avoir créé de nouveaux noms $\vec{n_1}$, et se transformer ainsi en $(\nu\vec{n_1})\langle m_1\rangle.P_1'$, alors P_2 doit pouvoir faire de même (éventuellement en créant des noms différents, en nombre différent), *i.e.* se transformer en $(\nu\vec{n_2})\langle m_2\rangle.P_2'$ et les processus obtenus, P_1' et P_2' doivent être en relation.

En ce qui concerne le dernier point, le fait que P_1' soit en relation avec P_2' est testé dans un environnement e' étendant e, mais ne distinguant pas m_1 de m_2. Comme le remarquent les auteurs de [AG98], aucune condition de minimalité n'est imposée à e'. La raison de ce choix est que, d'une part, il n'empêche pas la bisimilarité contextuelle d'être correcte par rapport à l'équivalence par tests et, d'autre part, la condition est plus simple à vérifier lorsque l'on cherche à prouver que deux processus sont en relation. La quantification existentielle n'est cependant pas très adaptée pour une vérification effective. [Hüt02] explique comment se limiter à un sous-ensemble fini des extensions de $(fr, th \cup \{(m_1, m_2)\})$ (ce qui permet en partie de rendre la h-bisimilarité contextuelle décidable, voir à ce sujet la section suivante) et [EHHO99] propose de ne considérer qu'une extension, cohérente et minimale, donnée par un algorithme (ceci conduit à une nouvelle notion de bisimilarité, la bisimilarité gardée, que les auteurs pensaient être équivalente à la bisimilarité contextuelle, mais qui en est en fait un sous ensemble strict, voir à ce sujet [BN02]). Pour notre part, nous présenterons dans la section suivante la h-bisimilarité contextuelle, puis dans la section 16.4, nous l'adapterons à la manière de la bisimilarité gardée, pour obtenir la h-bisimilarité gardée. □

Adaptant le théorème principal de [AG98] au cas des processus finis, nous obtenons le résultat suivant.

Proposition – 16.2.1

Soient $P_1, P_2 \in \mathrm{Proc}_{\mathrm{Fin}}$ *et* $n \in \mathrm{Name}$ *tels que* $n \notin \mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)$*, on a alors :*

$$(\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \{n\}, \varnothing) \vdash P_1 \underset{\mathrm{ctx}}{\sim} P_2 \quad \Rightarrow \quad P_1 \underset{\mathrm{tst}}{\sim} P_2$$

Démonstration :

Constatons d'abord que, si $e \vdash P_1 \underset{\mathrm{ctx}}{\sim} P_2$ (suivant notre définition), alors P_1 et P_2 sont également équivalents au sens de la bisimilarité contextuelle *du spi-calcul* [AG98]. On en déduit, par application du résultat de correction de [AG98], que P_1 et P_2 sont équivalents par tests *dans le spi-calcul*, il sont donc *a fortiori* équivalents par tests, suivant notre définition. ■

Remarque : La réciproque est fausse, comme on peut le constater en considérant les processus :

$$\begin{aligned} P_1 &\hat{=} (\nu n).\overline{n}\langle n\rangle.0 \parallel n(x).0 \\ P_2 &\hat{=} 0 \end{aligned}$$

qui sont équivalents pour l'équivalence par tests (aucun des deux ne peut effectuer une action visible), mais pas pour la bisimilarité contextuelle (le premier peut effectuer une action invisible, mais pas le second). □

16.3 h-bisimilarité contextuelle

Cette relation est définie dans [Hüt02]. Dans cet article, H. Hüttel montre qu'elle est équivalente à la bisimilarité contextuelle sur des processus finis et des environnements de hauteur inférieure à h. Ce résultat lui permet d'obtenir un algorithme de décision pour la bisimilarité contextuelle (puisque la h-bisimilarité contextuelle est décidable). Pour notre part, nous renvoyons la question d'un algorithme de décision à la section suivante, où une troisième et dernière relation de bisimilarité sera présentée. Pour définir la h-bisimilarité contextuelle, nous commençons pas définir la hauteur des messages et des processus. Il s'agit en fait de compter combien de constructeurs (respectivement de destructeurs) de paires et de chiffrement comporte un message (respectivement un processus) donné.

Définition – 16.3.1

La hauteur d'un message $m \in \mathrm{Msg}$, notée $\mathrm{h}(m)$, est définie par induction sur m de la manière suivante :

$$\begin{array}{rcl} \mathrm{h}(n) & \hat{=} & 0 \\ \mathrm{h}(\langle m, m' \rangle) & \hat{=} & \max(\mathrm{h}(m), \mathrm{h}(m')) + 1 \\ \mathrm{h}(\{m\}_n) & \hat{=} & \mathrm{h}(m) + 1 \end{array}$$

De même, la hauteur d'un processus $P \in \mathrm{Proc}_{\mathrm{Fin}}$, notée $\mathrm{h}(P)$, est définie par induction sur P :

$$\begin{array}{rcl} \mathrm{h}(\text{case } m \text{ of } \langle x, y \rangle.P) = \mathrm{h}(\text{case } m \text{ of } \{x\}_k.P) & \hat{=} & \mathrm{h}(P) + 1 \\ \mathrm{h}(\overline{n}\langle m \rangle.P) = \mathrm{h}(n(x).P) & \hat{=} & \mathrm{h}(P) \\ \mathrm{h}((\nu n).P) = \mathrm{h}([m = m'].P) & \hat{=} & \mathrm{h}(P) \\ \mathrm{h}(P + Q) & \hat{=} & \max(\mathrm{h}(P), \mathrm{h}(Q)) \\ \mathrm{h}(P \parallel Q) & \hat{=} & \mathrm{h}(P) + \mathrm{h}(Q) \\ \mathrm{h}(0) & \hat{=} & 0 \end{array}$$

La définition d'une h-simulation contextuelle s'obtient à partir de celle d'une simulation contextuelle en limitant, dans la deuxième condition, la taille des environnements et des messages indistinguables à considérer (en fonction de h).

Définition – 16.3.2

Soit $h \geq 0$. Une h-simulation contextuelle est une relation contextuelle $\mathcal{S}$ telle que, si $e \vdash P_1 \,\mathcal{S}\, P_2$ avec $e = (fr, th)$, les conditions suivantes sont satisfaites :

1. *Si $P_1 \xrightarrow{\tau} P_1'$, alors il existe un processus P_2' tel que $P_2 \xrightarrow{\tau} P_2'$ et $e \vdash P_1' \,\mathcal{S}\, P_2'$;*
2. *Si $P_1 \xrightarrow{n} (x).P_1'$ avec $n \in fr$, alors il existe une abstraction $(x).P_2'$ telle que $P_2 \xrightarrow{n} (x).P_2'$ et, quels que soient $\{\vec{n}\} \subseteq \mathrm{Name}$ et $m_1, m_2 \in \mathrm{Msg}$ satisfaisant les conditions suivantes :*
   – *m_1 et m_2 sont clos, $\mathrm{h}(m_1) \leq h$ et $\mathrm{h}(m_2) \leq h$,*
   – *$|\vec{n}| \leq 2^h$ et $\{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing$,*
   – *$(fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2$;*
   on a $(fr \cup \{\vec{n}\}, th) \vdash P_1'[m_1/x] \,\mathcal{S}\, P_2'[m_2/x]$;
3. *si $P_1 \xrightarrow{\bar{n}} (\nu\vec{n_1})\langle m_1 \rangle.P_1'$ avec $n \in fr$ et $\{\vec{n_1}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(e_1)) = \varnothing$, alors il existe une concrétion $(\nu\vec{n_2})\langle m_2 \rangle.P_2'$ telle que $P_2 \xrightarrow{\bar{n}} (\nu\vec{n_2})\langle m_2 \rangle.P_2'$ et $\{\vec{n_2}\} \cap (\mathrm{fn}(P_2) \cup \mathrm{fn}(e_2)) = \varnothing$ et il existe $e' \in \mathrm{Env}$ tel que $e \leq e'$, $e' \vdash m_1 = m_2$ et $e' \vdash P_1' \,\mathcal{S}\, P_2'$.*

Une h-bisimulation contextuelle est une relation contextuelle $\mathcal{S}$ telle que $\mathcal{S}$ et $\mathcal{S}^{-1}$ sont des h-simulations contextuelles. La réunion de toutes les h-bisimulations contextuelles est encore une h-bisimulation contextuelle, on l'appelle relation de h-bisimilarité contextuelle et on la note $\overset{h}{\underset{\mathrm{ctx}}{\sim}}$.

[Hüt02] commence par montrer que la bisimilarité contextuelle est indécidable sur les processus à états finis. Il s'agit, intuitivement, des processus dont le graphe de contrôle est fini. Les processus finis, qui sont les processus à états finis dont le graphe de contrôle ne contient pas de cycle, forment un sous-ensemble strict des processus à états finis. [Hüt02] montre alors que la bisimilarité contextuelle est équivalente, sur les processus finis, à la h-bisimilarité contextuelle (pour un h bien choisi, dépendant des processus dont on veut tester l'équivalence, ainsi que de l'environnement de départ) et que cette dernière est décidable. Nous énonçons ici le résultat d'équivalence, laissant pour la section suivante la question de la décidabilité.

Proposition – 16.3.1

Soient $e = (fr, th) \in \mathrm{Env}$ *et* $P_1, P_2 \in \mathrm{Proc}_{\mathrm{Fin}}$. *On pose :*

$$h \;\hat{=}\; \left(\max_{(m_1,m_2)\in th} \max(\mathrm{h}(m_1), \mathrm{h}(m_2))\right) + \max(\mathrm{h}(P_1), \mathrm{h}(P_2))$$

On a alors :

$$e \vdash P_1 \underset{\mathrm{ctx}}{\overset{h}{\sim}} P_2 \quad \Leftrightarrow \quad e \vdash P_1 \underset{\mathrm{ctx}}{\sim} P_2$$

Démonstration :

Voir [Hüt02]. L'idée consiste à montrer qu'un processus de hauteur h ne peut distinguer des messages égaux sur les h constructions de paires et de chiffrement les plus extérieures. ■

16.4 h-bisimilarité gardée

Dans la définition de la h-bisimilarité contextuelle, la troisième condition contient une quantification existentielle sur les extensions d'un environnement de la forme $(fr, th \cup \{(m_1, m_2)\})$, ce qui empêche de conclure directement que la h-bisimilarité structurée est décidable. [Hüt02] montre qu'il est possible de ne considérer qu'un nombre fini d'extensions et obtient ainsi la décidabilité. Nous préférons, pour notre part, limiter les extensions à considérer dans la définition de la bisimilarité, à la manière de [EHHO99]. Les auteurs de cet article définissent une nouvelle notion de bisimilarité, la bisimilarité gardée (*fenced bisimilarity*) en ne considérant que les extensions de $(fr, th \cup \{(m_1, m_2)\})$ fournies au moyen d'un certain algorithme. Nous remplaçons ici l'exécution de cet algorithme par la recherche d'une forme normale pour une certaine relation de réduction. Cette dernière, partant d'un environnement e, permet d'obtenir les extensions cohérentes minimales de e.

Définition – 16.4.1

On définit sur Env *une relation binaire* _ → _ *au moyen des règles suivantes :*

$$\frac{}{(fr, th \cup \{(n, n)\}) \to (fr \cup \{n\}, th)}$$

$$\frac{}{(fr \cup \{n\}, th \cup \{(\{m_1\}_n, \{m_2\}_n)\}) \to (fr \cup \{n\}, th \cup \{(m_1, m_2)\})}$$

$$\frac{}{(fr, th \cup \{(\langle m_1, m'_1\rangle, \langle m_2, m'_2\rangle)\}) \to (fr, th \cup \{(m_1, m_2), (m'_1, m'_2)\})}$$

On constate immédiatement que cette relation est nœthérienne. Pour le justifier, considérons $e = (fr, th)$ et associons-lui la mesure $\mu(e) \hat{=} (c(th), \mathrm{card}(th))$ où $c(th)$ est le nombre de constructions de couples et de chiffrement apparaissant dans th et $\mathrm{card}(th)$ est simplement le cardinal de th. Si $e \to e'$, alors $\mu(e)$ est strictement plus grand que e', dans l'ordre lexicographique (qui est bien fondé). Pour $e \in \mathrm{Env}$, notons $\mathrm{Norm}(e)$ l'ensemble des formes normales de e pour cette relation de réduction qui sont, de plus, cohérentes. La proposition suivante caractérise les extensions cohérentes de e, tout en nous permettant de calculer celles qui sont minimales.

Proposition – 16.4.1

> *Quel que soit $e \in \mathrm{Env}$, $\mathrm{Norm}(e)$ est l'ensemble des $e' \in \mathrm{Ext}(e)$ qui sont minimaux (pour la relation d'extension).*

Démonstration :

Nous allons montrer les deux points suivants :

1. $\mathrm{Norm}(e) \subseteq \mathrm{Ext}(e)$;
2. quel que soit $e' \in \mathrm{Ext}(e)$, il existe $e'' \in \mathrm{Norm}(e)$ tel que $e'' \leq e'$.

Pour le premier point, il suffit de montrer que si $e \to e'$, alors $e \leq e'$. Notons $e = (fr, th)$ et raisonnons par cas sur la règle utilisée pour la réduction $e \to e'$:

– si c'est la première règle qui s'applique, alors :

$$e = (fr, th \cup \{(n, n)\}) \quad \text{et} \quad e' = (fr \cup \{n\}, th)$$

– si c'est la deuxième, alors :

$$e = (fr \cup \{n\}, th \cup \{(\{m_1\}_n, \{m_2\}_n)\}) \quad \text{et} \quad e' = (fr \cup \{n\}, th \cup \{(m_1, m_2)\})$$

– si c'est la troisième, alors :

$$e = (fr, th \cup \{(\langle m_1, m_1' \rangle, \langle m_2, m_2' \rangle)\}) \quad \text{et} \quad e' = (fr, th \cup \{(m_1, m_2), (m_1', m_2')\})$$

Dans chaque cas, il est facile de vérifier, au moyen de la proposition 16.1.1 (page 175), que $e \leq e'$. Par conséquent, si e' est une forme normale de e, alors $e \leq e'$ et, par définition de $\mathrm{Norm}(e)$ qui ne contient que des environnements cohérents, $\mathrm{Norm}(e) \subseteq \mathrm{Ext}(e)$.

Pour le second point, commençons par montrer que si $e, e' \in \mathrm{Env}$ avec $e \leq e'$, e' cohérent et e non cohérent, alors il existe $e'' \in \mathrm{Env}$ tel que $e \to e''$ et $e'' \leq e'$. Pour cela, constatons que si $e = (fr, th)$ n'est pas cohérent, alors on est dans un des cas suivants :

– il existe $(n, n) \in th$ avec $n \in \mathrm{Name}$;
– il existe $(\{m_1\}_n, \{m_2\}_n) \in th$ avec $n \in fr$;
– il existe $(\langle m_1, m_1' \rangle, \langle m_2, m_2' \rangle) \in th$;
– il existe $(n_1, n_2) \in th$ avec $n_1, n_2 \in \mathrm{Name}$ et $n_1 \neq n_2$;
– il existe $(\{m_1\}_{n_1}, \{m_2\}_{n_2}) \in th$ avec $n_1 \in fr$ (ou $n_2 \in fr$) et $n_1 \neq n_2$;
– il existe $(m_1, m_2) \in th$ tel que m_1 et m_2 ne sont ni tous les deux des noms, ni tous les deux des paires, ni tous les deux des messages chiffrés ;
– il existe $(m_1, m_2), (m_1', m_2') \in th$ tels que $m_1 = m_1'$ et $m_2 \neq m_2'$ (ou $m_1 \neq m_1'$ et $m_2 = m_2'$).

En réalité, seuls les trois premiers cas sont possibles (car les derniers cas empêchent l'existence d'une extension cohérente de e). Chacun des trois premiers cas permet à l'une des règles de la définition 16.4.1 de s'appliquer et on obtient ainsi un environnement e'' tel que $e \to e''$. Suivant la règle appliquée, montrons que $e'' \leq e'$:

– si la première règle s'applique, on peut écrire :

$$e \;=\; (fr, th \cup \{(n,n)\}) \quad \text{et} \quad e'' \;=\; (fr \cup \{n\}, th)$$

Pour montrer que $e'' \leq e'$, il suffit alors de montrer que $e' \vdash n = n$, ce qui est évident puisque $e \leq e'$ et $e \vdash n = n$;

– si la deuxième règle s'applique, alors on peut écrire :

$$e \;=\; (fr \cup \{n\}, th \cup \{(\{m_1\}_n, \{m_2\}_n)\}) \quad \text{et} \quad e'' \;=\; (fr \cup \{n\}, th \cup \{(m_1, m_2)\})$$

Pour montrer que $e'' \leq e'$, il suffit alors de montrer que $e' \vdash m_1 = m_2$. Or on sait que $e' \vdash \{m_1\}_n = \{m_2\}_n$ (puisque $e \vdash \{m_1\}_n = \{m_2\}_n$ et $e \leq e'$). Comme e' est supposé cohérent, on ne peut pas avoir $(\{m_1\}_n, \{m_2\}_n) \in th'$ (si on note $e' = (fr', th')$), donc nécessairement $e' \vdash m_1 = m_2$;

– si la troisième règle s'applique, alors on peut écrire :

$$e \;=\; (fr, th \cup \{(\langle m_1, m_1'\rangle, \langle m_2, m_2'\rangle)\}) \quad \text{et} \quad e'' \;=\; (fr, th \cup \{(m_1, m_2), (m_1', m_2')\})$$

Pour montrer que $e'' \leq e'$, il suffit alors de montrer que $e' \vdash m_1 = m_2$ et $e' \vdash m_1' = m_2'$. Or on sait que $e' \vdash \langle m_1, m_1'\rangle = \langle m_2, m_2'\rangle$ (puisque $e \vdash \langle m_1, m_1'\rangle = \langle m_2, m_2'\rangle$ et $e \leq e'$). Comme e' est supposé cohérent, on ne peut pas avoir $(\langle m_1, m_1'\rangle, \langle m_2, m_2'\rangle) \in th'$ (si on note $e' = (fr', th')$), donc nécessairement $e' \vdash m_1 = m_2$ et $e' \vdash m_1' = m_2'$.

Si maintenant $e' \in \mathrm{Ext}(e)$, on itère ce processus et on obtient une suite $e = e_0 \to e_1 \to \ldots \to e_n \leq e'$. Comme la relation de réduction est nœthérienne, on aboutit à une forme normale e'', nécessairement cohérente, *i.e.* un élément $e'' \in \mathrm{Norm}(e)$ tel que $e'' \leq e'$. ■

La définition d'une h-simulation gardée s'obtient alors à partir de celle d'une h-simulation contextuelle en ne considérant, dans la troisième condition, que les environnements qui appartiennent à $\mathrm{Norm}(fr, th \cup \{(m_1, m_2)\})$.

Définition – 16.4.2

Soit $h \geq 0$. Une h-simulation gardée est une relation contextuelle $\mathcal{S}$ telle que, si $e \vdash P_1 \;\mathcal{S}\; P_2$ avec $e = (fr, th)$, les conditions suivantes sont satisfaites :

1. *Si $P_1 \xrightarrow{\tau} P_1'$, alors il existe un processus P_2' tel que $P_2 \xrightarrow{\tau} P_2'$ et $e \vdash P_1' \;\mathcal{S}\; P_2'$;*
2. *Si $P_1 \xrightarrow{n} (x).P_1'$ avec $n \in fr$, alors il existe une abstraction $(x).P_2'$ telle que $P_2 \xrightarrow{n} (x).P_2'$ et, quels que soient $\{\vec{n}\} \subseteq \mathrm{Name}$ et $m_1, m_2 \in \mathrm{Msg}$ satisfaisant les conditions suivantes :*
   - *m_1 et m_2 sont clos, $\mathrm{h}(m_1) \leq h$ et $\mathrm{h}(m_2) \leq h$,*
   - *$|\vec{n}| \leq 2^h$ et $\{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing$,*
   - *$(fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2$;*

   on a $(fr \cup \{\vec{n}\}, th) \vdash P_1'[m_1/x] \;\mathcal{S}\; P_2'[m_2/x]$;
3. *si $P_1 \xrightarrow{\bar{n}} (\nu\vec{n_1})\langle m_1\rangle.P_1'$ avec $n \in fr$ et $\{\vec{n_1}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(e_1)) = \varnothing$, alors il existe une concrétion $(\nu\vec{n_2})\langle m_2\rangle.P_2'$ telle que $P_2 \xrightarrow{\bar{n}} (\nu\vec{n_2})\langle m_2\rangle.P_2'$ et $\{\vec{n_2}\} \cap (\mathrm{fn}(P_2) \cup \mathrm{fn}(e_2)) = \varnothing$ et il existe $e' \in \mathrm{Norm}(fr, th \cup \{(m_1, m_2)\})$ tel que $e' \vdash P_1' \;\mathcal{S}\; P_2'$.*

Une h-bisimulation gardée est une relation contextuelle $\mathcal{S}$ telle que $\mathcal{S}$ et $\mathcal{S}^{-1}$ sont des h-simulations gardées. La réunion de toutes les h-bisimulations gardées est encore une h-bisimulation gardée, on l'appelle relation de h-bisimilarité gardée et on la note $\overset{h}{\underset{\mathrm{grd}}{\sim}}$.

Comme la troisième condition de cette définition est plus restrictive que celle qui se trouve dans la définition de la h-bisimilarité structurée, on a immédiatement le résultat suivant.

PROPOSITION – 16.4.2

Soient $e \in \mathrm{Env}$, $P_1, P_2 \in \mathrm{Proc}_{\mathrm{Fin}}$ *et* $h \geq 0$. *On a :*

$$e \vdash P_1 \overset{h}{\underset{\mathrm{grd}}{\sim}} P_2 \quad \Rightarrow \quad e \vdash P_1 \overset{h}{\underset{\mathrm{ctx}}{\sim}} P_2$$

Remarque : Là encore, la réciproque est fausse. Pour le constater, on peut considérer l'exemple suivant (extrait de [BN02]) :

$$\begin{array}{rcl} P_1 & \hat{=} & (\nu n, k, l).\overline{a}\langle\{\{n\}_k\}_l\rangle.(\nu m).\overline{a}\langle m\rangle.0 \\ P_2 & \hat{=} & (\nu n, k).\overline{a}\langle\{n\}_k\rangle.(\nu m).\overline{a}\langle m\rangle.0 \\ e & \hat{=} & (\{a\}, \varnothing) \end{array}$$

Chacun de ces processus peut émettre un message sur le canal a et il faut alors tester l'équivalence des processus :

$$\begin{array}{rcl} P_1' & \hat{=} & (\nu m).\overline{a}\langle m\rangle.0 \\ P_2' & \hat{=} & (\nu m).\overline{a}\langle m\rangle.0 \end{array}$$

dans l'environnement e' étendant e avec le couple de messages $(\{\{n\}_k\}_l, \{n'\}_{k'})$. Remarquons que n, k, l doivent être distincts. Par conséquent, il existe $z \in \{n, k, l\} \setminus \{n', k'\}$. Si on choisit e' minimal (comme pour la bisimilarité gardée), alors P_2' peut effectuer la transition :

$$P_2' \xrightarrow{\bar{a}} (\nu z)\langle z\rangle.0$$

qui ne pourra pas être simulée par P_1' (puisque $z \in \mathrm{fn}(P_1')$). On en déduit que P_1 et P_2 ne sont pas équivalents dans e pour la bisimilarité gardée. Par contre, si comme pour la bisimilarité contextuelle, on peut choisir une extension quelconque de e avec le couple $(\{\{n\}_k\}_l, \{n'\}_{k'})$, on peut prendre l'environnement e' précédent, étendu avec le nom z. Dans ce cas, la transition $P_2' \xrightarrow{\bar{a}} (\nu z)\langle z\rangle.0$ n'est plus à considérer. Les processus sont alors équivalents, au sens de la bisimilarité contextuelle.

De manière plus informelle, supposons, par exemple, que $l \notin \{n', k'\}$. Si on procède à la manière de la bisimilarité gardée, on ne peut pas empêcher P_2' d'effectuer la transition $P_2' \xrightarrow{\bar{a}} (\nu l)\langle l\rangle.0$. Comme $l \in \mathrm{fn}(P_1')$, il sera par contre impossible à P_1' de mimer cette transition, car pour toute transition $P_1' \xrightarrow{\bar{a}} (\nu l')\langle l'\rangle.0$, on aura $l' \notin \mathrm{fn}(P_1')$ et donc $l' \neq l$. On peut par contre éviter de prendre en compte la transition $P_2' \xrightarrow{\bar{a}} (\nu l)\langle l\rangle.0$ en prenant une extension e' de e dans laquelle l apparaît mais, pour pouvoir faire ceci, il faut utiliser la bisimulation contextuelle, qui n'impose pas de condition de minimalité sur les extensions. $\square$

Corollaire – 16.4.1

Soient $P_1, P_2 \in \mathrm{Proc}_{\mathrm{Fin}}$ *et* $n \in \mathrm{Name}$ *tels que* $n \notin \mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)$*. On pose :*

$$h \quad \hat{=} \quad \max(\mathrm{h}(P_1), \mathrm{h}(P_2))$$

On a alors :

$$(\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \{n\}, \varnothing) \vdash P_1 \underset{\mathrm{grd}}{\overset{h}{\sim}} P_2 \quad \Rightarrow \quad P_1 \underset{\mathrm{tst}}{\sim} P_2$$

16.5 Unification des différentes bisimilarités

Les définitions des différentes notions de bisimilarité données dans ce chapitre varient peu d'une relation de bisimilarité à la suivante. Afin d'apporter une meilleure compréhension de ces différentes notions et des relations qui existent entre elles, nous proposons ici une définition générale qui permet de retrouver chacune des relations de bisimilarité évoquée. Comme on l'a fait remarquer en donnant chaque définition, la différence d'une notion de bisimilarité à l'autre est contenue dans les ensembles sur lesquels sont réalisées les quantifications dans les deuxième et troisième conditions (la deuxième condition comporte une quantification universelle sur un ensemble de triplets $(\{\vec{n}\}, m_1, m_2)$ où $\{\vec{n}\}$ est un ensemble fini de noms et m_1 et m_2 sont des messages, alors que la troisième condition contient une quantification existentielle sur un ensemble d'environnements). Nous sommes donc conduits à poser la définition suivante.

Définition – 16.5.1

Soient $f : \mathrm{Env} \times \mathrm{Proc} \times \mathrm{Proc} \to 2^{2^{\mathrm{Name}} \times \mathrm{Msg} \times \mathrm{Msg}}$ *et* $g : \mathrm{Env} \times \mathrm{Msg} \times \mathrm{Msg} \to 2^{\mathrm{Env}}$ *deux fonctions. Une* (f,g)*-simulation est une relation contextuelle* $\mathcal{S}$ *telle que, si* $e \vdash P_1 \mathrel{\mathcal{S}} P_2$ *avec* $e = (fr, th)$*, les conditions suivantes sont satisfaites :*

1. *Si* $P_1 \xrightarrow{\tau} P_1'$*, alors il existe un processus* P_2' *tel que* $P_2 \xrightarrow{\tau} P_2'$ *et* $e \vdash P_1' \mathrel{\mathcal{S}} P_2'$ *;*
2. *Si* $P_1 \xrightarrow{n} (x).P_1'$ *avec* $n \in fr$*, alors il existe une abstraction* $(x).P_2'$ *telle que* $P_2 \xrightarrow{n} (x).P_2'$ *et, quels que soient* $(\{\vec{n}\}, m_1, m_2) \in f(e, P_1, P_2)$*, on a* $(fr \cup \{\vec{n}\}, th) \vdash P_1'[m_1/x] \mathrel{\mathcal{S}} P_2'[m_2/x]$ *;*
3. *si* $P_1 \xrightarrow{\bar{n}} (\nu\vec{n_1})\langle m_1\rangle.P_1'$ *avec* $n \in fr$ *et* $\{\vec{n_1}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(e_1)) = \varnothing$*, alors il existe une concrétion* $(\nu\vec{n_2})\langle m_2\rangle.P_2'$ *telle que* $P_2 \xrightarrow{\bar{n}} (\nu\vec{n_2})\langle m_2\rangle.P_2'$ *et* $\{\vec{n_2}\} \cap (\mathrm{fn}(P_2) \cup \mathrm{fn}(e_2)) = \varnothing$ *et il existe* $e' \in g(e, m_1, m_2)$ *tel que* $e' \vdash P_1' \mathrel{\mathcal{S}} P_2'$*.*

Une (f,g)*-bisimulation est une relation contextuelle* $\mathcal{S}$ *telle que* $\mathcal{S}$ *et* $\mathcal{S}^{-1}$ *sont des* (f,g)*-simulations. La réunion de toutes les* (f,g)*-bisimulations est encore une* (f,g)*-bisimulation, on l'appelle relation de* (f,g)*-bisimilarité et on la note* $\underset{\mathrm{f,g}}{\sim}$*.*

Comme f est associée à une quantification universelle et g à une quantification existentielle, si les fonctions f' et g' sont telles que $f \subseteq f'$ et $g' \subseteq g$ (*i.e.* quels que soient $e \in Env$, $P_1, P_2 \in \mathrm{Proc}$ et $m_1, m_2 \in \mathrm{Msg}$, $f(e, P_1, P_2) \subseteq f'(e, P_1, P_2)$ et $g'(e, m_1, m_2) \subseteq g(e, m_1, m_2)$), on a $\underset{\mathrm{f',g'}}{\sim} \subseteq \underset{\mathrm{f,g}}{\sim}$. Montrons maintenant comment retrouver les notions de bisimilarités définies plus haut au moyen de cette nouvelle notion. On pose tout d'abord, pour $e = (fr, th) \in Env$,

$P_1, P_2 \in \mathrm{Proc}$ et $m_1, m_2 \in \mathrm{Msg}$:

$$\begin{array}{rcl} f_{\mathrm{ctx}}(e, P_1, P_2) & \hat{=} & \{(\{\vec{n}\}, m_1, m_2) \mid \begin{array}[t]{l} m_1, m_2 \text{ clos}, \\ (fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2, \\ \{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing\} \end{array} \\ g_{\mathrm{ctx}}(e, m_1, m_2) & \hat{=} & \{e' \mid e \leq e' \text{ et } e' \vdash m_1 = m_2\} \end{array}$$

On a alors $\underset{\mathrm{ctx}}{\sim} = \underset{\mathrm{f_{ctx}, g_{ctx}}}{\sim}$. Posons ensuite, pour $h \geq 0$:

$$\begin{array}{rcl} f^h_{\mathrm{ctx}}(e, P_1, P_2) & \hat{=} & \{(\{\vec{n}\}, m_1, m_2) \mid \begin{array}[t]{l} m_1, m_2 \text{ clos}, \\ (fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2, \\ \mathrm{h}(m_1) \leq h, \mathrm{h}(m_2) \leq h, |\vec{n}| \leq 2^h, \\ \{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing\} \end{array} \\ g^h_{\mathrm{ctx}}(e, m_1, m_2) & \hat{=} & g_{\mathrm{ctx}}(e, m_1, m_2) \end{array}$$

On a maintenant $\overset{h}{\underset{\mathrm{ctx}}{\sim}} = \underset{\mathrm{f^h_{ctx}, g^h_{ctx}}}{\sim}$. Notons que, comme on n'a pas $f_{\mathrm{ctx}} \subseteq f'^h_{\mathrm{ctx}}$, il faut une preuve spéciale pour établir une relation entre $\overset{h}{\underset{\mathrm{ctx}}{\sim}}$ et $\underset{\mathrm{ctx}}{\sim}$ (c'est la proposition 16.3.1, issue de [Hüt02]). Posons enfin :

$$\begin{array}{rcl} f^h_{\mathrm{grd}}(e, P_1, P_2) & \hat{=} & f^h_{\mathrm{ctx}} \\ g^h_{\mathrm{grd}}(e, m_1, m_2) & \hat{=} & \mathrm{Norm}(fr, th \cup \{m_1, m_2\}) \end{array}$$

de sorte que $\overset{h}{\underset{\mathrm{grd}}{\sim}} = \underset{\mathrm{f^h_{grd}, g^h_{grd}}}{\sim}$. Comme on sait que $g^h_{\mathrm{grd}} \subseteq g^h_{\mathrm{ctx}}$, on a clairement $\overset{h}{\underset{\mathrm{grd}}{\sim}} \subseteq \overset{h}{\underset{\mathrm{ctx}}{\sim}}$. Après avoir clarifié les définitions et les liens qui existent entre les différentes notions de bisimilarité présentées ici, nous revenons maintenant au problèmes d'opacité.

16.6 Retour aux problèmes d'opacité

Un algorithme de décision pour la h-bisimilarité gardée. Compte-tenu du fait que les processus considérés ici sont finis (en particulier, il ne peut y avoir de chaînes infinies de transitions partant d'un processus donné), le fait que P_1 et P_2 soient bisimilaires (pour la h-bisimilarité gardée) dans un environnement $e = (fr, th)$, ce que l'on notera $B((fr, th), P_1, P_2)$, peut être défini par induction sur P_1 et P_2 au moyen des énoncés regroupés dans la figure 16.6.1. Explicitons la signification des énoncés intermédiaires qui apparaissent dans cette description :

- $B^h((fr, th), P_1, P_2)$ signifie que P_1 et P_2 sont en relation pour la h-bisimilarité gardée dans l'environnement (fr, th) ;
- $S^h_1((fr, th), P_1, P_2)$ (respectivement $S^h_2((fr, th), P_1, P_2)$) signifie que toute transition de P_1 (respectivement P_2) peut être simulée par une transition de P_2 (respectivement P_1), les processus résiduels étant bisimilaires ;
- $T^h((fr, th), \alpha, P_1, P'_1, P_2, P'_2)$ signifie que les transitions $P_1 \xrightarrow{\alpha} P'_1$ et $P_2 \xrightarrow{\alpha} P'_2$ se correspondent bien, au sens de la h-bisimulation gardée. La définition de $T((fr, th), \alpha, P_1, P'_1, P_2, P'_2)$ dépend par conséquent de α :
  - si $\alpha = \tau$, il suffit de vérifier que P'_1 et P'_2 sont bisimilaires,
  - si $\alpha = n$, il faut augmenter l'environnement avec au plus 2^h nouveaux noms et vérifier que, si dans le nouvel environnement m_1 et m_2 sont indistinguables et de hauteur au plus h, alors $P'_1[m_1/x]$ et $P'_2[m_2/x]$ sont bisimilaires,

$$
\begin{aligned}
B^h((fr, th), P_1, P_2) &\ \hat{=}\ S_1^h((fr, th), P_1, P_2) \wedge S_2^h((fr, th), P_1, P_2)\\
S_1^h((fr, th), P_1, P_2) &\ \hat{=}\ \bigwedge_{\substack{n \in fr\\ \alpha \in \{n, \bar{n}, \tau\}\\ P_1 \xrightarrow{\alpha} P_1'}} \ \bigvee_{P_2 \xrightarrow{\alpha} P_2'} T^h((fr, th), \alpha, P_1, P_1', P_2, P_2')\\
S_2^h((fr, th), P_1, P_2) &\ \hat{=}\ \bigwedge_{\substack{n \in fr\\ \alpha \in \{n, \bar{n}, \tau\}\\ P_2 \xrightarrow{\alpha} P_2'}} \ \bigvee_{P_1 \xrightarrow{\alpha} P_1'} T^h((fr, th), \alpha, P_1, P_1', P_2, P_2')\\
T^h((fr, th), \tau, P_1, P_1', P_2, P_2') &\ \hat{=}\ B^h((fr, th), P_1', P_2')\\
T^h((fr, th), n, P_1, (x).P_1', P_2, (x).P_2') &\ \hat{=}\\
& \bigwedge_{\substack{\{\vec{n}\} \subseteq \text{Name}, |\vec{n}| \leq 2^h\\ \{\vec{n}\} \cap (\text{fn}(P_1) \cup \text{fn}(P_2) \cup \text{fn}(e_1) \cup \text{fn}(e_2)) = \varnothing\\ (fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2\\ \text{h}(m_1) \leq h, \text{h}(m_2) \leq h}} B^h((fr \cup \{\vec{n}\}, th), P_1'[m_1/x], P_2'[m_2/x])\\
T^h((fr, th), \bar{n}, P_1, (\nu \vec{n_1})\langle m_1 \rangle.P_1', P_2, (\nu \vec{n_2})\langle m_2 \rangle.P_2') &\ \hat{=}\\
& \bigvee_{(fr', th') \in \text{Norm}(fr, th \cup \{(m_1, m_2)\})} B^h((fr', th'), P_1', P_2')
\end{aligned}
$$

FIG. 16.6.1 – *Description de la h-bisimilarité gardée*

- si $\alpha = \bar{n}$, il faut vérifier qu'il existe une extension cohérente et minimale de $(fr, th \cup \{(m_1, m_2)\})$ dans laquelle P_1' et P_2' sont bisimilaires.

On pose finalement :

$$B(P_1, P_2) \quad \hat{=} \quad B^h((\text{fn}(P_1) \cup \text{fn}(P_2) \cup \{n\}, \varnothing), P_1, P_2)$$

où n est un nom qui n'est ni dans $\text{fn}(P_1)$, ni dans $\text{fn}(P_2)$ et $h \mathrel{\hat{=}} \max(\text{h}(P_1), \text{h}(P_2))$. On a alors :

$$
\begin{aligned}
B(P_1, P_2) &\Leftrightarrow (\text{fn}(P_1) \cup \text{fn}(P_2) \cup \{n\}, \varnothing) \vdash P_1 \overset{h}{\underset{\text{grd}}{\sim}} P_2\\
B(P_1, P_2) &\Rightarrow P_1 \underset{\text{tst}}{\sim} P_2
\end{aligned}
$$

Lien avec les problèmes d'opacité. Commençons par revenir sur l'exemple du protocole de vote donné page 171.

Exemple : Dans le cas où il n'y a que trois participants, les processus décrivant ce protocole s'écrivent :

$$
\begin{aligned}
P_{A_i} &\ \hat{=}\ \overline{c}\langle \{V_1\}_{\text{k}(A_i, S)} \rangle.0 \ (i \in \{1, 2, 3\})\\
S_{A_i} &\ \hat{=}\ c(x).\text{case } x \text{ of } \{y\}_{\text{k}(A_i, S)}.\overline{c}\langle y \rangle.0 \ (i \in \{1, 2, 3\})\\
S &\ \hat{=}\ S_{A_1} \parallel S_{A_2} \parallel S_{A_3}\\
P &\ \hat{=}\ P_{A_1} \parallel P_{A_2} \parallel P_{A_3} \parallel S
\end{aligned}
$$

Le système observé associé à ce protocole est construit sur l'ensemble des mondes :

$$W \quad \hat{=} \quad \text{Name}^3 \times \{0,1\}^3$$

et nous nous intéressons à l'opacité de l'attribut qui décrit la correspondance entre les identités et le vote qu'elles ont effectué :

$$\begin{array}{rlcl} c: & W & \rightarrow & C = 2^{\text{Name}\times\{0,1\}} \\ & w & \mapsto & \{(a_i, v_i) \mid 1 \leq i \leq 3\} \end{array}$$

Le domaine abstrait sera l'ensemble des fonctions qui, à $a \in \text{Name}$, associent un sous-ensemble de $\{0,1\}$ (qui contient les valeurs possibles pour le vote de a). La propriété d'opacité considérée sera :

$$\varphi \quad \hat{=} \quad \{a \in A \mid \forall a_i.\ a(a_i) = \varnothing \text{ ou } a(a_i) = \{0,1\}\}$$

cette propriété énonçant que, si a_i est un participant, on peut savoir s'il a voté ou non mais pas la valeur du vote qu'il a effectué. Remarquons qu'il est impossible de prouver cette propriété dans le cas où tous les votants ont fait le même choix. Nous fixerons donc un monde particulier $w \in W$, contenant au moins deux votes différents, par exemple :

$$w \;=\; ((a_1,a_2,a_3),(v_1,v_2,v_3)) \quad \hat{=} \quad ((a_1,a_2,a_3),(1,0,0))$$

On peut, pour montrer la propriété d'opacité φ, considérer l'ensemble de mondes :

$$\begin{array}{rl} B_w \;\hat{=} & \{w' = ((a'_1,a'_2,a'_3),(v'_1,v'_2,v'_3)) \in W \mid \\ & \quad B(P[a_1/A_1,\ldots,a_3/A_3,v_1/V_1,\ldots,v_3/V_3], P[a'_1/A_1,\ldots,a'_3/A_3,v'_1/V_1,\ldots,v'_3/V_3]\} \end{array}$$

On sait que, pour $w' \in B_w$, on a $w \sim w'$. Comme W est fini, et petit, et B décidable, il serait facile de calculer B_w. On peut aussi essayer de « deviner » des éléments de B_w permettant de prouver φ. Par exemple, posons :

$$\begin{array}{rcl} w^1 &=& ((a'_1,a'_2,a'_3),(v'_1,v'_2,v'_3)) \quad \hat{=} \quad ((a_1,a_2,a_3),(0,1,0)) \\ w^2 &=& ((a'_1,a'_2,a'_3),(v'_1,v'_2,v'_3)) \quad \hat{=} \quad ((a_1,a_2,a_3),(0,0,1)) \end{array}$$

On pourrait vérifier sans peine (ce serait un peu fastidieux à la main, mais c'est tout à fait possible si on dispose d'une implémentation de B) que $w_1, w_2 \in B_w$. On a :

$$\begin{array}{rcl} c(w) &=& \{(a_1,1),(a_2,0),(a_3,0)\} \\ c(w^1) &=& \{(a_1,0),(a_2,1),(a_3,0)\} \\ c(w^2) &=& \{(a_1,0),(a_2,0),(a_3,1)\} \end{array}$$

et, par conséquent :

$$\{c(w), c(w^1), c(w2)\}^\sharp \quad = \quad \{(a_1,\{0,1\}),(a_2,\{0,1\}),(a_3,\{0,1\})\} \in \varphi$$

On en déduit donc que φ est satisfaite en w. $\square$

Voyons maintenant comment appliquer cette approche dans le cas général. Considérons un système observé $\mathcal{W} = (W, \sim, c, 2^C \underset{\flat}{\overset{\sharp}{\rightleftharpoons}} A)$, une propriété d'opacité $\varphi \subseteq A$ et $w \in W$. On suppose que $(W, \sim, c)$ est défini au moyen d'un processus $P \in \mathrm{Proc}_{\mathrm{Fin}}$, de variables libres $x_1, \ldots, x_n$, en posant :

$$w_1 \sim w_2 \quad \Leftrightarrow \quad P[w_1/(x_1, \ldots, x_n)] \underset{\mathrm{tst}}{\sim} P[w_2/(x_1, \ldots, x_n)]$$

(les éléments w' de W tenant lieu de valeurs pour instancier les variables $x_1, \ldots, x_n$). On cherche à savoir si $\mathcal{W}, w \models \varphi$, autrement dit si $\overline{c}(w)^\sharp \in \varphi$. D'après ce qui précède, on a :

$$B_w \;\hat{=}\; \{w' \in W \mid B(P[w/(x_1, \ldots, x_n)], P[w'/(x_1, \ldots, x_n)]\} \;\subseteq\; \{w' \in W \mid w \sim w'\}$$

Supposons qu'il existe une énumération (non nécessairement exhaustive) des éléments de B_w, de la forme :

$$\{w^1, w^2, \ldots\} \;\subseteq\; B_w$$

on pose alors, pour $i \geq 0$:

$$a^i(w) \quad \hat{=} \quad \{c(w^1), \ldots, c(w^i)\}^\sharp$$

de sorte que :

$$\exists i \geq 0.\ a^i(w) \in \varphi \quad \Rightarrow \quad \mathcal{W}, w \models \varphi$$

Comme le prédicat unaire $B_w \subseteq W$ est décidable, on obtient, dans le cas où W est récursivement énumérable, un semi-algorithme, nous permettant de certifier que $\mathcal{W}, w \models \varphi$. Il consiste à énumérer les éléments de W, sélectionner ceux qui appartiennent à B_w et calculer les éléments correspondant de la suite $(a^i(w))_{i \geq 0}$. Ceci n'est cependant pas applicable en pratique. Un autre moyen, plus efficace mais demandant une plus grande participation de la part de celui qui cherche à montrer une propriété d'opacité, consiste à essayer de « deviner » un certain nombre $w^1, \ldots, w^r$ d'éléments de W dont on conjecture qu'ils appartiennent à B_w, à vérifier au moyen de l'algorithme qu'ils sont effectivement dans B_w et à calculer $a \;\hat{=}\; \{c(w^1), \ldots, c(w^r)\}^\sharp$, en espérant que $a \in \varphi$ (cette approche a été mise en pratique dans l'exemple précédent, nous donnerons plus de détails et l'appliquerons également au dîner des cryptographes au chapitre 18).

Pour rendre cette approche moins dépendante de l'utilisateur, nous proposons d'utiliser l'algorithme de décision de la h-bisimilarité gardée et de le transformer de telle manière qu'il retourne lui-même (sous une certaine forme) des éléments de W qui sont dans B_w.

Utilisation de contraintes. Dans le chapitre suivant, nous allons montrer comment associer à $w \in W$ une contrainte Γ_w sur W (en fait, une contrainte sur les variables $x_1, \ldots, x_n$) telle que les solutions w' de Γ_w appartiennent à B_w. L'intérêt est de produire directement des éléments de B_w et d'éviter ainsi d'énumérer tout W pour sélectionner les éléments qui conviennent. Pour des raisons techniques (en fait, éviter les conjonctions et disjonctions infinies, ou mal définies), nous nous contenterons de le faire pour la h-similarité gardée, définie comme la réunion de toutes les h-simulations gardées. L'algorithme correspondant à cette dernière relation est le même que celui de la h-bisimilarité gardée, à ceci près que les tests réalisés dans S_2^h ne sont pas effectués (voir la figure 16.6.2). De la même manière que pour la h-bisimilarité gardée, on pose :

$$S(P_1, P_2) \quad \hat{=} \quad S^h((\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \{n\}, \varnothing), P_1, P_2)$$

$$
\begin{aligned}
S^h((fr, th), P_1, P_2) \ &\hat{=} \ \bigwedge_{\substack{n \in fr \\ \alpha \in \{n, \bar{n}, \tau\} \\ P_1 \xrightarrow{\alpha} P_1'}} U^h((fr, th), \alpha, P_1, P_1', P_2) \\
U^h((fr, th), \alpha, P_1, P_1', P_2) \ &\hat{=} \ \bigvee_{P_2 \xrightarrow{\alpha} P_2'} T^h((fr, th), \alpha, P_1, P_1', P_2, P_2') \\
T^h((fr, th), \tau, P_1, P_1', P_2, P_2') \ &\hat{=} \ S^h((fr, th), P_1', P_2') \\
T^h((fr, th), n, P_1, (x).P_1', P_2, (x).P_2') \ &\hat{=} \\
&\bigwedge_{\substack{\{\vec{n}\} \subseteq \mathrm{Name}, |\vec{n}| \leq 2^h \\ \{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing \\ (fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2 \\ \mathrm{h}(m_1) \leq h, \mathrm{h}(m_2) \leq h}} S^h((fr \cup \{\vec{n}\}, th), P_1'[m_1/x], P_2'[m_2/x]) \\
T^h((fr, th), \bar{n}, P_1, (\nu \vec{n_1}) \langle m_1 \rangle . P_1', P_2, (\nu \vec{n_2}) \langle m_2 \rangle . P_2') \ &\hat{=} \\
&\bigvee_{(fr', th') \in \mathrm{Norm}(fr, th \cup \{(m_1, m_2)\})} S^h((fr', th'), P_1', P_2')
\end{aligned}
$$

FIG. 16.6.2 – *Description de la h-similarité gardée*

où n est un nom qui n'est ni dans $\mathrm{fn}(P_1)$, ni dans $\mathrm{fn}(P_2)$ et $h \hat{=} \max(\mathrm{h}(P_1), \mathrm{h}(P_2))$. Dans le chapitre suivant, nous expliquerons tout d'abord comment réaliser, en général, une telle transformation sur des prédicats définis au moyen d'énoncés logiques (comme c'est le cas ici), puis nous appliquerons ces techniques au cas de la h-similarité gardée. Nous disposerons alors d'une fonction qui, à $w \in W$, associera une contrainte Γ_w telle que pour toute solution $w' \in W$ de Γ_w, on ait $S(P[w/(x_1, \ldots, x_n)], P[w'/(x_1, \ldots, x_n)])$. Si on parvient à obtenir une énumération $w'^1, w'^2, \ldots$ des solutions de Γ_w, il ne restera plus qu'à utiliser l'algorithme de décision de B pour en extraire une suite $w^1, w^2, \ldots$ d'éléments de B_w. Le recours à l'algorithme B, dans la dernière phase, est nécessaire. On sait en effet que chaque w'^i conduit à un processus similaire à w, mais rien ne prouve qu'ils sont bisimilaires[1].

[1]En fait, même si P_1 est similaire à P_2 et P_2 similaire à P_1, il se peut que P_1 et P_2 ne soient pas bisimilaires.

Chapitre 17

Utilisation de contraintes

Nous supposons ici fixé un processus $P \in \mathrm{Proc}_{\mathrm{Fin}}$, de variables libres $x_1, \ldots, x_n$. Soient $v_1, \ldots, x_n$ (respectivement $v'_1, \ldots, v'_n$) des valeurs pouvant instancier ces variables. Le chapitre précédent a montré comment décider si $P[v_1/x_1, \ldots, v_n/x_n]$ et $P[v'_1/x_1, \ldots, v'_n/x_n]$ sont équivalents pour la relation de h-bisimilarité gardée (avec $h \mathrel{\hat{=}} \mathrm{h}(P)$), ceci au moyen d'un énoncé noté B, dont on a montré qu'il était décidable. Nous avons aussi expliqué que, dans le but de vérifier des propriétés d'opacité, on s'intéresse particulièrement à calculer, à $(v_1, \ldots, v_n)$ fixé, un sous-ensemble de :

$$S_{(v_1,\ldots,v_n)} \quad \hat{=} \quad \{(v'_1, \ldots, v'_n) \mid S(P[v_1/x_1, \ldots, v_n/x_n], P[v'_1/x_1, \ldots, v'_n/x_n])\}$$

Pour cela, nous allons montrer dans ce chapitre comment obtenir, à partir de l'algorithme de décision de la h-similarité gardée, une fonction qui, à $(v_1, \ldots, v_n)$, associera une contrainte $\Gamma_{(v_1,\ldots,v_n)}$ sur les variables $x_1, \ldots, x_n$ dont les solutions $(v'_1, \ldots, v'_n)$ seront des éléments de $S_{(v_1,\ldots,v_n)}$.

Pour répondre à cette question, nous nous plaçons dans un cadre plus général. On suppose définis deux ensembles D et $D^{\intercal}$. Le premier sera dit clos et le second ouvert (on peut imaginer, par exemple, que $D^{\intercal}$ est l'ensemble des termes (avec variables) sur une certaine signature et que D est le sous-ensemble de $D^{\intercal}$ constitué des termes clos). On supposera que D est un sous-ensemble de $D^{\intercal}$ et qu'il existe un certain ensemble Subst, dont les éléments seront appelés des substitutions, qui agit sur $D^{\intercal}$ au moyen d'une application :

$$\begin{array}{lcl} D^{\intercal} \times \mathrm{Subst} & \to & D \\ (d, \sigma) & \mapsto & d\sigma \end{array}$$

surjective et telle que, quels que soient $d \in D$ et $\sigma \in \mathrm{Subst}$, on ait $d\sigma = d$. Dans ce cadre, la question précédente se formalise de la manière suivante : on considère un prédicat $\mathcal{D} \subseteq D$ (il s'agit simplement d'un sous-ensemble de D, que l'on peut également voir comme une relation unaire) et on cherche à construire une fonction qui, à $d \in D^{\intercal}$, associe les substitutions $\sigma \in \mathrm{Subst}$ telles que $d\sigma \in \mathcal{D}$. Nous représenterons de telles substitutions au moyen de contraintes qui seront supposées appartenir à un ensemble Cstr. Le lien entre substitutions et contraintes sera donné au moyen d'une relation binaire entre Subst et Cstr, appelée relation de satisfaction et notée $\models$. La fonction représentant $\mathcal{D}$ devra alors être une fonction de $\mathcal{D}^{\intercal}$ vers Cstr telle que, si $\Gamma = \mathcal{D}^{\intercal}(d)$ et $\sigma \models \Gamma$ (avec $d \in D^{\intercal}$ et $\sigma \in \mathrm{Subst}$), alors $d\sigma \in \mathcal{D}$. $\mathcal{D}^{\intercal}$ nous fournira ainsi un sous-ensemble de $\mathcal{D}$, ce qui sera suffisant pour notre propos. On peut tout de même vouloir aussi que tous les éléments de $\mathcal{D}$ puissent être obtenus de cette façon, ce qui nous conduira à considérer des notions plus fortes de représentations. Ce chapitre est organisé en deux sections : la première est consacrée à la définition des différentes notions de représentations et à leur construction et la seconde à l'application de ces résultats à la h-similarité gardée.

On suppose fixés, une fois pour toutes, les ensembles Subst et Cstr, ainsi que la relation $\models$. Nous supposerons que l'ensemble Cstr est muni des opérations suivantes :

– une opération de conjonction (respectivement disjonction) arbitraire qui, à toute famille de contraintes $(\Gamma_i)_{i\in I}$, associe une contrainte $\bigwedge_{i\in I}\Gamma_i$ (respectivement $\bigvee_{i\in I}\Gamma_i$) telle que, quelle que soit $\sigma \in \mathrm{Subst}$:

$$\sigma \models \bigwedge_{i\in I}\Gamma_i \quad\Leftrightarrow\quad \sigma \models \Gamma_i \text{ quel que soit } i \in I$$

(respectivement :

$$\sigma \models \bigvee_{i\in I}\Gamma_i \quad\Leftrightarrow\quad \sigma \models \Gamma_i \text{ pour un certain } i \in I)$$

– une opération de négation (elle ne sera utilisé qu'à la fin de la première section) qui, à toute contrainte Γ, associe une contrainte $\neg\Gamma$ telle que, quelle que soit $\sigma \in \mathrm{Subst}$:

$$\sigma \models \neg\Gamma \quad\Leftrightarrow\quad \sigma \not\models \Gamma$$

17.1 Représentations d'un prédicat

Définition – 17.1.1

> *Soient D et $D^\intercal$ deux ensembles et une application $a : D^\intercal \times \mathrm{Subst} \to D$. On dit que $D^\intercal$ représente D (*via *a) si les conditions suivantes sont satisfaites :*
>
> 1. *D est un sous-ensemble de $D^\intercal$;*
> 2. *Quel que soit $d \in D$, $a(d, \sigma) = d$.*

Dans la suite, lorsqu'un couple d'ensembles sera noté sous la forme $(D, D^\intercal)$, on supposera toujours que $D^\intercal$ représente D, *via* une application a sous-entendue. Pour $d \in D^\intercal$ et $\sigma \in \mathrm{Subst}$, $a(d,\sigma)$ sera noté plus simplement $d\sigma$. Cette application est nécessairement surjective, puisque si $d \in D \subseteq D^\intercal$ et $\sigma \in \mathrm{Subst}$ est quelconque, on a $d\sigma = d$. Remarquons aussi que, si $I^\intercal$ représente I et $D^\intercal$ représente D, alors $I^\intercal \times D^\intercal$ représente $I \times D$, *via* l'application qui, à $(i,d) \in I^\intercal \times D^\intercal$ et $\sigma \in \mathrm{Subst}$ associe $(i\sigma, d\sigma)$. Nous nous intéressons maintenant aux représentations d'un prédicat $\mathcal{D} \subseteq D$ par des fonctions qui, à $d \in D^\intercal$, associent une contrainte caractérisant les substitutions qui envoient d sur un élément de $\mathcal{D}$. Nous allons donner un paramètre supplémentaire à ces fonctions, sous la forme d'une contrainte également. Un couple (Γ, d) où Γ est une contrainte et $d \in D^\intercal$ peut être interprété comme la question : quelles sont les substitutions satisfaisant Γ qui envoient d sur un élément de $\mathcal{D}$? La contrainte Γ' retournée par la fonction correspond à la réponse : au moins les substitutions qui satisfont Γ' (en plus de Γ). De telles représentations permettent de retrouver un sous-ensemble du prédicat de départ $\mathcal{D}$. Nous qualifierons de complètes les représentations qui permettent de retrouver $\mathcal{D}$ tout entier et nous définirons un troisième type de représentations, intéressantes d'un point de vue technique (pour assurer la complétude d'autres représentations).

Définition – 17.1.2

Soient $\mathcal{D} \subseteq D$ *un prédicat et une fonction* $\mathcal{D}^{\intercal} : \mathrm{Cstr} \times D^{\intercal} \to \mathrm{Cstr}$. *On dit que :*

- $\mathcal{D}^{\intercal}$ *représente* $\mathcal{D}$ *si, quels que soient* $d \in D^{\intercal}$, $\sigma \in \mathrm{Subst}$ *et* $\Gamma \in \mathrm{Cstr}$ *tels que* $\sigma \models \Gamma \wedge \mathcal{D}^{\intercal}(\Gamma, d)$, *on a* $d\sigma \in \mathcal{D}$ *;*
- $\mathcal{D}^{\intercal}$ *représente complètement* $\mathcal{D}$ *si* $\mathcal{D}^{\intercal}$ *représente* $\mathcal{D}$ *et, de plus, quels que soient* $d \in D^{\intercal}$, $\sigma \in \mathrm{Subst}$ *et* $\Gamma \in \mathrm{Cstr}$ *tels que* $d\sigma \in \mathcal{D}$ *et* $\sigma \models \Gamma$, *il existe* $\sigma' \in \mathrm{Subst}$ *telle que* $\sigma' \models \Gamma \wedge \mathcal{D}^{\intercal}(\Gamma, d)$ *et* $d\sigma = d\sigma'$ *;*
- $\mathcal{D}^{\intercal}$ *représente proprement* $\mathcal{D}$ *si* $\mathcal{D}^{\intercal}$ *représente* $\mathcal{D}$ *et, de plus, quels que soient* $d \in D^{\intercal}$, $\sigma \in \mathrm{Subst}$ *et* $\Gamma \in \mathrm{Cstr}$ *tels que* $d\sigma \in \mathcal{D}$ *et* $\sigma \models \Gamma$, *on a* $\sigma \models \mathcal{D}^{\intercal}(\Gamma, d)$.

Remarque : On comprend mieux ces définitions au moyen des notations suivantes :

$$\begin{array}{rcl} A_{\Gamma,d} & \hat{=} & \{d\sigma \mid \sigma \in \mathrm{Subst}, \sigma \models \Gamma \text{ et } d\sigma \in \mathcal{D}\} \\ B_{\Gamma,d} & \hat{=} & \{d\sigma \mid \sigma \in \mathrm{Subst} \text{ et } \sigma \models \Gamma \wedge \mathcal{D}^{\intercal}(\Gamma, d)\} \end{array}$$

pour $\Gamma \in \mathrm{Cstr}$ et $d \in D^{\intercal}$. $A_{\Gamma,d}$ représente les instanciations de d par des substitutions satisfaisant Γ et qui aboutissent dans $\mathcal{D}$ et $B_{\Gamma,d}$ représente les instanciations de d par des substitutions satisfaisant Γ et $\mathcal{D}^{\intercal}(\Gamma, d)$. $\mathcal{D}^{\intercal}$ est alors une représentation (respectivement une représentation complète) de $\mathcal{D}$ si, et seulement si, quels que soient $\Gamma \in \mathrm{Cstr}$ et $d \in D^{\intercal}$, on a $B_{\Gamma,d} \subseteq A_{\Gamma,d}$ (respectivement $B_{\Gamma,d} = A_{\Gamma,d}$). Une représentation propre est une représentation complète pour laquelle on peut toujours choisir de prendre $\sigma' = \sigma$. $\square$

Nous allons maintenant expliquer comment construire des représentations de manière inductive. Un cas typique de telle construction est celui où on connaît des représentations de deux prédicats et où l'on cherche une représentation de leur conjonction (ou disjonction). La proposition qui suit permet de répondre à cette question dans un cas très général.

Proposition – 17.1.1

Soit $\mathcal{R}$ *un prédicat sur* $I \times D$ *représenté par* $\mathcal{R}^{\intercal} : \mathrm{Cstr} \times I^{\intercal} \times D^{\intercal} \to \mathrm{Cstr}$. *On pose :*

$$\begin{array}{rcl} \cap_I \mathcal{R} & \hat{=} & \{d \in D \mid \forall i \in I.\ (i, d) \in \mathcal{R}\} \\ \cup_I \mathcal{R} & \hat{=} & \{d \in D \mid \exists i \in I.\ (i, d) \in \mathcal{R}\} \end{array}$$

Alors, les fonctions :

$$\begin{array}{rccl} \wedge_I \mathcal{R}^{\intercal} & : & \mathrm{Cstr} \times D^{\intercal} & \to & \mathrm{Cstr} \\ & & (\Gamma, d) & \mapsto & \displaystyle\bigwedge_{i \in I^{\intercal}} \mathcal{R}^{\intercal}(\Gamma, i, d) \\ \vee_I \mathcal{R}^{\intercal} & : & \mathrm{Cstr} \times D^{\intercal} & \to & \mathrm{Cstr} \\ & & (\Gamma, d) & \mapsto & \displaystyle\bigvee_{i \in I^{\intercal}} \mathcal{R}^{\intercal}(\Gamma, i, d) \end{array}$$

sont des représentations de $\cap_I \mathcal{R}$ *et* $\cup_I \mathcal{R}$, *respectivement.*

Démonstration :

On pose :

$$\begin{array}{ll} \mathcal{P} \;\hat{=}\; \cap_I \mathcal{R} & \mathcal{Q} \;\hat{=}\; \cup_I \mathcal{R} \\ \mathcal{P}^\intercal \;\hat{=}\; \wedge_I \mathcal{R}^\intercal & \mathcal{Q}^\intercal \;\hat{=}\; \vee_I \mathcal{R}^\intercal \end{array}$$

Montrons que $\mathcal{P}^\intercal$ (respectivement $\mathcal{Q}^\intercal$) est une représentation de $\mathcal{P}$ (respectivement $\mathcal{Q}$). Soient pour cela $d \in D^\intercal$, $\sigma \in$ Subst et $\Gamma \in$ Cstr. Supposons tout d'abord que $\sigma \models \Gamma \wedge \mathcal{P}^\intercal(\Gamma, d)$, *i.e.* :

$$\sigma \models \Gamma \wedge \bigwedge_{i \in I^\intercal} \mathcal{R}^\intercal(\Gamma, i, d)$$

Soit $i \in I$ quelconque, on a $\sigma \models \Gamma \wedge \mathcal{R}^\intercal(\Gamma, i, d)$, d'où l'on déduit, puisque $\mathcal{R}^\intercal$ est une représentation de $\mathcal{R}$, que $(i\sigma, d\sigma) = (i, d\sigma) \in \mathcal{R}$. Ainsi, $(i, d\sigma) \in \mathcal{R}$ et ce, quel que soit $i \in I$, donc $d\sigma \in \mathcal{P}$. Si maintenant $\sigma \models \Gamma \wedge \mathcal{Q}^\intercal(\Gamma, d)$, *i.e.* :

$$\sigma \models \Gamma \wedge \bigvee_{i \in I^\intercal} \mathcal{R}^\intercal(\Gamma, i, d)$$

alors il existe $i \in I^\intercal$ tel que $\sigma \models \Gamma \wedge \mathcal{R}^\intercal(\Gamma, i, d)$. On en déduit que $(i\sigma, d\sigma) \in \mathcal{R}$ et, par conséquent, que $d\sigma \in \mathcal{Q}$. ■

Remarque : On rencontre souvent des disjonctions et conjonctions de la forme :

$$\bigcup_{i \in I, \mathcal{S}(i,d)} \mathcal{T}(i, d) \quad \text{et} \quad \bigcap_{i \in I, \mathcal{S}(i,d)} \mathcal{T}(i, d)$$

où $\mathcal{S}$ et $\mathcal{T}$ sont des prédicats sur $I \times D$. Ces disjonctions et conjonctions peuvent s'écrire respectivement :

$$\cup_I \mathcal{R}_1 \quad \text{et} \quad \cap_I \mathcal{R}_2$$

si on a posé :

$$\mathcal{R}_1 \;\hat{=}\; \mathcal{S} \cap \mathcal{T} \quad \text{et} \quad \mathcal{R}_2 \;\hat{=}\; ((I \times D) \setminus \mathcal{S}) \cup \mathcal{T}$$

Il s'agit donc d'un cas particulier d'application de la proposition précédente. □

Dans le cas de la conjonction, la représentation obtenue n'est, en général, pas complète, alors que la représentation de la disjonction l'est.

Proposition – 17.1.2

Avec les notations de la proposition 17.1.1, si $\mathcal{R}^\intercal$ est une représentation complète de $\mathcal{R}$, alors $\vee_I \mathcal{R}^\intercal$ est une représentation complète de $\cup_I \mathcal{R}$.

Démonstration :

Considérons $d \in D^\intercal$, $\sigma \in$ Subst et $\Gamma \in$ Cstr tels que $d\sigma \in \cup_I \mathcal{R}$ et $\sigma \models \Gamma$. Il existe alors $i \in I$ tel que $(i, d\sigma) = (i\sigma, d\sigma) \in \mathcal{R}$. Comme $\mathcal{R}^\intercal$ est une représentation complète de $\mathcal{R}$, il existe $\sigma' \in$ Subst telle que $\sigma' \models \Gamma \wedge \mathcal{R}^\intercal(\Gamma, i, d)$ et $(i\sigma, d\sigma) = (i, d\sigma) = (i, d\sigma') = (i\sigma', d\sigma')$ et, par conséquent, $\sigma' \models \Gamma \wedge \vee_I \mathcal{R}^\intercal(\Gamma, d)$. ■

En revanche, si on part de représentations propres, les représentations obtenues le sont aussi (même dans le cas d'une conjonction).

Proposition – 17.1.3

Avec les notations de la proposition 17.1.1, si $\mathcal{R}^\intercal$ est une représentation propre de $\mathcal{R}$, alors $\wedge_I \mathcal{R}^\intercal$ et $\vee_I \mathcal{R}^\intercal$ sont des représentations propres de $\cap_I \mathcal{R}$ et $\cup_I \mathcal{R}$, respectivement.

Démonstration :

Soient $d \in D^\intercal$, $\sigma \in \mathrm{Subst}$ et $\Gamma \in \mathrm{Cstr}$ tels que $\sigma \models \Gamma$. Supposons tout d'abord que $d\sigma \in \cap_I \mathcal{R}$. Alors, quel que soit $i \in I^\intercal$, on a $(i\sigma, d\sigma) \in \mathcal{R}$. Comme $\mathcal{R}^\intercal$ est supposée être une représentation propre de $\mathcal{R}$, on a $\sigma \models \Gamma \wedge \mathcal{R}^\intercal(\Gamma, i, d)$, donc $\sigma \models \Gamma \wedge (\wedge_I \mathcal{R}^\intercal(\Gamma, d))$. Supposons maintenant que $d\sigma \in \cap_I \mathcal{R}$. Il existe alors $i \in I$ tel que $(i, d\sigma) = (i\sigma, d\sigma) \in \mathcal{R}$. Comme $\mathcal{R}^\intercal$ est supposée être une représentation propre de $\mathcal{R}$, on a $\sigma \models \Gamma \wedge \mathcal{R}^\intercal(\Gamma, i, d)$, donc $\sigma \models \Gamma \wedge (\vee_I \mathcal{R}^\intercal(\Gamma, d))$. ■

Si on désire obtenir une représentation complète d'une conjonction, sans nécessairement passer par des représentations propres, on peut utiliser le résultat suivant (qui n'est valable que dans le cas de conjonctions finies). Noter la différence entre la conjonction définie ici et celle de la proposition 17.1.1.

Proposition – 17.1.4

Soient $\mathcal{D}_1$ et $\mathcal{D}_2$ deux prédicats sur D représentés (respectivement représentés complètement, représentés proprement) par $\mathcal{D}_1^\intercal$ et $\mathcal{D}_2^\intercal$. La fonction :

$$\begin{array}{lcll} \mathcal{D}_1^\intercal \wedge \mathcal{D}_2^\intercal & : & \mathrm{Cstr} \times D^\intercal & \rightarrow & \mathrm{Cstr} \\ & & (\Gamma, d) & \mapsto & \mathcal{D}_1^\intercal(\Gamma, d) \wedge \mathcal{D}_2^\intercal(\Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d), d) \end{array}$$

est alors une représentation (respectivement une représentation complète, une représentation propre) de $\mathcal{D}_1 \cap \mathcal{D}_2$.

Démonstration :

Posons $\mathcal{D} \mathrel{\hat{=}} \mathcal{D}_1 \cap \mathcal{D}_2$ et $\mathcal{D}^\intercal \mathrel{\hat{=}} \mathcal{D}_1^\intercal \wedge \mathcal{D}_2^\intercal$ et montrons tout d'abord que $\mathcal{D}^\intercal$ est une représentation de $\mathcal{D}$. Considérons pour cela $d \in D^\intercal$, $\sigma \in \mathrm{Subst}$ et $\Gamma \in \mathrm{Cstr}$ tels que $\sigma \models \Gamma \wedge \mathcal{D}^\intercal$, *i.e.* $\sigma \models \Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d) \wedge \mathcal{D}_2^\intercal(\Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d), d)$. On a en particulier $\sigma \models \Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d)$, donc $d\sigma \in \mathcal{D}_1$, et $\sigma \models (\Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d)) \wedge \mathcal{D}_2^\intercal(\Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d), d)$, donc $d\sigma \in \mathcal{D}_2$. On en déduit que $d\sigma \in \mathcal{D}$ et, par conséquent, que $\mathcal{D}^\intercal$ est une représentation de $\mathcal{D}$.

Supposons maintenant que $\mathcal{D}_1^\intercal$ et $\mathcal{D}_2^\intercal$ sont des représentations complètes et montrons que $\mathcal{D}^\intercal$ est complète. Supposons donc que $d\sigma \in \mathcal{D}$ et $\sigma \models \Gamma$. On a alors $d\sigma \in \mathcal{D}_1$ et, comme $\mathcal{D}_1^\intercal$ est complète, il existe σ' telle que $\sigma' \models \Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d)$ et $d\sigma = d\sigma'$. Comme $\mathcal{D}_2^\intercal$ est complète et $d\sigma' \in \mathcal{D}_2$, il existe σ'' telle que $\sigma'' \models \Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d) \wedge \mathcal{D}_2^\intercal(\Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d), d)$ et $d\sigma'' = d\sigma'$. Ainsi, $\sigma'' \models \Gamma \wedge \mathcal{D}^\intercal(\Gamma, d)$ et $d\sigma'' = d\sigma$, et $\mathcal{D}^\intercal$ est complète.

Finalement, supposons que $\mathcal{D}_1^\intercal$ et $\mathcal{D}_2^\intercal$ sont des représentations propres et montrons que c'est également le cas pour $\mathcal{D}^\intercal$. Si $d\sigma \in \mathcal{D}$ et $\sigma \models \Gamma$, alors $\sigma \models \mathcal{D}_1^\intercal(\Gamma, d)$ (car $\mathcal{D}_1^\intercal$ est une représentation propre et $d\sigma \in \mathcal{D}_1$) et $\sigma \models \mathcal{D}_2^\intercal(\Gamma \wedge \mathcal{D}_1^\intercal(\Gamma, d), d)$ (car $\mathcal{D}_2^\intercal$ est une représentation propre et $d\sigma \in \mathcal{D}_2$). On en déduit que $\mathcal{D}^\intercal$ est aussi une représentation propre. ■

Un autre cas usuel est celui où un prédicat $\mathcal{D}$ est défini sur D comme plus petit sous-ensemble de D satisfaisant un certain nombre de règles. Ces règles doivent être de la forme :

$$R = \frac{d_1 \in \mathcal{D} \quad \ldots \quad d_n \in \mathcal{D} \quad (d_1, \ldots, d_n) \in \mathcal{P}}{f^\intercal(d_1, \ldots, d_n) \in \mathcal{D}}$$

où $d_1, \ldots, d_n \in D^\intercal$, $\mathcal{P}$ est un prédicat défini sur D^n et $f^\intercal$ est une fonction de $D^{\intercal n}$ vers $D^\intercal$ représentant une fonction f de D^n vers D, au sens suivant : quels que soient $d_1, \ldots, d_n \in D^\intercal$

et $\sigma \in \mathrm{Subst}$, $(f^{\intercal}(d_1,\ldots,d_n))\sigma = f(d_1\sigma,\ldots,d_n\sigma)$. On dit que le prédicat $\mathcal{D} \subseteq D$ satisfait la règle R si, quelle que soit $\sigma \in \mathrm{Subst}$ telle que :

$$d_1\sigma \in \mathcal{D},\ldots,d_n\sigma \in \mathcal{D},(d_1\sigma,\ldots,d_n\sigma) \in \mathcal{P}$$

on a $(f^{\intercal}(d_1,\ldots,d_n))\sigma = f(d_1\sigma,\ldots,d_n\sigma) \in \mathcal{D}$. On dit que le prédicat satisfait un ensemble de règles si il satisfait chaque règle de l'ensemble. Il est clair que l'intersection (arbitraire) de prédicats satisfaisant un ensemble donné de règles satisfait encore cet ensemble de règles et ceci nous autorise à considérer le plus petit prédicat satisfaisant un ensemble de règles.

Proposition – 17.1.5

Soit $\mathcal{D}$ un prédicat défini sur D au moyen d'un ensemble de règles et supposons que l'on dispose d'une représentation $\mathcal{P}^{\intercal}$ de chaque prédicat $\mathcal{P}$ apparaissant dans ces règles. On associe à la règle R (cf. ce qui précède) la règle :

$$\frac{(\Gamma,d_1,\Gamma_1) \in (\mathcal{D}^{\intercal}) \quad (\Gamma\wedge\Gamma_1,d_2,\Gamma_2) \in (\mathcal{D}^{\intercal}) \quad \ldots \quad (\Gamma\wedge\Gamma_1\wedge\ldots\wedge\Gamma_{n-1},d_n,\Gamma_n) \in (\mathcal{D}^{\intercal})}{(\Gamma,f^{\intercal}(d_1,\ldots,d_n),\Gamma_1\wedge\ldots\wedge\Gamma_n\wedge\mathcal{P}^{\intercal}(\Gamma\wedge\Gamma_1\wedge\ldots\wedge\Gamma_n,(d_1,\ldots,d_n))) \in (\mathcal{D}^{\intercal})}$$

et on note $(\mathcal{D}^{\intercal}) \subseteq \mathrm{Cstr}\times\mathcal{D}^{\intercal}\times\mathrm{Cstr}$ la relation définie par les règles et $\mathcal{D}^{\intercal}$ la fonction :

$$\begin{array}{lcl} \mathrm{Cstr}\times D^{\intercal} & \to & \mathrm{Cstr} \\ (\Gamma,d) & \mapsto & \displaystyle\bigvee_{\Gamma' \ tq\ (\Gamma,d,\Gamma')\in(\mathcal{D}^{\intercal})} \Gamma' \end{array}$$

$\mathcal{D}^{\intercal}$ est alors une représentation de $\mathcal{D}$.

Démonstration :

Commençons par montrer que, si $(\Gamma,d,\Gamma') \in (\mathcal{D}^{\intercal})$ et $\sigma \models \Gamma\wedge\Gamma'$, alors $d\sigma \in \mathcal{D}$. On procède par induction sur la dérivation de $(\Gamma,d,\Gamma') \in (\mathcal{D}^{\intercal})$. Supposons que la règle :

$$\frac{(\Gamma,d_1,\Gamma_1) \in (\mathcal{D}^{\intercal}) \quad (\Gamma\wedge\Gamma_1,d_2,\Gamma_2) \in (\mathcal{D}^{\intercal}) \quad \ldots \quad (\Gamma\wedge\Gamma_1\wedge\ldots\wedge\Gamma_{n-1},d_n,\Gamma_n) \in (\mathcal{D}^{\intercal})}{(\Gamma,f^{\intercal}(d_1,\ldots,d_n),\Gamma_1\wedge\ldots\wedge\Gamma_n\wedge\mathcal{P}^{\intercal}(\Gamma\wedge\Gamma_1\wedge\ldots\wedge\Gamma_n,(d_1,\ldots,d_n))) \in (\mathcal{D}^{\intercal})}$$

soit la dernière appliquée et que $\sigma \models \Gamma\wedge\Gamma_1\wedge\ldots\wedge\Gamma_n\wedge\Gamma'$ (où on a posé $\Gamma' = \mathcal{P}^{\intercal}(\Gamma\wedge\Gamma_1\wedge\ldots\wedge\Gamma_n,(d_1,\ldots,d_n))$). On a par hypothèse d'induction $d_1\sigma,d_2\sigma,\ldots,d_n\sigma \in \mathcal{D}$. Comme $\mathcal{P}^{\intercal}$ est une représentation de $\mathcal{P}$, on a $(d_1,\ldots,d_n)\sigma = (d_1\sigma,\ldots,d_n\sigma) \in \mathcal{P}$. Par conséquent $f(d_1\sigma,\ldots,d_n\sigma) = (f^{\intercal}(d_1,\ldots,d_n))\sigma \in \mathcal{D}$, d'où le résultat.

Si maintenant $\sigma \models \Gamma\wedge\mathcal{D}^{\intercal}(\Gamma,d)$, c'est qu'il existe $(\Gamma,d,\Gamma') \in (\mathcal{D}^{\intercal})$ tel que $\sigma \models \Gamma\wedge\Gamma'$ et, d'après ce qui précède, $d\sigma \in \mathcal{D}$. ■

Pour finir, et bien que ce résultat ne soit pas utilisé par la suite, étudions comment prendre la négation d'une représentation.

Proposition – 17.1.6

Soit $\mathcal{D}$ un prédicat sur D, représenté proprement par $\mathcal{D}^{\intercal} : \mathrm{Cstr}\times D^{\intercal} \to \mathrm{Cstr}$. Alors la fonction :

$$\begin{array}{rlcl} \neg\mathcal{D}^{\intercal}: & \mathrm{Cstr}\times D^{\intercal} & \to & \mathrm{Cstr} \\ & (\Gamma,d) & \mapsto & \neg\mathcal{D}^{\intercal}(\Gamma,d) \end{array}$$

est une représentation propre de $\overline{\mathcal{D}} \hat{=} \{d \in D \mid d \notin \mathcal{D}\}$.

Démonstration :

Commençons par montrer que $\neg\mathcal{D}^\intercal$ représente $\overline{\mathcal{D}}$. Soient donc $d \in D^\intercal$, $\sigma \in$ Subst et $\Gamma \in$ Cstr tels que $\sigma \models \Gamma \wedge \neg\mathcal{D}^\intercal(\Gamma, d)$. On a alors $\sigma \models \Gamma$ et $\sigma \not\models \mathcal{D}^\intercal(\Gamma, d)$. Si on avait $d\sigma \in \mathcal{D}$, on aurait $\sigma \models \mathcal{D}^\intercal(\Gamma, d)$ (car $\mathcal{D}^\intercal$ est une représentation propre de $\mathcal{D}$), ce qui est impossible. Donc $d\sigma \notin \mathcal{D}$ et, par conséquent, $d\sigma \in \overline{\mathcal{D}}$. Montrons maintenant que cette représentation est propre. Supposons donc que $d\sigma \in \overline{\mathcal{D}}$ et $\sigma \models \Gamma$. Si on avait $\sigma \models \mathcal{D}^\intercal(\Gamma, d)$, on aurait $d\sigma \in \mathcal{D}$ (car $\mathcal{D}^\intercal$ représente $\mathcal{D}$), ce qui est impossible. Donc $\sigma \not\models \mathcal{D}^\intercal(\Gamma, d)$ et, par conséquent, $\sigma \models \neg\mathcal{D}^\intercal(\Gamma, d)$. ■

Ce dernier résultat met en valeur l'importance de la notion de représentation propre lorsque l'on cherche à obtenir des négations de représentations (vu la démonstration, il est clair qu'une représentation, même complète, de $\mathcal{D}$ ne permet, en général, pas d'obtenir une représentation de son complémentaire). C'est la raison pour laquelle, en pratique, nous n'utiliserons pas ce résultat. En effet, un grand nombre des représentations que nous utiliserons sera obtenu au moyen de la proposition 17.1.5, qui ne permet pas d'assurer que les représentations en question sont propres.

Quelques notations. La définition de représentations pour les différentes relations qui interviennent dans la h-similarité fait intervenir à la fois des notations propres au processus et des notations issues du calcul des représentations. Pour plus de clarté, nous proposons de maintenir une distinction entre ces deux mondes au moyen de la notation suivante : si $\mathcal{D} \subseteq D$ est un prédicat représenté par $\mathcal{D}^\intercal : \text{Cstr} \times D^\intercal \to \text{Cstr}$, le fait que $\Gamma' = \mathcal{D}^\intercal(\Gamma, d)$ sera noté :

$$\lfloor\Gamma\rfloor\ d \in \mathcal{D}^\intercal\ \lfloor\Gamma'\rfloor \quad \text{ou bien} \quad \lfloor\Gamma\rfloor\ \mathcal{D}^\intercal(d)\ \lfloor\Gamma'\rfloor$$

17.2 Application à la simulation

Nous utiliserons comme contraintes les énoncés obtenus en combinant les propositions atomiques suivantes (et leurs négations) au moyen de conjonctions et disjonctions :

$$\begin{array}{rcl}
\Gamma, \Gamma', \ldots \in \text{Cstr} & ::= & nx \in \text{fn}(P) \\
& | & nx \in \text{fv}(P) \\
& | & m \in \text{Name} \\
& | & m = m' \\
& | & m = \{_\}__ \\
& | & m = \{_\}_{m'} \\
& | & \neg\Gamma \\
& | & \Gamma \wedge \Gamma' \\
& | & \Gamma \vee \Gamma'
\end{array}$$

(où nx désigne un élément de $\text{Name} \cup \text{Var}$). Pour définir une relation de satisfaction entre substitutions et contraintes, il suffit de la définir sur les contraintes atomiques, les opérateurs logiques étant traités de la manière habituelle. On pose donc :

- $\sigma \models nx \in \text{fn}(P)$ si $nx\sigma \in \text{fn}(P\sigma)$ (autrement dit si l'image de nx par σ apparaît dans les noms libres de $P\sigma$) ;
- $\sigma \models nx \in \text{fv}(P)$ si $nx\sigma \in \text{fv}(P\sigma)$ (autrement dit si l'image de nx par σ apparaît dans les variables libres de $P\sigma$) ;
- $\sigma \models m \in \text{Name}$ si $m\sigma \in \text{Name}$;

– $\sigma \models m = m'$ si $m\sigma = m'\sigma$;
– $\sigma \models m = \{\_\}_\_$ si $m\sigma$ est un message chiffré ;
– $\sigma \models m = \{\_\}_{m'}$ si $m\sigma$ est un message chiffré par $m'\sigma$ (ce qui implique que $m'\sigma$ est une clé).

Signalons quelques abréviations que nous utiliserons dans la suite :
– pour un ensemble de noms (ou variables) $\{nx_1, \ldots, nx_n\}$ et un processus P, on pose :

$$\{nx_1, \ldots, nx_n\} \cap \mathrm{fn}(P) = \varnothing \quad \hat{=} \quad \neg nx_1 \in \mathrm{fn}(P) \wedge \ldots \wedge \neg nx_n \in \mathrm{fn}(P)$$

– pour un nom ou variable nx et un message m, on pose :

$$nx \in \mathrm{fn}(m) \quad \hat{=} \quad nx \in \mathrm{fn}([m = m].0)$$

Munis de cet ensemble de contraintes, nous allons définir, pas à pas, une représentation de la relation de simulation (figure 16.6.2). Pour cela, nous allons successivement reprendre les différents objets définis au cours des chapitres 15 et 16 (environnements, agents, relation de transition et simulation) et expliquer quelles sont les représentations choisies et comment en construire d'autres.

Environnements. Nous représentons les environnements au moyen d'éléments de l'ensemble $\mathrm{Env}^\intercal$, constitué des couples (fr, th) où fr est un sous-ensemble fini de Name et th est un ensemble fini de couples de messages (m_1, m_2) où m_1 est clos (mais pas nécessairement m_2). Ce choix est motivé par le fait que, lors du calcul de $S(P_1, P_2)$, nous connaîtrons parfaitement le processus P_1 mais seulement une représentation de P_2. Dans un couple de messages (m_1, m_2) apparaissant dans une représentation de l'environnement, m_1 sera donc clos (puisqu'il provient de P_1) mais pas nécessairement m_2. Nous utilisons la proposition 17.1.5 (page 196) pour obtenir, à partir de la définition de la relation de réduction sur les environnements (définition 16.4.1, page 180), une représentation de cette relation, notée $\_ \rightarrow^\intercal \_$ et définie à la figure 17.2.1. Conformément à la proposition 17.1.5, on a $\_ \rightarrow^\intercal \_ \subseteq \mathrm{Cstr} \times \mathrm{Env}^\intercal \times \mathrm{Env}^\intercal \times \mathrm{Cstr}$.

$$\frac{}{\lfloor\Gamma\rfloor \ (fr, th \cup \{(n, m)\}) \rightarrow^\intercal (fr \cup \{n\}, th) \ \lfloor n = m \rfloor}$$

$$\frac{}{\lfloor\Gamma\rfloor \ (fr \cup \{n\}, th \cup \{(\{m_1\}_n, \{m_2\}_{n'})\}) \rightarrow^\intercal (fr \cup \{n\}, th \cup \{(m_1, m_2)\}) \ \lfloor n = n' \rfloor}$$

$$\frac{}{\lfloor\Gamma\rfloor \ (fr, th \cup \{(\langle m_1, m_1'\rangle, \langle m_2, m_2'\rangle)\}) \rightarrow^\intercal (fr, th \cup \{(m_1, m_2), (m_1', m_2')\}) \ \lfloor \mathrm{true} \rfloor}$$

FIG. 17.2.1 – *Représentation de* $\_ \rightarrow \_ \subseteq \mathrm{Env} \times \mathrm{Env}$

Nous utilisons ensuite directement la définition 16.1.2 (page 174) de consistance d'un environnement pour en déduire une représentation du prédicat $\mathrm{cons} \subseteq \mathrm{Env}$ au moyen d'une fonction $\mathrm{cons}^\intercal$ de $\mathrm{Cstr} \times \mathrm{Env}^\intercal$ vers Cstr (voir la figure 17.2.2).

Munis de ces deux relations, nous pouvons les combiner de manière à former une représentation de $\mathrm{Norm} \subseteq \mathrm{Env} \times 2^{\mathrm{Env}}$. Il suffit pour cela, partant de $e \in \mathrm{Env}^\intercal$ et $\Gamma \in \mathrm{Cstr}$ de réduire le couple (Γ, e) à l'aide de la relation $\_ \rightarrow^\intercal \_$ tout en propageant les contraintes. On note $f(\Gamma, e)$

$$
\begin{aligned}
\mathrm{cons}^{\mathsf{T}}(\Gamma, (fr, th)) \quad \hat{=} \quad & \bigwedge_{(m_1,m_2)\in th} m_1 = \{\_\}_\_ \wedge m_2 = \{\_\}_\_ \\
\wedge & \bigwedge_{(m_1,m_2)\in th, n\in fr} \neg m_1 = \{\_\}_n \wedge \neg m_2 = \{\_\}_n \\
\wedge & \bigwedge_{(m_1,m_2),(m_1',m_2')\in th} m_1 = m_1' \Leftrightarrow m_2 = m_2'
\end{aligned}
$$

FIG. 17.2.2 – *Représentation de* cons $\subseteq$ Env

l'ensemble des formes normales ainsi obtenues (il s'agit de couples $(e', \Gamma') \in \mathrm{Env}^{\mathsf{T}} \times \mathrm{Cstr}$) et on pose ensuite :

$$
\mathrm{Norm}^{\mathsf{T}}(\Gamma, e) \quad \hat{=} \quad \{(e', \Gamma' \wedge \Gamma'') \mid (e', \Gamma') \in f(\Gamma, e) \text{ et } \Gamma'' = \mathrm{cons}^{\mathsf{T}}(\Gamma \wedge \Gamma', e')\}
$$

Agents. Nous utiliserons des agents (possédant éventuellement des variables libres) pour représenter les agents clos. Il nous faut maintenant, sur cette représentation symbolique, définir des représentations pour les notions de restriction d'un agent, mise en parallèle avec un processus et interaction de deux agents de la définition 15.1.3 (page 165). Ces différentes opérations sont définies dans la figure 17.2.3. La principale différence avec la définition 15.1.3 tient dans les conditions à respecter entre x, R, n' et $\vec{n}$, qui se trouvent maintenant dans les contraintes.

$$
\begin{aligned}
\lfloor\Gamma\rfloor\ (\nu n').((x).P) \quad &=^{\mathsf{T}} \quad (x).((\nu n').P)\ \lfloor \mathrm{true} \rfloor \\
\lfloor\Gamma\rfloor\ R \parallel ((x).P) \quad &=^{\mathsf{T}} \quad (x).(R \parallel P)\ \lfloor \neg x \in \mathrm{fv}(R) \rfloor \\
\lfloor\Gamma\rfloor\ ((x).P) \parallel R \quad &=^{\mathsf{T}} \quad (x).(P \parallel R)\ \lfloor \neg x \in \mathrm{fv}(R) \rfloor \\
\lfloor\Gamma\rfloor\ (\nu n').((\nu \vec{n})\langle m \rangle.Q) \quad &=^{\mathsf{T}} \quad (\nu n', \vec{n})\langle m \rangle.Q\ \lfloor n' \in \mathrm{fn}(m) \wedge \neg n' \in \{\vec{n}\} \rfloor \\
\lfloor\Gamma\rfloor\ (\nu n').((\nu \vec{n})\langle m \rangle.Q) \quad &=^{\mathsf{T}} \quad (\nu \vec{n})\langle m \rangle.((\nu n').Q)\ \lfloor \neg n \in \mathrm{fn}(m) \wedge \neg n \in \{\vec{n}\} \rfloor \\
\lfloor\Gamma\rfloor\ R \parallel ((\nu \vec{n})\langle m \rangle.Q) \quad &=^{\mathsf{T}} \quad (\nu \vec{n})\langle m \rangle.(R \parallel Q)\ \lfloor \vec{n} \cap \mathrm{fn}(R) = \varnothing \rfloor \\
\lfloor\Gamma\rfloor\ ((\nu \vec{n})\langle m \rangle.Q) \parallel R \quad &=^{\mathsf{T}} \quad (\nu \vec{n})\langle m \rangle.(Q \parallel R)\ \lfloor \vec{n} \cap \mathrm{fn}(R) = \varnothing \rfloor \\
\\
\lfloor\Gamma\rfloor\ (x).P \,@ (\nu \vec{n})\langle m \rangle.Q \quad &=^{\mathsf{T}} \quad (\nu \vec{n}).(P[m/x] \parallel Q)\ \lfloor \vec{n} \cap \mathrm{fn}(P) = \varnothing \rfloor \\
\lfloor\Gamma\rfloor\ (\nu \vec{n})\langle m \rangle.Q \,@ (x).P \quad &=^{\mathsf{T}} \quad (\nu \vec{n}).(Q \parallel P[m/x])\ \lfloor \vec{n} \cap \mathrm{fn}(P) = \varnothing \rfloor
\end{aligned}
$$

FIG. 17.2.3 – *Représentation des opérations sur les agents*

Relations de réduction et transition. Nous utiliserons des processus (possédant éventuellement des variables libres) pour représenter les processus clos. La figure 17.2.4 décrit une représentation possible pour la relation de réduction en un pas entre processus (définition 15.1.1, page 164). Nous appliquons ensuite la proposition 17.1.5 (page 196) pour obtenir une représentation de la relation de transition entre processus et agents (définition 15.1.4, page 165), donnée à la figure 17.2.5.

$$
\begin{array}{rcl}
\lfloor\Gamma\rfloor \ \text{case } m \text{ of } \langle x,y\rangle.P' & \rhd^{\mathsf{T}} & P' \ \lfloor m = \langle x,y\rangle\rfloor \\
\lfloor\Gamma\rfloor \ \text{case } m \text{ of } \{x\}_n.P' & \rhd^{\mathsf{T}} & P' \ \lfloor m = \{x\}_n\rfloor \\
\lfloor\Gamma\rfloor \ [m = m'].P' & \rhd^{\mathsf{T}} & P' \ \lfloor m = m'\rfloor
\end{array}
$$

FIG. 17.2.4 – *Représentation de* $\rhd \subseteq \mathrm{Proc} \times \mathrm{Proc}$

$$
\frac{}{\lfloor\Gamma\rfloor \ n(x).P \xrightarrow{n}{}^{\mathsf{T}} (x).P \ \lfloor\text{true}\rfloor}
\qquad
\frac{}{\lfloor\Gamma\rfloor \ \overline{n}\langle m\rangle.P \xrightarrow{\bar{n}}{}^{\mathsf{T}} (\nu)\langle m\rangle.P \ \lfloor\text{true}\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{n}{}^{\mathsf{T}} F \ \lfloor\Gamma_1\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\rfloor \ Q \xrightarrow{\bar{n'}}{}^{\mathsf{T}} C \ \lfloor\Gamma_2\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\wedge\Gamma_2\rfloor \ F \,@\, C =^{\mathsf{T}} A \ \lfloor\Gamma_3\rfloor}{\lfloor\Gamma\rfloor \ P \parallel Q \xrightarrow{\tau}{}^{\mathsf{T}} A \ \lfloor\Gamma_1\wedge\Gamma_2\wedge\Gamma_3\wedge n = n'\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{\bar{n}}{}^{\mathsf{T}} C \ \lfloor\Gamma_1\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\rfloor \ Q \xrightarrow{n'}{}^{\mathsf{T}} F \ \lfloor\Gamma_2\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\wedge\Gamma_2\rfloor \ C \,@\, F =^{\mathsf{T}} A \ \lfloor\Gamma_3\rfloor}{\lfloor\Gamma\rfloor \ P \parallel Q \xrightarrow{\tau}{}^{\mathsf{T}} A \ \lfloor\Gamma_1\wedge\Gamma_2\wedge\Gamma_3\wedge n = n'\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma'\rfloor}{\lfloor\Gamma\rfloor \ P + Q \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma'\rfloor}
\qquad
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma'\rfloor}{\lfloor\Gamma\rfloor \ Q + P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma'\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma_1\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\rfloor \ A \parallel Q =^{\mathsf{T}} A' \ \lfloor\Gamma_2\rfloor}{\lfloor\Gamma\rfloor \ P \parallel Q \xrightarrow{\alpha}{}^{\mathsf{T}} A' \ \lfloor\Gamma_1\wedge\Gamma_2\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma_1\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\rfloor \ Q \parallel A =^{\mathsf{T}} A' \ \lfloor\Gamma_2\rfloor}{\lfloor\Gamma\rfloor \ Q \parallel P \xrightarrow{\alpha}{}^{\mathsf{T}} A' \ \lfloor\Gamma_1\wedge\Gamma_2\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma_1\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\rfloor \ (\nu n).A =^{\mathsf{T}} A' \ \lfloor\Gamma_2\rfloor}{\lfloor\Gamma\rfloor \ (\nu n).P \xrightarrow{\alpha}{}^{\mathsf{T}} A' \ \lfloor\Gamma_1\wedge\Gamma_2\wedge\neg\alpha = n\wedge\neg\alpha = \bar{n}\rfloor}
$$

$$
\frac{\lfloor\Gamma\rfloor \ P \rhd^{\mathsf{T}} Q \ \lfloor\Gamma_1\rfloor \quad \lfloor\Gamma\wedge\Gamma_1\rfloor \ Q \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma_2\rfloor}{\lfloor\Gamma\rfloor \ P \xrightarrow{\alpha}{}^{\mathsf{T}} A \ \lfloor\Gamma_1\wedge\Gamma_2\rfloor}
$$

FIG. 17.2.5 – *Représentation de* $\rightarrow \subseteq \mathrm{Proc} \times \mathrm{Act} \times \mathrm{Agent}$

Simulation. La définition de la h-simulation gardée (figure 16.6.2, page 189) combine les relations dont on a précédemment défini des représentations au moyen de disjonctions et conjonctions. Pour en obtenir une représentation, il suffit donc de lui appliquer le résultat des propositions 17.1.1 (page 193) et 17.1.4 (uniquement dans le cas de conjonctions finies). Voyons comment procéder en pratique.

Notons tout d'abord que, parmi les deux processus P_1 et P_2 passés en paramètres à S^h, seul P_2 sera supposé contenir des variables (P_1 sera connu). C'est sur ces variables que l'on cherchera une contrainte Γ de sorte que, si Γ est satisfaite, alors P_1 et P_2 sont similaires. Les représentations que nous allons définir ne concernent donc pas le processus P_1. C'est pourquoi nous commençons par réécrire la définition de S^h en changeant le statut du paramètre P_1 (voir la figure 17.2.6). Il s'agit de remplacer la relation $S^h \subseteq \mathrm{Env} \times \mathrm{Proc} \times \mathrm{Proc}$ par une famille $(S^h[P_1])_{P_1 \in \mathrm{Proc}}$ où, pour chaque $P_1 \in \mathrm{Proc}$, $S^h[P_1] \subseteq \mathrm{Env} \times \mathrm{Proc}$ est une relation. C'est de cette relation dont nous allons chercher une représentation. Il est en est de même pour les relations $U^h \subseteq \mathrm{Env} \times \mathrm{Act} \times \mathrm{Proc} \times \mathrm{Agent} \times \mathrm{Proc}$ qui est remplacée par une famille $(U^h[\alpha, P_1, P_1'])_{\alpha, P_1, P_1'}$ et $T^h \subseteq \mathrm{Env} \times \mathrm{Act} \times \mathrm{Proc} \times \mathrm{Agent} \times \mathrm{Proc} \times \mathrm{Agent}$ qui est remplacée par une famille $(T^h[\alpha, P_1, P_1'])_{\alpha, P_1, P_1'}$. Supposons donc $P_1 \in \mathrm{Proc}$ clos fixé et cherchons une

$$
\begin{aligned}
S^h[P_1]((fr, th), P_2) \;&\hat{=}\; \bigwedge_{\substack{n \in fr \\ \alpha \in \{n, \bar{n}, \tau\} \\ P_1 \xrightarrow{\alpha} P_1'}} U^h[\alpha, P_1, P_1']((fr, th), P_2) \\
U^h[\alpha, P_1, P_1']((fr, th), P_2) \;&\hat{=}\; \bigvee_{P_2 \xrightarrow{\alpha} P_2'} T^h[\alpha, P_1, P_1']((fr, th), P_2, P_2') \\
T^h[\tau, P_1, P_1']((fr, th), P_2, P_2') \;&\hat{=}\; S^h[P_1']((fr, th), P_2') \\
T^h[n, P_1, (x).P_1']((fr, th), P_2, (x).P_2') \;&\hat{=} \\
&\bigwedge_{\substack{\{\vec{n}\} \subseteq \mathrm{Name}, |\vec{n}| \leq 2^h \\ \{\vec{n}\} \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \cup \mathrm{fn}(e_1) \cup \mathrm{fn}(e_2)) = \varnothing \\ (fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2 \\ \mathrm{h}(m_1) \leq h, \mathrm{h}(m_2) \leq h}} S^h[P_1'[m_1/x]]((fr \cup \{\vec{n}\}, th), P_2'[m_2/x]) \\
T^h[\bar{n}, P_1, (\nu \vec{n_1})\langle m_1 \rangle.P_1']((fr, th), P_2, (\nu \vec{n_2})\langle m_2 \rangle.P_2') \;&\hat{=} \\
&\bigvee_{(fr', th') \in \mathrm{Norm}(fr, th \cup \{(m_1, m_2)\})} S^h[P_1']((fr', th'), P_2')
\end{aligned}
$$

FIG. 17.2.6 – *Description alternative de la h-similarité gardée*

représentation de $S^h[P_1]$. Rappelons que les représentations d'environnements sont des couples (fr, th) où fr est un ensemble fini de noms et th est un ensemble fini de couples de messages (m_1, m_2) où m_1 est clos. Ainsi, si $e = (fr, th) \in \mathrm{Env}^{\mathsf{T}}$, l'ensemble :

$$\{(\alpha, P_1') \mid \exists n \in fr.\ \alpha \in \{n, \bar{n}, \tau\} \text{ et } P_1 \xrightarrow{\alpha} P_1'\}$$

est fini, on le note $\{(\alpha^1, P_1^1), \ldots, (\alpha^r, P_1^r)\}$. On a alors :

$$S^h[P_1](e, P_2) \;=\; \bigwedge_{i=1}^{r} U^h[\alpha^i, P_1, P_1^i](e, P_2)$$

Connaissant une représentation $U^h[\alpha, P_1, P_1']^\intercal$ de chaque $U^h[\alpha, P_1, P_1']$, il est alors facile d'en déduire une représentation $S^h[P_1]^\intercal$ de $S^h[P_1]$ par application iérée de la proposition 17.1.4. Il suffit en effet d'écrire :

$$\begin{gathered}
\lfloor\Gamma\rfloor\ U^h[\alpha^1, P_1, P_1^1]^\intercal(e, P_2)\ \lfloor\Gamma_1\rfloor \\
\lfloor\Gamma \wedge \Gamma_1\rfloor\ U^h[\alpha^2, P_1, P_1^2]^\intercal(e, P_2)\ \lfloor\Gamma_2\rfloor \\
\vdots \\
\lfloor\Gamma \wedge \Gamma_1 \wedge \dots \Gamma_{i-1}\rfloor\ U^h[\alpha^i, P_1, P_1^i]^\intercal(e, P_2)\ \lfloor\Gamma_i\rfloor \\
\vdots \\
\lfloor\Gamma \wedge \Gamma_1 \wedge \dots \Gamma_{r-1}\rfloor\ U^h[\alpha^r, P_1, P_1^r]^\intercal(e, P_2)\ \lfloor\Gamma_r\rfloor
\end{gathered}$$

puis de poser :

$$\lfloor\Gamma\rfloor\ S^h[P_1]^\intercal(e, P_2)\ \lfloor\Gamma_1 \wedge \dots \wedge \Gamma_r\rfloor$$

La représentation de chaque $U^h[\alpha, P_1, P_1']$ s'obtient à partir de celle de $T^h[\alpha, P_1, P_1']$ en appliquant la proposition 17.1.1. On écrit pour cela :

$$\Gamma' \quad \hat{=} \quad \bigvee_{\substack{\lfloor\Gamma\rfloor\ P_2 \xrightarrow{\alpha}{}^\intercal P_2'\ \lfloor\Gamma_1\rfloor \\ \lfloor\Gamma\wedge\Gamma_1\rfloor\ T^h[\alpha, P_1, P_1']^\intercal(e, P_2, P_2')\ \lfloor\Gamma_2\rfloor}} \Gamma_1 \wedge \Gamma_2$$

puis on pose :

$$\lfloor\Gamma\rfloor\ U^h[\alpha, P_1, P_1']^\intercal(e, P_2)\ \lfloor\Gamma'\rfloor$$

La représentation $T^h[\alpha, P_1, P_1']^\intercal$ de $T^h[\alpha, P_1, P_1']$ est obtenue à partir des représentations de chaque $S^h[P_1']^\intercal$ (pour des processus P_1' de taille inférieure à P_1). On procède suivant la forme de α :

- si $\alpha = \tau$, on écrit :

$$\lfloor\Gamma\rfloor\ S^h[P_1']^\intercal(e, P_2')\ \lfloor\Gamma'\rfloor$$

  puis on pose :

$$\lfloor\Gamma\rfloor\ T^h[\alpha, P_1, P_1']^\intercal(e, P_2, P_2')\ \lfloor\Gamma'\rfloor$$

- si $\alpha = n$, on note $\vec{n}$ un ensemble de noms disjoint de tous les noms utilisés jusqu'à présent et de taille 2^h. La contrainte :

$$\Gamma' \quad \hat{=} \quad \{\vec{n}\} \cap \mathrm{fn}(P_2) = \varnothing \wedge \{\vec{n}\} \cap \mathrm{fn}(e_2) = \varnothing$$

  permettra de plus d'assurer que les instanciations des variables libres présentes dans P_2 et e ne produiront pas par la suite de noms libres présents dans $\vec{n}$. On remarque ensuite que l'ensemble :

$$\{(m_1, m_2) \mid (fr \cup \{\vec{n}\}, th) \vdash m_1 = m_2 \text{ et } \mathrm{h}(m_1) \leq h\}$$

  est fini et se calcule facilement (rappelons que $fr \cup \{n\}$ est clos, de même que chaque message m_1 lorsque $(m_1, m_2) \in th$), on note donc cet ensemble $\{(m_1^1, m_2^1), \dots, (m_1^r, m_2^r)\}$, ce qui permet ensuite d'écrire :

$$T^h[\alpha, P_1, (x).P_1'](e, P_2, (x).P_2') \quad = \quad \bigwedge_{i=1}^{r} S^h[P_1'[m_1^i/x]]((fr \cup \{\vec{n}\}, th), P_2'[m_2^i/x])$$

Itérant alors le résultat de la proposition 17.1.4, on voit qu'il suffit de poser :

$$\lfloor \Gamma \rfloor \ S^h[P_1'[m_1^1/x]]^\intercal((fr \cup \{\vec{n}\}, th), P_2'[m_2^1/x]) \ \lfloor \Gamma_1 \rfloor$$
$$\vdots$$
$$\lfloor \Gamma \wedge \Gamma_1 \wedge \ldots \wedge \Gamma_{i-1} \rfloor \ S^h[P_1'[m_1^i/x]]^\intercal((fr \cup \{\vec{n}\}, th), P_2'[m_2^i/x]) \ \lfloor \Gamma_i \rfloor$$
$$\vdots$$
$$\lfloor \Gamma \wedge \Gamma_1 \wedge \ldots \wedge \Gamma_{r-1} \rfloor \ S^h[P_1'[m_1^r/x]]^\intercal((fr \cup \{\vec{n}\}, th), P_2'[m_2^r/x]) \ \lfloor \Gamma_r \rfloor$$

puis :

$$\lfloor \Gamma \rfloor \ T^h[n, P_1, (x).P_1']^\intercal(e, P_2, (x).P_2') \ \lfloor \Gamma_r \wedge \ldots \wedge \Gamma_r \wedge \Gamma' \rfloor$$

pour obtenir une représentation $T^h[n, P_1, (x).P_1']^\intercal$ de $T^h[n, P_1, (x).P_1']$;

- si $\alpha = \bar{n}$, on cherche alors une représentation $T^h[\bar{n}, P_1, (\nu\vec{n_1})\langle m_1\rangle.P_1']^\intercal$ de $T^h[\bar{n}, P_1, (\nu\vec{n_1})\langle m_1\rangle.P_1']$. On applique pour cela la proposition 17.1.1 à la représentation $\mathrm{Norm}^\intercal$ de Norm et à la représentation $S^h[P_1']^\intercal$ de $S^h[P_1']$ en posant :

$$\Gamma' \ \hat{=} \bigvee_{\substack{\lfloor \Gamma \rfloor \ e' \in \mathrm{Norm}^\intercal((fr, th \cup \{(m_1, m_2)\})) \ \lfloor \Gamma_1 \rfloor \\ \lfloor \Gamma \wedge \Gamma_1 \rfloor \ S^h[P_1']^\intercal(e', P_2') \ \lfloor \Gamma_2 \rfloor}} \Gamma_1 \wedge \Gamma_2$$

puis ensuite :

$$\lfloor \Gamma \rfloor \ T^h[\bar{n}, P_1, (\nu\vec{n_1})\langle m_1\rangle.P_1']^\intercal(e, P_2, (\nu\vec{n_2})\langle m_2\rangle.P_2') \ \lfloor \Gamma' \rfloor$$

Le résultat de ceci est une représentation $S^h[P_1]^\intercal$ de $S^h[P_1]$, pour chaque $P_1 \in \mathrm{Proc}$. Pour voir comment nous allons l'utiliser, reprenons l'exemple du protocole de vote rappelé page 186 (nous traiterons des exemples en détail dans le chapitre suivant).

Exemple : On considère le cas à trois participants du protocole de vote qui s'écrit alors :

$$\begin{aligned} P_{A_i} &\ \hat{=}\ \bar{c}\langle\{V_1\}_{\mathrm{k}(A_i,S)}\rangle.0 \ (i \in \{1,2,3\}) \\ S_{A_i} &\ \hat{=}\ c(x).\mathrm{case}\ x\ \mathrm{of}\ \{y\}_{\mathrm{k}(A_i,S)}.\bar{c}\langle y\rangle.0 \ (i \in \{1,2,3\}) \\ S &\ \hat{=}\ S_{A_1} \parallel S_{A_2} \parallel S_{A_3} \\ P &\ \hat{=}\ P_{A_1} \parallel P_{A_2} \parallel P_{A_3} \parallel S \end{aligned}$$

et l'ensemble des mondes associé est :

$$W \ \hat{=} \ \mathrm{Name}^3 \times \{0,1\}^3$$

Dans cet ensemble de mondes, nous avions considéré le monde particulier :

$$w = ((a_1, a_2, a_3), (v_1, v_2, v_3)) \ \hat{=} \ ((a_1, a_2, a_3), (1, 0, 0))$$

et nous avions expliqué qu'une étape essentielle à la preuve d'une propriété d'opacité en w consistait à trouver des mondes $w' \sim w$ (peut importe ici quels sont l'attribut et la propriété d'opacité considérés). On calcule pour cela la contrainte Γ_w telle que :

$$\lfloor \mathrm{true} \rfloor \ S^3[P[a_1/A_1, a_2/A_2, a_3/A_3, v_1/V_1, v_2/V_2, v_3/V_3]]^\intercal((\{n\}, \varnothing), P) \ \lfloor \Gamma_w \rfloor$$

Les mondes w' correspondant à des solutions de Γ_w seront alors équivalents à w. □

Notons enfin que, si on avait voulu considérer la H-bisimilarité gardée (au lieu de la h-similarité gardée) nous aurions eu à gérer des conjonctions infinie (par exemple la conjonction définissant S^h en fonction de U^h).

Chapitre 18

En pratique

Dans ce dernier chapitre sur les propriétés d'opacité, nous allons expliquer (essentiellement sur la base d'exemples) comment utiliser les ingrédients fournis par les chapitres précédents afin de prouver des propriétés. Nous avons pour cela implémenté l'algorithme du chapitre 16 (inspiré de celui de H. Hüttel [Hüt02]) pour décider la h-bisimilarité gardée ainsi que l'algorithme du chapitre 17 qui retourne une contrainte dont les solutions correspondent à des processus équivalents (au sens de la h-similarité gardée) au processus fourni en entrée. Comme ces algorithmes ne sont cependant pas très efficaces, nous allons les utiliser sur des versions simplifiées du dîner des cryptographes et du protocole de vote. Nous allons procéder suivant les deux méthodes décrites au chapitre 16. La première consiste à proposer des mondes qui permettraient de prouver la propriété d'opacité, sous réserve qu'ils soient équivalents au monde de départ considéré. On utilise alors l'algorithme de décision de la h-bisimilarité gardée pour vérifier que les mondes proposés sont effectivement équivalents. Nous appellerons cette méthode « deviner, vérifier ». La seconde méthode consiste à appliquer l'algorithme modifié retournant des contraintes, en espérant qu'il retourne suffisamment de solutions pour que la propriété d'opacité puisse être prouvée. Comme cet algorithme concerne la h-simulation gardée il faut bien entendu, pour chaque solution fournie, vérifier qu'elle correspond à un monde équivalent au monde de départ. La première section explique comment utiliser ces méthodes dans le cas du dîner des cryptographes et la deuxième est consacrée au protocole de vote. La dernière section sera notre conclusion sur cette partie.

18.1 Le dîner des cryptographes

Nous commençons par faire quelques rappels sur ce protocole (il s'agit d'ailleurs d'une version simplifiée du protocole qui a été utilisé jusqu'à présent) et les propriétés d'opacité considérées, puis nous appliquons chacune des deux méthodes présentées plus haut.

Le protocole. Le protocole se trouve page 157 et ses exécutions sont décrites par les processus qui se trouvent page 160. Cependant, par souci de simplification, et pour garder des temps d'exécution raisonnables, nous remplaçons l'utilisation de clés symétriques entre les participants et leurs voisins de droite par des canaux de communication privés. Ceci nous évite d'avoir à manipuler des chiffrements et déchiffrements. Si on note $c_{1\to2}$, $c_{2\to3}$ et $c_{3\to1}$ des canaux de communication privés entre chaque participant et son voisin de droite (nous nous limitons à trois participants), le protocole peut alors être représenté au moyen des processus suivants :

$$\begin{array}{rcl} P_{A_1} & \hat{=} & \overline{c_{1\to 2}}\langle Q_1\rangle.c_{3\to 1}(q_3).\dots \\ P_{A_2} & \hat{=} & c_{1\to 2}(q_1).\overline{c_{2\to 3}}\langle Q_2\rangle.\dots \\ P_{A_3} & \hat{=} & c_{2\to 3}(q_2).\overline{c_{3\to 1}}\langle Q_1\rangle.\dots \\ P & \hat{=} & P_{A_1} \parallel P_{A_2} \parallel P_{A_3} \end{array}$$

(nous n'avons pas repris le détail du calcul du « ou exclusif » qui se trouve page 160). Nous avons expliqué au chapitre 15 comment associer un système observé à ce protocole. L'ensemble des mondes est :

$$W \;\hat{=}\; \{((a_1, a_2, a_3), (q_1, q_2, q_3), (p_1, p_2, q_3)) \in \mathrm{Name}^3 \times \{0,1\}^6 \mid p_1 + p_2 + p_3 \leq 1\}$$

et, si pour $w \in W$ on note $P(w)$ le processus P décrivant les exécutions du protocole instancié avec les valeurs contenues dans w, on dira que w et w' sont équivalents, et on notera $w \sim w'$, si $P(w) \underset{\mathrm{tst}}{\sim} P(w')$. Nous avons ensuite considéré les attributs :

$$\begin{array}{rrcl} c_1 : & W & \to & C_1 = 2^{\mathrm{Name}} \\ & w & \mapsto & \{a_i \mid 1 \leq i \leq 3 \text{ et } p_i = 1\} \end{array} \qquad \text{et} \qquad \begin{array}{rrcl} c_2 : & W & \to & C_2 = \{0,1\}^3 \\ & w & \mapsto & (q_1, q_2, q_3) \end{array}$$

qui associent, à chaque monde possible, le participant distingué dans ce monde (s'il y en a un) et les tirages aléatoires effectués par les participants. Nous avions choisi, pour étudier l'opacité de ces attributs, d'utiliser les domaines abstraits :

$$A_1 \;=\; 2^{C_1} \quad \text{et} \quad A_2 \;=\; 2^{C_2}$$

munis des relations d'inclusion, la notion de compatibilité avec les domaines concrets correspondant à la relation d'appartenance. Enfin, nous avions considéré les propriétés d'opacité :

$$\begin{array}{rcl} \varphi_1 & \hat{=} & \{a \in A_1 \mid \mathrm{card}(a) \geq 3\} \\ \varphi_2 & \hat{=} & \{a \in A_2 \mid \forall i \in \{1,2,3\}.\ \exists (q_1, q_2, q_3), (q_1', q_2', q_3') \in a.\ q_i \neq q_i'\} \end{array}$$

Dans le reste de cette section, nous fixons une exécution particulière du protocole, *i.e.* un monde $w \in W$, par exemple :

$$w \;\hat{=}\; ((a_1, a_2, a_3), (0,1,0), (1,0,0))$$

(c'est donc le premier participant qui est distingué) et nous cherchons à montrer :
- qu'il existe des mondes $w' \sim w$ dans lesquels le deuxième (respectivement troisième) participant est distingué ;
- qu'il existe des mondes $w' \sim w$ dans lesquels le tirage effectué par le participant a_1 (respectivement a_2, a_3) est différent de celui effectué dans w.

Ces deux points nous permettraient de montrer que $\mathcal{W}, w \models \varphi_1$ et $\mathcal{W}, w \models \varphi_2$.

Méthode « deviner, vérifier ». Nous voulons montrer qu'il existe des mondes w', équivalents à w, dans lesquels ce n'est pas a_1 qui est distingué. Nous cherchons donc des mondes de la forme :

$$w' \;=\; ((a_1, a_2, a_3), (q_1', q_2', q_3'), (p_1', p_2', p_3'))$$

équivalents à w et tels que $p'_1 = 0$. Il n'y a que 24 cas possibles, nous pouvons donc les traiter de manière exhaustive. Pour chaque choix possible des q_i et p_i dans $\{0,1\}$ (rappelons que l'on doit toujours avoir $p_1 + p_2 + p_3 = 1$), on lance l'algorithme du chapitre 16, qui retourne un résultat dans $\{\text{true}, \text{false}\}$ suivant que le monde w' est équivalent ou non à w. La réponse fournie par l'implémentation se trouve à la figure 18.1.1. À la lecture de ces résultats, nous

$$
\begin{array}{lcl}
q'_1 = 0, q'_2 = 0, q'_3 = 0, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{false} \\
q'_1 = 0, q'_2 = 0, q'_3 = 0, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{false} \\
q'_1 = 0, q'_2 = 0, q'_3 = 0, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{false} \\
q'_1 = 0, q'_2 = 0, q'_3 = 1, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{false} \\
q'_1 = 0, q'_2 = 0, q'_3 = 1, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{true} \\
q'_1 = 0, q'_2 = 0, q'_3 = 1, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{false} \\
q'_1 = 0, q'_2 = 1, q'_3 = 0, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{true} \\
q'_1 = 0, q'_2 = 1, q'_3 = 0, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{false} \\
q'_1 = 0, q'_2 = 1, q'_3 = 0, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{false} \\
q'_1 = 0, q'_2 = 1, q'_3 = 1, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{false} \\
q'_1 = 0, q'_2 = 1, q'_3 = 1, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{false} \\
q'_1 = 0, q'_2 = 1, q'_3 = 1, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{true} \\
q'_1 = 1, q'_2 = 0, q'_3 = 0, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{false} \\
q'_1 = 1, q'_2 = 0, q'_3 = 0, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{false} \\
q'_1 = 1, q'_2 = 0, q'_3 = 0, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{true} \\
q'_1 = 1, q'_2 = 0, q'_3 = 1, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{true} \\
q'_1 = 1, q'_2 = 0, q'_3 = 1, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{false} \\
q'_1 = 1, q'_2 = 0, q'_3 = 1, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{false} \\
q'_1 = 1, q'_2 = 1, q'_3 = 0, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{false} \\
q'_1 = 1, q'_2 = 1, q'_3 = 0, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{true} \\
q'_1 = 1, q'_2 = 1, q'_3 = 0, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{false} \\
q'_1 = 1, q'_2 = 1, q'_3 = 1, p'_1 = 1, p'_2 = 0, p'_3 = 0 & : & \text{false} \\
q'_1 = 1, q'_2 = 1, q'_3 = 1, p'_1 = 0, p'_2 = 1, p'_3 = 0 & : & \text{false} \\
q'_1 = 1, q'_2 = 1, q'_3 = 1, p'_1 = 0, p'_2 = 0, p'_3 = 1 & : & \text{false}
\end{array}
$$

FIG. 18.1.1 – *Dîner des cryptographes : d'autres mondes possibles*

constatons :
- qu'il existe des mondes équivalents à w dans lesquels a_2 (respectivement a_3) est distingué ;
- qu'il existe des mondes équivalents à w dans lesquels le tirage effectué par a_1 (respectivement a_2, a_3) est différent de celui effectué dans w.

On en déduit que les propriétés φ_1 et φ_2 sont satisfaites dans le monde w. On pourrait faire de même dans les autres mondes.

Utilisation de contraintes. Imaginons que l'on cherche s'il existe des mondes équivalents à w dans lesquels le participant a_2 est distingué. Au lieu d'examiner tous les cas possibles afin de trouver des mondes w' équivalents à w de la forme :

$$w' \quad = \quad ((a_1, a_2, a_3), (q'_1, q'_2, q'_3), (0, 1, 0))$$

on utilise l'algorithme modifié qui retourne une contrainte sur q'_1, q'_2, q'_3. La contrainte retournée dans ce cas se trouve à la figure 18.1.2 (les points de suspensions désignent des contraintes omises qui sont peu pertinentes dans le cas présent). Ces contraintes nous permettent d'une

$$\begin{array}{ll}
 & [\ldots] \wedge q'_1 = 0 \wedge q'_2 = 0 \wedge q'_3 = 1 \\
\vee & [\ldots] \wedge q'_3 = 1 \wedge q'_1 = 0 \wedge q'_2 = 0 \\
\vee & [\ldots] \wedge q'_2 = 0 \wedge q'_3 = 1 \wedge q'_1 = 0 \\
\vee & [\ldots] \wedge q'_1 = 1 \wedge q'_2 = 1 \wedge q'_3 = 0 \\
\vee & [\ldots] \wedge q'_3 = 0 \wedge q'_1 = 1 \wedge q'_2 = 1 \\
\vee & [\ldots] \wedge q'_2 = 1 \wedge q'_3 = 0 \wedge q'_1 = 1
\end{array}$$

FIG. 18.1.2 – *Dîner des cryptographes : contrainte retournée*

part de connaître les tirages aléatoires fournissant le même résultat que w dans le cas où a_2 est distingué. Elles nous permettent également de déduire que φ_1 est vraie, puisqu'il existe des mondes équivalents à w dans lesquels a_2 est distingué. Notons que, en toute rigueur, pour déduire ce résultat, il faudrait avant tout :

- vérifier qu'ils existe des mondes équivalents à w dans lesquels a_3 est distingué (mais il suffit de procéder de la même manière que pour a_2) ;
- vérifier que les mondes obtenus sont bien équivalents au sens de la h-bisimilarité gardée et pas seulement pour la h-similarité gardée (mais ces calculs ont déjà été faits à la section précédente).

18.2 Le protocole de vote

Le protocole. Revenons maintenant sur le protocole de vote donné page 171. Nous supposons là encore qu'il n'y a que trois participants et nous utilisons des canaux privés de la forme $c_{i \to S}$ pour éviter d'utiliser un chiffrement à clé symétrique dans les communications entre le votant A_i et le serveur. Les processus qui entrent en jeu sont donc les suivants :

$$\begin{array}{rcl}
P_{A_i} & \hat{=} & \overline{c_{1 \to S}}\langle V_1 \rangle.0 \ (i \in \{1, 2, 3\}) \\
S_{A_i} & \hat{=} & c_{1 \to S}(y).\overline{c}\langle y \rangle.0 \ (i \in \{1, 2, 3\}) \\
S & \hat{=} & S_{A_1} \parallel S_{A_2} \parallel S_{A_3} \\
P & \hat{=} & P_{A_1} \parallel P_{A_2} \parallel P_{A_3} \parallel S
\end{array}$$

Le système observé associé à ce protocole est construit sur l'ensemble des mondes :

$$W \quad \hat{=} \quad \text{Name}^3 \times \{0, 1\}^3$$

et nous nous intéressons à l'opacité de l'attribut :

$$\begin{array}{rccl} c: & W & \to & C = 2^{\mathrm{Name}\times\{0,1\}} \\ & w & \mapsto & \{(a_i, v_i) \mid 1 \leq i \leq 3\} \end{array}$$

Nous utiliserons comme domaine abstrait l'ensemble des fonctions qui, à chaque $a \in \mathrm{Name}$, associent un sous-ensemble de $\{0,1\}$ (qui contient les valeurs possibles pour le vote de a) et nous chercherons à montrer la propriété d'opacité suivante :

$$\varphi \quad \hat{=} \quad \{a \in A \mid \forall a_i.\ a(a_i) = \varnothing \text{ ou } a(a_i) = \{0,1\}\}$$

Remarquons qu'il est impossible de prouver cette propriété dans le cas où tous les votants ont fait le même choix. Nous fixerons donc un monde particulier $w \in W$, contenant au moins deux votes différents, par exemple :

$$w \quad \hat{=} \quad ((a_1, a_2, a_3), (1, 0, 0))$$

et nous chercherons des mondes $w' \sim w$ pour lesquels les votes des participants sont différents de ceux effectués dans w.

Méthode « Deviner, vérifier ». Nous cherchons s'il existe des mondes de la forme :

$$w' \quad = \quad ((a_1, a_2, a_3), (v'_1, v'_2, v'_3))$$

équivalents à w. En essayant tous les cas possibles, comme dans la section précédente, on trouve le résultat donné à la figure 18.2.1. Comme on pouvait s'y attendre, les mondes équivalents

$$\begin{array}{rcl}
v'_1 = 0, v'_2 = 0, v'_3 = 0 & : & \text{false} \\
v'_1 = 0, v'_2 = 0, v'_3 = 1 & : & \text{true} \\
v'_1 = 0, v'_2 = 1, v'_3 = 0 & : & \text{true} \\
v'_1 = 0, v'_2 = 1, v'_3 = 1 & : & \text{false} \\
v'_1 = 1, v'_2 = 0, v'_3 = 0 & : & \text{true} \\
v'_1 = 1, v'_2 = 0, v'_3 = 1 & : & \text{false} \\
v'_1 = 1, v'_2 = 1, v'_3 = 0 & : & \text{false} \\
v'_1 = 1, v'_2 = 1, v'_3 = 1 & : & \text{false}
\end{array}$$

FIG. 18.2.1 – *Vote : d'autres mondes possibles*

sont ceux ou le résultat du vote (en nombre de votes) est le même. En particulier, il existe un monde équivalent où a_1 a voté 0, un autre où a_2 a voté 1 et un où a_3 a voté 1. On en déduit que la propriété d'opacité φ est satisfaite dans w.

Utilisation de contraintes. Utilisons l'algorithme pour obtenir une contrainte sur v'_1, v'_2, v'_3 telle que le monde :

$$w' \quad = \quad ((a_1, a_2, a_3), (v'_1, v'_2, v'_3))$$

soit équivalent à w. La figure 18.2.2 présente une partie des résultats obtenus (comme pour le dîner des cryptographes, nous avons omis une partie des contraintes, peu pertinente, mais nous avons également supprimé des contraintes qui ne possédaient pas de solutions). Sans surprise, on retrouve les instanciations qui conduisent à un même résultat pour le vote. On peut également vérifier que, pour chaque participant, il existe une instanciation possible dans laquelle il a voté 0 et une autre dans laquelle il a voté 1.

$$
\begin{array}{ll}
 & [\ldots] \\
\vee & [\ldots] \wedge 1 = v'_1 \wedge v'_2 = v'_3 \wedge 0 = v'_3 \wedge v'_3 = v'_2 \wedge 0 = v'_2 \wedge v'_3 = 0 \wedge v'_2 = 0 \wedge v'_1 = 1 \\
\vee & [\ldots] \wedge 1 = v'_1 \wedge v'_3 = v'_2 \wedge 0 = v'_2 \wedge v'_2 = v'_3 \wedge 0 = v'_3 \wedge v'_2 = 0 \wedge v'_3 = 0 \wedge v'_1 = 1 \\
\vee & [\ldots] \wedge 1 = v'_1 \wedge v'_2 = v'_3 \wedge 0 = v'_3 \wedge v'_3 = v'_2 \wedge 0 = v'_2 \wedge v'_3 = 0 \wedge v'_2 = 0 \wedge v'_1 = 1 \\
\vee & [\ldots] \wedge 1 = v'_1 \wedge v'_3 = v'_2 \wedge 0 = v'_2 \wedge v'_2 = v'_3 \wedge 0 = v'_3 \wedge v'_2 = 0 \wedge v'_3 = 0 \wedge v'_1 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_3 = 0 \wedge v'_1 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_1 = 0 \wedge v'_3 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_3 = 0 \wedge v'_1 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_1 = 0 \wedge v'_3 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_3 = 0 \wedge v'_1 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_1 = 0 \wedge v'_3 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_3 = 0 \wedge v'_1 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_2 \wedge v'_3 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_3 \wedge 0 = v'_3 \wedge v'_1 = 0 \wedge v'_3 = 0 \wedge v'_2 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_2 = 0 \wedge v'_1 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_1 = 0 \wedge v'_2 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_2 = 0 \wedge v'_1 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_1 = 0 \wedge v'_2 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_2 = 0 \wedge v'_1 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_1 = 0 \wedge v'_2 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_2 = 0 \wedge v'_1 = 0 \wedge v'_3 = 1 \\
\vee & [\ldots] \wedge 1 = v'_3 \wedge v'_2 = v'_1 \wedge 0 = v'_1 \wedge v'_1 = v'_2 \wedge 0 = v'_2 \wedge v'_1 = 0 \wedge v'_2 = 0 \wedge v'_3 = 1
\end{array}
$$

FIG. 18.2.2 – *Vote : contrainte retournée*

18.3 Conclusion

Dans cette partie, nous avons considéré les propriétés d'opacité qui généralisent l'anonymat. Nous avons tout d'abord proposé un modèle pour étudier ces propriétés, dont les ingrédients principaux étaient une notion d'équivalence observationnelle et une correspondance de Galois. Nous avons ensuite décrit un certain nombre de relations de bisimilarité qui permettent de raffiner l'équivalence observationnelle considérée, jusqu'à l'obtention d'une relation décidable. Munis de cette relation décidable et d'une énumération des différentes manières d'instancier le protocole, nous avons expliqué comment assurer que les propriétés d'opacité étaient satisfaites, mais d'une manière peu efficace. Nous avons ensuite proposé une amélioration consistant à transformer l'algorithme de décision de sorte qu'il retourne directement (sous forme d'une contrainte) les instances possibles du protocole qui seraient similaires à une instance donnée. Au moyen de quelques exemples, nous avons expliqué comment se passait ensuite (en pratique) la vérification. Voyons maintenant dans quelles directions il serait possible de développer ces travaux. Plaçons-nous, pour cela, dans le cas où on cherche à montrer qu'une propriété d'opacité φ est satisfaite en $w_0 \in W$ sur un système observé $\mathcal{W} = (W, \sim, c, (2^C, \subseteq) \overset{\sharp}{\underset{\flat}{\rightleftharpoons}} (A, \sqsubseteq))$. Suivant la définition 15.3.2 (page 169), on cherche donc à montrer que $\overline{c}(w_0)^\sharp \in \varphi^\flat$. Rappelons aussi que nous nous sommes toujours placés dans le cas où la relation d'équivalence sur W était induite par la relation d'équivalence par tests du spi-calcul, au moyen d'un processus P (élément de $\mathrm{Proc}_{\mathrm{Fin}}$), de variables libres $x_1, \ldots, x_n$, pour lequel les éléments de W tiennent lieu de paramètres.

L'approximation. On est en droit de se demander, au vu des travaux présentés dans cette partie, à quel endroit se situe l'approximation et quelle quantité d'information est perdue. Examinons donc quelles sont les différentes approximations qui interviennent lorsque l'on cherche à monter que $\mathcal{W}, w \models \varphi$, au moyen de notre technique d'abstraction :

1. L'équivalence par tests est remplacée par la h-bisimilarité gardée et nous avons montré (au moyen d'exemples dans les remarques page 178 et 183) que l'inclusion entre ces deux relations était stricte. Cependant, il nous semble que les différences sont suffisamment fines pour qu'une attaque contre une propriété d'opacité n'existant que pour la h-bisimilarité gardée doive être considérée comme inquiétante. En d'autres termes, l'approximation réalisée ici est négligeable, en ce sens que même les attaques qui n'existent que grâce à cette approximation doivent être prises en compte ;

2. Notons $w_0, w_1, \ldots$ une énumération de la classe d'équivalence de w_0 pour $\sim$. Au lieu de montrer que $\overline{c}(w_0)^\sharp = \{c(w_0), c(w_1), \ldots\}^\sharp \in \varphi$, on montre qu'il existe $i \geq 0$ tel que $\{c(w_0), c(w_1), \ldots, c(w_i)\}^\sharp \in \varphi$. Ceci revient à remplacer la connaissance (qu'a l'observateur) de toute une classe d'équivalence par celle de sous-ensembles finis de cette classe. On sait que cette approximation est correcte, elle serait complète si, quel que soit $\mathcal{C} \subseteq C$, on avait la propriété suivante :

$$\mathcal{C}^\sharp \in \varphi \quad \Leftrightarrow \quad \exists\, \mathcal{C}' \text{ fini. } \mathcal{C}' \subseteq \mathcal{C} \text{ et } \mathcal{C}'^\sharp \in \varphi$$

   Cette propriété rappelle la notion de compacité pour les relations d'ordre [AL91]. Il est possible de dire, pour chaque correspondance de Galois utilisée si cette propriété est satisfaite ou non (dans les exemples que nous avons étudié, elle est clairement satisfaite) ;

3. L'algorithme testant la h-bisimilarité gardée est modifié pour retourner une contrainte, suivant le formalisme décrit dans le chapitre 17. L'algorithme opérant sur les contraintes est une représentation de l'algorithme de départ, mais rien ne nous assure de la complétude de cette représentation. En effet, nous avons utilisé à plusieurs reprises la proposition 17.1.5 (page 196) qui n'assure en aucun cas la complétude. Ceci dit, nous n'avons pas en notre possession de contre-exemple qui permettrait de montrer que la complétude n'est pas satisfaite. Pour ce point, la question de l'importance de l'approximation reste donc ouverte.

On peut donc noter que, sur les trois approximations effectuées, la complétude de la première importe peu, celle de la deuxième est une propriété de la correspondance de Galois considérée (en pratique facilement prouvable ou réfutable) et celle de la troisième reste ouverte.

L'opacité, syntaxiquement. Nous avons fait remarquer au chapitre 15, page 169 que les propriétés d'opacité n'étaient pas exprimées de manière syntaxique (dans notre formalisme) mais comme des sous-ensembles de l'ensemble des connaissances possibles pour l'observateur. Nous pensons qu'il serait intéressant de développer un langage dans lequel pourraient s'exprimer les propriétés d'opacité. Une approche possible serait, peut-être, d'utiliser dans ce but les contraintes qui, dans les travaux qui ont été présentés ici, servent seulement en tant qu'outils. En effet, l'opacité d'un attribut paraît pouvoir être exprimée par le fait que cet attribut (ou quelque chose le concernant) n'est pas déductible des contraintes. Cependant, d'autres approches sont sans doute possibles.

Conclusion

Nous nous sommes intéressés, dans cette thèse, à la preuve de propriétés de protocoles cryptographiques. Notre étude a été organisée suivant trois grandes classes de propriétés et la démarche suivie a toujours été la même : tout d'abord proposer une modélisation pour les protocoles et propriétés considérés, puis définir des abstractions finies et correctes pour ces modèles et, enfin, étudier le résultat de l'application de ces techniques d'abstraction au problème de la vérification pratique des propriétés considérées. Des conclusions spécifiques ont été présentées dans chaque partie (pages 84, 153 et 210). Nous aimerions ici retracer ces travaux, mais dans une perspective unificatrice. Nous allons donc rappeler les choix qui ont été faits, dans les différentes modélisations, tout d'abord, puis dans la conception des abstractions.

En ce qui concerne la modélisation, nous avons utilisé successivement des modèles de la logique du premier ordre (pour les propriétés de traces), des systèmes de transitions alternés (pour les propriétés de jeux) et des systèmes observés (pour les propriétés d'opacité). Notons que les traces décrites au moyen de modèles de la logique du premier ordre, dans la première partie, pourraient sans doute apparaître comme les traces d'un système de transitions bien choisi. Les systèmes de transitions sont également présents dans la troisième partie, sous forme de processus. Il y a donc, dans chacune de nos modélisations, un système de transitions sous-jacent, même s'il n'est pas explicité.

Le principe général des abstractions dans le cadre de la logique du premier ordre était d'utiliser un morphisme d'algèbres, afin de relier les domaines abstrait et concret, et de donner pour chaque interprétation de prédicat dans la structure de départ, deux interprétations dans la structure abstraite, l'une étant une sur-approximation et l'autre une sous-approximation. Nous avons ensuite expliqué comment traduire les énoncés sur la structure concrète en énoncés sur la structure abstraite, de sorte qu'une propriété de correction soit satisfaite. Dans la deuxième partie, nous avons abstrait un système de transitions alterné en remplaçant l'ensemble de ses états par un ensemble fini (reliés au moyen d'une surjection) et en dédoublant la relation de transition en deux approximations, l'une supérieure et l'autre inférieure. Nous avons conservé les énoncés tels quels, mais nous avons redéfini leur sémantique dans le cadre des systèmes de transitions alternés abstraits et prenant garde, là aussi, à obtenir une propriété de correction. En ce qui concerne les propriétés d'opacité, finalement, nous avons vu que plusieurs approximations entrent en jeu, mais elles concourent toutes à remplacer les classes d'équivalence des différentes sessions du protocole, pour l'équivalence observationnelle, par des sous-ensembles finis de ces classes. On observe donc, dans chacune de ces trois manières de construire des abstractions, un dénominateur commun : il y a, dans chaque modèle, des relations (on les appelle aussi prédicats) qui sont abstraites au moyen de deux relations chacune, l'une étant une sur-approximation de la relation de départ et l'autre une sous-approximation. Parfois, seule une de ces deux approximations nous intéresse. Parfois aussi, l'approximation est relative à une surjection.

Nous avons proposé, dans les conclusions spécifiques à chaque partie, des perceptives de développement ultérieurs. Nous faisons de même ici, mais toujours dans une perspective unificatrice.

Unification. Comme le résumé précédent vient de le faire apparaître, chacune de nos trois approches utilise plus ou moins les même ingrédients, plus ou moins de la même manière. Il serait peut-être possible d'obtenir à partir de là une modélisation générale des protocoles cryptographiques, pouvant prendre en compte chaque type de propriété, ainsi qu'un cadre général pour réaliser les abstractions.

Comparaison. Un autre moyen d'éprouver les relations qui existent entre les trois types de propriétés, de modélisations et d'abstractions présentés ici serait d'essayer de trouver des traductions entre ces trois formalismes. Ceci semble cependant un peu ambitieux. Une approche plus simple serait de prendre un protocole et une propriété simple (de secret par exemple), de représenter cette propriété comme une propriété de trace, puis comme une propriété de jeu et enfin comme une propriété d'opacité. On pourrait alors, en faisant varier le protocole, observer quand une propriété serait vraie dans chaque formalisme tout en étant fausse dans les autres. Nous somme convaincus qu'un travail de cette sorte produirait des résultats intéressants.

Bibliographie

[Aba00] Martín ABADI : Security protocols and their properties : Dans F.L. BAUER et R. STEINBRUEGGEN, éditeurs, *Volume for the 20th International Summer School on Foundations of Secure Computation, Marktoberdorf, Germany, 1999*, NATO Science Series, pages 39–60. IOS Press, 2000.

[ABK+02] Egidio ASTESIANO, Michel BIDOIT, Hélène KIRCHNER, Bernd KRIEG-BRÜCKNER, Peter D. MOSSES, Donald SANNELLA et Andrzej TARLECKI : CASL : The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, septembre 2002.

[AG97] Martín ABADI et Andrew D. GORDON : Reasoning about cryptographic protocols in the spi calculus : Dans Antoni MAZURKIEWICZ et Józef WINKOWSKI, éditeurs, *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97), Warsaw, Poland, July 1-4, 1997*, volume 1243 de *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1997.

[AG98] Martín ABADI et Andrew D. GORDON : A bisimulation method for cryptographic protocols : Dans Chris HANKIN, éditeur, *Programming Languages and Systems – Proceedings of the 7th European Symposium on Programming (ESOP'98), held as part of the European Joint Conferences on the Theory and Practice of Software (ETAPS'98), Lisbon, Portugal, March 28 - April 4, 1998*, volume 1381 de *Lecture Notes in Computer Science*, pages 12–26. Springer-Verlag, 1998.

[AG99] Martín ABADI et Andrew D. GORDON : A calculus for cryptographic protocols : The spi calculus. *Information and Computation*, 148(1):1–70, janvier 1999.

[AHK98] Rajeev ALUR, Thomas A. HENZINGER et Orna KUPFERMAN : Alternating-time temporal logic : Dans Willem P. de ROEVER, Hans LANGMAACK et Amir PNUELI, éditeurs, *Compositionality : The Significant Difference – Revised Lectures of the International Symposium on Compositionality (COMPOS'97), Bad Malente, Germany, September 8-12, 1997*, volume 1536 de *Lecture Notes in Computer Science*, pages 23–60. Springer-Verlag, 1998.

[AKKB99] Egidio ASTESIANO, Hans-Jörg KREOWSKI et Bernd KRIEG-BRÜCKNER, éditeurs : *Algebraic Foundations of Systems Specification.* IFIP State-of-the-Art Reports. Springer-Verlag, 1999.

[AL91] Andrea ASPERTI et Giuseppe LONGO : *Categories, Types and Structures. An introduction to Category Theory for the working computer scientist.* The MIT Press, 1991.

[ASW98a] N. ASOKAN, Victor SHOUP et Michael WAIDNER : Asynchronous protocols for optimistic fair exchange. Dans *Proceedings of the 1998 IEEE Symposium on Se-*

curity and Privacy (S&P'98), Oakland, CA, USA, May 3-6, 1998, pages 86–99. IEEE Computer Society Press, 1998.

[ASW98b] N. Asokan, Victor Shoup et Michael Waidner : Optimistic fair exchange of digital signatures : Dans Kaisa Nyberg, éditeur, *Advances in Cryptology – Proceeding of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'98), Espoo, Finland, May 31 - June 4, 1998*, volume 1403 de *Lecture Notes in Computer Science*, pages 591–606. Springer-Verlag, 1998.

[BAN89] Michael Burrows, Martin Abadi et Roger Needham : A logic of authentication. SRC Research Report 39, Digital Systems Research Center, février 1989.

[BB01] Michel Bidoit et Alexandre Boisseau : Algebraic abstractions : Dans Maura Cerioli et Gianna Reggio, éditeurs, *Recent Trends in Algebraic Development Techniques – Selected Papers from the 15th International Workshop on Algebraic Development Techniques (WADT'01), Joint with the CoFI WG Meeting, Genova, Italy, April 1-3, 2001*, volume 2267 de *Lecture Notes in Computer Science*, pages 21–47. Springer-Verlag, 2001.

[Bla02] Bruno Blanchet : From secrecy to authenticity in security protocols : Dans Manuel V. Hermenegildo et German Puebla, éditeurs, *Proceedings of the 9th International Symposium Static Analysis (SAS 2002), Madrid, Spain, September 17-20, 2002*, volume 2477 de *Lecture Notes in Computer Science*, pages 342–359. Springer-Verlag, 2002.

[BLP03] Liana Bozga, Yassine Lakhnech et Michael Périn : Pattern-based abstraction for verifying secrecy in protocols : Dans Hubert Garavel et John Hatcliff, éditeurs, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2003), Warsaw, Poland, April 7-11, 2003*, volume 2619 de *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 2003.

[BM01] Michel Bidoit et Peter D. Mosses : A gentle introduction to CASL v1.0.1. Invited tutorial. 4th European Joint Conferences on Theory and Practice of Software (ETAPS'2001), Genova, Italy, avril 2001. `http://www.lsv.ens-cachan.fr/~bidoit/CASL/`.

[BN02] Johannes Borgström et Uwe Nestmann : On bisimulations for the spi calculus : Dans Hélène Kirchner et Christophe Ringeissen, éditeurs, *Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology (AMAST 2002), Saint-Gilles-les-Bains, Reunion Island, France, September 9-13, 2002*, volume 2422 de *Lecture Notes in Computer Science*, pages 287–303. Springer-Verlag, 2002.

[Bol97] Dominique Bolignano : Towards a mechanization of cryptographic protocol verification : Dans Orna Grumberg, éditeur, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97), Haifa, Israel, June 22-25, 1997*, volume 1254 de *Lecture Notes in Computer Science*, pages 131–142. Springer-Verlag, 1997.

[Bol99] Dominique Bolignano : Using abstract interpretation for the safe verification of security protocols : Dans Stephen Brookes, Achim Jung, Michael Mislove et

Andre Scedrov, éditeurs, *MFPS XV Mathematical Foundations of Programming Semantics, Fifteenth Conference, Tulane University, New Orleans, LA, April 28-May 1, 1999*, volume 20 de *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 1999.

[BT96] Michel Bidoit et Andrzej Tarlecki : Behavioural satisfaction and equivalence in concrete model categories : Dans Hélène Kirchner, éditeur, *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming (CAAP'96), Linköping, Sweden, April, 22-24, 1996*, volume 1059 de *Lecture Notes in Computer Science*, pages 241–256. Springer-Verlag, 1996.

[CC77] Patrick Cousot et Radhia Cousot : Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. Dans *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages (POPL), Los Angeles, California, January 1977*, pages 238–252. ACM Press, 1977.

[CC92a] Patrick Cousot et Radhia Cousot : Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2,3):103–179, juillet 1992.

[CC92b] Patrick Cousot et Radhia Cousot : Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, août 1992.

[CCM01] Hubert Comon, Véronique Cortier et John Mitchell : Tree automata with one memory, set constraints, and ping-pong protocols : Dans Fernando Orejas, Paul G. Spirakis et Jan van Leeuwen, éditeurs, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete, Greece, July 8-12, 2001*, volume 2076 de *Lecture Notes in Computer Science*, pages 682–693. Springer-Verlag, 2001.

[CD99] Kevin J. Compton et Scott Dexter : Proof techniques for cryptographic protocols : Dans Jirí Wiedermann, Peter van Emde Boas et Mogens Nielsen, éditeurs, *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP'99), Prague, Czech Republic, July 11-15, 1999*, volume 1644 de *Lecture Notes in Computer Science*, pages 25–39. Springer-Verlag, 1999.

[CES86] Edmund M. Clarke, E. Allen Emerson et A. Prasad Sistla : Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, avril 1986.

[CGL94] Edmund M. Clarke, Orna Grumberg et David E. Long : Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, septembre 1994.

[CGP99] Edmund M. Clarke, Orna Grumberg et Doron A. Peled : *Model Checking*. The MIT Press, 1999.

[Cha81] David L. Chaum : Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM (CACM)*, 24(2):84–88, février 1981.

[Cha88] David L. Chaum : The dining cryptographers problem : Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[CJ97] John A. CLARK et Jeremy L. JACOB : A survey of authentication protocol literature. Rapport technique, University of York, Department of Computer Science, novembre 1997.

[CJM98] Edmund M. CLARKE, Somesh JHA et Wilfredo R. MARRERO : Using state space exploration and a natural deduction style message derivation engine to verify security protocols : Dans David GRIES et Willem P. de ROEVER, éditeurs, *Programming Concepts and Methods, IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods (PROCOMET'98), 8-12 June 1998, Shelter Island, New York, USA*, volume 125 de *IFIP Conference Proceedings*, pages 87–106. Chapman & Hall, 1998.

[CKRT03] Yannick CHEVALIER, Ralf KÜSTERS, Michaël RUSINOWITCH et Mathieu TURUANI : An NP decision procedure for protocol insecurity with xor. Dans *LICS 2003*, 2003. To appear.

[CKS01] Rohit CHADHA, Max KANOVICH et Andre SCEDROV : Inductive methods and contract-signing protocols : Dans Pierangela SAMARATI, éditeur, *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001), Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 176–185. ACM Press, 2001.

[CLC03] Hubert COMON-LUNDH et Véronique CORTIER : Security properties : two agents are sufficient : Dans Pierpaolo DEGANO, éditeur, *Programming Languages and Systems – Proceedings of the 12th European Symposium on Programming (ESOP 2003), held as part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2003), Warsaw, Poland, April 7-11, 2003*, volume 2618 de *Lecture Notes in Computer Science*, pages 99–113. Springer-Verlag, 2003.

[CMSS03] Rohit CHADHA, John MITCHELL, Andre SCEDROV et Vitaly SHMATIKOV : Contract signing, optimism, and advantage. Dans *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03), Marseille, France, September 3-5, 2003*, 2003. (To appear).

[Cof] Cofi : The common framework initiative for algebraic specification and development. `http://www.brics.dk/Projects/CoFI/`.

[Cor02] Véronique CORTIER : Observational equivalence and trace equivalence in an extension of Spi-calculus. Application to cryptographic protocols analysis. Extended version. Research Report LSV-02-3, Laboratoire Spécification et Vérification, ENS de Cachan, mars 2002.

[Cou97] Patrick COUSOT : Types as abstract interpretations. Dans *Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97), Papers Presented at the Symposium, Paris, France, January 15-17, 1997*, pages 316–331. ACM Press, 1997.

[CS03] Hubert COMON-LUNDH et Vitaly SHMATIKOV : Constraint solving, exclusive or and the decision of confidentiality for security protocols assuming a bounded number of sessions. Research Report LSV-03-1, Laboratoire Spécification et Vérification, ENS de Cachan, janvier 2003.

[CT03] Hubert COMON-LUNDH et Ralf TREINEN : Easy intruder deductions. Research Report LSV-03-8, Laboratoire Spécification et Vérification, ENS de Cachan, avril 2003.

[DGG97] Dennis DAMS, Rob GERTH et Orna GRUMBERG : Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):253–291, mars 1997.

[DH76] Whitfield DIFFIE et Martin HELLMAN : New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, novembre 1976.

[DLMS99] Nancy DURGIN, Patrick LINCOLN, John MITCHELL et Andre SCEDROV : Undecidability of bounded security protocols : Dans Nevin HEINTZE et Edmund CLARKE, éditeurs, *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99), Trento, Italy, July 1999*, 1999.

[DS81] Dorothy E. DENNING et Giovanni M. SACCO : Timestamps in key distribution systems. *Communications of the ACM (CACM)*, 24(8):533–536, août 1981.

[DY83] Danny DOLEV et Andrew C. YAO : On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, mars 1983.

[EHHO99] Anders Strandløv ELKJÆR, Micheal HÖHLE, Hans HÜTTEL et Kasper OVERGÅRD : Towards automatic bisimilarity checking in the spi calculus : Dans C. S. CALUDE et M. J. DINNEEN, éditeurs, *Proceedings of the 2nd International Conference on Discrete Mathematics and Theoretical Computer Science (DMTCS'99) and the 5th Australasian Theory Symposium (CATS'99), held as part of the Australasian Computer Science Week (ACSW'99), Auckland, New Zealand, January 18-21, 1999*, Discrete Mathematics and Theoretical Computer Science. Springer-Verlag, 1999.

[EU00] Article « Cryptologie », Encyclopædia Universalis, 2000.

[EY80] Shimon EVEN et Yacov YACOBI : Relations among public key signature systems. Rapport technique 175, Technion, Haifa, Israel, mars 1980.

[FHG99] F. Javier Thayer FÁBREGA, Jonathan C. HERZOG et Joshua D. GUTTMAN : Strand spaces : Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999.

[GB92] Joseph GOGUEN et Rod BURSTALL : Institutions : Abstract model theory for specification and programming. *Journal of the ACM (JACM)*, 39(1):95–146, janvier 1992.

[GJM99] Juan A. GARAY, Markus JAKOBSSON et Philip MACKENZIE : Abuse-free optimistic contract signing : Dans Michael J. WIENER, éditeur, *Advances in Cryptology – Proceedings of the 19th Annual International Cryptology Conference (CRYPTO'99), Santa Barbara, California, USA, August 15-19, 1999*, volume 1666 de *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.

[GK00] Thomas GENET et Francis KLAY : Rewriting for cryptographic protocol verification : Dans David A. MCALLESTER, éditeur, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17), Pittsburgh, PA, USA, June 17-20, 2000*, volume 1831 de *Lecture Notes in Computer Science*, pages 271–290. Springer-Verlag, 2000.

[GL00] Jean GOUBAULT-LARRECQ : A method for automatic cryptographic protocol verification : Dans José D. P. ROLIM, éditeur, *Proceedings of the 15 International Parallel and Distributed Processing Symposium (IPDPS) Workshops, Cancun, Mexico, May 1-5, 2000*, volume 1800 de *Lecture Notes in Computer Science*, pages 977–984. Springer-Verlag, 2000.

[Gon93] Li GONG : Variations on the themes of message freshness and replay or, the difficulty in devising formal methods to analyze cryptographic protocols. Dans *Proceedings of the 6th IEEE Computer Security Foundations Workshop (CSFW'93), Franconia, New Hampshire, USA, June 15-17, 1993*, pages 131–136. IEEE Computer Society Press, 1993.

[HLS00] James HEATHER, Gavin LOWE et Steve SCHNEIDER : How to prevent type flaw attacks on security protocols. Dans *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00), Cambridge, England, UK, July 3-5, 2000*, pages 255–268. IEEE Computer Society Press, 2000.

[HMMR00] Thomas A. HENZINGER, Rupak MANJUMDAR, Freddy Y.C. MANG et Jean-François RASKIN : Abstract interpretation of game properties : Dans Jens PALSBERG, éditeur, *Proceedings of the 7th International Symposium on Static Analysis (SAS 2000), Santa Barbara, CA, USA, June 29 - July 1, 2000*, volume 1824 de *Lecture Notes in Computer Science*, pages 220–239. Springer-Verlag, 2000.

[HS03] Dominic HUGHES et Vitaly SHMATIKOV : Information hiding, anonymity and privacy : A modular approach. *Journal of Computer Security*, 2003. À pararaître.

[Hüt02] Hans HÜTTEL : Deciding framed bisimilarity : Dans Antonin KUCERA et Richard MAYR, éditeurs, *Proceedings of the Fourth International Workshop on Verification of Infinite-State Systems (INFINITY'02), Brno, Czech Republic, August 24, 2002*, volume 68 de *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2002.

[JN95] Neil. D. JONES et Flemming NIELSON : Abstract interpretation : Semantics-based tool for program analysis : Dans S. ABRAMSKY, D. M. GABBAY et T. S. E. MAIBAUM, éditeurs, *Semantic Modelling*, volume 4 de *Handbook of Logic in Computer Science*. Clarendon Press, 1995.

[JRV00] Florent JACQUEMARD, Michaël RUSINOWITCH et Laurent VIGNERON : Compiling and verifying security protocols : Dans Michel PARIGOT et Andrei VORONKOV, éditeurs, *Logic for Programming and Automated Reasoning – Proceedings of the 7th International Conference (LPAR 2000), Reunion Island, France, November 11-12, 2000*, volume 1955 de *Lecture Notes in Computer Science*, pages 131–160. Springer-Verlag, 2000.

[Ker83] Auguste KERCKHOFFS : La cryptographie militaire. *Journal des Sciences Militaires*, IX, 1883.

[KM00] Steve KREMER et Olivier MARKOWITCH : Optimistic non-repudiable information exchange : Dans Jan BIEMOND, éditeur, *Proceedings of the 21st Symposium on Information Theory in the Benelux (BENELUX-IT 2000), Wassenaar, The Netherlands, May, 2000*, pages 139–146. Werkgemeenschap Informatie- en Communicatietheorie, 2000.

[KR01] Steve KREMER et Jean-François RASKIN : A game-based verification of non-repudiation and fair exchange protocols : Dans Kim Guldstrand LARSEN et Mogens NIELSEN, éditeurs, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01), Aalborg, Denmark, August 20-25, 2001*, volume 2154 de *Lecture Notes in Computer Science*, pages 551–565. Springer-Verlag, 2001.

[KR02] Steve Kremer et Jean-François Raskin : Game analysis of abuse-free contract signing. Dans *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02), Cape Breton, Nova Scotia, Canada, June 24-26, 2002*, pages 206–220. IEEE Computer Society Press, 2002.

[LEW96] Jacques Loeckx, Hans-Dieter Ehrich et Markus Wolf : *Specification of Abstract Data Types.* Wiley & Teubner, 1996.

[Low96] Gavin Lowe : Breaking and fixing the Needham-Shroeder public-key protocol using FDR : Dans Tiziana Margaria et Bernhard Steffen, éditeurs, *Tools and Algorithms for Construction and Analysis of Systems – Proceedings of the Second International Workshop (TACAS'96), Passau, Germany, March 27-29, 1996*, volume 1055 de *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[Low97a] Gavin Lowe : Casper : A compiler for the analysis of security protocols. Dans *Proceedings of 10th IEEE Computer Security Foundations Workshop (CSFW'97), Rockport, Massachusetts, USA, June 10-12, 1997*, pages 18–30. IEEE Computer Society Press, 1997.

[Low97b] Gavin Lowe : A hierarchy of authentication specifications. Dans *Proceedings of 10th IEEE Computer Security Foundations Workshop (CSFW'97), Rockport, Massachusetts, USA, June 10-12, 1997*, pages 31–44. IEEE Computer Society Press, 1997.

[Low98] Gavin Lowe : Towards a completeness result for model checking of security protocols. Dans *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW'98), Rockport, Massachusetts, USA, June 9-11 , 1998*, pages 96–105. IEEE Computer Society Press, 1998.

[LR97] Gavin Lowe et Bill Roscoe : Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, octobre 1997.

[Mac98] Saunders Mac Lane : *Categories for the working mathematician.* Graduate Texts in Mathematics. Springer-Verlag, 2nd. edition édition, 1998.

[Mea96] Catherine Meadows : The NRL protocol analyser : An overview. *Journal of Logic Programming*, 26(2):113–131, février 1996.

[Mer97] Stephan Merz : Rules for abstraction : Dans R. K. Shyamasundar et Kazunori Ueda, éditeurs, *Advances in Computing Science – Proceedings of the Third Asian Computing Science Conferences (ASIAN'97), Kathmandu, Nepal, December 9-11, 1997*, volume 1345 de *Lecture Notes in Computer Science*, pages 32–45. Springer-Verlag, 1997.

[Mil90] Jonathan K. Millen : The interrogator : A tool for cryptographic protocol security. Dans *Proceedings of the 1984 IEEE Symposium on Security and Privacy (S&P'84), Oakland, California, USA, April 1984*, pages 134–141. IEEE Computer Society Press, 1990.

[MMS97] John C. Mitchell, Mark Mitchell et Ulrich Stern : Automated analysis of cryptographic protocols using murφ. Dans *Proceedings of the 1997 IEEE Symposium on Security and Privacy (S&P'97), Oakland, CA, USA, May 4-7, 1997*, pages 141–153. IEEE Computer Society Press, 1997.

[Moc] Mocha : Exploiting modularity in model checking. `http://www-cad.eecs.berkeley.edu/~mocha/`.

[MPW92a] Robin MILNER, Joachim PARROW et David WALKER : A calculus of mobile processes, i. *Information and Computation*, 100(1):1–40, septembre 1992.

[MPW92b] Robin MILNER, Joachim PARROW et David WALKER : A calculus of mobile processes, II. *Information and Computation*, 100(1):41–77, septembre 1992.

[MTP97] Till MOSSAKOWSKI, Andrzej TARLECKI et Wieslaw PAWLOWSKI : Combining and representing logical systems using model-theoretic parchments : Dans Francesco PARISI-PRESICCE, éditeur, *Recent Trends in Algebraic Development Techniques – Selected Papers from the 12th International Workshop on Algebraic Development Techniques (WADT'97), Tarquinia, Italy, June 1997*, volume 1376 de *Lecture Notes in Computer Science*, pages 349–364. Springer-Verlag, 1997.

[Nau65] P. NAUR : Checking of operand types in ALGOL compilers. *BIT*, 5(3):151–163, 1965.

[NH84] Rocco De NICOLA et Matthew HENNESSY : Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[NS78] Roger M. NEEDHAM et Michael D. SCHROEDER : Using encryption for authentication in large networks of computers. *Communications of the ACM (CACM)*, 21(12):993–999, décembre 1978.

[Oos95] Jaap van OOSTEN : Basic category theory. BRICS Lecture Series LS-95-1, Department of Computer Science, University of Aarhus, janvier 1995.

[Pau98] Lawrence C. PAULSON : The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1,2):85–128, 1998.

[Paw01] Wieslaw PAWLOWSKI : Presentations for abstract context institutions : Dans Maura CERIOLI et Gianna REGGIO, éditeurs, *Recent Trends in Algebraic Development Techniques – Selected Papers from the 15th International Workshop on Algebraic Development Techniques (WADT'01), Joint with the CoFI WG Meeting, Genova, Italy, April 1-3, 2001*, volume 2267 de *Lecture Notes in Computer Science*, pages 256–279. Springer-Verlag, 2001.

[PK00] Andreas PFITZMANN et Marit KÖHNTOPP : Anonymity, unobservability, and pseudonymity – A proposal for terminology : Dans Hannes FEDERRATH, éditeur, *Designing Privacy Enhancing Technologies – Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000*, volume 2009 de *Lecture Notes in Computer Science*, pages 1–9. Springer-Verlag, 2000.

[QS81] Jean-Pierre QUEILLE et Joseph SIFAKIS : Specification and verification of concurrent systems in CESAR. Dans *Proceedings of the Fifth International Symposium in Programming*, 1981.

[RSA78] Ronald L. RIVEST, Adi SHAMIR et Leonard ADLEMAN : A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM (CACM)*, 21(2):120–126, février 1978.

[RSG+01] Peter RYAN, Steve SCHNEIDER, Michael GOLDSMITH, Gavin LOWE et Bill ROSCOE : *Modelling and Analysis of Security Protocols.* Addison-Wesley, 2001.

[RT01] Michaël RUSINOWITCH et Mathieu TURUANI : Protocol insecurity with finite number of sessions is NP-complete. Dans *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01), Cape Breton, Nova Scotia, Canada, June 11-13, 2001*, pages 174–187. IEEE Computer Society Press, 2001.

[Sat89] Mahadev SATYANARAYANAN : Integrating security in a large distributed system. *ACM Transactions on Computer Systems (TOCS)*, 7(3):247–280, août 1989.

[SBB+99] Philippe SCHNOEBELEN, Béatrice BÉRARD, Michel BIDOIT, François LAROUSSINIE et Antoine PETIT : *Vérification de logiciels : Techniques et outils du model-checking*. Vuibert, 1999.

[Sch96] Steve SCHNEIDER : Security properties and CSP. Dans *Proceedings of the 1996 IEEE Symposium on Security and Privacy (S&P'96), Oakland, CA, USA*, pages 174–187. IEEE Computer Society Press, 1996.

[SM00a] Vitaly SHMATIKOV et John MITCHELL : Analysis of a fair exchange protocol. Dans *Proceedings of the seventh Symposium on Network and Distributed Systems Security (NDSS'00), San Diego, California, USA, February 2-4, 2000*, pages 119–128. Internet Society, 2000.

[SM00b] Vitaly SHMATIKOV et John MITCHELL : Analysis of abuse-free contract signing : Dans Yair FRANKEL, éditeur, *Proceedings of the 4th International Conference on Financial Cryptography (FC 2000), Anguilla, British West Indies, February 20-24, 2000*, volume 1962 de *Lecture Notes in Computer Science*, pages 174–191. Springer-Verlag, 2000.

[SM02] Vitaly SHMATIKOV et John MITCHELL : Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, juin 2002.

[SMC00] Paul SYVERSON, Catherine MEADOWS et Iliano CERVESATO : Dolev-yao is no better than machiavelli. Dans *Workshop on Issues in the Theory of Security (WITS'00)*, 2000.

[Spo] Security protocols open repository. `http://www.lsv.ens-cachan.fr/spore/`.

[SS96] Steve SCHNEIDER et Abraham SIDIROPOULOS : CSP and anonymity : Dans Elisa BERTINO, Helmut KURTH, Giancarlo MARTELLA et Emilio MONTOLIVO, éditeurs, *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS 96), Rome, Italy, September 25-27, 1996*, volume 1146 de *Lecture Notes in Computer Science*, pages 198–218. Springer-Verlag, 1996.

[SS98] Vitaly SHMATIKOV et Ulrich STERN : Efficient finite-state analysis for large security protocols. Dans *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW'98), Rockport, Massachusetts, USA, June 9-11 , 1998*, pages 105–115. IEEE Computer Society Press, 1998.

[SS99] Paul F. SYVERSON et Stuart G. STUBBLEBINE : Group principals and the formalization of anonymity : Dans Jeanette M. WING, Jim WOODCOCK et Jim DAVIES, éditeurs, *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99), Volume I, Toulouse, France, September 20-24, 1999*, volume 1708 de *Lecture Notes in Computer Science*, pages 814–833. Springer-Verlag, 1999.

[Sto01] Scott D. STOLLER : A bound on attacks on payment protocols. Dans *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01),* *Boston, Massachusetts, USA, June 16-19 2001*, pages 61–70. IEEE Computer Society Press, 2001.

[TMN89] Makoto TATEBAYASHI, Ntsume MATSUZAKI et David B. NEWMAN, Jr. : Key distribution protocol for digital mobile communication systems : Dans Gilles BRASSARD, éditeur, *Advances in Cryptology – Proceedings of the 9th 9th Annual International Cryptology Conference (CRYPTO'89), Santa Barbara, California, USA, August 20-24, 1989*, volume 435 de *Lecture Notes in Computer Science*, pages 324–334. Springer-Verlag, 1989.

[Wei99] Christoph WEIDENBACH : Towards an automatic analysis of security protocols in first-order logic : Dans Harald GANZINGER, éditeur, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16), Trento, Italy, July 7-10, 1999*, volume 1632 de *Lecture Notes in Computer Science*, pages 314–328. Springer-Verlag, 1999.