Résolution ordonnée avec sélection et classes décidables de la logique du premier ordre

Jean Goubault-Larrecq
LSV/CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan
61 avenue du président-Wilson, F-94235 Cachan Cedex
goubault@lsv.ens-cachan.fr
Phone: +33-1 47 40 75 68 Fax: +33-1 47 40 24 64

5 octobre 2010

Résumé

Ce document sert de notes de cours pour le cours de démonstration automatique de théorèmes (2-5), première partie, magistère MPRI 2ème année, éditions 2006–2007, 2007-08, 2008-09, 2009-10. Il s'agit de la version 13 du 05 octobre 2010, faisant suite à la version 12 du 09 octobre 2009, faisant suite à la version 11 du 24 novembre 2006, faisant suite à la version 10 du 19 septembre 2006, la version 9 du 12 octobre 2005, à la version 8 du 02 décembre 2004, à la version 7 du 08 novembre 2004, à la version 6 du 13 octobre 2004, à la version 5 du 12 octobre 2004, et à la version 4 du 06 octobre 2004.

Les versions précédentes servaient de notes de cours pour le cours de vérification automatique de protocoles cryptographiques de DEA Programmation, édition 2002–2003. Il s'agissait de la version 3. La version 1 date du vendredi 31 janvier 2002. La version 2 date du mercredi 19 mars 2003. La version 3 date du mercredi 23 avril 2003.

Il est question ici de techniques de résolution, c'est-à-dire de démonstration automatique, de techniques d'automates d'arbres et de contraintes ensemblistes. Le parti pris de ce document est de ramener tous les problèmes d'automates et de contraintes ensemblistes systématiquement à des problèmes de démonstration automatique, ce qui fournit une certaine unité au sujet.

Table des matières

Remerciements

Merci à Muriel Roger, qui a trouvé plusieurs erreurs dans une première version de ce document ; et à Arnaud Spiwack, Michaël Lienhardt, Bruno Marnette, et Sergiu Bursuc, qui en ont trouvé quelques autres dans des versions ultérieures.

1 Préliminaires

Fixons une fois pour toutes un ensemble infini dénombrable de symboles de prédicats unaires. Un *atome* est un object de la forme $P(t_1, \ldots, t_n)$, où P est un symbole de prédicat et t_1, …, t_n sont des termes du premier ordre sur la signature Σ.

Dans la suite, nous nous restreindrons toujours au cas $n = 1$, sauf temporairement au début de la section 7. Autrement dit, les atomes seront de la forme $P(t)$, où P est un symbole de prédicat unaire et t est un terme du premier ordre sur la signature Σ. Ceci simplifiera les notations. D'autre part, nous nous intéresserons spécifiquement plus tard à une sous-classe de la classe de la logique du premier ordre (la classe *monadique*) présentant naturellement cette restriction. Finalement, ceci n'entache pas la généralité de nos énoncés, puisque l'on peut toujours coder $P(t_1, \ldots, t_n)$ sous la forme $P(f_P(t_1, \ldots, t_n))$, où P est maintenant unaire et il y a bijection entre les symboles P est les symboles de fonction n-aires f.

Un *littéral* est soit un littéral *positif* $+P(t)$, soit un littéral *négatif* $-P(t)$. Une *clause* est une disjonction de littéraux $\pm_1 P_1(t_1) \vee \pm_2 P_2(t_2) \vee \ldots \vee \pm_k P_k(t_k)$. (Par *disjonction* de littéraux, on entend ici un ensemble fini de littéraux.) Si $k = 0$, on écrit cette disjonction $\Box$; il s'agit de la *clause vide*.

Certaines clauses sont particulièrement importantes, il s'agit des clauses *de Horn*, qui sont celles contenant au plus un littéral positif. Nous ne nous limiterons pas au cas des clauses de Horn, mais mentionnerons à l'occasion certains résultats intéressants qui ne sont valables que pour des ensembles de clauses de Horn.

Les clauses de Horn sont regroupées en clauses définies et clauses buts. Les *clauses définies* sont les clauses ayant exactement un littéral positif. On notera typiquement la clause définie $+P(t) \vee -P_1(t_1) \vee \ldots \vee -P_n(t_n)$ comme suit :

$$P(t) \Leftarrow P_1(t_1), \ldots, P_n(t_n) \tag{1}$$

Dans le cas où $n = 0$, on notera aussi cette clause $P(t)$, et on l'appellera un *fait*. La notation $\Leftarrow$ est censée rappeler la notation d'implication. Le langage Prolog utilise en général la notation `:-` au lieu de $\Leftarrow$.

Un *but*, aussi appelé *clause négative*, est une clause ne contenant que des littéraux négatifs. Il s'ensuit que tout but est une clause de Horn. On écrit parfois le but $-P_1(t_1) \vee \ldots \vee -P_n(t_n)$ sous la forme :

$$\bot \Leftarrow P_1(t_1), \ldots, P_n(t_n) \tag{2}$$

où $\bot$ dénote le faux.

Symétriquement, une clause *positive* en est une qui ne contient que des littéraux positifs.

2 Sémantique

2.1 Sémantique de Tarski

Une *structure* I est la donnée d'un ensemble non vide D (le *domaine*), de sous-ensembles I_P de D, un pour chaque prédicat P, et d'applications $I_f : D^n \to D$ pour chaque fonction $f \in \Sigma$, d'arité n. Étant donné un *environnement* ρ, c'est-à-dire une application qui associe à chaque variable un élément de D, la *valeur* $I[\![t]\!]\rho$ d'un terme t est définie par :

– $I[\![x]\!]\rho = \rho(x)$ pour chaque variable x ;

– $I[\![f(t_1, \ldots, t_n)]\!]\rho = I_f(I[\![t_1]\!]\rho, \ldots, I[\![t_n]\!]\rho)$.

On définit la relation $I, \rho \models P(t)$, et l'on dit que $P(t)$ est *vraie* dans I, ρ, si et seulement si $I[\![t]\!]\rho$ est dans I_P.

Pour toute clause C, on pose $I, \rho \models C$ si et seulement s'il existe un littéral $+P(t)$ dans C tel que $I, \rho \models P(t)$, ou un littéral $-P(t)$ dans C tel que $I, \rho \not\models P(t)$. Ceci peut être reformulé dans le cas de clauses de Horn, comme suit. Par convention, posons que $I, \rho \not\models \bot$. On a $I, \rho \models C$, où C est une clause de Horn $A \Leftarrow A_1, \ldots, A_n$, si et seulement si $I, \rho \not\models A_i$ pour au moins un i, $1 \leq i \leq n$, ou $I, \rho \models A$.

La structure I est un *modèle* de la clause C si et seulement si $I, \rho \models C$ pour tout environnement ρ ; on écrit alors $I \models C$. I est un modèle d'un ensemble S de clauses si et seulement si $I \models C$ pour toute clause C dans S ; on écrit $I \models S$ dans ce cas.

On dit qu'une clause, resp. un ensemble S de clauses, est *satisfiable* si et seulement elle (resp. il) a un modèle.

2.2 Sémantique de Herbrand

Un terme, un atome, un littéral, une clause est *clos(e)* si et seulement s'il (si elle) ne contient aucune variable libre. Une *substitution* σ est une application des variables vers les termes. On note $t\sigma$ le résultat de l'application de la substitution σ au terme t : $x\sigma = \sigma(x)$, $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. Une *instance* de t est tout terme de la forme $t\sigma$, pour une substitution σ. On notera aussi $\sigma\theta$ la *composition* de σ et θ, c'est-à-dire l'unique substitution telle que $t(\sigma\theta) = (t\sigma)\theta$ pour tout terme t : $\sigma\theta$ associe à toute variable x le terme $\sigma(x)\theta$. Un *renommage* ϱ est une substitution qui est une bijection entre variables.

L'*univers de Herbrand* H est l'ensemble de tous les termes clos. Une *structure de Herbrand* est toute structure I dont le domaine est H, et telle que I_f envoie tout n-uplet de termes clos t_1, …, t_n vers le terme clos $f(t_1, \ldots, t_n)$, pour tout f d'arité n. (On suppose ici, sans perte de généralité, que Σ contient au moins une constante, c'est-à-dire un symbole de fonction d'arité 0, de sorte que H est non vide.) Il est facile de voir que ceci est bien défini, et que

$$I[\![t]\!]\rho = t\rho \tag{3}$$

où ρ est vu comme une substitution sur le côté droit et comme un environnement sur le côté gauche. Un *modèle de Herbrand* de S est une structure de Herbrand qui est un modèle de S.

On a le lemme facile suivant :

Lemme 1 *Soit σ une substitution, I une structure, ρ un environnement. Posons $\rho[\sigma]$ l'environnement qui à tout x dans le domaine de σ associe $I[\![x\sigma]\!]\rho$, et à tout autre x associe $\rho(x)$. Alors, pour tout terme t, $I[\![t\sigma]\!]\rho = I[\![t]\!](\rho[\sigma])$.*

Démonstration. Récurrence immédiate sur la structure de t. □

Proposition 2 *Un ensemble de clauses S est satisfiable si et seulement s'il a un modèle de Herbrand.*

Démonstration. Si S a un modèle de Herbrand, S est clairement satisfiable. Réciproquement, supposons que S est satisfiable, et soit I un modèle de S. On définit la structure de Herbrand I' par $I'_P = \{u \text{ clos} | I[\![u]\!] \in I_P\}$. (Comme t est clos, $I[\![t]\!]\rho$ est indépendant de ρ ; on note cette valeur $I[\![t]\!]$.) Comme I est un modèle de S, pour toute clause C de S, pour tout environnement ρ, on a $I, \rho \models C$. C'est en particulier le cas lorsque ρ est définissable, c'est-à-dire qu'il existe une substitution ρ' envoyant les variables vers des termes clos telle que $\rho(x) = I[\![x\rho']\!]$ pour toute variable x, autrement dit telle que $\rho = \mathrm{id}[\rho']$. Donc $I \models C\rho'$ par le lemme 1, d'où $I', \rho' \models C$ par définition de I' et l'équation (3). Donc I' est un modèle de Herbrand de S. □

Remarque 1 *Nous ne considérons que des clauses ici. Dans le cas de formules générales du premier ordre, ce lemme est toujours valide, à condition de ne considérer que des formules universelles, c'est-à-dire de la forme $\forall x_1, \ldots, x_n \cdot F$, où F ne contient aucun quantificateur. Ce lemme est faux pour les formules existentielles.*

Une structure de Herbrand I peut être caractérisée, d'une autre façon, par un ensemble d'atomes clos : à savoir les atomes clos $P(t)$ tels que $I \models P(t)$. (Encore une fois, l'environnement ρ est superflu ici, donc omis.) Réciproquement, si E est un ensemble d'atomes clos, on peut définir l'interprétation de Herbrand correspondante I par $I_P = \{t \text{ clos} | P(t) \in E\}$. On considérera dans la suite, sans le dire explicitement, qu'une structure de Herbrand *est* un ensemble d'atomes clos.

2.3 Clauses de Horn et plus petits modèles

En particulier, les structures de Herbrand peuvent alors être ordonnées par inclusion $\subseteq$. On peut montrer (exercice !) que tout ensemble satisfiable de clauses de Horn a un *plus petit modèle de Herbrand*. Ceci n'est pas en général vrai pour un ensemble de clauses qui n'est pas de Horn.

Une conséquence immédiate est que tout ensemble S de clauses définies a un plus petit modèle. En effet, S a un modèle, à savoir celui qui contient tous les atomes clos.

Une autre caractérisation des plus petits modèles de Herbrand, qui est plus évidente pour les spécialistes de Prolog, est la suivante. Fixons un ensemble S de clauses de Horn. Soit $\mathcal{F}$ l'ensemble des atomes clos, union $\bot$. Posons par convention $\bot\sigma = \bot$ pour toute substitution σ. On considère $\bot$ comme étant clos. On définit l'opérateur T_S de $\mathbb{P}(\mathcal{F})$ vers $\mathbb{P}(\mathcal{F})$ par :

$$\begin{aligned} T_S(I) \;=\; & \{A\sigma | A \Leftarrow A_1, \ldots, A_n \in S, \\ & \quad A\sigma \text{ clos}, A_1\sigma \in I, \ldots, A_n\sigma \in I\} \end{aligned}$$

T_S est monotone pour l'ordre d'inclusion. Par le théorème de Tarski, il a un plus petit point fixe. (Voir annexe A.1.) En fait, il est facile de voir que ce plus petit point fixe est $\bigcup_{n\in\mathbb{N}} T_S^n(\emptyset)$ (exercice). Ce plus petit point fixe est alors le plus petit modèle de Herbrand de S dans le cas où il ne contient pas $\bot$. S'il contient $\bot$, alors S est insatisfiable.

2.4 Langages et reconnaissabilité

Avant de passer aux techniques de démonstration automatique sur des ensembles de clauses du premier ordre, notons que les ensembles de clauses de Horn définissent des *langages de termes*. Ceci n'est pas sans rappeler la définition de langages reconnus par automates. Nous verrons en effet que les automates (d'arbres) sont des cas particuliers d'ensembles de clauses de Horn.

Soit S un ensemble satisfiable de clauses de Horn, et P un symbole de prédicat. Le *langage* $L_P(S)$ de S à l'*état* P est l'ensemble des termes clos t tels que $P(t)$ est dans le plus petit modèle de Herbrand de S. Les éléments de $L_P(S)$ sont appelés les termes *reconnus* à P par S.

Par abus de langage, on dira que P est *vide* dans S si et seulement si $L_P(S)$ est vide, et de même pour toute autre propriété. Les lemmes suivants sont faciles. Le premier caractérise la reconnaissabilité, sémantiquement.

Lemme 3 *Soit S un ensemble satisfiable de clauses de Horn. Le terme clos t est reconnu à P par S si et seulement si S plus la clause $\bot \Leftarrow P(t)$ est insatisfiable.*

Démonstration. Si $t \in L_P(S)$, alors par définition $P(t)$ est dans le plus petit modèle de Herbrand de S, donc dans tout modèle de Herbrand de S ; donc S plus $\bot \Leftarrow P(t)$ est insatisfiable. Réciproquement, si S plus $\bot \Leftarrow P(t)$ est insatisfiable, alors tout modèle de S doit ne pas satisfaire $\bot \Leftarrow P(t)$, et donc contenir $P(t)$; donc $t \in L_P(S)$. □

Le deuxième lemme caractérise la vacuité.

Lemme 4 *Étant donné un ensemble S satisfiable de clauses de Horn, P est vide dans S si et seulement si S plus la clause $\bot \Leftarrow P(x)$ est satisfiable.*

On appellera parfois dans la suite des clauses de la forme $\bot \Leftarrow P(x)$ des *requêtes*.

Démonstration. Si P est vide, alors le plus petit modèle de Herbrand de S ne contient pas d'atome clos de la forme $P(t)$, donc satisfait $\bot \Leftarrow P(x)$.

Réciproquement, si S plus $\bot \Leftarrow P(x)$ est satisfiable, alors son plus petit modèle de Herbrand ne contient pas d'atome clos de la forme $P(t)$. Comme tout modèle de S plus $\bot \Leftarrow P(x)$ est aussi un modèle de S, le plus petit modèle de Herbrand de S est inclus dans celui de S plus $\bot \Leftarrow P(x)$, donc ne contient pas d'atome clos de la forme $P(t)$ non plus ; c'est-à-dire que P est vide dans S. □

Le troisième lemme caractérise la *vacuité d'intersection*, c'est-à-dire, étant donnés un nombre fini de symboles de prédicats P_1, …, P_n, la question de la vacuité de $L_{P_1}(S) \cap \ldots \cap L_{P_n}(S)$. (On dira par abus de langage que l'intersection de P_1, …, P_n est vide dans ce cas.)

Lemme 5 *Étant donné un ensemble S satisfiable de clauses de Horn, l'intersection de P_1, …, P_n est vide dans S si et seulement si S plus la clause $\bot \Leftarrow P_1(x), \ldots, P_n(x)$ est satisfiable.*

On appellera de telles clauses des *requêtes conjonctives*.

Démonstration. La démonstration est similaire à celle du lemme 4, et laissée en exercice. □

Remarque 2 *Il est à noter que la vacuité se traduit en une propriété de satisfiabilité. Dans le cadre des protocoles cryptographiques, un modèle du codage en clauses de Horn d'un protocole est une preuve de sécurité, une observation due à Selinger [Sel01]. Dans ce cadre, une propriété de sécurité d'un protocole cryptographique se traduit en une propriété de vacuité d'un langage. On pourra penser "sécurité = vacuité de l'ensemble des attaques", encore que ceci soit davantage un moyen mnémotechnique qu'une véritable correspondance.*

3 Résolution

Comment peut-on détecter si un ensemble de clauses est insatisfiable ? Une des techniques de démonstration automatique les plus connues, et les plus efficaces en pratique, est la *résolution*, inventée par J. Alan Robinson en 1965.

La résolution au premier ordre utilise la notion d'*unification*. Un *unificateur* σ de deux termes ou atomes s et t est une substitution telle que $s\sigma = t\sigma$. La substitution σ est *plus générale* que σ' si et seulement s'il existe une troisième substitution θ telle que $\sigma' = \sigma\theta$. Un *unificateur le plus général*, ou *mgu* (abrégeant l'expression anglaise "*most general unifier*") de s et de t est un unificateur de s et t qui est plus général que tout autre unificateur de s et t.

Si s et t sont unifiables, alors ils ont un unificateur le plus général. On notera $\mathrm{mgu}\ (s \doteq t)$ l'un de ces mgu. On peut montrer que les unificateurs les plus généraux ne diffèrent que par un renommage ; c'est-à-dire, si σ et σ' en sont deux, alors il existe un renommage ϱ tel que $\sigma = \sigma'\rho$. On peut aussi montrer qu'il existe parmi les unificateurs les plus généraux certains σ qui sont *idempotents* ; autrement dit, $\sigma\sigma = \sigma$. Cette condition est équivalente au fait qu'aucune variable du domaine $\mathrm{dom}\ \sigma = \{x | \sigma(x) \neq x\}$ n'est libre dans aucun term $\sigma(x)$, $x \in \mathrm{dom}\ \sigma$.

La règle de *résolution* est :

$$\frac{C \vee +A \quad C' \vee -A'}{C\sigma \vee C'\sigma}\ \sigma = \mathrm{mgu}\ (A \doteq A')$$

où la condition $\sigma = \mathrm{mgu}\ (A \doteq A')$ signifie que A et A' sont unifiables, et que σ est leur mgu. D'autre part, la notation $C \vee +A$ en prémisse sous-entend que $+A$ n'est pas dans la disjonction C, et similairement pour $C' \vee -A'$. On sous-entend aussi que les deux prémisses ont des ensembles de variables libres disjoints, ce qui peut être effectué en renommant l'une ou l'autre des prémisses. On appelle la conclusion le *résolvant binaire* des deux prémisses.

Cette règle se spécialise dans le cas des clauses de Horn à la règle :

$$\frac{A \Leftarrow A_1, \ldots, A_m \quad B \Leftarrow B_1, \ldots, B_n}{A\sigma \Leftarrow B_1\sigma, \ldots, B_n\sigma, A_2\sigma, \ldots, A_m\sigma}\ \sigma = \mathrm{mgu}\ (A_1 \doteq B)$$

La règle de résolution binaire est complète pour les clauses de Horn : si un ensemble S de clauses de Horn est insatisfiable, alors on peut déduire la clause vide $\bot$ en un nombre fini d'étapes de résolution binaire.

Dans le cas de clauses générales, la résolution binaire n'est *pas* complète. Par exemple, l'ensemble de clauses suivant est insatisfiable :

$$+P(x) \vee +P(y) \qquad -P(x) \vee -P(y) \tag{4}$$

Ici x, y sont deux variables distinctes. Cependant, l'unique résolvant binaire, à renommage près, de ces deux clauses est $+P(x) \vee -P(x)$, et à partir de celui-là, aucune nouvelle clause n'est produite ; en particulier, la clause vide ne sera jamais produite.

Ceci est facile à corriger ; il suffit de nous donner la possibilité d'utiliser une règle additionnelle, dite de *factorisation* (positive) :

$$\frac{C \vee +A \vee +A'}{C\sigma \vee +A\sigma}\,\sigma = \mathrm{mgu}\,(A \doteq A')$$

On notera que cette règle n'est jamais applicable sur une clause de Horn — on applique la même convention que plus haut, et l'on comprend que $C \vee +A \vee +A'$ en prémisse sous-entend que A et A' sont distincts et hors de C.

On a alors le résultat que la combinaison des deux règles de résolution binaire et de factorisation positive est complète : tout ensemble insatisfiable de clauses admet une dérivation de la clause vide $\Box$ à l'aide de ces deux règles. (Cette combinaison des deux règles est appelée la *résolution*.) Nous montrerons un théorème plus fort en section 3.2.

Dans l'exemple ci-dessus, on dérive $+P(x)$ résolvant $-P(x) \vee -P(y)$ avec $+P(x)$, on obtient $-P(y)$; en résolvant cette dernière clause avec $+P(x)$ de nouveau, on obtient la clause vide.

On appelle *facteur* (positif) d'une clause C toute clause C' que l'on peut obtenir à partir de C par 0, 1, ou plusieurs applications de la règle de factorisation (positive). On appelle *résolvant* de deux clauses C_1 et $C' \vee -A'$ tout résolvant binaire $C \vee C'$ entre un facteur $C \vee +A$ de C_1 et $C' \vee -A'$.

On a le résultat facile :

Lemme 6 (Correction) *Soit S un ensemble de clauses, et S' n'importe quel ensemble de clauses obtenues à partir de S par résolution binaire et factorisation positive. Tout modèle de S est un modèle de S'.*

En particulier, si $\Box$ est déductible par résolution à partir de S, alors S est insatisfiable.

Exercice 1 *On considère la règle de* factorisation négative *:*

$$\frac{C \vee -A \vee -A'}{C\sigma \vee +A\sigma}\,\sigma = \mathrm{mgu}\,(A \doteq A')$$

On définit les facteurs négatifs de manière similaire aux facteurs positifs, et la règle de résolution générale *par :*

$$\frac{C \vee +A_1 \vee \ldots \vee +A_m \quad C' \vee -A'_1 \vee \ldots \vee -A'_n}{C\sigma \vee C'\sigma}$$

où $m \geq 1$, $n \geq 1$, *et* $\sigma = \mathrm{mgu}\ (A_1 \doteq A_2, \ldots, A_{m-1} \doteq A_m, A'_1 \doteq A'_2, \ldots, A'_{n-1} \doteq A'_n, A_m \doteq A'_n)$. *Montrer par récurrence sur* $n \geq 1$ *que tout résolvant général est déjà un résolvant, autrement dit que la règle de factorisation négative est superflue. (Ne pas oublier que les deux prémisses sont supposées avoir des ensembles de variables libres disjoints.)*

3.1 Raffinements de la résolution, résolution ordonnée avec sélection

Il existe un certain nombre de restrictions de la règle de résolution qui restent complètes :

- On peut restreindre la prémisse $C \vee +A$ de la règle de résolution binaire à être une clause positive ; cette restriction est appelée la règle d'*hyperrésolution positive*.
  Dans le cas des clauses de Horn, ceci revient à restreindre cette prémisse à être une clause réduite à un seul littéral positif $+A$, c'est-à-dire à être un fait.
  La restriction de la règle de résolution binaire où l'une des deux prémisses de la règle de résolution binaire est une clause *unitaire*, c'est-à-dire ne contenant qu'un littéral, est appelée *résolution unitaire*. La résolution unitaire est donc complète pour les clauses de Horn, mais elle ne l'est *pas* dans le cas de clauses générales. Considérer par exemple l'ensemble insatisfiable :
  $$\begin{array}{ll} +P(a) \vee +P(b) & +P(a) \vee -P(b) \\ -P(a) \vee +P(b) & -P(a) \vee -P(b) \end{array} \tag{5}$$
- On peut restreindre la prémisse $C \vee -A$ de la règle de résolution binaire à être une clause négative ; on obtient ainsi l'*hyperrésolution négative*.
- En général, la *résolution sémantique* est complète, où l'une des prémisses est contrainte à être fausse dans une interprétation I fixée. On notera que l'on retrouve l'hyperrésolution positive en prenant I l'interprétation de Herbrand fausse de tout atome clos, et l'hyperrésolution négative en choisissant pour I l'interprétation de I vraie de toute atome clos.
- La *résolution ordonnée* demande que :
  - A soit maximal dans la clause $C \vee +A$, et A' maximal dans $C' \vee -A'$, où $C \vee +A$ et $C' \vee -A'$ sont les prémisses de la règle de résolution binaire,
  - et A, A' soient tous les deux maximaux dans la prémisse $C \vee +A \vee +A'$ de factorisation,

  ceci pour un ordre strict stable $\succ$ sur les atomes. On dit que $\succ$ est *stable* si et seulement si $A \succ B$ implique $A\sigma \succ B\sigma$ pour toute substitution telle que $A\sigma$ et $B\sigma$ sont clos. (Il est en fait possible de relaxer la condition de stabilité, voir [dN95].)
  Précisons qu'un atome A est *maximal* dans une clause C s'il n'existe pas de littéral $\pm B$ dans C tel que $B \succ A$.
  La résolution ordonnée est un raffinement complet de la résolution, qui est très restrictif.
- La *résolution ordonnée avec sélection* est similaire, mais utilise en plus une fonction sel , dite de *sélection*, qui à chaque clause C associe un sous-ensemble (possiblement vide) de ses littéraux négatifs. L'idée est que l'on utilise la fonction de sélection pour déterminer sur quel littéral résoudre ; on n'utilise l'ordre que par défaut.
  Regardons d'abord le cas (plus simple) où $\mathrm{sel}\ (C)$ retourne soit l'ensemble vide (on dit qu'aucun littéral n'est sélectionné dans C) soit un singleton $\{-A\}$ (on dit que $-A$ est le

littéral sélectionné dans C). Précisément, la règle de résolution ordonnée avec sélection est alors :

$$\frac{C \vee +A_1 \vee \ldots \vee +A_n \quad C' \vee -A'}{C\sigma \vee C'\sigma}$$

où $n \geq 1$, $\sigma = \mathrm{mgu}\ (A_1 \doteq A', \ldots, A_n \doteq A')$, aucun littéral n'est sélectionné dans $C \vee +A_1 \vee \ldots \vee +A_n$ et $A_1, \ldots, A_n$ sont maximaux dans $C \vee +A_1 \vee \ldots \vee +A_n$; et $-A'$ est sélectionné dans $C' \vee -A'$ ou bien aucun littéral n'y est sélectionné mais A' est maximal dans $C' \vee -A'$.

On remarquera que l'on a combiné résolution binaire et factorisation en une seule règle. La prémisse de droite $C' \vee -A'$ s'appelle la prémisse *principale*, celle de gauche la prémisse *auxiliaire*.

En général, on peut demander que *plusieurs* littéraux soient sélectionnés, $-A'_1$, ..., $-A'_\ell$ au lieu d'un seul, $-A'$, dans la prémisse principale de droite. On unifiera chacun avec une nouvelle prémisse auxiliaire. Dans le cas général, donc, sel est une fonction quelconque qui à toute clause associe un sous-ensemble, possiblement vide, de ses littéraux négatifs, et la règle de résolution ordonnée avec sélection est :

$$\frac{\overbrace{C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}}^{1 \leq i \leq \ell} \quad C' \vee -A'_1 \vee \ldots \vee -A'_\ell}{C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma}$$

où :

(i) $n_i \geq 1$ pour tout i, $1 \leq i \leq \ell$;

(ii) $\sigma = \mathrm{mgu}\ \{A_{ij} \doteq A'_i | 1 \leq i \leq \ell, 1 \leq j \leq n_i\}$;

(iii) $\mathrm{sel}\ (C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}) = \emptyset$ et $A_{i1}, \ldots, A_{in_i}$ sont maximaux dans $C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}$, pour tout i, $1 \leq i \leq \ell$;

(iv) $\mathrm{sel}\ (C' \vee -A'_1 \vee \ldots \vee -A'_\ell) = \{-A'_1, \ldots, -A'_\ell\}$ et $\ell \geq 1$, ou bien aucun littéral n'est sélectionné, $\ell = 1$ et $-A'_1$ est maximal dans $C' \vee -A'_1 \vee \ldots \vee -A'_\ell = C' \vee -A'_1$.

Il est à noter que la fonction sel est *quelconque*, et n'est restreinte que par le fait qu'elle doit sélectionner des littéraux négatifs. (On pourrait les choisir positifs, ceci ne changerait rien en terme de complétude.)

Si l'on choisit comme fonction de sélection la fonction qui retourne toujours l'ensemble vide, on note qu'on retrouve la résolution ordonnée. Si au contraire $\mathrm{sel}\ (C)$ retourne l'ensemble de tous les littéraux négatifs de C, on obtient un raffinement de l'hyperrésolution positive où les littéraux positifs sur lesquels on résout sont de plus restreints à être maximaux dans leur clause. La résolution ordonnée avec sélection est complète, et donc ces derniers raffinements aussi, comme nous allons le montrer.

D'autres raffinements complets peuvent être trouvés dans [CL73] : la lock-résolution de Boyer, la stratégie "set-of-support", la stratégie linéaire notamment. Certains raffinements ne sont complets que sur des ensembles de clauses de Horn : résolution unitaire, résolution input [CL73], résolution avec sélection libre [dN95].

3.2 Complétude, arbres sémantiques

Soit $A_1, A_2, \ldots, A_i, \ldots$ une énumération des atomes clos. Une *interprétation partielle* sur cette énumération est une liste finie $\pm_1 A_1, \pm_2 A_2, \ldots, \pm_k A_k$. Si A_i apparaît avec le signe $+$, on dit que A_i est *vrai* dans l'interprétation partielle ; il est *faux* s'il apparaît avec le signe $-$. Sinon, A_i est *indéfini* dans l'interprétation partielle.

L'*arbre de Herbrand* est l'arbre dont les sommets sont les interprétations partielles. L'interprétation partielle $\pm_1 A_1, \pm_2 A_2, \ldots, \pm_k A_k$ a deux fils, qui sont $\pm_1 A_1, \pm_2 A_2, \ldots, \pm_k A_k, -A_{k+1}$ et $\pm_1 A_1, \pm_2 A_2, \ldots, \pm_k A_k, +A_{k+1}$. (Ceci, bien sûr, à condition que A_{k+1} existe ; sinon, $\pm_1 A_1$, $\pm_2 A_2, \ldots, \pm_k A_k$ n'a pas de fils.) La racine de l'arbre ϵ est l'interprétation partielle vide.

Les branches maximales (infinies en général) de l'arbre de Herbrand sont naturellement en bijection avec les interprétations de Herbrand. Si I est une interprétation de Herbrand, on obtient une branche maximale qui passe par les nœuds ϵ, puis $\pm_1 A_1$, puis $\pm_1 A_1, \pm_2 A_2$, etc., où $\pm_i$ est le signe $+$ si $A_i \in I$, $-$ sinon. Réciproquement, sur toute branche maximale, on note que tout atome A_i apparaît avec un signe unique $\pm_i$ dans toute interprétation partielle de longueur au moins i ; on obtient donc une interprétation de Herbrand contenant exactement les A_i tels que $\pm_i$ est le signe $+$.

Théorème 7 (Herbrand) *Étant donné un ensemble de clauses S du premier ordre, les trois conditions suivantes sont équivalentes :*

1. *S est insatisfiable ;*
2. *l'ensemble $S\downarrow$ des instances closes de S est insatisfiable ;*
3. *il existe un ensemble fini S_0 d'instances closes de S qui est insatisfiable.*

Démonstration. Supposons S satisfiable, et I un modèle de Herbrand de S (cf. proposition 2). Alors pour toute instance close $C\rho$ d'une clause C de S, comme $I, \rho \models C$, par l'équation (3), on a $I \models C\rho$. Donc I est un modèle de $S\downarrow$. On en déduit que 2 implique 1.

Réciproquement, si I est une interprétation de Herbrand qui satisfait $S\downarrow$, par l'équation (3) de nouveau, I satisfait S. On en déduit que 1 implique 2.

Clairement, 3 implique 2. Il ne reste qu'à démontrer que 2 implique 3. Fixons une énumération quelconque A_1, A_2, A_3, …, des atomes clos. Disons qu'une clause close C est *fausse* au nœud $N = \pm_1 A_1, \pm_2 A_2, \ldots, \pm_k A_k$ si et seulement si pour tout littéral $\pm A$ de C, le littéral opposé $\mp A$ est dans la liste N. On remarque que si C est fausse au nœud N, alors C est fausse dans toute interprétation I passant par le nœud N. Réciproquement, si C est fausse dans l'interprétation I, alors il existe un nœud N où C est fausse : si l'on pose $C = \pm_1 A_{i_1} \vee \ldots \vee \pm_m A_{i_m}$, $i_1 < \ldots < i_m$, alors on peut prendre pour N n'importe quel nœud qui contient $\mp_1 A_{i_1}$, …, $\mp_m A_{i_m}$.

On remarque maintenant que, pour toute interprétation de Herbrand I, par 2 il existe une instance close $C\sigma$ d'une clause C de S qui est fausse dans I. Par la remarque ci-dessus, $C\sigma$ est donc fausse à un nœud N de l'arbre sémantique. Appelons *nœud d'échec* pour S tout nœud minimal rendant fausse au moins une instance close d'une clause de S. (Par minimal on entend minimal pour la longueur du nœud vu comme liste.) L'*arbre sémantique clos* est le sous-arbre de l'arbre sémantique de même racine ϵ, et dont les feuilles sont les nœuds d'échec. L'arbre sémantique clos n'a pas de branche infinie : sinon elle définirait une interprétation I, pour laquelle

au moins une instance close $C\sigma$ d'une clause C de S serait fausse, menant à l'existence d'un nœud d'échec sur la branche I, ce qui serait absurde. D'autre part, l'arbre sémantique clos est à branchement fini, donc d'après le lemme de König, il est fini. (Voir annexe A.2 pour plus d'informations sur le lemme de König.) En particulier, il a un nombre fini de feuilles, c'est-à-dire de nœuds d'échec. Choisissant pour chaque nœud d'échec N une instance close $C_N\theta_N$ d'une clause C_N de S qui est fausse en N, on obtient que l'ensemble fini des $C_N\theta_N$ est insatisfiable. □

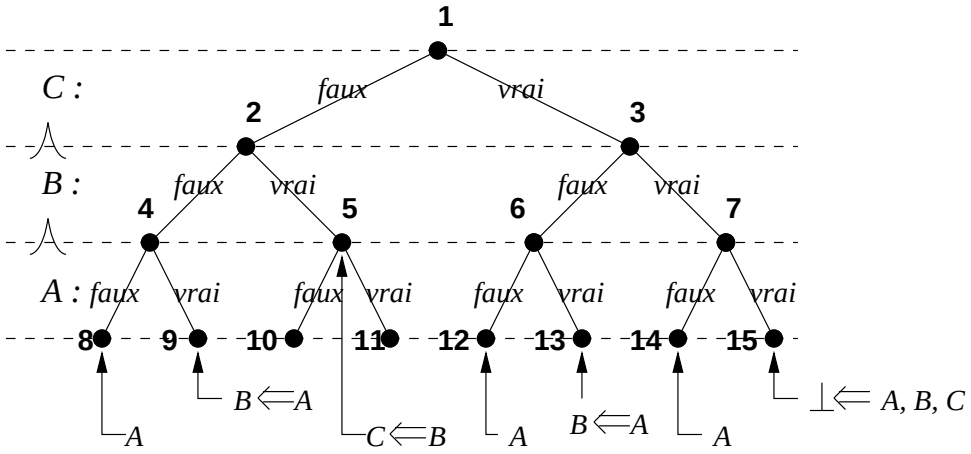


FIG. 1 – Un arbre sémantique

Illustrons les notions employées dans la preuve sur un exemple. En figure 1, on a représenté un arbre sémantique (fini) sur les trois atomes C, B, A, dans cet ordre. Autrement dit, $A_1 = C$, $A_2 = B$, $A_3 = A$. Le nœud 1 est l'interprétation partielle vide ϵ, le nœud 2 est l'interprétation partielle $-C$, le nœud 3 est $+C$, le nœud 4 est $-C, -B$, le nœud 5 est $-C, +B$, etc. Notons que la clause $C \Leftarrow B$, c'est-à-dire $+C \vee -B$, est fausse en $-C, +B$ (nœud 5). On note ainsi que si S est l'ensemble des clauses $+A$; $B \Leftarrow A$; $C \Leftarrow B$; $\bot \Leftarrow A, B, C$, alors toute branche rencontre un nœud qui rend au moins une des clauses fausse. Si l'on choisit les nœuds minimaux (les plus hauts possibles) parmi ces derniers, on obtient les nœuds d'échec, que l'on a représentés en figure 1 par des nœuds N sur lesquels une flèche pointe ; la source de la flèche étant la clause C_N choisie qui est fausse en N.

On peut déjà en déduire la complétude de la résolution, par récurrence sur la taille de l'arbre sémantique clos. Si la racine ϵ est un nœud d'échec, alors nécessairement C_ϵ est la clause vide, car aucune clause non vide n'est fausse à ϵ. Sinon, il existe un *nœud d'inférence*, c'est-à-dire un nœud dont les deux fils sont des nœuds d'échec. (Choisir un nœud non d'échec maximal, c'est-à-dire le plus bas possible.) En figure 1, les nœuds d'inférence sont 4, 6, et 7. Soit N un nœud d'inférence, et $N, -A$ et $N, +A$ ses deux fils. On rappelle que $C_{N,-A}\theta_{N,-A}$ est fausse en $N, -A$ mais pas en N, donc $C_{N,-A}\theta_{N,-A}$ est de la forme $C_1 \vee +A$. (De plus, C_1 est fausse en N.) En particulier, $C_{N,-A}$ peut s'écrire $C \vee +B_1 \vee \ldots \vee +B_m$, où $m \geq 1$ et $B_1\theta_{N,-A} = \ldots = B_m\theta_{N,-A} = A$. De même $C_{N,+A}\theta_{N,+A}$ peut s'écrire $C'_1 \vee -A$ où C'_1 est fausse en N, et $C_{N,+A}$ peut s'écrire $C' \vee -B'_1 \vee \ldots \vee -B'_n$, où $n \geq 1$ et $B'_1\theta_{N,+A} = \ldots = B'_m\theta_{N,+A} = A$. Posons σ_0 l'union

(disjointe si l'on suppose que $C_{N,-A}$ et $C_{N,+A}$ ont des ensembles de variables libres disjoints) de $\theta_{N,-A}$ et de $\theta_{N,+A}$. Par construction σ_0 unifie tous les B_i et tous les B'_j, donc est une instance $\sigma\theta$ de $\sigma = \text{mgu}\ (B_1 \doteq B_2, \ldots, B_{m-1} \doteq B_m, B'_1 \doteq B'_2, \ldots, B'_{n-1} \doteq B'_n, B_m \doteq B'_n)$. On peut donc appliquer la règle de résolution générale (exercice 1) et déduire un résolvant général $C\sigma \vee C'\sigma$. Il est à noter que cette dernière clause a comme instance close $(C\sigma \vee C'\sigma)\theta = C_1 \vee C'_1$, qui est fausse au nœud N. En appliquant judicieusement la règle de résolution générale à deux clauses bien choisies, on a donc créé un nœud d'échec au moins aussi haut que le nœud N. (Par l'exercice 1, on peut obtenir ce résolvant général sans utiliser la factorisation négative, donc par résolution.) Ceci diminue strictement la taille de l'arbre sémantique clos, et l'on peut appliquer l'hypothèse de récurrence : la clause vide est déductible par résolution à partir de S.

Théorème 8 (Complétude) *La résolution ordonnée avec sélection est complète : pour tout ordre $\succ$ stable, pour toute fonction de sélection* sel *, l'ensemble de clauses S est insatisfiable si et seulement si l'on peut dériver la clause vide $\square$ à partir de S par la seule règle de résolution ordonnée avec sélection.*

Démonstration. La direction "si" est évidente. Réciproquement, fixons $A^0_1, \ldots, A^0_n$ l'ensemble fini des atomes clos figurant dans l'ensemble fini S_0 d'instances closes de S du théorème 7.3. On suppose les A^0_i dans un ordre tel que si $A^0_i \prec A^0_j$, alors $i < j$. (En d'autres termes, on étend $\succ$ à un ordre total sur les A^0_i.) Appelons un arbre sémantique clos *adapté* si ses nœuds sont de la forme $\pm_1 A^0_1, \ldots, \pm A^0_k$ avec $k \leq n$. Par construction, il existe un arbre sémantique clos adapté, tel qu'à tout nœud d'échec N est associé une clause C_N de S et une substitution θ_N telle que $C_N\theta_N$ est une clause close qui est fausse en N.

Appelons *arbre décoré* pour un ensemble de clauses S' tout uplet $(T, C_\bullet, \theta_\bullet)$, où T est un arbre sémantique clos adapté, $C_\bullet$ associe à chaque nœud d'échec (feuille de T) N une clause C_N de S', et $\theta_\bullet$ associe à chaque nœud d'échec N une substitution θ_N telle que $C_N\theta_N$ est close et fausse à N. Il existe donc un arbre décoré pour S.

Étant donné un arbre décoré $(T, C_\bullet, \theta_\bullet)$ pour S', soit la racine ϵ est un nœud d'échec, et alors C_ϵ est nécessairement la clause vide, soit l'on trouve un nœud (qui n'est pas d'échec) dans T en calculant comme suit. Le plan de la démonstration consiste à montrer que ce nœud est caractéristique de clauses sur lesquelles on peut appliquer la règle de résolution ordonnée avec sélection. L'appliquer engendre une nouvelle clause, qui aura un arbre décoré plus petit, au sens où une certaine mesure bien fondée de cet arbre décroîtra ; ce qui montrera la terminaison.

Choix des clauses sur lesquelles résoudre. Soit donc S' un ensemble de clauses et un arbre décoré $(T, C_\bullet, \theta_\bullet)$ pour S'. Soit, pour chaque nœud d'échec N, $C_N\theta_N$ une clause close qui est fausse en N. Soit $\pm_N H_N$ le littéral $\pm A^0_i$ de $C_N\theta_N$ avec i maximal—autrement dit, le littéral le plus grand, ou le plus bas sur la branche menant à N. Si $\mathcal{N}$ est un ensemble fini d'atomes $B_1, \ldots, B_k$, notons $-\mathcal{N}$ la disjonction $-B_1 \vee \ldots \vee -B_k$. Si $\mathcal{P}$ est un ensemble fini d'atomes $B_1, \ldots, B_k$, notons $+\mathcal{P}$ la disjonction $+B_1 \vee \ldots \vee +B_k$.

On peut écrire $C_N\theta_N$ de façon unique sous la forme $\pm_N H_N \vee +\mathcal{P}_N \vee -\mathcal{N}_N \vee \mathcal{S}_N$, où $\mathcal{P}_N$ est l'ensemble des atomes de $C_N\theta_N$ (hors $\pm_N H_N$) qui apparaissent sous le signe $+$, $\mathcal{S}_N = \{L\theta_N | L \in \text{sel}\ (C_N)\}$, et $\mathcal{N}_N$ est l'ensemble des atomes de $C_N\theta_N$ (hors $\pm_N H_N$) qui apparaissent sous le signe $-$, resp. non sélectionnées et sélectionnés.

Disons que C_N, et par extension $C_N\theta_N$, est *génératrice* si et seulement si H_N apparaît positivement, c'est-à-dire $\pm_N = +$, et si aucun littéral n'y est sélectionné ($\mathcal{S}_N = \emptyset$). Si C_N est génératrice, $C_N\theta_N$ est donc de la forme $+H_N \vee +\mathcal{P}_N \vee -\mathcal{N}_N$.

On construit une interprétation propositionnelle I par étapes, comme suit. Soit $I_0 = \emptyset$. Supposons I_k déjà construite, et considérons toutes les clauses génératrices C_N telles que $H_N = A^0_{k+1}$, le $k+1$-ième atome. S'il en existe une telle que $\mathcal{N}_N \subseteq I_k$ (tous les atomes niés sont vrais) et $\mathcal{P}_N \cap I_k = \emptyset$ (tous les atomes positifs sont faux), posons $I_{k+1} = I_k \cup \{A^0_{k+1}\}$. Sinon, poser $I_{k+1} = I_k$. La suite s'arrête à l'indice n, lorsqu'on a énuméré tous les atomes, et l'on pose $I = I_n$. L'interprétation I a les propriétés suivantes :

(I.1) Pour toute clause génératrice C_N telle que I rend vrais tous les atomes de $\mathcal{N}_N$ et faux tous les atomes de $\mathcal{P}_N$, I rend H_N vrai.

(I.2) Si H est un atome vrai dans I, alors il existe une clause génératrice C_N telle que $H_N = H$. De plus, tous les atomes de $\mathcal{N}_N$ sont vrais dans I, et tous les atomes de $\mathcal{P}_N$ sont faux dans I.

La vérification de ces deux propriétés est laissée en exercice. Elle dépend du fait qu'un littéral devenu vrai à l'étape k ne pourra jamais redevenir faux ; et, plus subtilement, du fait qu'un littéral de $\mathcal{P}_N$ qui était faux à l'étape k, au moment où $H_N = A^0_{k+1}$, restera faux à toutes les étapes suivantes (parce que H_N est plus bas que tout atome de $\mathcal{P}_N$).

I étant une interprétation de Herbrand, est aussi une branche de l'arbre sémantique, et définit donc une unique branche de l'arbre décoré $(T, C_\bullet, \theta_\bullet)$. Au bout de cette branche dans T, on trouve donc un unique nœud d'échec N_I. L'idée est que C_{N_I} va être la prémisse principale dans la règle de résolution ordonnée avec sélection.

Prémisse principale. Montrons donc en premier que C_{N_I} peut être présentée de sorte à obéir aux restrictions imposées aux prémisses principales, c'est-à-dire à la condition *(iv)*.

Si $\mathrm{sel}\ (C_{N_I})$ est non vide, alors c'est évident : on pose $\{-A'_1, \ldots, -A'_\ell\} = \mathrm{sel}\ (C_{N_I})$, et $\ell \geq 1$ par hypothèse. Suppons donc qu'aucun littéral ne soit sélectionné dans C_{N_I}. Considérons le littéral $\pm_{N_I} H_{N_I}$. Si le signe $\pm_{N_I}$ était $+$, C_{N_I} serait donc génératrice. Mais comme C_{N_I} est fausse dans I, tous les atomes de $\mathcal{N}_{N_I}$ sont dans I et aucun des atomes de $\mathcal{P}_{N_I}$ n'est dans I. Par la propriété (I.1), ceci impliquerait que H_I soit aussi dans I, ce qui rendrait C_{N_I} vraie dans I, contradiction. Donc le signe $\pm_{N_I}$ est $-$. Soit $-A'_1$ n'importe quel littéral négatif de C_{N_I} tel que $A'_1\theta_{N_I} = H_{N_I}$. Alors C_{N_I} s'écrit $C' \vee -A'_1$, où aucun littéral n'est sélectionné, et où $-A'_1$ est maximal. Donc la propriété *(iv)* est vérifiée dans tous les cas.

Prémisses auxiliaires. Considérons les A'_i comme définis ci-dessus. Comme $C_{N_I}\theta_{N_I}$ est fausse dans I, tous les $A'_i\theta_{N_I}$ sont vrais dans I. Par la propriété (I.2) (première partie), il existe une clause génératrice C_{N_i} telle que $H_{N_i} = A'_i\theta_{N_I}$. Elle est nécessairement telle que $C_{N_i}\theta_{N_i}$ contient le littéral $+A'_i\theta_{N_I}$.

Soient donc $+A_{i1}$, …, $+A_{in_i}$, $n_i \geq 1$, tous les littéraux L de C_{N_i} tels que $L\theta_{N_i} = +A'_i\theta_{N_I}$, et soit C_i la disjonction de tous les autres littéraux de C_{N_i}. (Notez bien qu'il se peut qu'il y ait *plusieurs* tels littéraux L ; c'est ce qui justifie le fait que n_i n'est en général pas égal à 1, d'où le besoin de la factorisation.) On a ainsi trouvé la prémisse auxiliaire $C_{N_i} = C_i \vee +A_{i1} \vee \ldots \vee +A_{in_i}$. Comme $n_i \geq 1$, la condition *(i)* est vérifiée.

Comme C_{N_i} est une clause génératrice, aucun littéral n'y est sélectionné. De plus, les $+A_{ij}$ y sont maximaux : sinon il existerait un littéral L dans C_{N_i} tel que $L \succ +A_{ij}$, donc $L\theta_{N_i} \succ +A_{ij}\theta_{N_i} = +A'_i\theta_{N_I} = H_{N_i}$ (par stabilité de $\succ$). Mais H_{N_i} est par définition le grand littéral (le plus bas dans l'arbre) de $C_{N_i}\theta_{N_i}$, contradiction. Donc la condition *(iii)* est vérifiée.

Finalement, on a $A_{ij}\theta_{N_i} = A'_i\theta_{N_I}$. Comme on a supposé (sans perte de généralité) que A_{ij} et $A_{i'j'}$ n'avaient aucune variable libre en commun si $i \neq i'$, et que A_{ij} et $A'_{i'}$ n'avaient aucune variable en commun (pour tous i, i', j), la substitution $\theta_{N_1} \cup \theta_{N_2} \cup \ldots \cup \theta_{N_\ell} \cup \theta_{N_I}$ a un sens, et est clairement un unificateur des A_{ij} et des A'_i.

Soit σ leur mgu : la condition *(ii)* est donc vérifiée. De plus, $\theta_{N_1} \cup \theta_{N_2} \cup \ldots \cup \theta_{N_\ell} \cup \theta_{N_I}$ est une instance $\sigma\theta$ de σ, pour une certaine substitution θ. (Nous utiliserons ceci plus bas.)

La clause $C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma$ est donc déductible par résolution ordonnée avec sélection.

Terminaison. Montrons que ce processus termine. Définissons le nouvel arbre décoré $(T', C'_\bullet, \theta'_\bullet)$ à partir de $(T, C_\bullet, \theta_\bullet)$ comme suit. Soit S' l'ensemble de clauses dont $(T, C_\bullet, \theta_\bullet)$ est un arbre décoré, et soit S'' l'ensemble de clauses S' plus le résolvant $C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma$.

On montre d'abord que $(C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma)\theta = C_1\theta_{N_1} \vee \ldots \vee C_\ell\theta_{N_\ell} \vee C'\theta_{N_I}$ est fausse au nœud N_I, ce qui revient à dire que $C'\theta_{N_I}$ est fausse en N_I, et tous les $C_i\theta_{N_i}$, $1 \leq i \leq \ell$, sont faux en N_I. Le fait que $C'\theta_{N_I}$ soit fausse en N_I est évident, car $C'\theta_{N_I}$ est incluse dans $C_{N_I}\theta_{N_I}$, qui décore le nœud d'échec N_I. Ensuite, C_{N_i}, qui est génératrice, est égale à $+H_{N_i} \vee +\mathcal{P}_{N_i} \vee -\mathcal{N}_{N_i}$. Par construction de C_{N_i} (voir plus haut) et la propriété $(I.2)$ (deuxième partie), tous les atomes de $\mathcal{N}_{N_i}$ sont vrais dans I et tous ceux de $\mathcal{P}_{N_i}$ sont faux dans I. Comme par construction, $C_i\theta_{N_i}$ est exactement la sous-clause $+\mathcal{P}_{N_i} \vee -\mathcal{N}_{N_i}$, elle est fausse dans I. Donc $(C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma)\theta$ est faux dans I. Comme cette dernière clause ne contient que des atomes inférieurs ou égaux (plus haut dans l'arbre) que ceux des prémisses, elle est fausse au nœud N_I.

Il est donc sensé de considérer le nœud N' le plus haut au-dessus de N_I qui rend le résolvant $(C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma)\theta$ faux. Alors :

(a) Si N' est strictement plus haut que N_I dans T, alors soit T' l'arbre sémantique clos dont les nœuds d'échec sont N', plus tous les nœuds d'échec de T qui ne sont pas en-dessous de N'. On pose $C'_{N'}$ égale au résolvant $C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma$, et $\theta'_{N'} := \theta$, alors que $C'_{N''} = C_{N''}$ et $\theta'_{N''} = \theta_{N''}$ pour tout $N'' \neq N'$.

(b) Si $N' = N_I$, on pose $T' = T$, C'_{N_I} égale au résolvant $C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma$ (qui remplace donc au nœud d'échec N_I la clause $C_{N_I} = C' \vee -A'_1 \vee \ldots \vee -A'_\ell$), $\theta'_{N_I} := \theta$ et $C'_{N''} = C_{N''}$ et $\theta'_{N''} = \theta_{N''}$ pour tout $N'' \neq N_I$.

Ce dernier cas ne peut se produire que si l'atome le plus grand (le plus bas) de $(C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma)\theta$ est le même que celui de $C_{N_I}\theta_{N_I}$, soit H_{N_I}. Considérons les autres littéraux de $(C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma)\theta = C_1\theta_{N_1} \vee \ldots \vee C_\ell\theta_{N_\ell} \vee C'\theta_{N_I}$. Les littéraux des $C_i\theta_{N_i}$, $1 \leq i \leq \ell$, sont par définition de C_i strictement plus petits (strictement plus haut) que $H_{N_i} = A'_i\theta_{N_I}$, qui est un atome de $C_{N_I}\theta_{N_I}$, donc plus petit ou égal à (plus haut que) H_{N_I}. Les littéraux de $C_i\theta_{N_i}$ sont donc toujours strictement plus hauts que H_{N_I}. La seule raison pour laquelle H_{N_I} pourrait donc apparaître dans $(C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma)\theta = C_1\theta_{N_1} \vee \ldots \vee C_\ell\theta_{N_\ell} \vee C'\theta_{N_I}$, c'est qu'il apparaisse dans $C'\theta_{N_I}$. Ce qui est important, c'est qu'en passant de $C_{N_I}\theta_{N_I}$ à $C_1\theta_{N_1} \vee \ldots \vee C_\ell\theta_{N_\ell} \vee C'\theta_{N_I}$ comme décoration du nœud N_I, on a moralement remplacé des "grands" littéraux H_{N_i} par des clauses $C_i\theta_{N_i}$, qui ne contiennent que des littéraux strictement plus petits.

Formellement, on définit la mesure $\mu_1(C_N, \theta_N)$ du nœud d'échec N d'un arbre décoré $(T, C_\bullet, \theta_\bullet)$ comme étant le multi-ensemble contenant autant de fois l'entier i qu'il y a de littéraux $\pm A'$ de C_N tels que $A'\theta_N = A_i^0$ (le $i$ième atome clos de l'énumération choisie). Dans le cas *(b)*, où $N' = N_I$ (et en posant $N = N_I$), ce multi-ensemble décroît, dans l'ordre multi-ensemble, lors du remplacement de C_{N_I} par $C_1\sigma \vee \ldots \vee C_\ell\sigma \vee C'\sigma$.

On rappelle en passant que cet ordre multi-ensemble est bien fondé, voir l'annexe A.3 pour plus de détails. (On rappelle qu'un ordre est *bien fondé* s'il n'admet aucune chaîne infinie décroissante. Il s'agit d'un anglicisme [well-founded], que nous préférons pour des raisons de clarté au mot français *bon ordre*.)

On définit ensuite la mesure $\mu^-(T, C_\bullet, \theta_\bullet)$ comme étant le multi-ensemble des $\mu_1(C_N, \theta_N)$, N parcourant les nœuds d'échec de T. Dans le cas *(b)*, $\mu_1(C_N, \theta_N)$ décroît strictement, les autres $\mu_1(C_{N''}, \theta_{N''})$ étant inchangés. Donc la mesure μ^- décroît dans l'ordre multi-ensemble dans le cas *(b)*.

D'un autre côté, la taille $|T|$ décroît strictement dans le cas *(a)*, et est inchangée dans le cas *(b)*. Donc la mesure $\mu(T, C_\bullet, \theta_\bullet)$ définie comme le couple $(|T|, \mu^-(T, C_\bullet, \theta_\bullet))$ ordonné lexicographiquement, décroît lors du passage de $(T, C_\bullet, \theta_\bullet)$ à $(T', C'_\bullet, \theta'_\bullet)$. L'ordre lexicographique sur les couples formés d'un entier et d'un multi-ensemble de multi-ensembles d'entiers étant bien fondé, le processus termine, et l'on peut donc dériver la clause vide en un nombre fini d'étapes de résolution ordonnée avec sélection. □

On pourra remarquer l'utilisation de μ^- pour montrer par des moyens purement sémantiques (c'est-à-dire sans utiliser l'exercice 1) que la factorisation négative n'est pas nécessaire.

Remarque 3 *L'utilisation d'une fonction de sélection est parfois encore un peu trop restrictive, et l'on pourrait garder juste une* relation *de sélection ; en termes basiques, on pourrait décider de sélectionner un ensemble de littéraux négatifs différent pour deux occurrences de la même clause. Ceci n'a, en pratique, aucun intérêt : si on élimine les clauses subsumées comme nous le ferons dans la suite, nous ne fabriquerons en particulier jamais deux fois la même clause, et le problème ne se pose donc pas.*

4 Élimination de redondances

En pratique, il est utile, voire fondamental, de combiner la règle de résolution avec différentes règles de *simplification* de l'ensemble de clauses courant. Les deux plus importantes règles de simplification sont :

- L'*élimination de tautologies* : on dit qu'une clause C est une *tautologie* si et seulement si elle est de la forme $C_1 \vee +A \vee -A$ pour un certain atome A. On peut alors éliminer toutes les tautologies de S.
- L'*élimination de clauses subsumées* : on dit qu'une clause C *subsume* une clause C' si et seulement si C' est de la forme $C\sigma \vee C_1$. On notera que C implique $C\sigma$, qui implique $C\sigma \vee C_1$. Donc si C subsume C', alors C implique C'. Intuitivement, on doit pouvoir éliminer toute clause C' impliquée par une autre C. L'intuition est fausse, mais l'on peut très souvent éliminer les clauses subsumées.

– L'*élimination des clauses pures* : on dit qu'une clause C est pure dans un ensemble de clauses S si et seulement si C est de la forme $C_1 \vee \pm A$, et il n'existe aucune clause de la forme $C'_1 \vee \mp A'$ dans S telle que A et A' sont unifiables (on dit que $\pm A$ est *pur*). L'ensemble S est alors satisfiable si et seulement si $S \setminus \{C\}$ est satisfiable.

Le fait que $S \setminus \{C\}$ est satisfiable si et seulement si S est satisfiable est une bonne indication que l'on peut enlever C de S. Mais ce n'est pas une condition suffisante. Un cas intéressant est celui où l'on sépare la règle de résolution en résolution binaire et factorisation positive : tout facteur $C\sigma$ de C est subsumé par C, donc si l'on pouvait ne serait-ce qu'éliminer toute clause subsumée, alors la résolution binaire serait complète sans factorisation ; on a vu que ce n'était pas le cas.

En général, le problème peut s'expliquer comme suit. Supposons que S_1 est un ensemble de clauses insatisfiables, et que la plus courte dérivation de la clause vide par résolution (ordonnée, avec sélection) est de longueur 10. Enlevons de S_1 une clause C redondante (tautologique, subsumée, pure), et posons $S_2 = S_1 \setminus \{C\}$. S_2 est toujours insatisfiable, mais rien n'empêche a priori que la plus courte dérivation de la clause vide soit de longueur 20 (par exemple). Effectuons une étape de résolution, obtenant S_3, qui n'est plus qu'à 19 étapes de la clause vide. Il est plausible qu'éliminer une autre clause redondante mène à S_4, qui est disons à 30 étapes de la clause vide. On voit ainsi que l'on engendre une suite infinie d'ensembles de clauses, tous insatisfiables, mais qui ne se rapprochent pas d'un ensemble de clauses contenant la clause vide.

Nous allons regarder quelles clauses peuvent être éliminées de façon sûre en regardant finement la preuve du théorème 8. L'idée sera de garantir que la mesure $\mu(T, C_\bullet, \theta_\bullet)$ (ou un raffinement) décroisse.

4.1 Élimination de tautologies

Une tautologie $C_1 \vee +A \vee -A$ n'a que des instances closes vraies. Donc aucune clause $C_N\theta_N$ n'est une instance d'une telle clause.

Raffinons la mesure des arbres décorés en posant $\mu'(S, T, C_\bullet, \theta_\bullet) = (\mu(T, C_\bullet, \theta_\bullet, |S|)$ ordonnée lexicographiquement, où $|S|$ est la *taille* de l'ensemble de clauses S, et où l'on contraint $(T, C_\bullet, \theta_\bullet)$ à être un arbre décoré pour S. Comme aucune tautologie de S n'est une clause C_N, l'élimination de tautologies laisse $\mu(T, C_\bullet, \theta_\bullet)$ inchangé, mais fait décroître $|S|$.

On en conclut que l'on peut intercaler entre toutes étapes de résolution ordonnée avec sélection n'importe quel nombre d'étapes d'effacement de tautologies. Notamment, on peut décider d'éliminer systématiquement toutes les tautologies.

4.2 Élimination de clauses subsumées

Supposons que S contient C et une clause subsumée $C\sigma \vee C_1$. (Dans $C\sigma \vee C_1$, on suppose implicitement qu'il n'y aucun littéral commun à $C\sigma$ et C_1.) Supposons que N est un nœud rendant faux une instance close $(C\sigma \vee C_1)\theta$ de $C\sigma \vee C_1$. Comme N rend fausse la clause $C\sigma\theta \vee C_1\theta$, en particulier N rend fausse la clause $C\sigma\theta$, qui est une instance close de C. Donc il existe un nœud d'échec N' au-dessus de N.

Si N' est strictement au-dessus de N, comme en section 4.1, éliminer $C\sigma \vee C_1$ fait décroître $\mu(T, C_\bullet, \theta_\bullet)$. Ce cas ne se présente en fait jamais, puisque N est un nœud d'échec : il n'y a aucun nœud au-dessus de N' où une clause de S serait déjà fausse.

Donc $N' = N$. Soit C_N n'est pas la clause $C\sigma \vee C_1$, et le même argument s'applique, soit C_N est la clause $C\sigma \vee C_1$, et l'on peut remplacer la clause C_N au nœud d'échec N par C, à condition que $\mu_1(C, \sigma\theta)$ soit plus petit ou égal dans l'ordre multi-ensemble que $\mu_1(C\sigma \vee C_1, \theta)$. (Le cas plus petit est clair ; si on a l'égalité, noter que $|S|$ décroît.)

C'est notamment le cas si σ n'unifie aucun littéral dans C (autrement dit, ne calcule pas un facteur de C). En effet, dans ce cas $\mu_1(C\sigma \vee C_1, \theta) = \mu_1(C\sigma, \theta) \uplus \mu_1(C_1, \theta) \geq \mu_1(C\sigma, \theta) = \mu_1(C, \sigma\theta)$. Noter qu'en général $\mu_1(C\sigma, \theta) \leq \mu_1(C, \sigma\theta)$, et on n'a l'égalité que si σ n'unifie aucun littéral dans C.

Nous appellerons une clause subsumée $C\sigma \vee C_1$ telle que σ n'unifie aucune paire de littéraux de C une clause *subsumée linéairement*. La discussion ci-dessus montre que l'on peut éliminer toute clause subsumée linéairement, à tout moment au cours de la recherche de preuve. (Répéter l'argument ci-dessus pour *tous* les nœuds N tels que $C = C_N$; il peut y en avoir plus d'un.) En particulier, on peut supprimer les clauses subsumées linéairement le plus tôt possible.

Par exemple, la clause $+P(x) \vee +P(y)$ subsume linéairement les clauses :

- $+P(a) \vee +P(y)$ (σ instancie x par a)
- $+P(a) \vee +P(y) \vee +Q(z)$
- $+P(x) \vee +P(y) \vee +Q(z)$

et elle subsume, mais pas linéairement :

- $+P(z)$ (σ instancie x et y par z)
- $+P(a)$ (σ instancie x et y par a)

A noter que réciproquement, $+P(z)$ subsume linéairement $+P(x) \vee +P(y)$.

Remarque 4 *On peut en fait fournir des critères d'élimination de clauses subsumées plus libéraux, à condition de changer de mesure $\mu(T, C_\bullet, \theta_\bullet)$. Un cas simple est celui de la résolution ordonnée, c'est-à-dire où* $\mathrm{sel}\ (C) = \emptyset$ *pour toute clause C. La complétude de ce cas peut se montrer en prenant $\mu_1(C, \theta) = |T|$; alors on peut éliminer toute clause subsumée, même si elle ne l'est pas linéairement, à tout moment.*

Dans le cas de la résolution ordonnée avec sélection, on peut à la place raffiner $\mu_1(C, \theta)$. Étendons l'ordre $\succ$ sur les atomes à un ordre sur les littéraux par le produit lexicographique de l'ordre $\succ$ sur l'atome sous-jacent avec l'ordre $- > +$ sur le signe. Ordonnons les clauses C par l'ordre multi-ensemble $\gg$ sur les littéraux, ordonnés par l'ordre précédent. La preuve du théorème 8 tient toujours en choisissant $\mu_1(C, \theta)$ égale à C, ordonnée par $\gg$. D'autre part, on a toujours $C\sigma \vee C_1 \gg C$, sauf si $C\sigma \vee C_1$ subsume elle aussi C. (Exercice.) On en conclut que l'on peut éliminer toute clause C strictement subsumée *par une autre clause C', c'est-à-dire toute clause C subsumée par C' telle que C' n'est pas subsumée par C. D'autre part, on peut aussi éliminer tout résolvant (ordonné, avec sélection) qui est subsumé (pas nécessairement strictement) par une autre clause de S. Ceci est prouvé notamment en [BG01] par une technique apparemment très différente de celle présentée ici, mais (probablement) en réalité isomorphe.*

Remarque 5 *Tester si une clause C est subsumée par une autre, C', est un problème NP-complet. Il est clairement dans NP : il suffit de deviner quels littéraux de C proviennent d'instanciation de littéraux de C', puis de résoudre en temps polynomial le problème de pattern-matching qui en résulte. Il est d'autre part NP-complet, mais différentes techniques algorithmiques permettent de détecter toutes les clauses subsumées par une clause donnée, ou réciproquement si une clause donnée est subsumée par une des clauses d'un ensemble donnée, extrêmement efficacement [RSV01]. Dans le cas de formats de clauses comme ceux que l'on considérera en section 7.6, il sera facile de voir que le test de subsomption est faisable en temps polynomial.*

4.3 Élimination de clauses pures

L'élimination de clauses pures est une stratégie d'élimination de clauses redondantes qui est parfois très efficace : supprimer une clause redondante (pure notamment) peut rendre pures d'autres clauses, et ainsi de suite en chaîne.

Un argument facile montrant qu'on peut éliminer les clauses pures est le suivant : tout résolvant d'une clause pure $C \vee \pm A$ (où $\pm A$ est pur, c'est-à-dire ne s'unifie avec aucun $\mp A'$ dans aucune clause) avec une clause C' est de la forme $C'' \vee \pm A\sigma$, où $\pm A\sigma$ ne s'unifie toujours avec aucun $\mp A'$ dans aucune clause. Donc tout résolvant d'une clause pure est pur. Comme la clause vide n'est pas pure, aucune des étapes de résolution construites dans la preuve du théorème 8 ne peut utiliser de clause pure. On peut donc toutes les éliminer.

En pratique, un démonstrateur automatique fondé sur la résolution ordonnée avec sélection maintient des tables des littéraux qui s'unifient en position maximale ou sélectionnée dans les clauses. Il est alors facile de détecter quelles clauses sont de la forme $C \vee +A$ (A maximal, $\mathrm{sel}\ (C \vee +A) = \emptyset$) telles que $+A$ est pur, et quelles clauses sont de la forme $C \vee -A$ (A sélectionné, ou bien $\mathrm{sel}\ (C \vee -A) = \emptyset$ et A maximal) avec $-A$ pur. Il est aussi plus facile de donner un argument direct de complétude d'élimination de ces clauses pures particulières par la technique de preuve du théorème 8, les C_N ne pouvant être pures dans ce sens.

4.4 Divers

Bachmair et Ganzinger [BG01] mentionnent d'autres cas de redondance. En particulier, la règle de *résolution-subsomption* permet, à partir des deux prémisses $C \vee \pm A$ et $C' \vee C\sigma \vee \mp A\sigma$, d'inférer $C' \vee C\sigma$. Il s'agit d'une étape de résolution (non ordonnée, sans sélection), qui est donc toujours correcte et préserve la complétude. L'avantage de cette règle est que la conclusion $C' \vee C\sigma$ subsume linéairement la prémisse $C' \vee C\sigma \vee \mp A\sigma$, que l'on peut donc effacer immédiatement.

Joyner [JJ76] utilise une autre forme de mélange entre résolution et subsomption. Posons $C \rightsquigarrow C\sigma$, où $C\sigma$ est un facteur général de C, si et seulement si C est de la forme $C\sigma \vee C'$. Noter que dans ce cas, C et son facteur $C\sigma$ se subsument mutuellement. De plus, $C\sigma$ subsume C linéairement, donc on peut effacer C dès que $C \rightsquigarrow C\sigma$. Cette règle s'appelle la *condensation*.

Pour voir quand cette règle est utile, considérons la clause $C = +P(x) \vee +P(y) \vee +P(a)$. On a $C \rightsquigarrow +P(x) \vee +P(a)$ (par l'unificateur $y := x$ ou $y := a$) $\rightsquigarrow +P(a)$. En général, $\rightsquigarrow$ définit un système de réécriture qui termine (évident) et est confluent. Joyner appelle l'unique forme

normale d'une clause C pour $\rightsquigarrow$ sa *condensation*. Remplacer systématiquement une clause par sa condensation est correct et préserve la complétude.

5 Splitting

Considérons une clause de la forme $C \vee C'$, où C et C' ne partagent aucune variable libre, et un ensemble de clauses de la forme $S \cup \{C \vee C'\}$. Ce dernier est insatisfiable si et seulement si $S \cup \{C\}$ et $S \cup \{C'\}$ sont insatisfiables.

Par exemple, $S \cup \{+P(x) \vee +Q(y)\}$ est satisfiable par tout modèle qui satisfait S d'une part, et tel que P est vrai de tout x, ou bien Q est vrai de tout y. Donc S est satisfiable si et seulement s'il existe un modèle de $S \cup \{+P(x)\}$, ou un modèle de $S \cup \{+Q(y)\}$. On remarquera qu'il est important que C (ici, $+P(x)$) et C' (ici, $+Q(y)$) ne partagent aucune variable.

On va examiner plusieurs façons d'exploiter cette remarque.

5.1 Splitting et tableaux

La première approche consiste à découper l'ensemble de clauses $S \cup \{C \vee C'\}$ en deux ensembles de clauses, $S \cup \{C\}$ et $S \cup \{C'\}$. On a alors besoin de manipuler plusieurs ensembles de clauses. Appelons *tableau* $\mathcal{T}$ tout multi-ensemble fini non vide d'ensembles de clauses. On notera $S_1|S_2|\ldots|S_n$ le tableau formé des ensembles de clauses S_1, S_2, …, S_n. En général, on notera $\mathcal{T}|\mathcal{T}'$ l'union multi-ensemble des tableaux $\mathcal{T}$ et $\mathcal{T}'$. Chaque ensemble de clauses S_i est appelé traditionnellement une *branche* du tableau $\mathcal{T}$. On dit qu'une branche est *close* si et seulement si elle contient la clause vide $\Box$. On dit qu'un tableau $\mathcal{T}$ est *clos* si et seulement si toutes ses branches sont closes.

Pour modéliser le splitting, soit $\triangleright$ une relation binaire entre ensembles de clauses, appelée relation de déduction. On pense ici typiquement à la relation définie par $S \triangleright S'$ si et seulement si S' est obtenu par l'ajout à S d'un résolvant ordonnée avec sélection entre clauses de S, ou S' est obtenu par élimination d'une tautologie ou d'une clauses subsumée linéairement par une autre clause de S. Cependant, notre argumentation restera valable pour de nombreuses autres relations de déduction.

On définit la relation binaire $\multimap$ entre tableaux par les règles :

$$\frac{S \triangleright S'}{S|\mathcal{T} \multimap S'|\mathcal{T}}\,(\triangleright) \qquad \frac{C, C' \text{ n'ont aucune variable libre en commun}}{S \cup \{C \vee C'\}|\mathcal{T} \multimap S \cup \{C\}|S \cup \{C'\}|\mathcal{T}}\,(Split)$$

Disons qu'une interprétation I *satisfait* un tableau $\mathcal{T}$ si et seulement si I satisfait au moins une branche de $\mathcal{T}$. En particulier, $\mathcal{T}$ est insatisfiable si et seulement si toutes ses branches sont insatisfiables.

Lemme 9 (Correction) *Supposons que $\triangleright$ est correcte, au sens où si $S \triangleright S'$ et S est satisfiable, alors S' est satisfiable.*

Alors $\multimap$ est correcte, au sens où si $\mathcal{T} \multimap^* \mathcal{T}'$ et $\mathcal{T}$ est satisfiable, alors $\mathcal{T}'$ est satisfiable. En particulier, si $\mathcal{T} \multimap^* \mathcal{T}'$ et $\mathcal{T}'$ est un tableau clos, alors $\mathcal{T}$ est insatisfiable.

Démonstration.

- (Règle ($\triangleright$)) si $S \triangleright S'$ et $S'|\mathcal{T}$ est satisfiable, soit I une interprétation satisfaisant $S|\mathcal{T}$. Si I satisfait S, par hypothèse il existe une interprétation I' satisfaisant S', donc I' satisfait $S'|\mathcal{T}$.
- (Règle $(Split)$) si C et C' sont deux clauses n'ayant aucune variable en commun, soit I est une interprétation qui satisfait $S \cup \{C \vee C'\}|\mathcal{T}$. Si I satisfait $S \cup \{C \vee C'\}$, comme on l'a vu plus haut, I satisfait $S \cup \{C\}$ ou $S \cup \{C'\}$, donc $S \cup \{C\}|S \cup \{C'\}|\mathcal{T}$. La même conclusion est clairement obtenue si I satisfait $\mathcal{T}$. □

La complétude est, comme d'habitude, un problème plus délicat. Notons d'abord que la relation $\rightarrow\!\triangleright$ est complète, de façon triviale, dès que $\triangleright$ est complète : il suffit de ne jamais appliquer $(Split)$. On souhaite cependant utiliser $(Split)$ à volonté, par exemple dès qu'elle est applicable.

Pour ceci, et bien qu'on puisse le démontrer dans un cadre général, supposons que $\triangleright$ est l'union des relations $\triangleright^r$ et $\triangleright^{eff}$, où $S \triangleright^r S'$ si et seulement si S' est S union un résolvant ordonné avec sélection de clauses de S, et $S \triangleright^{eff} S'$ si et seulement si S' est S moins une tautologie, ou moins une clause de S subsumée linéairement par une autre clause de S.

Définissons une *stratégie d'effacement et de splitting* (dans la suite, on dira juste *stratégie*) comme étant une fonction f qui à chaque tableau $\mathcal{T}$ associe soit le symbole spécial $*$, soit un tableau $\mathcal{T}'$ tel que soit $\mathcal{T} \rightarrow\!\triangleright \mathcal{T}'$ est déductible par la règle $(Split)$, soit $\mathcal{T} \triangleright^{eff} \mathcal{T}'$. Une *$f$-dérivation* est une dérivation

$$\mathcal{T}_0 \rightarrow\!\triangleright \mathcal{T}_1 \rightarrow\!\triangleright \ldots \rightarrow\!\triangleright \mathcal{T}_k$$

où pour tout j, $1 \leq j \leq k$, soit $f(\mathcal{T}_{i-1}) = \mathcal{T}_i$, soit $f(\mathcal{T}_{i-1}) = *$. On dit que $\triangleright^r$ est *f-complète* si et seulement si, pour tout tableau $\mathcal{T}$ insatisfiable, il existe une dérivation finie comme ci-dessus telle que $\mathcal{T} = \mathcal{T}_0$ et $\mathcal{T}_k$ est clos. On a alors :

Théorème 10 (Complétude) *La résolution ordonnée avec sélection est f-complète pour toute stratégie f d'effacement et de splitting.*

Démonstration. On mesure chaque branche S d'un tableau par le couple $\mu(T, C_\bullet, \theta_\bullet), |S|$, comme en Section 4, et un tableau complet $\mathcal{T}$ par le multi-ensemble de ces quantités lorsque S parcourt les branches de $\mathcal{T}$. Il est clair que toute application de $(Split)$ ou de $\triangleright^{eff}$ fait décroître cette mesure, alors que la construction de la preuve du théorème 8 fournit une étape de résolution ordonnée avec splitting $\triangleright^r$ qui fait décroître cette mesure à partir de toute branche non close de tout tableau $\mathcal{T}$ si $f(\mathcal{T}) = *$. □

En d'autres termes, la résolution ordonnée avec sélection, augmentée de l'effacement à volonté de tautologies et de clauses linéairement subsumées, et de l'utilisation à volonté de la règle de splitting $(Split)$, est complète.

Dans la suite, on utilisera toujours une forme particulière de splitting. Pour tout ensemble de clauses S, toute clause C de S peut être découpée en $C_1 \vee \ldots \vee C_k$, où chaque C_i est une sous-clause minimale non vide telle que C_i et C_j ne partagent aucune variable libre, pour aucun $i \neq j$. On dira que les C_i sont les *blocs* de C, et que $C_1, \ldots, C_k$ est une *décomposition en blocs*

de C. On utilisera toujours la forme suivante de la règle de splitting :

$$\frac{C_1, \ldots, C_k \text{ décomposition en blocs de } C, k \geq 2}{S \cup \{C\} | \mathcal{T} \ \twoheadrightarrow \ S \cup \{C_1\} | \ldots | S \cup \{C_k\} | \mathcal{T}}$$

Cette règle est simulable par $k-1$ applications de la règle $(Split)$. Réciproquement, il est clair qu'utiliser cette dernière règle au lieu de $(Split)$ reste complet, pour toute stratégie d'effacement et de splitting (exercice).

5.2 Splitting et non-déterminisme

La modélisation des tableaux comme des multi-ensembles de branches est pratique pour démontrer la correction et la complétude, mais ce n'est pas la bonne façon de voir ce qui se passe réellement. Illustrons ceci sur un exemple. Considérons l'ensemble de clauses

$$(a) \quad P(f(x,y)) \Leftarrow P_1(x), P_2(y) \qquad (b) \quad Q_1(x) \Leftarrow P(f(x,y)), Q_2(y)$$
$$(c) \quad P_2(g(g(y))) \Leftarrow P_2(y) \qquad (d) \quad P_2(g(a))$$
$$(e) \quad Q_2(g(g(g(y)))) \Leftarrow Q_2(y) \qquad (f) \quad Q_2(g(g(a)))$$
$$(g) \quad \bot \Leftarrow Q_1(g(a)) \qquad (h) \quad P_1(g(x))$$

Lorsqu'on résout entre les deux premières clauses (a) et (b) sur $P(f(x,y))$, on obtient la clause $Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y)$, qui se splitte en deux. On poursuit ensuite le raisonnement sur chaque branche *indépendamment* :

$$(i) \quad Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y) \quad \text{par } (a), (b)$$

$$\begin{array}{c|c} (j) \quad Q_1(x) \Leftarrow P_1(x) & (k) \quad \bot \Leftarrow P_2(y), Q_2(y) \\ (l) \quad \bot \Leftarrow P_1(g(a)) \quad \text{par } (j), (g) & (m) \quad \bot \Leftarrow P_2(y), Q_2(g(g(y)) \quad \text{par } (k), (c) \\ \square \quad \text{par } (l), (h) & (n) \quad \bot \Leftarrow P_2(g(y)), Q_2(y) \quad \text{par } (m), (e) \\ & (o) \quad \bot \Leftarrow P_2(y), Q_2(g(y)) \quad \text{par } (n), (c) \\ & (p) \quad \bot \Leftarrow Q_2(g(g(a)) \quad \text{par } (o), (d) \\ & (q) \quad \square \quad \text{par } (p), (f) \end{array}$$

La véritable forme d'une dérivation en présence de splitting n'est donc pas une suite d'opérations, en pratique, mais un *arbre*. Sur une machine de Turing non déterministe, on réalisera le splitting en remplaçant non-déterministiquement toute clause C ayant une décomposition en blocs $C_1, \ldots, C_k$ par l'un des C_j, $1 \leq j \leq k$. Appelons cette règle le *splitting non déterministe*. Disons aussi qu'un ensemble de clauses S est *saturé* pour un certain raffinement de la résolution et splitting non déterministe si toute clause obtenue par splitting non déterministe d'un résolvant de deux clauses de S (pour le raffinement choisi) est subsumée linéairement par une clause de S. Si un ensemble de clauses S mène par un certain raffinement de la résolution et splitting non déterministe à un ensemble de clauses saturé S_∞, alors on en déduit que S est satisfiable.

Si S est dans une classe $\mathcal{C}$ de clauses telle que le nombre maximum d'étapes de résolution et de splitting non déterministe est polynomial en la taille de S, ceci fournira un algorithme de décision de la satisfiabilité pour la classe $\mathcal{C}$ qui est donc dans NP. Si le nombre maximum d'étapes est exponentiel (c'est-à-dire borné par l'exponentielle d'un polynôme en la taille de S), alors la satisfiabilité sera dans NEXPTIME. Nous utiliserons cette remarque plusieurs fois dans la suite.

5.3 Splitting sans splitting

On peut simuler le splitting en utilisant l'équivalence entre $C \vee C'$ et $\exists q \cdot (q \Rightarrow C) \wedge (\neg q \Rightarrow C')$, lorsque C et C' ne partagent aucune variable libre. Ceci permet de remplacer la disjonction $C \vee C'$ par la *conjonction* des deux clauses $-q \vee C$ et $+q \vee C'$, où q est un symbole de prédicat frais d'arité 0.

Ceci permet d'éviter d'avoir à parler de tableaux, et de ne raisonner de nouveau que sur des ensembles de clauses. Dans la suite, pour éviter d'avoir à parler de symboles d'arité 0, on écrira $q(*)$ à la place de q, où $*$ est une constante fixée.

Pour formaliser ceci, nous avons besoin de préciser ce qu'un symbole de prédicat *frais* signifie. Fixons donc l'ensemble $\mathcal{P}$ des symboles de prédicats pouvant apparaître dans un ensemble de clauses S fourni en entrée de notre procédure de résolution. Soit $\mathcal{Q}$ un ensemble de symboles de prédicats, disjoint de $\mathcal{P}$. Supposons de plus que $\mathcal{Q}$ est en bijection avec l'ensemble des clauses formées à l'aide de prédicats dans $\mathcal{P}$ seulement, modulo renommage. Autrement dit, il existe une application qui à chaque clause C formée à l'aide de prédicats dans $\mathcal{P}$ uniquement associe un symbole $\ulcorner C \urcorner$ de $\mathcal{Q}$, telle que $\ulcorner C \urcorner = \ulcorner C' \urcorner$ si et seulement s'il existe un renommage ϱ tel que $C = C'\varrho$. Cette construction existe toujours : il suffit de choisir pour $\mathcal{Q}$ exactement le quotient de l'ensemble des clauses formées à l'aide de prédicats dans $\mathcal{P}$ par la relation d'équivalence reliant toute clause C à tout renommage $C\varrho$.

La règle de *splitting sans splitting* est alors :

$$\frac{C \vee C'}{C \vee -\ulcorner C' \urcorner(*) \quad + \ulcorner C' \urcorner(*) \vee C'}$$

par laquelle on entend que C et C' sont deux sous-clauses non vides, ne partageant aucune variable, telles que C' est formée de symboles de prédicats dans $\mathcal{P}$ uniquement, et que C contient au moins un atome $P(t)$ avec $P \in \mathcal{P}$; l'effet de la règle est de remplacer $C \vee C'$ par l'ensemble des deux clauses $C \vee -\ulcorner C' \urcorner(*)$ et $+\ulcorner C' \urcorner(*) \vee C'$. (On pourrait lever la restriction que C' est formée de symboles de prédicats dans $\mathcal{P}$ uniquement, mais nous n'en aurons pas besoin. En revanche, nous lèverons cette restriction dans le cas du splitting non clos en section 5.4.)

Soit I une interprétation de Tarski, de domaine D, des symboles de prédicats dans $\mathcal{P}$. Son *extension standard* $I^{\mathcal{Q}}$ est telle que $I^{\mathcal{Q}}_P = I_P$ pour tout $P \in \mathcal{P}$, et $I^{\mathcal{Q}}_{\ulcorner C \urcorner}$ vaut D si $I \not\models C$, et $\emptyset$ si $I \models C$. On dira qu'une interprétation I est *standard* si et seulement si, pour toute clause C formée à l'aide de symboles de prédicats de $\mathcal{P}$ uniquement, $I_{\ulcorner C \urcorner}$ est vide si et seulement si $I \models C$. Toute extension standard est clairement standard.

Lemme 11 (Correction) *La résolution ordonnée avec sélection, l'élimination de tautologies et de clauses subsumées, et la règle de splitting sans splitting sont correctes au sens où, s'il existe une interprétation standard I telle que $I \models S$ et si S' est déduite de S par l'une de ces règles, alors $I \models S'$.*

En particulier, si on peut déduire la clause vide de S par ces règles, et si S ne contient que des clauses formées à l'aide de symboles de prédicats de $\mathcal{P}$, alors S est insatisfiable.

Démonstration. Immédiat pour toutes les règles sauf éventuellement la règle de splitting sans splitting. Si $I \models C \vee C'$, alors on a deux cas. Si $I \models C'$, comme I est standard, $I \not\models +\ulcorner C' \urcorner(*)$,

donc $I \models C \vee -\ulcorner C' \urcorner(*)$; d'autre part $I \models +\ulcorner C' \urcorner(*) \vee C'$, donc I est un modèle de toutes les clauses en conclusion de la règle. Si $I \not\models C'$, comme I est standard, $I \models +\ulcorner C' \urcorner(*)$, donc $I \models +\ulcorner C' \urcorner(*) \vee C'$; d'autre part comme $I \not\models C'$ mais $I \models C \vee C'$ et C et C' ne partagent aucune variable libre, nécessairement $I \models C$, donc $I \models C \vee -\ulcorner C' \urcorner(*)$. Donc I est encore un modèle de toutes les clauses en conclusion de la règle.

Finalement, si on peut déduire la clause vide à partir de S, S ne peut donc avoir aucun modèle standard. Mais toute extension standard d'un modèle de S est un modèle standard de S. Donc S n'a aucun modèle : S est insatisfiable. □

L'ajout de $\mathcal{Q}$ modifie la base de Herbrand, c'est-à-dire l'ensemble des atomes clos à considérer : on doit maintenant considérer non seulement tous les atomes clos $P(t)$, $P \in \mathcal{P}$, mais aussi les atomes clos de la forme $Q(t)$, où $Q \in \mathcal{Q}$. Pour tout ordre stable $\succ$, on étend $\succ$ de sorte que $P(t) \succ Q(*)$ pour tout terme clos t, et pour tous $P \in \mathcal{P}$, $Q \in \mathcal{Q}$. Un tel ordre sera appelé *admissible*. On peut définir comme en section 5.1 une notion de *stratégie* d'effacement et de splitting sans splitting (exercice). On a alors :

Théorème 12 (Complétude) *Soit $\succ$ un ordre stable admissible. La résolution ordonnée pour $\succ$ avec sélection, élimination de tautologies, de clauses subsumées linéairement et splitting sans splitting est complète : pour tout ensemble insatisfiable S de clauses formées à partir de symboles de prédicats dans $\mathcal{P}$, pour toute stratégie f, il existe une f-dérivation de la clause vide à partir de S.*

Démonstration. Il suffit de remarquer que, si N est un nœud d'échec, et C_N est de la forme $C \vee C'$, où ni C ni C' n'est vide, alors soit $\ulcorner C' \urcorner(*)$ est fausse en N et alors $+\ulcorner C' \urcorner(*) \vee C'\theta_N$ est fausse en N, soit $\ulcorner C' \urcorner(*)$ est vraie en N et alors $C\theta_N \vee -\ulcorner C' \urcorner(*)$ est fausse en N.

Dans le deuxième cas, $\mu_1(C \vee -\ulcorner C' \urcorner(*), \theta_N)$ est plus petite que $\mu_1(C \vee C', \theta_N)$, car C' est non vide, construite uniquement à partir de symboles de $\mathcal{P}$, donc pour tout atome A de C', $A\theta_N \succ \ulcorner C' \urcorner(*)$, puisque $\succ$ est admissible.

Dans le premier cas, de même, $\mu_1(+\ulcorner C' \urcorner(*) \vee C', \theta_N)$ est plus petite que $\mu_1(C \vee C', \theta_N)$, cette fois-ci parce que C contient au moins un atome de la forme $P(t)$ avec $P \in \mathcal{P}$, donc tel que $P(t)\theta_N \succ \ulcorner C' \urcorner$. □

Note : L'argument du théorème 12 contient une erreur, qui m'a été signalée en 2005 par K. Neeraj Verma. Je préfère ne pas la corriger pour éveiller l'attention du lecteur sur un point subtil. Il se trouve que $\ulcorner C' \urcorner(*)$ est fausse en N, ou $\ulcorner C' \urcorner(*)$ est vraie en N, *ou encore* $\ulcorner C' \urcorner(*)$ n'est pas interprétée du tout en N. Il suffit que $\ulcorner C' \urcorner(*)$ tombe en-dehors de l'ensemble des atomes $A^0_1, \ldots, A^0_n$ garanti par le théorème de compacité. On peut corriger le théorème de différentes façons :

1. Dans le cas où l'on saurait par avance que l'on ne peut fabriquer qu'un nombre fini d'atomes de la forme $\ulcorner C' \urcorner(*)$ au cours de la résolution — ce sera le cas à la section 6.5 — il suffit de rajouter à $A^0_1, \ldots, A^0_n$ l'ensemble fini de tous ces atomes, et la démonstration fonctionne alors sans aucun autre changement.

2. Dans le cas général, on peut encore ajouter tous les atomes $\ulcorner C' \urcorner(*)$ à l'énumération des A_i^0, ce qui en fait une énumération infinie. On doit alors considérer des arbres sémantiques transfinis [Rus89], ce qui complique la démonstration.
3. Une version plus simple consiste à utiliser la méthode de démonstration de Bachmair et Ganzinger [BG01]. Le splitting sans splitting est en effet un cas particulier de leur critère général de redondance. Il y a des parallèles entre leur méthode et celle des arbres sémantiques [GLJ05].

L'exemple de la section 5.2 devient alors :

$$
\begin{array}{rll}
(i) & Q_1(x) \Leftarrow P_1(x), P_2(y), Q_2(y) & \text{par } (a), (b) \\
(j') & Q_1(x) \Leftarrow P_1(x), q(*) & \\
(k') & q(*) \Leftarrow P_2(y), Q_2(y) & \\
(m') & q(*) \Leftarrow P_2(y), Q_2(g(g(y))) & \text{par } (k'), (c) \\
(n') & q(*) \Leftarrow P_2(g(y)), Q_2(y) & \text{par } (m'), (e) \\
(o') & q(*) \Leftarrow P_2(y), Q_2(g(y)) & \text{par } (n'), (c) \\
(p') & q(*) \Leftarrow Q_2(g(g(a))) & \text{par } (o'), (d) \\
(q') & q(*) & \text{par } (p'), (f) \\
(j'') & Q_1(x) \Leftarrow P_1(x) & \text{par } (q'), (j') \\
(l') & \bot \Leftarrow P_1(g(a)) & \text{par } (j''), (g) \\
 & \Box & \text{par } (l'), (h)
\end{array}
$$

où $q = \ulcorner \bot \Leftarrow P_2(y), Q_2(y) \urcorner$. On pourra remarquer que les étapes (k')–(q') simulent exactement la branche droite du tableau de la section 5.1, les étapes suivant (j'') simulant la branche gauche.

Le splitting sans splitting a de nombreux avantages. D'abord, en pratique, les démonstrateurs automatiques par résolution sont écrits dans des langages impératifs, et effectuer du splitting est difficile dans ce contexte : il ne faut pas qu'une étape de déduction sur une branche modifie par inadvertance les données d'une autre branche. Ceci n'est pas un problème avec le splitting sans splitting, puisqu'il n'y a jamais qu'une branche dans ce cas. Ensuite, l'utilisation du splitting sans splitting permet l'utilisation de toutes les stratégies d'élimination de redondances de la section 4, alors que le splitting interdit l'élimination des clauses d'une branche qui sont redondantes par rapport à des clauses d'une autre branche. Finalement, l'utilisation de fonctions de sélection adaptées nous permettra d'obtenir grâce au splitting sans splitting des bornes de complexité fines, notamment en section 6.5. Ces bornes seront en particulier meilleures que par l'utilisation du splitting ordinaire.

5.4 Splitting non clos

On peut pousser plus loin l'idée du splitting sans splitting. Supposons que l'on ait une clause $C \vee C'$, mais que C et C' partagent une variable x. Alors $C \vee C'$ est équivalent à $\exists q \cdot \forall x \cdot (q(x) \Rightarrow C) \wedge \forall x \cdot (\neg q(x) \Rightarrow C')$, où q est un symbole de prédicat frais d'arité 1.

Soit $\mathcal{Q}_1$ un ensemble de symboles de prédicats disjoint de $\mathcal{P}$ et de $\mathcal{Q}$, en bijection avec les paires (x, C) formées d'une variable x et d'une clause C formée à partir de symboles de prédicats dans P, modulo renommage. Autrement dit, il existe une application qui à x et C associe $\ulcorner x \cdot C \urcorner$ de $\mathcal{Q}_1$, telle que $\ulcorner x \cdot C \urcorner = \ulcorner x' \cdot C' \urcorner$ si et seulement s'il existe un renommage ϱ tel que $C = C'\varrho$ et $x = \varrho(x')$.

La règle de *splitting non clos* est alors :

$$\frac{C \vee C'}{C \vee -\ulcorner x \cdot C' \urcorner(x) \quad + \ulcorner x \cdot C' \urcorner(x) \vee C'}$$

par laquelle on entend que C et C' sont deux sous-clauses non vides, ne partageant que la variable x au plus, telles que C' est formée de symboles de prédicats dans $\mathcal{P}$ uniquement, et que C contient au moins un atome $P(t)$ avec $P \in \mathcal{P}$; l'effet de la règle est de remplacer $C \vee C'$ par l'ensemble des deux clauses $C \vee -\ulcorner x \cdot C' \urcorner(x)$ et $+\ulcorner x \cdot C' \urcorner(x) \vee C'$.

Il sera utile de pouvoir étendre un tant soit peu cette règle, en assouplissant la condition sur C'. On autorise C' à être de la forme $C'_0 \vee -\ulcorner x \cdot C'_1 \urcorner(x) \vee \ldots \vee -\ulcorner x \cdot C'_k \urcorner(x)$. Noter que C' est intuitivement l'équivalent de la clause $C'_0 \vee C'_1 \vee \ldots \vee C'_k$. Par *splitting non clos*, nous comprendrons désormais la règle :

$$\frac{C \vee \overbrace{C'_0 \vee -\ulcorner x \cdot C'_1 \urcorner(x) \vee \ldots \vee -\ulcorner x \cdot C'_k \urcorner(x)}^{C'}}{C \vee -\ulcorner x \cdot C'_0 \vee C'_1 \vee \ldots \vee C'_k \urcorner(x) \quad + \ulcorner x \cdot C'_0 \vee C'_1 \vee \ldots \vee C'_k \urcorner(x) \vee C'} \tag{6}$$

où C et C' sont deux sous-clauses non vides, ne partageant que la variable x au plus, telles que C'_0 est formée de symboles de prédicats dans $\mathcal{P}$ uniquement, C'_i et C'_j ne partagent que la variable x au plus pour tous i, j, $0 \leq i < j \leq k$, et C contient au moins un atome $P(t)$ avec $P \in \mathcal{P}$.

Les arguments de la section 5.3 se répètent pratiquement mot pour mot. Une interprétation I de domaine D est *standard* si et seulement si elle est standard au sens de la section 5.3, et si pour toute clause C formée de symboles de prédicats dans $\mathcal{P}$ uniquement, pour toute variable x, $I_{\ulcorner x \cdot C \urcorner}$ est l'ensemble des valeurs $v \in D$ telles qu'il existe un environnement ρ tel que $\rho(x) = v$ et $I, \rho \not\models C$.

Lemme 13 (Correction) *La résolution ordonnée avec sélection, l'élimination de tautologies et de clauses subsumées, et les règles de splitting sans splitting et de splitting non clos, sont correctes au sens où, s'il existe une interprétation standard I telle que $I \models S$ et si S' est déduite de S par l'une de ces règles, alors $I \models S'$.*

En particulier, si on peut déduire la clause vide de S par ces règles, et si S ne contient que des clauses formées à l'aide de symboles de prédicats de $\mathcal{P}$, alors S est insatisfiable.

Appelons un ordre stable $\succ$ *admissible* si et seulement s'il est admissible au sens de la section 5.3 ($P(t) \succ Q(*)$ pour tout $P \in \mathcal{P}$, $Q \in \mathcal{Q}$), si $P(t) \succ Q(t')$ pour tous $P \in \mathcal{P}$, $Q \in \mathcal{Q}_1$, et tous termes clos t et t', et enfin si $\ulcorner x \cdot C \urcorner \succ \ulcorner x \cdot C' \urcorner$ dès que C subsume C' strictement.

Théorème 14 (Complétude) *Soit $\succ$ un ordre stable admissible. La résolution ordonnée pour $\succ$ avec sélection, élimination de tautologies, de clauses subsumées linéairement, splitting sans*

splitting et splitting non clos, est complète : pour tout ensemble insatisfiable S de clauses formées à partir de symboles de prédicats dans $\mathcal{P}$, pour toute stratégie f, il existe une f-dérivation de la clause vide à partir de S.

Nous laissons la démonstration de ces deux résultats en exercice ; les arguments sont similaires à ceux du lemme 11 et du théorème 12 respectivement.

Donnons un exemple de splitting non clos, que nous réutiliserons par la suite. Considérons l'ensemble de clauses :

$$(a)\ P_0(f(x,y)) \Leftarrow P_1(x), P_4(y) \quad (b)\ P_3(f(x,y)) \Leftarrow P_1(x), P_2(y) \quad (c)\ P_4(f(x,y)) \Leftarrow P_4(x), P_3(y)$$
$$(d)\ P_1(a) \quad (e)\ P_2(a) \quad (f)\ P_4(a)$$
$$(g)\ Q(x) \Leftarrow P_0(x), P_1(x) \quad (h)\ P_1(x) \Leftarrow P_3(x), P_4(x) \quad (i)\ P_3(x) \Leftarrow P_4(x)$$

Choisissons un ordre $\succ$ tel que $P(f(u,v)) \succ P'(u), P'(v)$ pour tous P, P', u, v. On a les étapes de résolution, où l'on attire l'attention sur les étapes de splitting non clos (k_0)–(k_2) et (n_0)–(n_2) :

(j)	$P_1(f(x,y)) \Leftarrow P_1(x), P_2(y), P_4(f(x,y))$	par $(h), (b)$
(k)	$P_1(f(x,y)) \Leftarrow P_1(x), P_2(y), P_4(x), P_3(y)$	par $(j), (c)$
(k_0)	$P_1(f(x,y)) \Leftarrow p_{14}(x), p_{23}(y)$	par splitting non clos, avec
(k_1)	$p_{14}(x) \Leftarrow P_1(x), P_4(x)$	$p_{14} = \ulcorner x \cdot \bot \Leftarrow P_1(x), P_4(x) \urcorner$
(k_2)	$p_{23}(x) \Leftarrow P_2(x), P_3(x)$	$p_{23} = \ulcorner x \cdot \bot \Leftarrow P_2(x), P_3(x) \urcorner$
(l)	$Q(f(x,y)) \Leftarrow P_1(x), P_4(y), P_1(f(x,y))$	par $(g), (a)$
(m)	$Q(f(x,y)) \Leftarrow P_1(x), P_4(y), p_{14}(x), p_{23}(y)$	par $(l), (k_0)$
(n_0)	$Q(f(x,y)) \Leftarrow p_{14}(x), p_{234}(y)$	par splitting non clos (6), avec
(n_1)	$p_{234}(x) \Leftarrow P_4(x), p_{23}(x)$	$p_{234} = \ulcorner x \cdot \bot \Leftarrow P_2(x), P_3(x), P_4(x) \urcorner$
	$p_{14}(x) \Leftarrow P_1(x), p_{14}(x)$	[tautologie]

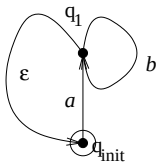
Remarque 6 *On peut définir une règle de splitting généralisé comme suit. À partir de n'importe quelle disjonction $C \vee C'$ de deux clauses non vides, telles que les variables partagées entre C et C' soient parmi $x_1, \ldots, x_n$, on fabrique les deux clauses $C \vee -q(x_1, \ldots, x_n)$ et $+q(x_1, \ldots, x_n) \vee C'$, où q est un symbole de prédicat frais. On laisse le soin de développer les détails au lecteur. On remarquera aussi que le cas $n = 0$ est le splitting sans splitting, et le cas $n = 1$ est ce que nous avons présenté ici comme étant le splitting non clos. Nous n'aurons jamais besoin du cas $n \geq 2$ dans la suite.*

6 Automates d'arbres, contraintes ensemblistes

Nous avons vu en section 2.4 que tout ensemble S de clauses de Horn définissait des langages $L_P(S)$. Ceci n'est pas sans rappeler la théorie des automates, et pour cause. Nous développons ici ce thème extensivement.

6.1 Automates d'arbres

Considérons par exemple l'automate de mots finis à droite. On suppose que l'état initial est q_{init}, et que les états finaux sont entourés (le seul état final est donc q_{init}). Cet automate reconnaît tous les mots de la forme $(ab^*)^*$, c'est-à-dire tous les mots sur l'alphabet $\{a, b\}$ commençant par a et le mot vide.

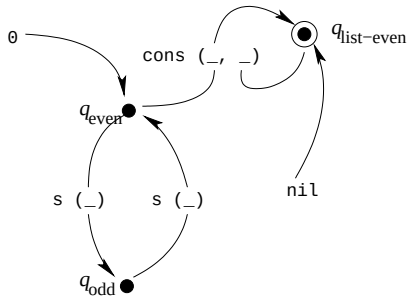


On peut exactement décrire l'ensemble des mots reconnus à chaque état q, sous forme de clauses de Horn, où $q(t)$ signifie que t est reconnu en q :

$$q_{init}(\epsilon) \qquad q_1(a(X)) \Leftarrow q_{init}(X)$$
$$q_1(b(X)) \Leftarrow q_1(X) \qquad q_{init}(X) \Leftarrow q_1(X)$$

Tout mot u est ici codé sous la forme du terme clos $u(\epsilon)$, où ϵ est une constante : en général, si u est le mot vide, alors $u(t) = t$, et si u est de la forme va, où a est une lettre, alors $u(t) = a(v(t))$.

On peut en général définir des automates d'*arbres*, qui sont des automates finis permettant de reconnaître des ensembles de termes clos. L'automate d'arbres de droite, par exemple, reconnaît (en $q_{\text{list}-\text{even}}$) l'ensemble de toutes les listes d'entiers pairs. Pour être précis, l'ensemble de toutes les listes $\texttt{cons}(t_1, \texttt{cons}(t_2, \ldots, \texttt{cons}(t_n, \texttt{nil})\ldots))$, où chaque t_i est de la forme $\texttt{S}^{n_i}(\texttt{0})$, n_i pair. Noter que la transition `0` (en haut à gauche) part de 0 état, alors que la transition $\texttt{cons}(\_, \_)$ (au milieu) part de deux états q_{even} et $q_{\text{list}-\text{even}}$.



Pour définir précisément la sémantique des automates d'arbres, le plus simple est de la décrire en clauses de Horn. Comme plus haut, chaque transition donnera lieu à exactement une clause :

$$q_{\text{even}}(\texttt{0}) \qquad q_{\text{even}}(\texttt{S}(X)) \Leftarrow q_{\text{odd}}(X) \qquad q_{\text{odd}}(\texttt{S}(X)) \Leftarrow q_{\text{even}}(X)$$
$$q_{\text{list}-\text{even}}(\texttt{cons}(X, Y)) \Leftarrow q_{\text{even}}(X), q_{\text{list}-\text{even}}(Y) \qquad q_{\text{list}-\text{even}}(\texttt{nil})$$

Noter que l'on n'a pas besoin de définir des états initiaux dans un automate d'arbre : `0` par exemple est reconnu en q_{even}, en utilisant la transition $q_{\text{even}}(\texttt{0})$, d'arité 0.

En général, on appellera *automate d'arbres* tout ensemble S de clauses de Horn de la forme

$$P(f(X_1, \ldots, X_n)) \quad \Leftarrow \quad P_1(X_1), \ldots, P_n(X_n) \tag{7}$$

où les variables X_1, ..., X_n sont deux à deux distinctes. Lorsque $n = 0$, on retrouve les clauses initiales telles que $q_{\text{even}}(\texttt{0})$.

Un langage de termes est dit *régulier* si et seulement s'il est de la forme $L_P(S)$, pour un prédicat P et un automate d'arbres S.

Test d'appartenance. On peut tester si un terme clos t est dans $L_P(S)$ en testant si S plus la clause $\bot \Leftarrow P(t)$ est insatisfiable, par le lemme 3. Soit $\rhd$ l'ordre *sous-terme*, c'est-à-dire le plus petit ordre strict tel que $f(t_1, \ldots, t_n) \rhd t_i$ pour tout i, $1 \leq i \leq n$, et tout f. Utilisons l'ordre $\succ_1$

sur les atomes défini par $P(t) \succ_1 P'(t')$ si et seulement si $t \rhd t'$. Il est facile de voir que, sur toute branche du tableau obtenu par résolution ordonnée (avec $\succ_1$) et splitting, on ne générera que des clauses de la forme $\bot \Leftarrow Q(u)$, où Q est un symbole de prédicat et u un sous-terme de t. Chaque branche est donc de longueur polynomiale en la taille de t et S, et le problème est donc dans coNP. En réalité, le problème est même dans P : on peut extraire de toute dérivation de $\Box$ par hyperrésolution *positive* cette fois une dérivation par hyperrésolution positive ne dérivant que des clauses unitaires de la forme $P(u)$, où u est un sous-terme de t ; de plus, si u est de la forme $f(u_1, \ldots, u_n)$, alors $P(u)$ est dérivé par une instance d'une clause (7) de la forme

$$P(f(u_1, \ldots, u_n)) \quad \Leftarrow \quad P_1(u_1), \ldots, P_n(u_n)$$

Il n'y a qu'un nombre polynomial de telles instances, lorsque u parcourt les sous-termes de t. De plus, ces instances sont closes, donc l'ensemble de ces instances est en fait un ensemble de clauses de Horn propositionnelles. Ceci reste vrai lorsqu'on rajoute la clause $\bot \Leftarrow P(t)$. Or l'hyperrésolution positive sur des clauses de Horn propositionnelles termine en temps polynomial. (On peut même la faire terminer en temps linéaire, en utilisant des structures d'indexation adaptées.) Donc l'appartenance d'un terme clos à $L_P(S)$ est décidable en temps polynomial.

Test du vide. On peut aussi tester si $L_P(S)$ est vide par résolution ordonnée et splitting, en utilisant le lemme 4. On ne génère que des clauses de la forme $Q(X)$, donc le problème est aussi dans coNP. Cependant, le problème de la vacuité est lui aussi, en fait, dans P. En effet, $L_P(S)$ est non vide si et seulement si l'ensemble des clauses obtenues à partir de S en oubliant les arguments des symboles des prédicats (c'est-à-dire en passant de $Q(t)$ à Q) plus la clause $\bot \Leftarrow P$ est insatisfiable. On obtient ainsi encore un ensemble de clauses propositionnelles. Leur insatisfiabilité est donc décidable par hyperrésolution positive en temps linéaire.

6.2 Clôture par les opérations booléennes

Il est possible de montrer que toute union finie, toute intersection finie, tout complément de langages reconnus par des automates d'arbres est encore reconnaissable par automates d'arbres.

Union. Le cas de l'union est trivial : si P est un état nouveau, et si l'on ajoute les deux clauses $P(X) \Leftarrow Q_1(X)$ et $P(X) \Leftarrow Q_2(X)$, alors le langage reconnu en P est l'union de ceux reconnus en Q_1 et Q_2. (Pour ceci, on suppose que les deux automates dont on souhaite calculer l'union ont été fusionnés en un seul, en prenant l'union de leurs ensembles de clauses, après renommage éventuel des symboles de prédicats de l'un des deux.)

Intersection. Le cas de l'intersection fait appel à la *construction produit*. Pour calculer l'intersection de deux automates d'arbres — ensembles de clauses — S_1 et S_2, on considère les symboles de prédicats $P^1 \cap P^2$, pour chaque symbole P^1 apparaissant dans S_1 et P^2 dans S_2 ; on construit l'automate dont les clauses sont

$$(P^1 \cap P^2)(f(X_1, \ldots, X_n)) \Leftarrow (P_1^1 \cap P_1^2)(X_1), \ldots, (P_n^1 \cap P_n^2)(X_n)$$

pour toute paire de clauses

$$\begin{aligned} P^1(f(X_1,\ldots,X_n)) &\Leftarrow P_1^1(X_1),\ldots,P_n^1(X_n) \\ P^2(f(X_1,\ldots,X_n)) &\Leftarrow P_1^2(X_1),\ldots,P_n^2(X_n) \end{aligned}$$

dans S_1 et S_2 respectivement, avec le même f. (Les X_i sont les mêmes, ce qu'on peut toujours supposer, moduler renommage.)

Au lieu d'utiliser la construction produit, on pourrait ajouter la simple clause $P(X) \Leftarrow Q_1(X), Q_2(X)$ pour reconnaître en P (P nouveau prédicat) l'intersection des langages reconnus en Q_1 et Q_2. Mais cette nouvelle clause n'est pas une clause de la forme (7). Nous verrons en section 6.3 que l'utilisation de résolution ordonnée avec sélection et splitting non clos permet en fait d'éliminer ces clauses, et de retrouver un automate d'arbres. Cette dernière construction est une optimisation de la construction produit, au sens où tous les produits $P^1 \cap P^2$ ne sont pas engendrés.

Le calcul de l'intersection binaire est faisable en temps polynomial, en fait quadratique. Le calcul de l'intersection de k automates de m clauses produit un automate d'au plus m^k clauses. On peut montrer que ceci est optimal, au sens où la vacuité de l'intersection d'une famille finie d'automates d'arbres est un problème DEXPTIME-complet [Sei94]. (Le problème analogue sur les automates de mots est PSPACE-complet [Koz77].)

Complémentaire. La construction du complémentaire de $L_P(S)$ est plus complexe. En particulier, il ne suffit pas de créer de nouveaux symboles de prédicats $\overline{P}$ dénotant le complémentaire de P, et d'exploiter le fait que l'on doit avoir $\overline{P}(X)$ si et seulement si non $P(X)$. La raison en est que $\overline{P}(t)$ doit être vrai si et seulement si $P(t)$ est faux *dans le plus petit modèle de Herbrand* de l'automate d'arbres S. Or passer à la négation nous fait passer du plus petit au plus grand modèle de Herbrand, ce qui ne répond pas à la question. Par exemple, considérons l'ensemble de clauses S vide, $L_P(S)$ est donc vide, et l'on s'attend à ce que $\overline{P}$ reconnaisse tous les termes clos, mais définir $\overline{P}$ comme la négation de P signifie ajouter les clauses

$$-P(X) \vee -\overline{P}(X) \qquad +P(X) \vee +\overline{P}(X)$$

Au passage, celle de droite n'est pas une clause de Horn, et il n'y a pas de plus petit modèle. On laisse le lecteur vérifier que ces deux clauses plus $\bot \Leftarrow P(X)$, ou bien plus $\bot \Leftarrow \overline{P}(X)$, forment des ensembles satisfiables (mais pour différents modèles). En particulier, ajouter $\bot \Leftarrow \overline{P}(X)$ ne produit pas d'insatisfiabilité (alors que le complémentaire de P est non vide... dans le plus petit modèle de l'ensemble vide de clauses).

À la place, pour chaque symbole de prédicat P, pour chaque symbole de fonction P, énumérons les clauses de S dont la tête (l'atome à gauche de $\Leftarrow$) est de la forme $P(f(X_1,\ldots,X_n))$:

$$P(f(X_1,\ldots,X_n)) \;\Leftarrow\; P_{i1}(X_1),\ldots,P_{in}(X_n) \qquad (1 \leq i \leq k)$$

Un terme clos $f(t_1,\ldots,t_n)$ tel que $t_1 \in L_{P_{i1}}(S)$, ..., $t_n \in L_{P_{in}}(S)$ est dans $L_P(S)$, clairement. Mais comme l'on considère le plus petit modèle de Herbrand de S, la réciproque est vraie : si

$f(t_1, \ldots, t_n)$ est reconnu en P, alors il existe i, $1 \leq i \leq k$, tel que t_1 est reconnu en P_{i1}, ..., t_n est reconnu en P_{in}. Autrement dit, la formule suivante est vraie dans le plus petit modèle de S :

$$P(f(X_1, \ldots, X_n)) \quad \Leftrightarrow \quad \bigvee_{i=1}^{k} P_{i1}(X_1) \wedge \ldots \wedge P_{in}(X_n)$$

Passant aux complémentaires, et réutilisant la notation $\overline{P}$ introduite plus haut,

$$\overline{P}(f(X_1, \ldots, X_n)) \quad \Leftrightarrow \quad \bigwedge_{i=1}^{k} \overline{P}_{i1}(X_1) \vee \ldots \vee \overline{P}_{in}(X_n)$$

Ceci peut se réécrire en ne conservant que la direction $\Leftarrow$ de l'équivalence ci-dessus (l'autre venant, de même, du fait que nous considérons toujours le plus petit modèle), et en introduisant de nouveaux prédicats Q_i servant à abréger les disjonctions de droite de l'équivalence ci-dessus :

$$\begin{array}{rcl} \overline{P}(X) & \Leftarrow & Q_1(X), \ldots, Q_k(X) \\ Q_i(f(X_1, \ldots, X_n)) & \Leftarrow & \overline{P}_{ij}(X_j) \quad (1 \leq i \leq k, 1 \leq j \leq n) \end{array}$$

Aucune de ces deux formes de clauses ne correspond à (7). Dans le cas de la deuxième forme, ceci est aisément réparable, quoique de façon peu élégante, en les réécrivant

$$Q_i(f(X_1, \ldots, X_n)) \quad \Leftarrow \quad \overline{P}_{ij}(X_j), \top(X_1), \ldots, \top(X_{j-1}), \top(X_{j+1}), \ldots, \top(X_n)$$

où $\top$ est un prédicat reconnaissant tous les termes : ajouter toutes les clauses $\top(g(Y_1, \ldots, Y_m)) \Leftarrow \top(Y_1), \ldots, \top(Y_m)$. Dans le cas de la première forme, on a des clauses intersection, que l'on peut éliminer par des constructions produits, comme plus haut.

Ceci prend un temps exponentiel en la taille des clauses. Une autre façon classique de calculer les unions, intersections, et compléments, est de déterminiser les automates d'arbres. Nous ne le traiterons pas ici, voir [GS97]. Dans tous les cas, la complexité exponentielle est inévitable : le problème de l'universalité d'un état P de l'automate S, c'est-à-dire le problème de la vacuité du complémentaire de $L_P(S)$, est DEXPTIME-complet (PSPACE-complet pour les automates de mots).

Un point intéressant est que la construction du complémentaire $\overline{P}$ au moyen de passages d'implications à des équivalences, qui sont valides dans le plus petit modèle de Herbrand, se généralise à n'importe quel format de clauses de Horn. Pour définir $\overline{P}$, énumérer toutes les clauses dont la tête commence par le prédicat P :

$$P(t_i) \quad \Leftarrow \quad P_{i1}(t_{i1}), \ldots, P_{in}(t_{in}) \quad (1 \leq i \leq k)$$

Alors, dans le plus petit modèle, on a :

$$P(X) \quad \Leftrightarrow \quad \bigvee_{i=1}^{k} \exists \vec{x}_i \cdot X = t_i \wedge P_{i1}(t_{i1}) \wedge \ldots \wedge P_{in}(t_{in})$$

où $\vec{x}_i$ est la liste des variables libres de la $i$ième clause ci-dessus, et X est une variable fraîche. (Ces équivalences forment ce qu'on appelle la *complétion de Clark* de S.) Donc

$$\overline{P}(X) \quad \Leftrightarrow \quad \bigwedge_{i=1}^{k} \forall \vec{x}_i \cdot (X = t_i \Rightarrow \overline{P}_{i1}(t_{i1}) \vee \ldots \vee \overline{P}_{in}(t_{in}))$$

Mais cette équivalence ne s'écrit pas nécessairement sous forme d'un ensemble fini de clauses en général, ce qui interdit de généraliser la construction du complémentaire aux clauses de Horn générales. Malgré cela, on peut *définir* la négation $\overline{P}$ de P comme étant le prédicat satisfaisant l'équivalence ci-dessus. Cette négation n'est pas la négation classique ; c'est la *négation par l'échec* du langage Prolog (au moins dans le cas de la sémantique dite stratifiée, mais nous ne rentrerons pas dans ces détails).

6.3 Automates bidirectionnels, alternants, et autres

On peut considérer des automates enrichis, utilisant d'autres formes de clauses, et pas seulement les clauses (7).

ϵ-transitions. Un premier ajout, bénin, est celui des *ϵ-transitions* :

$$P(X) \quad \Leftarrow \quad P'(X) \tag{8}$$

qui exprimer que tout terme reconnu en P' est aussi reconnu en P. Nous avons déjà utilisé une telle forme de clause dans notre premier exemple d'automate en section 6.1, sans le dire.

Un ensemble de clauses de forme (7) ou (8) sera appelé un *automate d'arbre avec ϵ-transitions*. Tous les résultats de décidabilité et de complexité des test d'appartenance et du vide, et des calculs d'unions, d'intersections, et de complémentaires, tiennent toujours en présence d'ϵ-transitions ; les modifications à apporter aux arguments des sections précédentes sont mineures.

Alternance. Un ajout plus intéressant, et que nous avons déjà fait en section 6.2 dans les calculs d'intersections et de complémentaires, est de considérer des *clauses intersection* :

$$P(X) \quad \Leftarrow \quad P_1(X), \ldots, P_n(X) \tag{9}$$

où $n \geq 1$. (On inclura donc le cas des ϵ-transitions dans les clauses intersection.)

Une collection de clauses de la forme (7) ou (9) est appelé un automate d'arbre *alternant*. La raison intuitive est que, pour reconnaître un terme clos t en P, on doit d'abord trouver (par un choix existentiel) une clause qui permet de dériver $P(t)$, et si cette clause est une clause intersection (9), on doit vérifier que t est reconnu en tout état P_i, $1 \leq i \leq n$ (c'est un choix universel). On reconnaît donc t en alternant des choix existentiels et universels.

En réalité, le problème de l'appartenance d'un terme clos t à $L_P(S)$ est dans P. Il suffit, comme dans le cas non alternant, d'énumérer les instances closes de clauses de S fabriquées à l'aide de sous-termes de t, et de tester l'insatisfiabilité par hyperrésolution positive.

Par contre, le problème de la vacuité est DEXPTIME-complet. Il est DEXPTIME-difficile, parce qu'il contient le problème de l'intersection d'automates d'arbres, qui est DEXPTIME-complet, comme on l'a vu en section 6.2. Il est dans DEXPTIME, ce qui est une conséquence du théorème suivant et du fait que la vacuité d'automates d'arbres est dans P. (On peut aussi donner une démonstration directe en utilisant la même stratégie de résolution que celle de la démonstration de ce théorème.)

Théorème 15 *Pour tout automate d'arbres alternant S, on peut calculer en temps exponentiel un automate d'arbres $reg(S)$ équivalent, au sens où pour tout symbole de prédicat P apparaissant dans S, $L_P(S) = L_P(reg(S))$.*

(En première approche, on pourra lire la démonstration à l'aide de l'exemple donné à sa suite.)

Démonstration. Saturons S par résolution ordonnée avec sélection, splitting non clos, et élimination de clauses subsumées linéairement. (L'élimination de clauses subsumées ne nous servira qu'à éliminer des clauses qui sont identiques modulo renommage.) Le plan de la preuve est le suivant : on montre que cette étape de saturation ne produit que des clauses d'un certain format, puis qu'il n'y a qu'un nombre exponentiel de clauses de ce format, donc la stratégie de résolution termine sur un ensemble saturé S' ; ensuite, on montre que toute déduction de la clause vide à partir de S' et de $\bot \Leftarrow P(t)$, permettant de tester si $t \in L_P(S') = L_P(S)$, n'utilise comme clauses de S' que celles qui sont des clauses d'automates d'arbres (7).

L'ordre choisi est $\succ_1$ (étendu à un ordre admissible quelconque), la fonction de sélection est telle que $\mathrm{sel}\ (C)$ est l'ensemble de tous les littéraux maximaux pour $\succ_1$ de C qui sont négatifs. Finalement, nous appliquons la règle de splitting non clos (6) dès que possible, sous la condition supplémentaire que C' soit une clause négative d'au moins deux littéraux. Soit $\mathcal{P}$ l'ensemble des symboles de prédicats apparaissant dans S.

On peut supposer sans perte de généralité que toute clause intersection (9) est telle que $1 \leq n \leq 2$; sinon, on peut réécrire $P(X) \Leftarrow P_1(X), \ldots, P_n(X)$ sous forme de $n-1$ clauses $P(X) \Leftarrow P_1(X), P'_1(X)$ et $P'_1(X) \Leftarrow P_2(X), P'_2(X)$ et … et $P'_{n-2}(X) \Leftarrow P_{n-1}(X), P_n(X)$.

Soit $\mathcal{Q}_{10}$ l'ensemble des symboles de la forme $\ulcorner x \cdot \bot \Leftarrow P_1(X), \ldots, P_n(X) \urcorner$, où $P_1, \ldots, P_n \in \mathcal{P}$. Les clauses obtenues dans toute dérivation partant de S sont de la forme :

1. $P(f(X_1, \ldots, X_n)) \Leftarrow Q_1(X_1), \ldots, Q_n(X_n)$, où $X_1, \ldots, X_n$ sont deux à deux distinctes, $P, Q_1, \ldots, Q_n \in \mathcal{P} \cup \mathcal{Q}_{10}$,
2. ou $P(f(X_1, \ldots, X_n)) \Leftarrow Q_1(X_1), \ldots, Q_n(X_n), P_1(f(X_1, \ldots, X_n))$, où $X_1, \ldots, X_n$ sont deux à deux distinctes, $m \geq 1$, $P, P_1, Q_1, \ldots, Q_n \in \mathcal{P} \cup \mathcal{Q}_{10}$,
3. ou $Q(X) \Leftarrow Q_1(X), \ldots, Q_n(X)$, où $1 \leq n \leq 2$ et $Q, Q_1, \ldots, Q_n \in \mathcal{P} \cup \mathcal{Q}_{10}$.

En effet, c'est clairement le cas initialement. Ensuite, on remarque que l'unique littéral maximal dans les clauses de type 1 est la tête ; en particulier, aucun littéral n'est sélectionné dans les clauses de type 1. Si l'on résout une clause de type 1

$$P_1(f(X_1, \ldots, X_n)) \Leftarrow Q_1(X_1), \ldots, Q_n(X_n)$$

sur sa tête $P_1(f(X_1, \ldots, X_n))$ avec une clause de type 2

$$P(f(X_1, \ldots, X_n)) \Leftarrow Q'_1(X_1), \ldots, Q'_n(X_n), P_1(f(X_1, \ldots, X_n))$$

on obtient :

$$\begin{array}{rcl} P(f(X_1,\ldots,X_n)) & \Leftarrow & Q_1(X_1),\ldots,Q_n(X_n), \\ & & Q'_1(X_1),\ldots,Q'_n(X_n) \end{array}$$

Par splitting non clos (6, on obtient les clauses :

$$\begin{array}{rcl} P(f(X_1,\ldots,X_n)) & \Leftarrow & Q''_1(X_1),\ldots,Q''_n(X_n) \\ Q''_1(X) & \Leftarrow & Q_1(X), Q'_1(X) \\ & \ldots & \\ Q''_n(X) & \Leftarrow & Q_n(X), Q'_n(X) \end{array}$$

où Q''_1, ..., Q''_n sont des symboles de prédicats dans $\mathcal{Q}_{10}$, et la première clause est de type 1. Si l'on résout une clause de type 1 avec une clause de type 3 (avec n littéraux négatifs, $1 \leq n \leq 2$), on obtient soit une clause de type 1 si $n = 1$ soit une clause de type 2 si $n = 2$. Comme les clauses de type 2 ou 3 contiennent au moins un littéral sélectionné, on ne peut pas résoudre sur leur tête.

On remarque ensuite qu'il n'y a qu'un nombre exponentiel de clauses de type 1, 2, ou 3. Soit p le nombre de symboles de prédicats dans $\mathcal{P}$, f le nombre de symboles de fonctions dans la signature Σ, a l'arité maximale des symboles de fonctions de Σ. Il y a en effet au plus 2^p symboles dans $\mathcal{Q}_{10}$, donc au plus $f2^{p(a+1)}$ clauses de type 1 à renommage près, au plus $f2^{p(a+2)}$ clauses de type 2, et au plus $2^{2p} + 2^{3p}$ clauses de type 3. On peut donc produire à partir de S un ensemble de clauses S' saturé en temps exponentiel.

Posons maintenant $reg(S)$ l'ensemble des clauses de S' qui sont de type 1. Clairement, $reg(S)$ est un automate d'arbres. Il ne reste qu'à montrer que pour tout $P \in \mathcal{P}$, $L_P(S) = L_P(reg(S))$.

Considérons n'importe quel ensemble S_1 de clauses de la forme

$$(*) \qquad \bot \Leftarrow P_1(t_1),\ldots,P_m(t_m)$$

tel que $P_1,\ldots,P_m \in \mathcal{P} \cup \mathcal{Q}_{01}$, et $t_1,\ldots,t_m$ sont des termes clos. Considérons une étape de la stratégie de résolution choisie, $S' \cup S_1 \rightarrow S''$, non triviale au sens où $S'' \neq S' \cup S_1$. Cette étape ne peut pas être une étape de résolution entre clauses de S', car S' est saturé. Ce ne peut pas être une étape de résolution entre clauses de S_1, car S_1 ne contient que des clauses négatives. Finalement, considérons une étape de résolution ordonnée avec sélection entre une clause $(*)$ et une clause C de S'. C est nécessairement de type 1, car c'est le seul type de clause où aucun littéral n'est sélectionné. Le résolvant obtenu est de la forme $(*)$ de nouveau.

On en conclut que dans toute dérivation $S' \cup S_1 = S'_1 \rightarrow S'_2 \rightarrow \ldots \rightarrow S'_i \rightarrow \ldots$, où l'on peut supposer sans perte de généralité que chaque étape $S'_i \rightarrow S'_{i+1}$ est non triviale, les seules clauses de S' qui sont utilisées sont de type 1, donc dans $reg(S)$. Pour tout terme clos t, si $t \in L_P(S)$, alors $S \cup \{\bot \Leftarrow P(t)\}$ a une dérivation de la clause vide comme ci-dessus, donc aussi $reg(S) \cup \{\bot \Leftarrow P(t)\}$, donc $t \in L_P(reg(S))$. La réciproque $t \in L_P(reg(S)) \Rightarrow t \in L_P(S)$ est évidente. □

Illustrons la procédure sur l'exemple des clauses (a)–(i) de la section 5.4 :

$$(a)\ P_0(f(x,y)) \Leftarrow P_1(x), P_4(y) \quad (b)\ P_3(f(x,y)) \Leftarrow P_1(x), P_2(y) \quad (c)\ P_4(f(x,y)) \Leftarrow P_4(x), P_3(y)$$
$$(d)\ P_1(a) \qquad (e)\ P_2(a) \qquad (f)\ P_4(a)$$
$$(g)\ Q(x) \Leftarrow P_0(x), P_1(x) \qquad (h)\ P_1(x) \Leftarrow P_3(x), P_4(x) \qquad (i)\ P_3(x) \Leftarrow P_4(x)$$

On pourra vérifier que l'ensemble des clauses déduites par la stratégie de résolution utilisée dans la preuve fournit les clauses (j)–(n_1) déjà décrites en section 5.4, plus d'autres :

(j)	$P_1(f(x,y)) \Leftarrow P_1(x), P_2(y), P_4(f(x,y))$	par $(h),(b)$
(k)	$P_1(f(x,y)) \Leftarrow P_1(x), P_2(y), P_4(x), P_3(y)$	par $(j),(c)$
(k_0)	$P_1(f(x,y)) \Leftarrow p_{14}(x), p_{23}(y)$	par splitting non clos, avec
(k_1)	$p_{14}(x) \Leftarrow P_1(x), P_4(x)$	$p_{14} = \ulcorner x \cdot \bot \Leftarrow P_1(x), P_4(x) \urcorner$
(k_2)	$p_{23}(x) \Leftarrow P_2(x), P_3(x)$	$p_{23} = \ulcorner x \cdot \bot \Leftarrow P_2(x), P_3(x) \urcorner$
(l)	$Q(f(x,y)) \Leftarrow P_1(x), P_4(y), P_1(f(x,y))$	par $(g),(a)$
(m)	$Q(f(x,y)) \Leftarrow P_1(x), P_4(y), p_{14}(x), p_{23}(y)$	par $(l),(k_0)$
(n_0)	$Q(f(x,y)) \Leftarrow p_{14}(x), p_{234}(y)$	par splitting non clos (6), avec
(n_1)	$p_{234}(x) \Leftarrow P_4(x), p_{23}(x)$	$p_{234} = \ulcorner x \cdot \bot \Leftarrow P_2(x), P_3(x), P_4(x) \urcorner$
	$p_{14}(x) \Leftarrow P_1(x), p_{14}(x)$	[tautologie]
(o)	$Q(a) \Leftarrow P_0(a)$	par $(g),(d)$
(p)	$P_1(a) \Leftarrow P_3(a)$	par $(h),(f)$
(q)	$P_3(a)$	par $(i),(f)$
(r)	$p_{23}(f(x,y)) \Leftarrow P_2(f(x,y)), P_1(x), P_2(y)$	par $(k_2),(b)$
(s)	$p_{14}(f(x,y)) \Leftarrow P_1(f(x,y)), P_4(x), P_3(y)$	par $(k_1),(c)$
(t)	$p_{234}(f(x,y)) \Leftarrow P_4(x), P_3(y), p_{23}(f(x,y))$	par $(n_1),(c)$
(u)	$p_{14}(a) \Leftarrow P_4(a)$	par $(k_1),(d)$
(v)	$p_{23}(a) \Leftarrow P_3(a)$	par $(k_2),(e)$
(w)	$p_{234}(a) \Leftarrow p_{23}(a)$	par $(n_1),(f)$
(x)	$p_{14}(a)$	par $(u),(f)$
(y)	$Q(f(x,y)) \Leftarrow P_0(f(x,y)), p_{14}(x), p_{23}(y)$	par $(k_0),(g)$
(z)	$p_{14}(f(x,y)) \Leftarrow p_{14}(x), p_{23}(y), P_4(f(x,y))$	par $(k_0),(k_1)$
(aa)	$P_1(a) \Leftarrow P_4(a)$	par $(q),(h)$
(ab)	$p_{23}(a) \Leftarrow P_2(a)$	par $(q),(k_2)$
	$P_1(a)$	par $(q),(p)$ [subsumé par (d)]
(ac)	$p_{23}(a)$	par $(q),(v)$
(ad)	$p_{234}(a) \Leftarrow P_4(a)$	par $(ac),(n_1)$
(ae)	$p_{234}(a)$	par $(ac),(w)$ [subsume (ad)]

L'automate d'arbres déduit par la procédure du théorème 15 est donc le sous-ensemble des clauses d'automates d'arbres, soit :

$$
\begin{array}{cc}
(a) & P_0(f(x,y)) \Leftarrow P_1(x), P_4(y) \\
(b) & P_3(f(x,y)) \Leftarrow P_1(x), P_2(y) \\
(c) & P_4(f(x,y)) \Leftarrow P_4(x), P_3(y) \\
(d) & P_1(a) \\
(e) & P_2(a) \\
(f) & P_4(a) \\
(k_0) & P_1(f(x,y)) \Leftarrow p_{14}(x), p_{23}(y) \\
(n_0) & Q(f(x,y)) \Leftarrow p_{14}(x), p_{234}(y) \\
(q) & P_3(a) \\
(x) & p_{14}(a) \\
(ac) & p_{23}(a) \\
(ae) & p_{234}(a)
\end{array}
$$

Automates bidirectionnels. On peut enrichir les automates d'arbres avec des clauses de la forme

$$P(X_i) \quad \Leftarrow \quad Q(f(X_1, \ldots, X_n)) \tag{10}$$

où $1 \leq i \leq n$ et les variables X_1, ..., X_n sont distinctes deux à deux. Alors que les clauses (7) reconnaissent un terme $f(t_1, \ldots, t_n)$ à un état P en le décomposant, et en essayant ensuite de reconnaître t_1 en P_1, ..., t_n en P_n (en lisant la clause (7) de gauche à droite), les clauses (10) tentent de reconnaître un terme t en P en devinant un terme plus gros $f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)$ reconnu en Q.

Un ensemble de clauses (7), (8) ou (10) est appelé un automate d'arbres *bidirectionnel*. Un ensemble de clauses (7), (9) ou (10) est appelé un automate d'arbres *bidirectionnel alternant*.

En vérification de protocoles cryptographiques, on a naturellement besoin de clauses qui sont des extensions naturelles du format (10) :

$$P(X_i) \quad \Leftarrow \quad Q(f(X_1, \ldots, X_n)), Q_1(X_{i_1}), \ldots, Q_k(X_{i_k}) \tag{11}$$

où $k \geq 0$, $1 \leq i_1, \ldots, i_k \leq n$ et les variables X_1, ..., X_n sont distinctes deux à deux. Par exemple, en supposant qu'un intrus de Dolev-Yao a réussi à récupérer un ensemble de messages défini par un prédicat E — autrement dit, le message t est récupéré si et seulement si $E(t)$ dans le plus petit modèle de Herbrand d'un ensemble de clauses donné —, alors le prédicat I défini par les clauses additionnelles :

$$
\begin{array}{rcl}
I(X) & \Leftarrow & E(X) \\
I(c(X,Y)) & \Leftarrow & I(X), I(Y)
\end{array}
$$

$$\begin{aligned} I(X) &\Leftarrow I(c(X,Y)), I(Y) \\ I(p(X,Y)) &\Leftarrow I(X), I(Y) \\ I(X) &\Leftarrow I(p(X,Y)) \\ I(Y) &\Leftarrow I(p(X,Y)) \end{aligned}$$

est tel que $I(t)$ est vrai dans le plus petit modèle de Herbrand si et seulement si t est déductible par l'intrus de Dolev-Yao, en lisant $c(u,v)$ "u chiffré en utilisant la clé symétrique v" et $p(u,v)$ "paire u, v". La première clause exprime que tout message récupéré est déductible, et est une ϵ-transition. La deuxième clause exprime que l'intrus peut déduire tout message qui est le chiffrement de deux messages déjà déduits, et est une transition d'automate d'arbres (7). La troisième clause exprime que l'intrus peut déchiffrer tout message chiffré $c(u,v)$ qu'il sait déduire à condition de pouvoir déduire aussi la clé v ; il s'agit d'une clause de la forme (11). Les trois dernières clauses, qui expriment que l'intrus peut déduire la paire $p(u,v)$ si et seulement s'il peut déduire à la fois u et v, sont une clause (7) et deux clauses (10).

Un ensemble de clauses (7), (8) ou (11) est appelé un automate d'arbres *bidirectionnel étendu*. Un ensemble de clauses (7), (9) ou (11) est appelé un automate d'arbres *bidirectionnel alternant étendu*.

Contraintes d'égalités entre frères. La restriction exprimant que les variables X_1, ..., X_n sont distinctes dans les clauses d'automates (7) et les clauses d'automates bidirectionnels (10) ou étendus (11) peut être relaxée. Considérons par exemple la clause d'automate (relaxée) suivante :

$$P(f(X,X,Y)) \Leftarrow Q_1(X), Q_2(X), Q_3(Y)$$

Elle exprime que $f(t_1,t_2,t_3)$ est reconnu en P dès que t_1 est reconnu en Q_1, t_2 en Q_2, t_3 en Q_3, *et* que $t_1 = t_2$ (c'est-à-dire que t_1 et t_2 sont identiques en tant que termes clos).

La contrainte d'égalité entre t_1 et t_2 porte sur des *frères* : t_1 et t_2 sont des arguments du même symbole de fonction f dans $f(t_1,t_2,t_3)$. Les ensembles de clauses définis par des clauses de la forme :

$$P(f(X_1,\ldots,X_n)) \Leftarrow P_1(X_1), \ldots, P_n(X_n)$$

sans aucune contrainte sur les variables X_1, ..., X_n, sont appelés des *automates d'arbres avec contraintes d'égalité entre frères*.

Il est à noter que les langages reconnus par automates d'arbres avec contraintes d'égalité entre frères sont plus riches que ceux reconnus par automates d'arbres. Par exemple, le langage L de tous les termes $f(t,t)$, t parcourant les termes clos, est reconnaissable en utilisant des égalités entre frères, mais il n'existe aucun automate d'arbres S tel que $L_P(S) = L$, pour aucun prédicat P.

Comme en section 6.1, on peut montrer que tester si un terme clos est reconnu à l'état P d'un automate d'arbre avec contraintes d'égalité entre frères est faisable en temps polynomial. L'union et l'intersection d'automates d'arbres avec contraintes d'égalité entre frères est calculable en temps polynomial sous forme d'automate d'arbres avec contraintes d'égalité entre frères, mais le

complémentaire n'est pas en général représentable sous forme d'automate d'arbre avec contrainte d'égalité entre frères. Pour remédier à ce problème, Bogaert et Tison [BT92] ont proposé d'utiliser des contraintes d'égalité *et* de différence $t_1 \neq t_2$ entre frères. Ces derniers automates sortent du cadre de ces notes, cependant.

On peut aussi étendre les contraintes d'égalité ou de différence à des sous-termes plus profonds, mais tester la vacuité des automates ainsi obtenus est indécidable, même pour des tests d'égalité dits entre cousins et pas de tests de différence. Pour plus d'information, voir [CDG$^+$97].

6.4 Décider la vacuité d'automates d'arbres généralisés

Généralisons généreusement le format de clauses que l'on peut rencontrer dans des automates d'arbres bidirectionnels alternants étendus, avec contraintes d'égalité entre frères. Notamment, ignorons le fait que toutes les clauses rencontrées sont de Horn.

Un *ϵ-bloc* est une clause de la forme :

$$\pm_1 P_1(X) \vee \ldots \vee \pm_n P_n(X)$$

où $n \geq 0$, et tous les prédicats P_i sont appliqués à la même variable X ; on notera souvent $B(X)$ un ϵ-bloc dont la seule variable libre (si elle existe) est X. Noter que la clause vide est un ϵ-bloc, que toute ϵ-transition est un ϵ-bloc, que toute clause intersection (9) est un ϵ-bloc.

Appelons *terme plat* tout terme de la forme $f(X_1, \ldots, X_m)$, où f est un symbole de fonction n-aire et X_1, ..., X_m sont des variables, non nécessairement distinctes.

Une *transition* est une clause de la forme :

$$\bigvee_{i=1}^{m} \pm_i P_i(t_i) \vee B_1(X_1) \vee \ldots \vee B_n(X_n)$$

où $m \geq 1$, les $B_j(x_j)$ sont des ϵ-blocs, et les t_i sont des termes plats dont les variables libres contiennent X_1, ..., X_n.

On appellera clause *automatique* tout clause qui est soit un ϵ-bloc soit une transition. On notera que toutes les clauses d'automates d'arbres bidirectionnels alternants étendus, avec contraintes d'égalité entre frères, sont des clauses automatiques, qui sont de plus des clauses définies. Les requêtes $\bot \Leftarrow P(X)$ (voir lemme 4) et les requêtes conjonctives $\bot \Leftarrow P_1(x), \ldots, P_n(x)$ (voir lemme 5) sont aussi des clauses automatiques.

Théorème 16 *La satisfiabilité d'ensembles S de clauses automatiques est décidable en D2EXPTIME, et dans NEXPTIME si l'arité des symboles de fonction est bornée par une constante.*

Démonstration. Par résolution ordonnée (voir section 3.1) avec l'ordre $\succ_1$, subsomption linéaire, et splitting. Rappelons que l'ordre $\succ_1$ est défini par $P(t) \succ_1 P'(t')$ si et seulement si $t \rhd t'$, c'est-à-dire si et seulement si t' est un sous-terme strict de t. Le plan de la preuve est : d'abord, montrer que la résolution ordonnée avec splitting ne fabrique que des clauses automatiques ; ensuite, montrer qu'il n'y a qu'un nombre fini, qu'on estimera, de clauses automatiques.

Appelons *projection* toute substitution σ telle que $\sigma(X)$ est une variable, pour toute variable X. Il est clair que, pour toute projection σ, pour tout ϵ-bloc C, $C\sigma$ est encore un ϵ-bloc, et pour toute transition C, $C\sigma$ est encore une transition. (On notera que cette dernière affirmation dépend du fait que nous autorisons les variables répétées, c'est-à-dire, dans le cas de Horn, les contraintes d'égalité entre frères.)

Tout facteur ordonné d'un ϵ-bloc est donc un ϵ-bloc. De plus, tout facteur ordonné d'une transition est une transition. En effet, soit $C \vee +A \vee +A'$ une transition avec A et A' unifiables et maximaux pour $\succ$. Nécessairement $A = P(t)$ et $A' = P(t')$ pour deux termes plats t et t'. Leur mgu est donc une projection, donc le facteur ordonné $(C \vee +A \vee +A')\sigma$ est une transition.

Montrons maintenant que tout résolvant binaire ordonné de deux clauses automatiques est une disjonction de clauses automatiques ne partageant aucune variable. En fait, on va montrer que tout résolvant binaire ordonné de deux clauses automatiques est soit une transition, soit une disjonction d'ϵ-blocs, que l'on peut toujours réécrire comme une disjonction d'ϵ-blocs ne partageant aucune variable ;

- Un résolvant binaire de deux ϵ-blocs est un ϵ-bloc.
- Si $C \vee +P(X)$ est un ϵ-bloc et $C' \vee -P(f(X_1, \ldots, X_n))$ est une transition, alors leur résolvant binaire est $C[X := f(X_1, \ldots, X_n)] \vee C'$, qui est une transition dès que C est non vide ou C' est une transition (c'est-à-dire contient au moins un atome $\pm Q(t)$, où t est un terme plat, nécessairement de variables libres $X_1, \ldots, X_n$). Sinon, C est vide et C' est de la forme $B_1(X_1) \vee \ldots \vee B_n(X_n)$, et le résolvant binaire est donc la disjonction des ϵ-blocs $B_1(X_1)$, …, et $B_n(X_n)$.
- Le cas symétrique $C \vee -P(X)$ un ϵ-bloc, $C' \vee +P(f(X_1, \ldots, X_n))$ une transition, est similaire.
- Finalement, si $C \vee +P(f(X_1, \ldots, X_n))$ et $C' \vee -P(f(Y_1, \ldots, Y_n))$ sont deux transitions, on remarque d'abord que le mgu servant à calculer le résolvant binaire est une projection. On a maintenant deux cas. Si C ou C' est une transition, c'est-à-dire contient au moins un atome $\pm Q(t)$, avec t plat, alors $C \vee C'$ est une transition, donc le résolvant binaire $(C \vee C')\sigma$ est encore une transition. Sinon, C et C' sont des disjonctions d'ϵ-blocs, donc $(C \vee C')\sigma$ est encore une disjonction d'ϵ-blocs.

On en déduit que tout résolvant ordonné de deux clauses automatiques est une disjonction de clauses automatiques ne partageant aucune variable. Donc la résolution ordonnée, avec splitting appliqué le plus tôt possible, produit des tableaux dont les branches ne contiennent que des clauses automatiques.

Comptons le nombre de clauses automatiques. Soit p le nombre de symboles de prédicats, f le nombre de symboles de fonctions dans la signature Σ, a l'arité maximale des symboles de fonctions de Σ. D'abord, il y a au plus $2^{2p} = 4^p$ ϵ-blocs, à renommage près. (Si l'on a deux ϵ-blocs $B(X)$ et $B(Y)$ qui ne diffèrent que par renommage, chacun subsume l'autre linéairement, et l'on peut donc supprimer n'importe lequel d'entre eux.) Pour compter les transitions, à renommage près, on peut supposer que les variables libres des transitions sont X_1, …, X_n ($0 \leq n \leq a$) dans cet ordre. Il y a n^n listes de n variables prises parmi ces dernières (on pose $0^0 = 1$), donc fn^n termes plats, donc $2pfn^n$ littéraux $\pm P(t)$, avec t plat, donc au plus $2^{2pfn^n} = 4^{pfn^n} \leq 4^{pfa^a}$ possibilités pour la partie des transitions qui porte sur des termes plats ; on a ensuite au plus 4^p ϵ-blocs $B_1(X_1)$, 4^p ϵ-blocs $B_2(X_2)$, etc. Donc on a au plus $4^{pfa^a}.4^{pa} = 4^{pfa^a+pa}$ transitions. Il

existe donc au plus $4^{pfa^a+pa} + 4^p$ clauses automatiques.

Cette borne est une double exponentielle en (un polynôme de) la taille de l'ensemble de clauses automatiques initial, et une simple exponentielle si l'arité maximale a est une constante. La satisfiabilité revenant à tester s'il existe une branche du tableau obtenu par résolution ordonnée et splitting qui est saturée et ne contient pas la clause vide est donc dans N2EXPTIME, et dans NEXPTIME si a est une constante.

On obtient la borne D2EXPTIME au lieu de N2EXPTIME en remarquant que, comme le splitting n'opère en fait que sur des disjonctions d'un nombre au plus a d'ϵ-blocs, et qu'il n'y a qu'un nombre exponentiel d'ϵ-blocs (4^p), le problème général se résout en temps doublement exponentiel sur une machine qui ne fait qu'un nombre exponentiel de choix non-déterministes (de splittings) ; donc au total dans D2EXPTIME. Un autre argument est d'utiliser le splitting sans splitting, et de constater qu'il n'y a qu'un nombre au plus doublement exponentiel de clauses obtenues avec cette règle (exercice). □

D2EXPTIME est une complexité très haute. C'est en grande partie un artefact du format très général des transitions, qui autorise notamment :

– à renommer les symboles de fonction en tête des termes :

$$P(f(X_1, \ldots, X_n)) \Leftarrow Q(g(X_1, \ldots, X_n))$$

permet de changer un g en un f au sommet d'un terme reconnu en Q pour en faire un reconnu en f ;

– à réordonner les sous-termes, par exemple :

$$P(f(X, Y)) \Leftarrow Q(g(Y, X))$$

permet de changer un g en f tout en échangeant leurs deux arguments ;

– et tout ceci tout en dupliquant des sous-termes ou en testant s'ils sont égaux, entre autres.

La très haute complexité du problème de la satisfiabilité des ensembles de clauses automatiques est aussi un artefact du fait que nous avons généralisé les automates de sorte à inclure des clauses qui ne sont pas de Horn. D'autre part, il est relativement improbable que le problème soit complet pour D2EXPTIME, mais la véritable complexité de ce problème est ouverte.

Un cas important est celui où, dans les transitions, tous les termes plats t_i sont de la forme $f_i(Y_1, \ldots, Y_m)$, avec la *même* liste de variables en argument Y_1, …, Y_m. C'était notamment le cas des automates d'arbres bidirectionnels alternants étendus, et même avec contraintes d'égalité entre frères. Appelons de telles transitions des transitions *uniformes*, et appelons clauses *automatiques uniformes* les ϵ-blocs et les transitions uniformes.

La résolution ordonnée avec splitting appliqué le plus tôt possible, partant de clauses automatiques uniformes, ne produit que des clauses automatiques uniformes, et il y a cette fois-ci au plus $4^{pf+pa} + 4^p$ clauses automatiques uniformes, d'où :

Proposition 17 *La satisfiabilité d'ensembles de clauses automatiques uniformes est décidable en NEXPTIME.*

Il se trouve que le problème est en fait NEXPTIME-complet, car les clauses automatiques uniformes incluent en particulier toutes les contraintes ensemblistes, dont la satisfiabilité est déjà NEXPTIME-complète (voir section 6.7).

Note : La satisfiabilité d'ensembles de clauses automatiques, même non uniformes, est en réalité décidable en NEXPTIME [SV05].

6.5 Le cas des automates généralisés de Horn

Dans le cas de Horn, qui est après tout notre première motivation ici, on peut ramener la complexité encore un peu plus bas, à condition de travailler un peu plus dur. En fait, nous allons montrer que le problème est dans DEXPTIME. (Les arguments présentés dans le reste de cette section, en particulier, demandent pour être maîtrisés à avoir bien compris la preuve du théorème 16.)

On pourrait penser qu'il suffit d'effectuer de la résolution ordonnée, et qu'aucun cas de splitting ne se présentera, mais c'est faux. Par exemple, l'inférence suivante par résolution produit une clause qui splitte :

$$\frac{Q_1(X) \Leftarrow P(f(X,Y,Z)), Q_2(Y), Q_3(Z) \qquad P(f(X,Y,Z)) \Leftarrow P_1(X), P_2(Y), P_3(Z)}{Q_1(X) \Leftarrow P_1(X), P_2(Y), Q_2(Y), P_3(Z), Q_3(Z)}$$

(On a aussi vu un exemple en section 5.2.) On peut cependant remarquer que les clauses obtenues par splitting sont, d'une part, une clause définie $Q_1(X) \Leftarrow P_1(X)$, et d'autre part deux clauses qui sont des requêtes conjonctives, $\bot \Leftarrow P_2(Y), Q_2(Y)$ et $\bot \Leftarrow P_3(Z), Q_3(Z)$. En d'autres termes, la conclusion de l'inférence ci-dessus peut se lire "$Q_1(X) \Leftarrow P_1(X)$ à condition que $P_2 \cap Q_2$ et $P_3 \cap Q_3$ soient non vides".

Ceci motive l'idée suivante. Utilisons le splitting sans splitting (section 5.3), ce qui abrégera la clause ci-dessus en :

$$\begin{gathered} Q_1(X) \Leftarrow P_1(X), q(*), q'(*) \\ q(*) \Leftarrow P_2(Y), Q_2(Y) \\ q'(*) \Leftarrow P_3(Z), Q_3(Z) \end{gathered}$$

On souhaite maintenant considérer la première clause ci-dessus comme étant la clause $Q_1(X) \Leftarrow P_1(X)$, mais gelée tant que $q(*)$ et $q'(*)$ n'ont pas été dérivés, c'est-à-dire tant qu'on n'a pas montré que $P_2 \cap Q_2$ et $P_3 \cap Q_3$ étaient non vides.

Pour ceci, il suffit de sélectionner $q(*)$ et $q'(*)$ dans cette clause, ce qui forcera la clause à n'être utilisée qu'avec des clauses de la forme $q(*) \Leftarrow \ldots$ et $q'(*) \Leftarrow \ldots$. Ensuite, arrangeons-nous pour qu'au moins un littéral soit sélectionné dans ces deux derniers types de clauses, dès qu'elles contiennent au moins un littéral négatif.

Précisément, reprenons le formalisme de la section 5.3, soit $\mathcal{P}$ un ensemble de symboles de prédicats, et $\mathcal{Q}$ l'ensemble des $\ulcorner C \urcorner$, pour toute clause C formée à partir de symboles de prédicats dans $\mathcal{P}$. Pour toute clause C (utilisant possiblement des symboles de Q), définissons $\mathrm{sel}\,(C)$ comme suit :

- Soit $\mathcal{Q}^-(C)$ l'ensemble des littéraux de la forme $-q(t)$ de C tels que $q \in \mathcal{Q}$; si $\mathcal{Q}^-(C) \neq \emptyset$, poser $\mathrm{sel}\,(C) = \mathcal{Q}^-(C)$.
- Sinon, soit $Max(C)$ l'ensemble des littéraux maximaux de C pour l'ordre $\succ_1$, alors $\mathrm{sel}\,(C)$ est le sous-ensemble des littéraux négatifs de $Max(C)$.

Rappelons que $P(t) \succ_1 P'(t')$ si et seulement si $t \rhd t'$, c'est-à-dire si et seulement si t' est un sous-terme strict de t.

Utilisons la résolution ordonnée (pour $\succ_1$) avec fonction de sélection sel , élimination de tautologies et de clauses subsumées linéairement, et la stratégie suivante de splitting sans splitting. Pour toute clause C de Horn, écrivons C sous la forme $C_{\mathcal{P}} \vee C_{\mathcal{Q}}$, où $C_{\mathcal{P}}$ est formée à partir de symboles de prédicats dans $\mathcal{P}$ uniquement, et $C_{\mathcal{Q}}$ est formée à partir de symboles de prédicats dans $\mathcal{Q}$ uniquement ; soit $C_1, \ldots, C_k$ une décomposition en blocs de $C_{\mathcal{P}}$. Il existe au plus une sous-clause C_i qui est définie ; on peut donc supposer que C_2, …, C_k sont des clauses négatives. Notre stratégie de splitting sans splitting consiste alors à remplacer C par

$$C_{\mathcal{Q}} \vee C_1 \vee -\ulcorner C_2 \urcorner(*) \vee \ldots \vee -\ulcorner C_k \urcorner(*)$$
$$+\ulcorner C_2 \urcorner(*) \vee C_2$$
$$\ldots$$
$$+\ulcorner C_k \urcorner(*) \vee C_k$$

si $k \geq 2$. (On notera que toutes ces clauses sont de Horn ; toutes sauf possiblement la première sont en fait des clauses définies.) D'autre part, on applique cette décomposition au plus tôt. Par les résultats de la section 5.3, c'est une stratégie complète.

Soit $\mathcal{Q}_0$ le sous-ensemble de $\mathcal{Q}$ formé des $\ulcorner C \urcorner$, où C est un ϵ-bloc négatif formé à partir de symboles de prédicats dans $\mathcal{P}$. On notera qu'il y a au plus 2^p éléments dans $\mathcal{Q}_0$, où p est le cardinal de $\mathcal{P}$. (Ceci garantit en particulier que la solution 1 au problème mentionné dans la note suivant le théorème 12 s'applique.)

Si S est un ensemble de clauses automatiques, formées à partir de symboles de prédicats dans $\mathcal{P}$ uniquement, toute clause obtenue par la stratégie ci-dessus est de Horn, et :

1. un ϵ-bloc formé à partir de symboles de prédicats dans $\mathcal{P}$, possiblement en disjonction avec un $+q(*)$, $q \in \mathcal{Q}_0$;
2. ou une transition formée à partir de symboles de prédicats dans $\mathcal{P}$, possiblement en disjonction avec un $+q(*)$, $q \in \mathcal{Q}_0$;
3. ou une clause $C \vee -q_1(*) \vee \ldots \vee -q_m(*)$, où C est un ϵ-bloc formé à partir de symboles de prédicats dans $\mathcal{P}$, $q_1, \ldots, q_m \in \mathcal{Q}_0$ et $1 \leq m \leq a$, où a est l'arité maximale des symboles de fonctions, possiblement en disjonction avec un $+q(*)$, $q \in \mathcal{Q}_0$.

Noter en effet que tout résolvant ordonné avec sélection des clauses de type 1 ou 2 fournit des clauses de type 1, 2, ou bien une disjonction d'ϵ-blocs (possiblement en disjonction avec un $+q(*)$, $q \in \mathcal{Q}_0$), qui fournissent des clauses de type 3 par splitting sans splitting. L'argument est similaire à celui de la preuve du théorème 16. Ensuite, la seule façon de résoudre une clause de type 3 est de résoudre sur un des littéraux $-q_i(*)$, $1 \leq i \leq m$, qui est sélectionné. On doit donc résoudre avec une clause C contenant $+q_i(*)$. Mais $+q_i(*)$ n'est pas maximal dans C si C est de type 1 ou 2, sauf si C est réduite au seul littéral $+q_i(*)$; et si C est de type 3, l'étape de résolution est impossible car alors $\mathrm{sel}\ (C) \neq \emptyset$. Donc on ne peut résoudre une clause de type 3 qu'avec un fait $+q_i(*)$, ce qui fournit une clause de type 3 ou 1.

Il y a au plus $4^p(2^p+1)$ clauses de type 1 : 4^p possibilités d'ϵ-blocs, 2^p possibilités pour $+q(*)$ si présent. (En fait, on en a au plus $2^p(2^p+p+1)$, en tenant compte du fait qu'il s'agit de clauses

de Horn, mais nous ne serons pas aussi fins dans la suite, pour éviter de rendre le comptage trop compliqué). Il y a au plus $4^{pfa^a+pa}.(2^p+1)$ clauses de type 2 ($(4^{pf+pa}+4^p).(2^p+1)$ dans le cas uniforme). Il y a au plus $4^p(2^p+1).\sum_{i=1}^{a} \mathrm{C}_{2^p}^i$ clauses de type 3. Or $\sum_{i=1}^{a} \mathrm{C}_{2^p}^i \leq \sum_{i=1}^{a} 2^{pi} = 2^p(2^{pa}-1)/(2^p-1)$. Il y a donc au plus $4^p(2^p+1)2^p(2^{pa}-1)/(2^p-1)$ clauses de type 3, une quantité simplement exponentielle. On en déduit :

Théorème 18 *La satisfiabilité d'ensembles de clauses automatiques de Horn est décidable en DEXPTIME si l'arité des symboles de fonction est bornée par une constante.*

La satisfiabilité d'ensembles de clauses automatiques uniformes de Horn est décidable en DEXPTIME.

Le problème est en fait DEXPTIME-complet, car la vacuité de l'intersection d'automates d'arbres est déjà DEXPTIME-complète (section 6.2).

6.6 Réduction des automates généralisés de Horn aux automates d'arbres

Lorsque S est satisfiable, la stratégie de la section 6.5 ne dérive pas la clause vide, mais l'on peut utiliser l'ensemble de clauses obtenu lorsque ce processus termine pour obtenir des informations supplémentaires. Notamment, nous obtiendrons la description d'un modèle de S sous forme d'un automate d'arbres (possiblement avec contraintes d'égalité entre frères). La technique est essentiellement la même que celle utilisée dans le théorème 15.

Proposition 19 *Pour tout ensemble S satisfiable de clauses automatiques de Horn, il existe un ensemble $norm_1(S)$ de clauses de la forme*

$$(a)\ P(f(X_1,\ldots,X_n)) \Leftarrow B_1(X_1),\ldots,B_n(X_n) \quad \textit{ou} \quad (b)\ P(X)$$

où B_1, …, B_n sont des blocs négatifs, et tel que $L_P(S) = L_P(norm_1(S))$ pour tout $P \in \mathcal{P}$. De plus, $norm_1(S)$ est calculable à partir de S en temps doublement exponentiel, et simplement exponentiel si l'arité des fonctions est bornée par une constante.

Démonstration. Appliquons la stratégie de résolution ordonnée avec sélection, splitting sans splitting et élimination de clauses redondantes de la section 6.5, obtenant un nouvel ensemble de clauses saturé S' en temps doublement exponentiel, simplement exponentiel si l'arité des fonctions est bornée. (Ce faisant, nous supposons sans perte de généralité que $\mathcal{P}$ contient P et tous les symboles de prédicats apparaissant dans S.) Pour simplifier la preuve, adaptons la stratégie de splitting sans splitting de sorte à ne splitter que des clauses non closes ; on vérifiera que ceci calcule le même ensemble S', avec les mêmes étapes de calcul. (On pourrait ne rien changer à cette stratégie, mais l'argument serait plus compliqué.)

Considérons n'importe quel ensemble S_1 de clauses de la forme

$$(*) \qquad \bot \Leftarrow P_1(t_1),\ldots,P_n(t_n)$$

où $n \geq 1$, $P_1,\ldots,P_n \in \mathcal{P}$, et t_1, …, t_n sont des termes clos. (Donc on ne splittera pas une telle clause.) Considérons une étape de la stratégie ci-dessus $S' \cup S_1 \rightarrow S''$, non triviale au sens où

$S'' \neq S' \cup S_1$. Cette étape ne peut pas être une étape de résolution entre clauses de S', car S' est saturé. Ce ne peut pas être une étape de résolution entre clauses de S_1, car S_1 ne contient que des clauses négatives. Finalement, considérons une étape de résolution ordonnée avec sélection entre une clause $(*)$ et une clause C de S'. D'abord, C ne peut pas contenir de littéral $+q(*)$, $q \in \mathcal{Q}_0$, car $q(*)$ ne s'unifie pas avec $P_i(t_i)$. Ensuite, $\mathrm{sel}\ (C) = \emptyset$, donc C ne peut pas être de type 1, sauf si elle est exactement la clause $P_i(X)$, donc de la forme (b). C ne peut être de type 2 que si C est une transition ne contenant pas de littéral de la forme $-P(f(X_1, \ldots, X_n))$, qui serait maximal donc sélectionné ; donc C est de la forme (a). Enfin C ne peut pas être de type 3, car sinon $\mathrm{sel}\ (C) \neq 0$.

De plus, chaque étape de résolution ci-dessus fabrique soit la clause vide soit une clause de la forme $(*)$, de nouveau. On en conclut que dans toute dérivation $S' \cup S_1 = S'_1 \rightarrow S'_2 \rightarrow \ldots \rightarrow S'_i \rightarrow \ldots$, où l'on peut supposer sans perte de généralité que chaque étape $S'_i \rightarrow S'_{i+1}$ est non triviale, les seules clauses de S' qui sont utilisées sont de forme (a) ou (b). On pose donc $norm_1(S) = \{C \in S' | C$ de forme (a) ou $(b)\}$, il s'ensuit que pour tout terme clos t, si $t \in L_P(S)$, alors $S \cup \{\bot \Leftarrow P(t)\}$ a une dérivation de la clause vide comme ci-dessus, donc aussi $norm_1(S) \cup \{\bot \Leftarrow P(t)\}$, donc $t \in L_P(norm_1(S))$. La réciproque $t \in L_P(norm_1(S)) \Rightarrow t \in L_P(S)$ est évidente. □

On peut maintenant remplacer toute clause de la forme (b), soit $P(X)$, par l'ensemble fini de clauses

$$P(f(X_1, \ldots, X_n)) \Leftarrow P(X_1), \ldots, P(X_n)$$

lorsque f parcourt la signature Σ. Ceci ne change le langage d'aucun prédicat de $norm_1(S)$. On peut ensuite transformer en temps polynomial toute clause (a) en les clauses

$$\begin{array}{rcl} P(f(X_1, \ldots, X_n)) & \Leftarrow & Q_1(X_1), \ldots, Q_n(X_n) \\ Q_1(X) & \Leftarrow & B_1(X) \\ & \ldots & \\ Q_n(X) & \Leftarrow & B_n(X) \end{array}$$

en introduisant des symboles de prédicats Q_1, …, Q_n frais.

En utilisant une démonstration similaire à celle du théorème 15, on en déduit :

Théorème 20 *Pour tout ensemble S satisfiable de clauses automatiques de Horn, il existe un automate d'arbres avec contraintes d'égalité entre frères $norm(S)$ tel que $L_P(S) = L_P(norm(S))$ pour tout $P \in \mathcal{P}$. De plus, $norm(S)$ est calculable à partir de S en temps doublement exponentiel, et simplement exponentiel si l'arité des fonctions est bornée par une constante.*

Démonstration. On obtient $norm(S)$ à partir de $norm_1(S)$ en éliminant les clauses intersection comme dans le théorème 15. De façon remarquable, et même si $norm(S)$ est de taille doublemenent exponentielle (simplement exponentielle si l'arité des fonctions est bornée par une constante) en celle de S en général, les nombres de symboles de prédicats p, l'arité maximale a des symboles de fonctions, le nombre de symboles de fonction f sont les mêmes dans S et dans $norm_1(S)$. Or le calcul de $norm(S)$ est exponentiel en p, f, a, d'où le résultat. □

Les mêmes arguments fournissent les deux autres résultats :

Théorème 21 *Pour tout ensemble S satisfiable de clauses automatiques uniformes de Horn, il existe un automate d'arbres avec contraintes d'égalité entre frères $norm(S)$ tel que $L_P(S) = L_P(norm(S))$ pour tout $P \in \mathcal{P}$. De plus, $norm(S)$ est calculable à partir de S en temps exponentiel.*

Théorème 22 (Réduction) *Pour tout automate d'arbres bidirectionnel alternant étendu S, on peut calculer en temps exponentiel un automate d'arbres $norm(S)$ tel que $L_P(S) = L_P(norm(S))$ pour tout $P \in \mathcal{P}$.*

Démonstration. Il suffit de remarquer que l'on peut restreindre toutes les clauses contenant un littéral de la forme $f(X_1, \ldots, X_n)$ de sorte que X_1, ..., X_n soient distinctes deux à deux. (Ceci ne serait pas vrai si l'on avait à utiliser la factorisation ; heureusement, nos clauses sont de Horn). □

En particulier, tout langage reconnu à tout état de n'importe quel automate d'arbres bidirectionnel alternant étendu est régulier.

Remarque 7 *On peut aussi utiliser $norm(S)$ pour extraire un modèle de S. En effet, tout automate d'arbres S_0 avec contraintes d'égalité entre frères a un modèle de Tarski I défini comme suit. Le domaine D est l'ensemble des parties de $\mathcal{P}$, où $\mathcal{P}$ est l'ensemble des symboles de prédicats de S_0. Ensuite, pour tout symbole n-aire I_f, on pose $I_f(v_1, \ldots, v_n)$ l'ensemble des prédicats P tels qu'il existe une clause de la forme*

$$P(f(X_1, \ldots, X_n)) \Leftarrow P_1(X_1), \ldots, P_n(X_n)$$

où $P_1 \in v_1$, ..., $P_n \in v_n$. Ce modèle est tel que $v \in I_P$ si et seulement si $P \in v$. On notera que ce modèle est fini, et de taille exponentielle en la taille de l'automate d'arbres avec contraintes d'égalité entre frères $S_0 = norm(S)$, donc doublement exponentielle en la taille de S.

6.7 Contraintes ensemblistes

Les contraintes ensemblistes sont un formalisme à première vue très différent, mais qui ne l'est pas tant. Elles ont été inventées par John Reynolds [Rey69], et ont été plus tard utilisées en inférence automatique de types pour programmes Prolog, en inférence de forme de données en ML, et en vérification de protocoles cryptographiques.

Informellement, si l'on considère les équations suivantes,

$$Nat = \mathtt{S}(Nat) \cup \mathtt{0} \quad Even = \mathtt{S}(\mathtt{S}(Even)) \cup \mathtt{0} \quad ListEven = \mathtt{cons}(Even, ListEven) \cup \mathtt{nil}$$

intuitivement, on a spécifié que Nat devait représenter l'ensemble des entiers naturels (écrits en unaire), $Even$ l'ensemble des entiers pairs, $ListEven$ l'ensemble des listes finies d'entiers pairs. La notation $\mathtt{S}(Even)$ dénote l'ensemble de tous les successeurs d'entiers pairs.

Plus généralement, on peut souhaiter écrire des inclusions entre expressions. (C'est plus général car on peut toujours réécrire une égalité $e_1 = e_2$ en deux inclusions $e_1 \subseteq e_2$ et $e_2 \subseteq e_1$.)

Par exemple, si I est l'ensemble des messages déductibles par un intrus de Dolev-Yao à partir d'un ensemble de messages E, on doit avoir :

$$E \subseteq I \quad c(I,I) \subseteq I \quad c(1,I) \cap I \subseteq c(I,1)$$

où c est l'opérateur de chiffrement. L'inclusion $E \subseteq I$ exprime que tout message de E est déductible par l'intrus, l'inclusion $c(I,I) \subseteq I$ exprime que tout message de la forme $c(M,N)$ avec $M \in I$, $N \in I$, est dans I (l'intrus sait chiffrer). Finalement, 1 dénote l'ensemble de tous les termes ; l'inclusion $c(1,I) \cap I \subseteq c(I,1)$ exprime alors que tout message de la forme $c(M,N)$ avec N dans I, et qui est dans I, est dans $c(I,1)$, c'est-à-dire que M est dans I. Autrement dit, l'intrus sait déchiffrer.

Formalisons ceci. Les *expressions ensemblistes* sont définies par la grammaire :

$$e ::= \xi \mid 0 \mid 1 \mid e \cap e \mid e \cup e \mid \complement e \mid f(e_1, \ldots, e_n)$$

où f est un symbole de fonction quelconque (d'arité n), et ξ, η, ..., sont des *variables ensemblistes*. L'ensemble des variables ensemblistes est supposé infini dénombrable. Dans les expressions de la forme $f_i^{-1}(e)$ (les *projections*), on demande $1 \leq i \leq n$.

La sémantique $[\![e]\!]\,\rho$ d'une expression ensembliste e dans un environnement ρ, qui associe à chaque variable ensembliste un ensemble de termes clos, est la suivante, où $\mathcal{T}$ est l'ensemble de tous les termes clos :

$$\begin{array}{ccc}
[\![\xi]\!]\,\rho = \rho(\xi) & [\![0]\!]\,\rho = \emptyset & [\![1]\!]\,\rho = \mathcal{T} \\
[\![e_1 \cap e_2]\!]\,\rho = [\![e_1]\!]\,\rho \cap [\![e_2]\!]\,\rho & [\![e_1 \cup e_2]\!]\,\rho = [\![e_1]\!]\,\rho \cup [\![e_2]\!]\,\rho & [\![\complement e]\!]\,\rho = \mathcal{T} \setminus [\![e]\!]\,\rho \\
\multicolumn{3}{c}{[\![f(e_1, \ldots, e_n)]\!]\,\rho = \{f(t_1, \ldots, t_n) | t_1 \in [\![e_1]\!]\,\rho, \ldots, t_n \in [\![e_n]\!]\,\rho\}}
\end{array}$$

Une *contrainte ensembliste* est une inclusion de la forme $e_1 \subseteq e_2$. Un *système de contraintes ensemblistes* K est un ensemble fini de contraintes ensemblistes. On a $\rho \models e_1 \subseteq e_2$ si et seulement si $[\![e_1]\!]\,\rho \subseteq [\![e_2]\!]\,\rho$, et $\rho \models K$ si et seulement si $\rho \models e_1 \subseteq e_2$ pour toute contrainte $e_1 \subseteq e_2$ dans K. On dit alors que ρ *satisfait* K. K est *satisfiable* si et seulement s'il existe un environnement qui satisfait K.

On peut toujours ramener la satisfiabilité de systèmes de contraintes ensemblistes à des systèmes de *contraintes élémentaires*, qui sont les contraintes listées dans la première colonne de la table ci-dessous. Il suffit en effet d'introduire de nouvelles variables ensemblistes pour nommer chaque sous-expression, et de simplifier. Par exemple, on peut remplacer $f(g(\xi)) \subseteq \eta$ par $g(\xi) \subseteq \xi'$, $f(\xi') \subseteq \eta$, ou bien on peut remplacer $\xi \subseteq \eta \cap \zeta$ par $\xi \subseteq \eta$ et $\xi \subseteq \zeta$. Cette transformation prend un temps polynomial.

On peut ensuite traduire chaque contrainte ensembliste élémentaire $e_1 \subseteq e_2$ en une conjonction S de clauses (montrée en vis-à-vis), telle que $\rho \models e_1 \subseteq e_2$ si et seulement si $\rho \models S$. Noter ici que l'environnement ρ n'est rien d'autre qu'une interprétation de Herbrand I, celle telle que $I_\xi = \rho(\xi)$ pour tout symbole de prédicat ξ. Cette traduction prend encore un temps polynomial.

Contrainte élémentaire	Clauses automatiques
$\xi \subseteq 0$	$-\xi(X)$
$1 \subseteq \xi$	$+\xi(X)$
$\xi \subseteq \eta$	$-\xi(X) \vee +\eta(X)$
$\xi \subseteq \eta \cup \zeta$	$-\xi(X) \vee +\eta(X) \vee +\zeta(X)$
$\xi \cap \eta \subseteq \zeta$	$-\xi(X) \vee -\eta(X) \vee +\zeta(X)$
$\xi \subseteq \complement\eta$	$-\xi(X) \vee -\eta(X)$
$\complement\xi \subseteq \eta$	$+\xi(X) \vee +\eta(X)$
$\xi \subseteq f(\xi_1, \ldots, \xi_n)$	$\begin{cases} -\xi(f(X_1, \ldots, X_n)) \vee +\xi_1(X_1) \\ \ldots \\ -\xi(f(X_1, \ldots, X_n)) \vee +\xi_n(X_n) \\ -\xi(g(X_1, \ldots, X_m)) \quad \text{(pour tout } g \neq f\text{)} \end{cases}$
$f(\xi_1, \ldots, \xi_n) \subseteq \xi$	$\bigvee_{i=1}^n -\xi_i(X_i) \vee +\xi(f(X_1, \ldots, X_n))$

Au total, on a réduit, en temps polynomial, le problème de la satisfiabilité de contraintes ensemblistes à celui de la satisfiabilité de clauses automatiques, qui plus est uniformes. Ceci est donc décidable en NEXPTIME, par la proposition 17. En fait, le problème est NEXPTIME-complet.

L'exemple de contrainte ensembliste modélisant la connaissance de l'intrus de Dolev-Yao, que nous avons vu plus haut, se traduit en :

$$\begin{array}{rrrr} I(X) \Leftarrow E(X) & I(c(X,Y)) \Leftarrow I(X), I(Y) & & \\ \xi(X) & \eta(c(X,Y)) \Leftarrow \xi(X), I(Y) & \zeta(X) \Leftarrow \eta(X), I(X) & \\ I(X) \Leftarrow \zeta(c(X,Y)) & \omega(Y) \Leftarrow \zeta(c(X,Y)) & \bot \Leftarrow \zeta(g(X_1, \ldots, X_m)) & (g \neq c) \end{array}$$

où l'on a d'abord traduit la contrainte ensembliste $c(1, I) \cap I \subseteq c(I, 1)$ en les contraintes élémentaires

$$1 \subseteq \xi \qquad c(\xi, I) \subseteq \eta \qquad \eta \cap I \subseteq \zeta \qquad \zeta \subseteq c(I, \omega)$$

On note que ces clauses sont des clauses de Horn, et en fait des clauses d'automates d'arbres bidirectionnels alternants.

Ceci motive la définition de la classe des *contraintes ensemblistes définies*, qui sont les contraites ensemblistes $e_1 \subseteq e_2$ où le symbole $\complement$ n'apparaît pas ni dans e_1 ni dans e_2, et où le symbole $\cup$ n'apparaît pas dans e_2. La traduction présentée ci-dessus fournit alors un ensemble de clauses automatiques uniformes *de Horn*.

Par le théorème 18, la satisfiabilité de systèmes de contraintes ensemblistes définies est DEXPTIME-complète.

Par le théorème 22, la plus petite solution d'un système de contraintes ensemblistes envoie chaque variable X vers un langage régulier, calculable en temps exponentiel.

7 Classes décidables de clauses du premier ordre

7.1 Indécidabilité du cas général

L'insatisfiabilité d'ensembles de clauses du premier ordre est un problème indécidable. Il y a un certain nombre de façons de le démontrer. En voici deux :

- Le problème de correspondance de Post (PCP) est indécidable. Ce problème est le suivant :
  ENTRÉE : un nombre fini de paires de mots (u_i, v_i), $1 \leq i \leq p$, sur un alphabet $\{a_1, \ldots, a_n\}$;
  QUESTION : existe-t-il une suite d'entiers $i_1, \ldots, i_k$ entre 1 et p (avec répétitions possibles) telle que $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$ et ce mot est non vide ?
  On peut le coder comme suit. D'abord, pour tout mot u et tout terme t, définissons le terme $u(t)$ par : si u est le mot vide alors $u(t) = t$; si u est le mot va, où a est une lettre, alors $u(t) = a(v(t))$. Il ne reste plus qu'à écrire les clauses :

$$
\begin{array}{rcll}
 & & P(\epsilon, \epsilon) & (12)\\
P(u_i(x), v_i(y)) & \Leftarrow & P(x, y) \qquad (1 \leq i \leq p) & (13)\\
 & & Q(a_j(\epsilon)) \qquad (1 \leq j \leq n) & (14)\\
Q(a_j(x)) & \Leftarrow & Q(x) \qquad (1 \leq j \leq n) & (15)\\
\bot & \Leftarrow & P(x, x), Q(x) & (16)
\end{array}
$$

  Le modèle minimal des quatre premières clauses est constitué de l'ensemble des $P(u_{i_1} u_{i_2} \ldots u_{i_k}(\epsilon), v_{i_1} v_{i_2} \ldots v_{i_k}(\epsilon))$, et des $Q(w(\epsilon))$ pour tout mot w non vide. La dernière clause permet de déduire faux si et seulement s'il existe un mot non vide $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$. Si l'on savait décider un tel ensemble de clauses, on saurait donc décider du PCP, ce qui est impossible.
- Le problème de l'accessibilité dans les machines à deux compteurs est indécidable. Il s'agit de :
  ENTRÉE : un ensemble fini d'*états* Q, un état *initial* $q_0 \in Q$, un état *final* $q_f \in Q$, un ensemble fini de *transitions*, qui sont des triplets $q \xrightarrow{a} q'$, où $q, q' \in Q$ et a est une expression de la forme $\mathtt{R}_i + +$, $\mathtt{R}_i - -$ ou $\mathtt{R}_i{=}{=}0$, $1 \leq i \leq 2$.
  QUESTION : existe-t-il $R_1, R_2 \in \mathbb{N}$ tels que $(q_0, 0, 0) \longrightarrow^* (q_f, R_1, R_2)$?
  Ici, on définit $(q, R_1, R_2) \longrightarrow (q', R_1', R_2')$ si et seulement s'il existe une transition $q \xrightarrow{a} q'$ telle que :
  – a est de la forme $\mathtt{R}_i + +$, $R_i' = R_i + 1$ et $R_{\overline{i}}' = R_{\overline{i}}$ (où $\overline{1} = 2$, $\overline{2} = 1$) ;
  – ou a est de la forme $\mathtt{R}_i - -$, $R_i \geq 1$, $R_i' = R_i - 1$ et $R_{\overline{i}}' = R_{\overline{i}}$;
  – ou a est de la forme $\mathtt{R}_i == 0$, $R_i = 0$, $R_i' = R_i$ et $R_{\overline{i}}' = R_{\overline{i}}$.
  On peut coder ceci en clauses de Horn de nouveau, où les configurations (q, R_1, R_2) sont codées sous forme d'atomes $q(\hat{R}_1, \hat{R}_2)$, où le codage des entiers est $\hat{0} = \mathtt{0}$, $\widehat{n+1} = \mathtt{S}(\hat{n})$:

$$
\begin{array}{rcll}
 & & q_0(\mathtt{0}, \mathtt{0}) & (17)\\
q'(\mathtt{S}(x), y) & \Leftarrow & q(x, y) \qquad \text{(pour toute transition } q \xrightarrow{\mathtt{R}_1 + +} q') & (18)\\
q'(x, \mathtt{S}(y)) & \Leftarrow & q(x, y) \qquad \text{(pour toute transition } q \xrightarrow{\mathtt{R}_2 + +} q') & (19)
\end{array}
$$

$$
\begin{array}{rcll}
q'(x,y) & \Leftarrow & q(\mathtt{S}(x),y) & \text{(pour toute transition } q \xrightarrow{\mathtt{R}_1 - -} q') \quad (20)\\
q'(x,y) & \Leftarrow & q(x,\mathtt{S}(y)) & \text{(pour toute transition } q \xrightarrow{\mathtt{R}_2 - -} q') \quad (21)\\
q'(\mathtt{0},y) & \Leftarrow & q(\mathtt{0},y) & \text{(pour toute transition } q \xrightarrow{\mathtt{R}_1 == 0} q') \quad (22)\\
q'(x,\mathtt{0}) & \Leftarrow & q(x,\mathtt{0}) & \text{(pour toute transition } q \xrightarrow{\mathtt{R}_2 == 0} q') \quad (23)\\
\bot & \Leftarrow & q_f(x,y) & \text{(condition d'accessibilité)} \quad (24)
\end{array}
$$

En revanche, il existe certains formats d'ensembles de clauses qui sont décidables : ce sont les *classes décidables* de la logique du premier ordre. La plus importante est certainement la classe monadique, que nous examinerons en section 7.6 et suivantes. Avant d'y arriver, mentionnons-en quelques autres.

7.2 La classe de Herbrand

Il s'agit de la classe des ensembles de clauses *unitaires*. (Une clause unitaire est une clause formée d'un seul littéral.) Elle est décidable par résolution : étant donné un ensemble S de clauses unitaires, soit il existe deux clauses $+A$ et $-A'$ de S qui s'unifient, alors S est insatisfiable, soit il n'en existe pas, la résolution termine sans produire la clause vide, et S est donc satisfiable. On note en particulier que ceci fonctionne en temps polynomial, l'unification du premier ordre étant décidable en temps polynomial (même linéaire).

7.3 La classe de Bernays-Schönfinkel

C'est la classe des clauses dont les littéraux sont de la forme $\pm P(t_1, \ldots, t_n)$, où chaque terme t_i est soit une variable, soit une constante. Il n'y a pas de symbole de fonction d'arité non nulle, l'univers de Herbrand est donc fini. On peut donc décider tout ensemble S de clauses dans cette classe en listant l'ensemble (fini !) $S\downarrow$ de toutes les instances clauses d'éléments de S, et en résolvant le problème de satisfiabilité résultant. La satisfiabilité propositionnelle est NP-complète, et $S\downarrow$ est en général de taille exponentielle en la taille de S ; en fait la satisfiabilité de la classe de Bernays-Schönfinkel est NEXPTIME-complète.

Bizarrement, on ne sait pas décider la classe de Bernays-Schönfinkel par aucun raffinement de la résolution. Le sous-cas de Horn est décidable, dans DEXPTIME, par hyperrésolution positive : on ne produit qu'un nombre au plus exponentielle de clauses unitaires.

La classe de Bernays-Schönfinkel est exactement la classe des mises en formes clausales de formules de la forme $\exists x_1, \ldots, x_m \cdot \forall y_1, \ldots, y_n \cdot F(x_1, \ldots, x_m, y_1, \ldots, y_n)$, où F est sans quantificateur et ne contient que des variables comme termes. On dit traditionnellement que cette classe est le *fragment* $\exists^*\forall^*$ de la logique du premier ordre.

7.4 La classe d'Ackermann

De même, la classe d'Ackermann est le fragment $\exists^*\forall\exists^*$. Les clauses résultantes C contiennent au plus une variable x, et consistent en des littéraux de la forme $\pm P(t_1, \ldots, t_n)$, où chaque t_i est une constante, ou la variable x, ou un terme de la forme $f(x)$ (avec la même variable x).

On peut montrer que la résolution ordonnée termine sur des ensembles de clauses de cette forme. Définissons la *profondeur* $d(t)$ d'un terme ou un atome t par $d(x) = 0$, $d(f(t_1, \ldots, t_n)) = \max_{1 \leq i \leq n} d(t_i) + 1$, et posons $A > B$ si et seulement si $d(A) > d(B)$. On peut alors démontrer que $>$ est un ordre stable, que tout résolvant ordonné entre clauses de la forme ci-dessus est encore de la forme ci-dessus, et que d'autre part il n'y a qu'un nombre fini (exponentiel...) de clauses de cette forme. La classe d'Ackermann est donc décidable, en DEXPTIME.

7.5 Divers

Le fragment $\exists^*\forall^*\exists^*$ est indécidable, mais un certain nombre de sous-fragments sont décidables. Notamment le fragment $\exists^*\forall\forall\exists^*$ (la classe de Gödel), ou le fragment d'ensembles de clauses obtenues à partir du fragment $\exists^*\forall^*\exists^*$ restreints aux clauses à au plus deux littéraux (la classe de Maslov). On pourra consulter [JJ76]. Montrer que ces deux classes sont décidables est difficile.

Le fragment des formules définies à l'aide de deux variables seulement est décidable lui aussi, mais le fragment à trois variables est indécidable. (Il s'agit ici de réutiliser au maximum les mêmes variables dans les diverses quantifications. Par exemple, $\forall x, y \cdot (P(x, y) \Rightarrow \exists x \cdot (P(y, x) \wedge \forall y \cdot \neg P(x, y)))$ est une formule à deux variables x et y.) La décidabilité du fragment à deux variables peut être montrée par résolution ordonnée, en utilisant une forme clausale définitionnelle (voir annexe C).

De même, le *fragment gardé* de de Nivelle est décidable par résolution ordonnée [dN98]. Il s'agit des formules de la forme

$$\begin{array}{rcl} F & ::= & \pm P(x_1, \ldots, x_n) | F \wedge F | F \vee F \\ & | & \forall x_{i_1}, \ldots, x_{i_k} \cdot P(x_1, \ldots, x_n) \Rightarrow F \\ & | & \exists x_{i_1}, \ldots, x_{i_k} \cdot P(x_1, \ldots, x_n) \wedge F \end{array}$$

où, dans les deux dernières lignes, les variables libres de F sont toutes dans $\{x_1, \ldots, x_n\}$, et $1 \leq x_{i_1}, \ldots, x_{i_k} \leq n$.

On pourra consulter [FLHT01] pour un panorama des classes décidables de la logique du premier ordre.

7.6 La classe monadique

La classe monadique est définie traditionnellement comme la classes des formules de la forme

$$\begin{array}{rcl} F & ::= & \pm P(x) | F \wedge F | F \vee F \\ & | & \forall x \cdot F | \exists x \cdot F \end{array}$$

À part la restriction que les arguments des symboles de prédicats sont des variables, comme dans les classes décidables mentionnées plus haut, la restriction essentielle de la classe monadique est que tout symbole de prédicat P est d'*arité* 1.

Montrer que cette classe est décidable est facile. Soient P_1, ..., P_n les symboles de prédicat (unaires). Soit I une structure quelconque, de domaine D : pour chaque i, $1 \leq i \leq n$, I_{P_i}

est un sous-ensemble de D. À chaque valeur v de D, on associe une suite $s(v)$ de symboles $\pm_1, \pm_2, \ldots, \pm_n$, définie par : $\pm_i$ vaut $+$ si $v \in I_{P_i}$, $-$ si $v \notin I_{P_i}$. Il est alors facile de voir que, pour toute formule monadique F construite sur $P_1, \ldots, P_n$, la valeur de vérité de F dans l'environnement ρ ne dépend que des $s(\rho(x))$, x libre dans F. Si F est satisfiable, et a un modèle I de domaine D, elle a alors un modèle *fini*, dont le domaine est l'ensemble des $s(v)$, $v \in D$. Ce modèle est de taille au plus 2^n, où n est le nombre de prédicats distincts de F. Une façon de tester la satisfiabilité de F est donc de deviner un ensemble D de symboles $\pm_1, \pm_2, \ldots, \pm_n$, et de tester si F est vraie sur le modèle I de domaine D et tel que $I_{P_i} = \{\pm_1, \ldots, \pm_{i-1}, +, \pm_{i+1}, \ldots, \pm_n | \forall j \neq i \cdot \pm_j \in \{+, -\}\}$. La classe monadique est donc décidable en NEXPTIME. De plus, cet algorithme est théoriquement optimal : la satisfiabilité de formules monadiques est un problème NP-complet.

Cet algorithme est cependant extrêmement lourd. En fait, il fonctionne en un temps qui est toujours le pire possible. On va voir (superficiellement) qu'on peut transformer ce problème en un problème de satisfiabilité de clauses, que l'on peut résoudre par résolution. Le premier intérêt est que l'on peut espérer trouver des preuves par résolution plus rapidement qu'en temps non-déterministe exponentiel dans les cas pratiques. On verra en section 6.7 que ceci nous permettra de plus une compréhension plus fine de la relation entre classe monadique et automates d'arbres.

La mise en forme clausale (standard) d'une formule F de la classe monadique fournit des clauses qui ont une forme particulière. Si l'on met d'abord F en forme prénexe, on obtient une formule de la forme $Q_1 z_1 \cdot \ldots \cdot Q_n z_n \cdot G$, où G est une formule sans quantificateur, et les Q_i sont des quantificateurs, dans $\{\exists, \forall\}$. Si l'on skolémise maintenant, on obtient la formule $G\sigma$, où σ est la substitution qui à z_i associe z_i si $Q_i = \forall$, et $g_i(\vec{z}_{\forall,<i})$ si $Q_i = \exists$, où les symboles de fonctions g_i sont deux à deux distincts, et $\vec{z}_{\forall,<i}$ dénote la liste de toutes les variables z_j telles que $j < i$ et $Q_j = \forall$. Il ne reste plus qu'à mettre $G\sigma$ en une forme clausale S. Comme F est monadique, les atomes de $G\sigma$ et donc de S sont de la forme $P(z_i)$ (avec $Q_i = \forall$) ou de la forme $P(f(\vec{z}_{\forall,<j}))$. Renumérotons les z_i tels que $Q_i = \forall$, et appelons-les $x_1, \ldots, x_m$ dans cet ordre. Renumérotons aussi les g_i sous la forme $f_1, \ldots, f_p$. Les atomes de S sont donc de la forme $P(x_i)$ ou $P(f_i(x_1, \ldots, x_{n_i}))$, avec $0 \leq n_1 \leq \ldots \leq n_p \leq m$.

Notons $\sqsubseteq$ l'ordre sur les termes plats défini par : $f(X_1, \ldots, X_m) \sqsubseteq g(Y_1, \ldots, Y_n)$ si et seulement s'il existe un entier k, $0 \leq k \leq n$, tel que $\{X_1, \ldots, X_m\} = \{Y_1, \ldots, Y_k\}$. Par exemple, $f(X) \sqsubseteq g(X, Y)$, $a \sqsubseteq f(X)$, $g(X, X) \sqsubseteq g(X, Y)$, mais $g(X, Y) \not\sqsubseteq f(X)$, $g(X, Y) \not\sqsubseteq g(X, X)$, $f(Y) \not\sqsubseteq g(X, Y)$. Il est clair que toute clause de S est formée d'atomes $P(t)$, où t est une variable ou un terme plat.

Toute clause ainsi obtenue a donc une décomposition en blocs d'une des deux formes :

1. Un ϵ-bloc
$$\pm_1 P_1(X) \vee \ldots \vee \pm_n P_n(X)$$
où $n \geq 0$, et tous les prédicats P_i sont appliqués à la même variable X ; on notera souvent $B(X)$ un ϵ-bloc dont la seule variable libre (si elle existe) est X.
2. Ou une *transition étendue* de la forme
$$\bigvee_{i=1}^{m} \pm_i P_i(t_i) \vee B_1(X_1) \vee \ldots \vee B_n(X_n)$$

où $m \geq 1$, les $B_j(x_j)$ sont des ϵ-blocs, les t_i sont des termes plats tels que
– $t_1 \sqsubseteq t_2 \sqsubseteq \ldots \sqsubseteq t_m$;
– et $X_1, \ldots, X_n$ sont toutes libres dans t_m.

En fait, la mise en forme clausale de formules de la classe monadique produit même des clauses un peu plus particulières, au sens où les X_i sont distinctes deux à deux dans les clauses complexes.

Par exemple, considérons la formule monadique

$$\begin{array}{ll} & (\forall x \cdot E(x) \Rightarrow I(x)) \\ \wedge & \left(\forall x, y \cdot \exists z \cdot \left\{ \begin{array}{ll} & I(y) \Rightarrow (I(x) \Leftrightarrow I(z)) \\ \wedge & (P(x) \wedge Q(y) \Rightarrow E(z)) \end{array} \right.\right) \\ \wedge & (\forall x, y \cdot \exists z \cdot (I(x) \wedge I(y)) \Leftrightarrow I(z)) \\ \wedge & \exists u, v \cdot E(v) \wedge P(u) \wedge Q(v) \wedge \neg I(u) \end{array}$$

c'est-à-dire, en respectant la syntaxe que nous avons donnée des formules monadiques :

$$\begin{array}{ll} & (\forall x \cdot -E(x) \vee +I(x)) \\ \wedge & \left(\forall x, y \cdot \exists z \cdot \left\{ \begin{array}{ll} & (-I(x) \vee -I(y) \vee +I(z)) \\ \wedge & (-I(z) \vee -I(y) \vee +I(x)) \\ \wedge & (-P(x) \vee -Q(y) \vee +E(z)) \end{array} \right.\right) \\ \wedge & \left(\forall x, y \cdot \exists z \cdot \left\{ \begin{array}{ll} & (-I(x) \vee -I(y) \vee +I(z)) \\ \wedge & (-I(z) \vee +I(x)) \\ \wedge & (-I(z) \vee +I(y)) \end{array} \right.\right) \\ \wedge & \exists u, v \cdot +E(v) \wedge +P(u) \wedge +Q(v) \wedge -I(u) \end{array}$$

Mise en forme clausale, cette formule devient :

$$\begin{array}{c} -E(X) \vee +I(X) \\ -I(X) \vee -I(Y) \vee +I(c(X,Y)) \\ -I(c(X,Y)) \vee -I(Y) \vee +I(X) \\ -P(X) \vee -Q(Y) \vee +E(c(X,Y)) \\ -I(X) \vee -I(Y) \vee +I(p(X,Y)) \\ -I(p(X,Y)) \vee +I(X) \\ -I(p(X,Y)) \vee +I(Y) \\ +E(k) \\ +P(m) \\ +Q(k) \\ -I(m) \end{array}$$

où m et k sont deux constantes, et c et p sont deux symboles binaires. Noter que, si l'on lit $c(X,Y)$ "X chiffré avec la clé Y" et $p(X,Y)$ "paire X, Y", les 7 premières clauses à l'exception

de la quatrième définissent exactement les capacités déductives I d'un intrus de Dolev-Yao à partir d'un ensemble E de messages dont il dispose.

La première clause, $-E(X) \vee +I(X)$, est un ϵ-bloc. On incite le lecteur à vérifier que toutes les autres sont des clauses complexes.

Le point important est que les clauses obtenues à partir de formules monadiques, de types 1 ou 2 décrits plus haut, ressemblent beaucoup aux clauses automatiques que nous avons vues en section 6.4. On y a relâché la condition selon laquelle tous les t_i devaient avoir le même ensemble de variables libres, au profit de la condition $t_1 \sqsubseteq t_2 \sqsubseteq \ldots \sqsubseteq t_m$, t_m contenant toutes les variables libres de la clause. Les mêmes techniques qu'en section 6 montrent que la résolution ordonnée (pour $\succ_1$) avec sélection, élimination de tautologies et de clauses subsumées linéairement, et splitting, termine en temps non-déterministe simplement exponentiel. (On laisse l'adaptation du théorème 16 en exercice.)

Le cas de la restriction de Horn de la classe monadique se traite lui en temps déterministe exponentiel. (De nouveau, on laisse l'adaptation du théorème 18 en exercice. On doit étendre $\succ_1$ de sorte que $t \sqsubset u$ implique $t \prec_1 u$.) De plus, toute formule monadique de Horn se réduit (en temps exponentiel, adaptation du théorème 20 laissée en exercice) en un automate d'arbres avec contraintes d'égalités entre frères.

8 Approximations d'ensembles de clauses de Horn par automates généralisés

La logique du premier ordre est indécidable (section 7.1), mais ceci ne doit pas nous empêcher de tester si un ensemble de clauses S donnée en entrée est satisfiable ou non. Il y a deux approches typiques de ce problème :

- Utiliser une méthode de démonstration automatique *complète*, comme la résolution et ses raffinements. Ceci a été le sujet des sections 3, 4, et 5, mais souffre du défaut qu'aucune stratégie de démonstration automatique ne terminera sur tous les cas.
- Ou utiliser des techniques d'*approximation*, c'est-à-dire des algorithmes de démonstration $\mathcal{A}$ qui peuvent se tromper. En d'autres termes, $\mathcal{A}(S)$ répond "oui" ou "non", mais on s'autorise à ne pas demander à la fois la correction (si la réponse est "non" alors S est insatisfiable) et la complétude (si S est insatisfiable alors la réponse est "non"). Le gain est que l'on peut espérer une garantie de terminaison.

Dans la suite, nous ne considérerons que des approximations qui terminent. Il y a alors deux types d'approximations :

- les *sous-approximations* $\mathcal{A}$ sont correctes mais pas nécessairement complètes. Si $\mathcal{A}(S)$ retourne "non", alors S est insatisfiable, sinon on ne sait rien.
  En termes de vérification de protocoles cryptographiques par exemple, une sous-approximation est telle que si $\mathcal{A}(S)$ retourne "non", où S décrit un protocole cryptographique, les hypothèses de sécurité et les propriétés à prouver, alors il y a une attaque sur le protocole, de façon sûre. En revanche, si $\mathcal{A}(S)$ retourne "oui", on ne peut pas garantir la sécurité du protocole.

Trouver une sous-approximation est facile. Il suffit de tester S sur des modèles finis par exemple. Autrement dit, il suffit de définir une interprétation de Tarski I quelconque de domaine D fini non vide, et de tester $I \models S$ en temps polynomial en la taille de S et de D. Plus finement, on peut définir le domaine D, fixer les sémantiques I_f des symboles de fonction f, mais pas les sémantiques I_P des symboles de prédicats ; on peut alors tester s'il existe un modèle I de S avec les D et les I_f fixés en énumérant toutes les instances de clauses de S lorsqu'on remplace les termes t par leurs valeurs $I[\![t]\!]\rho$; on se ramène ainsi au problème SAT de satisfiabilité de formules propositionnelles, qui est NP-complet (polynomial dans le cas de clauses de Horn).

– Les *sur-approximations* $\mathcal{A}$, d'un autre côté sont complètes mais pas nécessairement correctes : si S est insatisfiable, alors $\mathcal{A}(S) =$"non". En renversant la proposition, si $\mathcal{A}(S)$ retourne "oui", alors S est satisfiable, sinon on ne sait rien.
En termes de vérification de protocoles cryptographiques, une sur-approximation est telle que si $\mathcal{A}(S)$ retourne "oui", où S décrit un protocole cryptographique, les hypothèses de sécurité et les propriétés à prouver, alors il n'y a pas d'attaque sur le protocole, de façon sûre. On obtient ainsi une *preuve de sécurité*.
En revanche, si $\mathcal{A}(S)$ retourne "non", on ne peut pas garantir qu'il y ait une attaque.
Notre but dans cette section est de décrire une famille très générale de sur-approximations, fondées sur la théorie de la section 6.

Remarque 8 *À noter que l'on peut opérer une fertilisation croisée entre les deux approches. Nos sur-approximations vont transformer des ensembles de clauses générales S_0 en des automates d'arbres généralisés S. On peut décider de la satisfiabilité de S. Si S est satisfiable, alors S_0 sera satisfiable (d'où une preuve de sécurité, dans le cas des protocoles cryptographiques). Si S est insatisfiable, on peut cependant, au moins dans le cas de Horn, convertir le sous-ensemble des clauses définies de S, qui est toujours satisfiable, en un modèle fini (voir remarque 7). Ce dernier modèle peut ensuite être utilisé, au moins en principe, par l'algorithme de sous-approximation décrit plus haut pour tenter de montrer que S_0 est insatisfiable (c'est-à-dire l'existence d'une attaque, dans le cas des protocoles cryptographiques).*

8.1 Types descriptifs

Nous présentons une famille extrêmement efficace d'approximations pour le problème de la satisfiabilité : les *typage descriptifs*. Il s'agit de familles d'algorithmes de typage de programmes Prolog, c'est-à-dire d'ensembles de clauses définies, qui tentent de découvrir les types des variables qui mènent à une déduction réussie d'un but donné. Par exemple, considérons l'ensemble de clauses :

$$\texttt{double}(\texttt{0},\texttt{0}) \quad \texttt{double}(\texttt{S}(X),\texttt{S}(\texttt{S}(Y))) \Leftarrow \texttt{double}(X,Y)$$

Il est facile de se convaincre que les seules formules dérivables de la forme $\texttt{double}(m,n)$ sont telles que m est un entier en unaire, et n est un entier pair, qui vaut le double de m.

Un mécanisme de typage descriptif est un moyen d'attribuer à chaque variable X de chaque clause un *type* τ, c'est-à-dire un ensemble de termes clos, tel que toute utilisation de la clause

dans une déduction instanciera X à un terme du type τ (plus précisément, un terme dont toutes les instances closes sont dans τ). Dans l'exemple ci-dessus, on peut ainsi attribuer à X dans la deuxième clause le type Nat des entiers unaires et à Y le type $Even$ des entiers pairs. Nous avions décrits ces types en section 6.7 par les contraintes ensemblistes :

$$Nat = \mathtt{S}(Nat) \cup \mathtt{0} \quad Even = \mathtt{S}(\mathtt{S}(Even)) \cup \mathtt{0}$$

Connaissant ces types, on constate que, si on ajoute aux clauses ci-dessus la clause

$$\bot \Leftarrow \mathtt{double}(X, \mathtt{S}(\mathtt{0}))$$

il n'y a aucun moyen de déduire le faux. On peut bien sûr le voir en lançant un des outils de démonstration automatique mentionnés plus haut. Mais c'est encore plus visible par typage : si on pouvait déduire le faux $\bot$, ç'aurait été avec une valeur paire du deuxième argument de `double`, par typage. Mais on ne peut déduire $\bot$. qu'en passant la valeur 1 (soit $\mathtt{S}(\mathtt{0})$), qui n'est pas paire.

Il n'est a priori pas facile de trouver de tels types descriptifs. Frühwirth *et al.* [FSVY91] proposent une méthode extrêmement élégante et complètement automatisée. Elle procède en trois étapes, consistant à transformer l'ensemble de clauses initial S_0 en un nouvel ensemble de clauses S décrivant les types des variables.

- On crée un nouveau prédicat `type`, prenant un argument, et autant de symboles de fonctions f_P qu'il y a de symboles de prédicats P dans l'ensemble de clauses S_0 de départ. L'idée est que $\mathtt{type}(f_P(t_1, \ldots, t_n))$ sera déductible de S si $P(t_1, \ldots, t_n)$ est déductible de S_0 : `type` décrit un sur-ensemble de toutes les formules $P(t_1, \ldots, t_n)$ déductibles de S_0.
- Pour toute clause C de S_0, de la forme

  $$P(t_1, \ldots, t_n) \Leftarrow P_1(t_{11}, \ldots, t_{1n_1}), \ldots, P_m(t_{m1}, \ldots, t_{mn_m})$$

  et chaque variable libre X de C, on crée un prédicat $\mathtt{type\text{-}}C\mathtt{\text{-}}X$, prenant un argument. L'idée est que $\mathtt{type\text{-}}C\mathtt{\text{-}}X(t)$ sera déductible de S si t est une valeur possible pour la variable X dans une déduction utilisant la clause C, comme dans l'exemple ci-dessus. Soient $X_1, \ldots, X_k$ les variables libres dans la tête $P(t_1, \ldots, t_n)$, listées de gauche à droite. (Il est possible qu'une même variable soit donc listée plusieurs fois.) Soient $Y_1, \ldots, Y_k$ des variables fraîches, et distinctes deux à deux. Soient $\tilde{t}_1, \ldots, \tilde{t}_n$ les termes $t_1, \ldots, t_n$ où X_i est remplacée par Y_i, $1 \leq i \leq k$. On remplace alors C par les clauses :

  $$\begin{array}{rcl} \mathtt{type}(f_P(\tilde{t}_1, \ldots, \tilde{t}_n)) & \Leftarrow & \mathtt{type\text{-}}C\mathtt{\text{-}}X_1(Y_1), \ldots, \mathtt{type\text{-}}C\mathtt{\text{-}}X_k(Y_k) \\ \mathtt{type\text{-}}C\mathtt{\text{-}}X_1(X_1) & \Leftarrow & \mathtt{type}(f_{P_1}(t_{11}, \ldots, t_{1n_1})), \ldots, \mathtt{type}(f_{P_m}(t_{m1}, \ldots, t_{mn_m})) \\ & & \ldots \\ \mathtt{type\text{-}}C\mathtt{\text{-}}X_k(X_k) & \Leftarrow & \mathtt{type}(f_{P_1}(t_{11}, \ldots, t_{1n_1})), \ldots, \mathtt{type}(f_{P_m}(t_{m1}, \ldots, t_{mn_m})) \end{array}$$

Par exemple, en partant des clauses ci-dessus, on obtient :

$$\mathtt{type}(f_{\mathtt{double}}(\mathtt{0}, \mathtt{0}))$$

$$\begin{aligned}
\texttt{type}(f_{\texttt{double}}(\texttt{S}(X),\texttt{S}(\texttt{S}(Y)))) &\Leftarrow \texttt{type-}C_2\texttt{-}X(X),\texttt{type-}C_2\texttt{-}Y(Y)\\
\texttt{type-}C_2\texttt{-}X(X) &\Leftarrow \texttt{type}(f_{\texttt{double}}(X,Y))\\
\texttt{type-}C_2\texttt{-}Y(Y) &\Leftarrow \texttt{type}(f_{\texttt{double}}(X,Y))
\end{aligned}$$

On peut se demander quel est l'avantage de décrire les types des variables d'un ensemble de clauses S_0 par un autre ensemble de clauses. Il se résume en ceci : savoir si un type est vide est décidable, et le problème est DEXPTIME-complet [FSVY91]. (Le lecteur est invité à montrer que la vacuité des programmes uniformes de Frühwirth *et al.* est décidable en temps exponentiel par les arguments de la section 6.5.)

On pourra vérifier que l'ensemble de clauses ci-dessus est tel que le langage reconnu en $\texttt{type-}C_2\texttt{-}X$ est exactement le type Nat, et le langage reconnu en $\texttt{type-}C_2\texttt{-}Y$ est exactement le type $Even$ des entiers pairs.

8.2 Approximations par formules monadiques

Dans le même esprit, on peut définir une procédure de sur-approximation valide pour tout ensemble de clauses, même si elles ne sont pas de Horn. (Cette procédure sera similaire dans l'esprit à celle de la section 8.2, mais ne coincide pas exactement dans le cas de Horn.)

Faisons l'hypothèse que tout prédicat est unaire. C'est l'hypothèse que nous faisons depuis la section 1. L'utilisation du prédicat `type` de la section 8.1 est une façon de ramener le cas général au cas unaire.

Définissons maintenant une série de transformations sur des ensembles de clauses S. Tant qu'il existe un terme t qui n'est ni une variable ni un terme plat dans une clause $C = C_0 \vee \pm P(t)$ de S, posons $t = f(t_1, \ldots, t_n)$, puis :

1. Si $\pm$ est le signe $-$, créer n prédicats unaires frais P_j, $1 \leq j \leq n$, et remplacer C dans S par les $n+1$ clauses :

$$C_0 \vee -P_1(t_1) \vee \ldots \vee -P_n(t_n) \tag{25}$$
$$-P(f(X_1,\ldots,X_n)) \vee +P_j(X_j) \qquad (1 \leq j \leq n) \tag{26}$$

   où $X_1, \ldots, X_n$ sont n variables distinctes deux à deux.

2. Si $\pm$ est le signe $+$, créer n prédicats unaires frais P_j, $1 \leq j \leq n$, et remplacer C dans S par les $n+1$ clauses :

$$+P(f(X_1,\ldots,X_n)) \vee -P_1(X_1) \vee \ldots \vee -P_n(X_n) \tag{27}$$
$$C_0 \vee +P_j(t_j) \qquad (1 \leq j \leq n) \tag{28}$$

Si S' est obtenue par un tel remplacement, écrivons $S \rightsquigarrow S'$. La relation $\rightsquigarrow$ termine. En effet, soit $|t|$ la taille du terme t, définie par $|X| = 0$, $|f(t_1,\ldots,t_n)| = 1 + |t_1| + \ldots + |t_n|$, et la taille d'une clause $\pm_1 P_1(t_1) \vee \ldots \vee \pm_n P_n(t_n)$ par $\sum_{i=1}^n |t_i|$. Alors $S \rightsquigarrow S'$ implique que le multi-ensemble des tailles des clauses de S est strictement plus grand que celui des tailles des clauses de S'.

Il est aussi facile de voir que si $S \rightsquigarrow S'$, alors S' implique S, c'est-à-dire que toute modèle de S' est un modèle de S. C'est parce que la clause remplacée C est un résolvant (non ordonné) des $n+1$ clauses (25) et (26), respectivement (27) et (28). (Le lecteur intéressé pourra améliorer la procédure en constatant que l'on n'est pas obligé de créer les X_i distinctes deux à deux, mais qu'il suffit d'imposer que $t_i \neq t_j$ implique $X_i \neq X_j$.)

Soit S_1 une forme normale (pour $\rightsquigarrow$) quelconque de S_0 : $S_0 \rightsquigarrow^* S_1$, et $S_1 \not\rightsquigarrow S'$ pour aucun S'. S_1 est de plus obtenu en temps polynomial à partir de S_0, et S_1 implique S_0. Si S_1 est satisfiable, S_0 est donc satisfiable.

Bien que S ressemble à un automate d'arbres généralisé, il se peut qu'il n'en soit pas un. En effet, S peut contenir une clause C de la forme $C_0 \vee L_1 \vee L_2$, où L_1 et L_2 sont deux littéraux ayant des ensembles de variables libres distincts mais non disjoints.

Il y a ici deux possibilités. La plus précise consiste à transformer ce type de clauses en les clauses :

$$C_0 \vee -\mathtt{type}^-(h(X_1, \ldots, X_n))$$
$$+\mathtt{type}^-(h(X_1, \ldots, X_n)) \vee L_1 \vee L_2$$

si L_1 et L_2 sont des littéraux négatifs, et en :

$$C_0 \vee +\mathtt{type}^+(h(X_1, \ldots, X_n))$$
$$-\mathtt{type}^+(h(X_1, \ldots, X_n)) \vee L_1 \vee L_2$$

si L_1 ou L_2 est positif, où h est un symbole de prédicat frais. Ce processus termine et préserve la satisfiabilité, et aboutit à un ensemble de clauses S du type de celles obtenues en section 7.6, c'est-à-dire des clauses dont les blocs sont des ϵ-blocs ou des transitions étendues.

On peut aussi directement produire un automate d'arbres généralisé S (à splitting près) en remplaçant C par $C_0 \vee L_1 \vee C_0' \vee L_2'$, où $C_0' \vee L_2'$ est un renommage de $C_0 \vee L_2$ qui ne partage aucune variable libre avec $C_0 \vee L_1$. Noter que $C_0 \vee L_1 \vee C_0' \vee L_2'$ implique $C_0 \vee L_1 \vee L_2$ car cette dernière est un facteur de la première. L'itération de ce processus termine encore en temps polynomial.

Dans les deux cas, l'ensemble de clauses final S est obtenu en temps polynomial à partir de S, et S implique S_0. Si S est satisfiable, S_0 est donc satisfiable. L'algorithme prenant S_0 en entrée et retournant "oui" si S est satisfiable, "non" sinon est donc un algorithme de sur-approximation, qui termine.

Dans le cas où S_0 est un ensemble de clauses de Horn, la procédure d'approximation ci-dessus fournit toujours un ensemble de clauses de Horn. (Les règles ont été conçues dans ce but.) Ceci nous permet d'obtenir une garantie de terminaison en temps simplement exponentiel, et de retrouver des modèles descriptibles par automates d'arbres avec contraintes d'égalité entre frères (remarque 7) dans le cas où S est satisfiable.

8.3 Amélioration de la précision

On peut se demander si la méthode d'approximation des ensembles de clauses de Horn par des automates d'arbres avec contraintes d'égalité entre frères de la section 8.2 est suffisamment

précise. Il arrive, et notamment dans le cadre de la vérification de protocoles cryptographiques, que ce ne soit pas le cas. Autrement dit, l'algorithme de sur-approximation $\mathcal{A}(S_0)$ répond "oui" trop souvent alors que S_0 est insatisfiable.

Cependant, certaines transformations préliminaires de l'ensemble de clauses S_0 améliorent grandement la précision de l'algorithme de typage de la section 8.2. On pourra consulter [Gou02], section 4.5 pour une justification de l'intérêt de ces transformations. Toutes ces transformations opèrent sur des ensembles de clauses *de Horn*.

Inlining L'*inlining* est une technique venant de la compilation des langages de programmation, adaptée au cas de Prolog. L'idée est que, si l'on a plusieurs clauses définies

$$C_1 \vee -P(t_1)$$
$$\dots$$
$$C_n \vee -P(t_n)$$

ce que l'on peut voir comme plusieurs endroits dans le programme où l'on appelle le sous-programme P, on peut remplacer les appels à $P(t_i)$ par la définition de P. Autrement dit, si les clauses définies dont la tête est de la forme $P(\dots)$ sont :

$$\begin{aligned} P(u_1) &\Leftarrow B_1 \\ &\dots \\ P(u_m) &\Leftarrow B_k \end{aligned}$$

on peut remplacer la clause $C_i \vee -P_i(t_i)$ par les clauses

$$(C_i \vee B_j)\sigma_{ij}$$

où σ_{ij} est le mgu de t_i avec u_j, s'il existe.

Cette stratégie intuitive est cependant à la fois trop naïve et trop compliquée. De façon plus simple, pour *inliner* un prédicat P, on liste d'abord les clauses C_1, …, C_n qui ont un littéral de la forme $-P(...)$ mais aucun de la forme $+P(...)$. On crée ensuite n nouveaux prédicats P_1, …, P_n, et on remplace P par P_i dans C_i, pour chaque i entre 1 et n. La seconde étape consiste à créer les clauses définissant les nouveaux prédicats P_i, en recopiant la définition de P et de tous les prédicats utilisés par P, remplaçant P par P_i.

Formellement, soit $\mathcal{P}(P)$ le plus petit ensemble de prédicats contenant P et tel que si $Q \in \mathcal{P}(P)$ et il existe une clause de la forme $Q(...) \Leftarrow Q_1(...), \dots, Q_m(...)$, alors $Q_1, \dots, Q_m \in \mathcal{P}(P)$. Soit aussi $\mathcal{C}(P)$ l'ensemble des clauses dont la tête est de la forme $Q(...)$ avec $Q \in \mathcal{P}(P)$. Pour définir P_i il suffit de, d'abord, créer de nouveaux prédicats Q^i distincts deux à deux, pour chaque $Q \in \mathcal{P}(P_i)$, $1 \leq i \leq n$, ensuite de créer pour chaque clause $C \in \mathcal{C}(P)$, la clause obtenue en remplaçant tout prédicat Q apparaissant dans C par Q^i.

Cette méthode est appelée inlining, parce qu'elle correspond exactement à la technique d'inlining de procédures en compilation, ici adaptée à Prolog. L'inlining produit un ensemble de clauses S_1 à partir d'un ensemble de clauses S_0, tel que, pour tout prédicat *non inliné* P, $P(t_1, \dots, t_n)$

est déductible de S_0 si et seulement s'il est déductible de S_1 : c'est en cela que l'inlining est correct.

L'intérêt de l'inlining est que l'on remplace le calcul du langage reconnu en P, ce qui inclut les valeurs des arguments de P venant de toutes les clauses $C_1, \ldots, C_n$ de façon non discriminée, par les langages reconnus en P_1, ..., P_n séparément.

Partitionnement Le partitionnement consiste à constater que toute variable X a pour vocation, dans la sémantique de Herbrand, à être instanciée par un terme clos, qui est donc de la forme $f(t_1, \ldots, t_n)$, $f \in \Sigma$, $t_1, \ldots, t_n$ termes clos. Étant donnée une clause C, et une variable X libre dans C, on peut donc remplacer la clause C par les clauses

$$C[X := f(X_1, \ldots, X_n)]$$

lorsque f parcourt Σ, et $X_1, \ldots, X_n$ sont n variables fraîches, distinctes deux à deux, n étant l'arité de f.

On peut utiliser cette règle de transformation avant de lancer un démonstrateur automatique. Mieux : l'utilisation de la résolution ordonnée avec sélection et de cette règle est un raffinement complet de la résolution, comme on s'en convaincra en adaptant la preuve du théorème 8.

Il est cependant à noter que l'utilisation de cette règle au plus tôt provoque la non-terminaison dès que Σ contient un symbole de fonction d'arité non nulle. En revanche, si tout symbole de Σ est une constante, alors la résolution plus cette règle fournit une procédure de décision pour la classe de Bernays-Schönfinkel.

Cette transformation est davantage utile dans un cas typé, où toute variable ne peut être instanciée que par des termes du même type qu'elle. On consultera [Gou02].

Le partitionnement préserve tous les plus petits modèles de Herbrand, mais pas nécessairement les autres modèles.

Ensembles magiques Une autre transformation classique en Prolog est celle des *ensembles magiques*. Elle est utile en particulier en conjonction avec la transformation par partitionnement, qui a tendance à engendrer de nombreuses clauses qui ne servent à rien dans aucune dérivation. La technique des ensembles magiques permet de guider la recherche de preuves [BMSU86].

La technique est relativement simple encore une fois. Supposons que nous ayons une clause $A \Leftarrow B_1, \ldots, B_n$. Créons de nouveaux symboles de prédicats (que nous représenterons en ajoutant le préfixe `magic-`). La formule `magic-`A intuitivement sera vraie si on estime avoir besoin de prouver A. On modifie alors la clause ci-dessus en la clause

$$A \texttt{:- magic-} A, B_1, \ldots, B_n$$

qui exprime que l'on n'aura le droit d'utiliser cette clause que si `magic-`A est vraie, c'est-à-dire si on a demandé à prouver A. La technique est élégante, dans la mesure où l'on peut ainsi guider la recherche de preuve de sorte qu'elle ne cherche à dériver que des formules qui ont une chance de servir à la preuve, sans avoir à se plonger dans des détails algorithmiques : le formalisme logique se charge de tout.

On doit aussi rajouter des clauses exprimant les dépendances entre formules. On écrira notamment :

$$\begin{aligned} \texttt{magic-}B_1 &\Leftarrow \texttt{magic-}A \\ \texttt{magic-}B_2 &\Leftarrow \texttt{magic-}A \\ &\dots \\ \texttt{magic-}B_n &\Leftarrow \texttt{magic-}A \end{aligned}$$

Autrement dit, si on a besoin de dériver A, on a a priori à dériver B_1, ..., B_n. Ce sont les *ensembles magiques* ("magic sets"). Le raffinement des *motifs magiques* ("magic templates") introduit une interaction non triviale avec la recherche de preuve proprement dite. Au lieu des clauses ci-dessus, on écrit :

$$\begin{aligned} \texttt{magic-}B_1 &\Leftarrow \texttt{magic-}A \\ \texttt{magic-}B_2 &\Leftarrow \texttt{magic-}A, B_1 \\ &\dots \\ \texttt{magic-}B_n &\Leftarrow \texttt{magic-}A, B_1, \dots, B_{n-1} \end{aligned}$$

Autrement dit, si on a besoin de dériver A on doit dériver B_1. Mais on n'a à dériver B_2 que si d'une part on a besoin de dériver A, mais aussi si d'autre part on a réussi à prouver B_1.

Il est finalement nécessaire d'ajouter une clause à l'ensemble de clauses transformé exprimant quel est le résultat que nous souhaitons démontrer. Si l'on souhaite vérifier qu'il existe un terme t tel que $P(t)$ soit vrai dans le plus petit modèle de Herbrand de l'ensemble de clauses qu'on s'est donné, on va ajouter la clause

$$\bot \Leftarrow P(X)$$

ainsi que la clause magique

$$\texttt{magic-}P(X)$$

On consultera [BMSU86] pour le cas de formules du premier ordre (avec variables). Ce n'est pas spécialement plus compliqué.

Ces transformations ne préservent la dérivabilité que des atomes $P(t_1, \dots, t_n)$ pour lesquels on a ajouté la clause $\texttt{magic-}P(t_1, \dots, t_n)$. Ceci se démontre par récurrence sur les arbres de dérivation de clauses buts obtenus par hyperrésolution positive (exercice).

Références

[BG01] Leo Bachmair and Harald Ganzinger. *Resolution Theorem Proving*, chapter 2, pages 19–99. Volume I of Robinson and Voronkov [RV01], 2001.

[BMSU86] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In *Proceedings of the International Conference on Management of Data and Symposium on Principles of Database Systems*. ACM Press, 1986.

[BT92] B. Bogaert and Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *9th Annual Symposium on Theoretical Aspects of Computer Science (STACS'92)*, pages 161–171. Springer-Verlag LNCS 577, 1992.

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. `www.grappa.univ-lille3.fr/tata`, 1997.

[CKS81] A. K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J.ACM*, 28 :114–133, 1981.

[CL73] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science Classics. Academic Press, 1973.

[dN95] Hans de Nivelle. *Ordering Refinements of Resolution*. PhD thesis, Technische Universiteit Delft, 1995.

[dN98] Hans de Nivelle. Resolution decides the guarded fragment. Technical Report CT-1998-01, Institute of Language, Logic and Computation, 1998.

[FLHT01] Christian Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tammet. *Resolution Decision Procedures*, chapter 25, pages 1791–1849. Volume II of Robinson and Voronkov [RV01], 2001.

[FSVY91] Thom Frühwirth, Ehud Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic programs as types for logic programs. In *LICS'91*, 1991.

[GLJ05] Jean Goubault-Larrecq and Jean-Pierre Jouannaud. The blossom of finite semantic trees. Workshop on Programming Logics in Memory of Harald Ganzinger. Submitted for publication. `http://www.lsv.ens-cachan.fr/~goubault/DemAuto/goubault-larrecq-jouannaud-the-blossom-of-finite-semantic-trees.pdf`, 2005.

[Gou02] Jean Goubault-Larrecq. Vérification de protocoles cryptographiques : la logique à la rescousse ! In Jean Goubault-Larrecq, editor, *Actes du 1er workshop international sur la sécurité des communications sur Internet (SECI'02), Tunis, Tunisia, Sep. 2002*, pages 119–152. INRIA, 2002.

[GS97] Ferenc Gécseg and Magnus Steinby. Tree languages. In Grzegorz Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer Verlag, 1997.

[JJ76] William H. Joyner Jr. Resolution strategies as decision procedures. *Journal of the ACM*, 23(3) :398–417, July 1976.

[Joh90] David S. Johnson. A catalog of complexity classes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 2, pages 67–161. Elsevier Science Publishers b.v., 1990.

[Koz77] Dexter Kozen. Lower bounds for natural proof systems. In *18th FOCS*, pages 254–266, 1977.

[Rey69] John C. Reynolds. Automatic computation of data set definition. *Information Processing*, 68 :456–461, 1969.

[RSV01] I. V. Ramakrishnan, R. Sekar, and A. Voronkov. *Term Indexing*, chapter 26, pages 1853–1954. Volume II of Robinson and Voronkov [RV01], 2001.

[Rus89] Michaël Rusinowitch. *Démonstration Automatique*. Interéditions, 1989.

[RV01] J. Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. North-Holland, 2001.

[Sei94] Helmut Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2) :57–60, 1994.

[Sel01] Peter Selinger. Models for an adversary-centric protocol logic. *Electronic Notes in Theoretical Computer Science*, 55(1) :73–87, July 2001. Proceedings of the 1st Workshop on Logical Aspects of Cryptographic Protocol Verification (LACPV'01), J. Goubault-Larrecq, ed.

[Sha84] Ehud Y. Shapiro. Alternation and the computational complexity of logic programs. *Journal of Logic Programming*, 1(1) :19–33, 1984.

[SV05] Helmut Seidl and Kumar Neeraj Verma. Flat and one-variable clauses : Complexity of verifying cryptographic protocols with single blind copying. In *LPAR'2004*, pages 79–94. Springer-Verlag LNCS 3452, 2005.

A Quelques résultats fondamentaux

A.1 Théorème du point fixe de Tarski

Une *relation d'ordre* $\leq$ sur un ensemble L est une relation binaire réflexive ($x \leq x$), antisymétrique (si $x \leq y$ et $y \leq x$ alors $x = y$) et transitive (si $x \leq y$ et $y \leq z$ alors $x \leq z$). On dit aussi que $(L, \leq)$ est un *ensemble ordonné*.

Pour tout $F \subseteq L$, un *majorant* de F est un élément x de L tel que $x \geq y$ pour tout y dans F. Une *borne supérieure* de F est un majorant plus petit que tous les autres. Si elle existe, la borne supérieure de F est unique, par antisymétrie de $\leq$, et on la notera $\bigsqcup F$. Elle est caractérisée par les propriétés :

$$\forall y \in F \cdot y \leq \bigsqcup F \tag{29}$$

$$\forall x \cdot (\forall y \in F \cdot y \leq x) \Rightarrow \bigsqcup F \leq x \tag{30}$$

Similairement, un *minorant* de F est un élément x de L tel que $x \leq y$ pour tout y dans F. Le plus grand des minorants de F, s'il existe, est la *borne inférieure* $\bigsqcap F$ de F.

Un ensemble ordonné $(L, \leq)$ tel que toute partie F de L a une borne supérieure et une borne inférieure est appelé un *treillis complet*. On pourra remarquer qu'il suffit de demander que toute partie F ait une borne supérieure ; dans ce cas la borne inférieure existe toujours, et $\bigsqcap F$ est la borne supérieure de l'ensemble des minorants de F. Symétriquement, il suffit de demander que

toute partie F ait une borne inférieure ; dans ce cas la borne supérieure existe toujours, et $\bigsqcup F$ est la borne inférieure de l'ensemble des majorants de F.

Une application $f : (L, \leq) \to (L', \leq')$ est *monotone* si et seulement si $x \leq y$ implique $f(x) \leq f(y)$. Pour toute $f : L \to L$, un *point fixe* de f est un élément x de L tel que $f(x) = x$.

Théorème 23 (Tarski) *Soit $(L, \leq)$ un treillis complet, et $f : (L, \leq) \to (L, \leq)$ une application monotone. Alors f a un plus petit point fixe, et un plus grand point fixe.*

Démonstration. Soit $Post(f)$ l'ensemble des *post-points-fixes* de f, c'est-à-dire des élements x de L tels que $f(x) \subseteq x$. $Post(f)$ est non vide, car $\top$ est un post-point-fixe de f, où $\top = \bigsqcap \emptyset$ est le plus grand élément de L.

Posons $x_0 = \bigsqcap Post(f)$. Pour tout $x \in Post(f)$, $x_0 \leq x$, donc $f(x_0) \leq f(x)$ puisque f est monotone, donc $f(x_0) \leq x$ puisque $f(x) \leq x$ (définition de $x \in Post(f)$). Donc $f(x_0)$ est un minorant de $Post(f)$. Comme x_0 est le plus grand des minorants de $Post(f)$, $f(x_0) \leq x_0$. Mais ceci signifie que x_0 est dans $Post(f)$: x_0 est le *plus petit* post-point-fixe.

Comme $f(x_0) \leq x_0$, par monotonie $f(f(x_0)) \leq f(x_0)$, donc $f(x_0)$ est un post-point-fixe de f. Comme x_0 est le plus petit, $x_0 \leq f(x_0)$. Par antisymétrie, $x_0 = f(x_0)$, donc x_0 est un point fixe de f. Comme tout point fixe est un post-point-fixe, x_0 est donc aussi le plus petit point fixe de f.

De même, $\bigsqcup Pre(f)$ est le plus grand point fixe de f, où $Pre(f)$ est l'ensemble des *pré-points-fixes* de f, c'est-à-dire des éléments x de L tels que $x \leq f(x)$. □

On peut même démontrer que l'ensemble des points fixes de f forme un sous-treillis complet de $(L, \leq)$.

Notons $x > y$ si et seulement si $x \geq y$ et $x \neq y$. Un treillis complet a la *condition de chaîne croissante* si et seulement s'il n'existe aucune chaîne croissante infinie $x_0 < x_1 < \ldots < x_i < \ldots$; on a :

Proposition 24 *Soit $(L, \leq)$ un treillis complet ayant la condition de chaîne croissante, et $f : (L, \leq) \to (L, \leq)$ une application monotone. Alors le plus petit point fixe de f est $\bigsqcup_{n \in \mathbb{N}} f^n(\bot)$, où $\bot = \bigsqcup \emptyset$ est le plus petit élément de L.*

Démonstration. Posons $x = \bigsqcup_{n \in \mathbb{N}} f^n(\bot)$. On remarque que $f^n(\bot) \leq f^{n+1}(\bot)$ pour tout n, par récurrence sur n. C'est vrai pour $n = 0$, parce que $f^0(\bot) = \bot$ est le plus petit élément de L. Ensuite $f^{n-1}(\bot) \leq f^n(\bot)$ entraîne $f^n(\bot) \leq f^{n+1}(\bot)$ par monotonie de f.

La suite $(f^n(\bot))_{n \in \mathbb{N}}$ est donc croissante. Par la condition de chaîne croissante, elle est finie, c'est-à-dire qu'il existe un entier N tel que pour tout $n \geq N$, $f^n(\bot) = f^N(\bot)$. Donc $x = f^N(\bot)$. Il s'ensuit que $f(x) = f^{N+1}(\bot) = f^N(\bot) = x$, donc x est un point fixe de f.

Soit y un autre point fixe de f. On a $\bot \leq y$ car $\bot$ est le plus petit élement de L, donc $f^n(\bot) \leq f^n(y)$ pour tout y, car f est monotone. Donc $x = \bigsqcup_{n \in \mathbb{N}} f^n(\bot) \leq \bigsqcup_{n \in \mathbb{N}} f^n(y) = \bigsqcup_{n \in \mathbb{N}} y = y$ (car y est un point fixe). Donc x est le plus petit point fixe de f. □

La preuve montre que, dans ce cas, le plus petit point fixe de f est en fait *calculable* (pourvu que f soit elle-même calculable) par le programme :

$x := \bot$;
tant que ($y := f(x)$, $y \neq x$)
$x := y$;

A.2 Arbres et lemme de König

Un ensemble ordonné $(L, \leq)$ est une *forêt* si et seulement si :

- Pour tout x dans L, le segment initial $x \downarrow = \{y \in L | y \leq x\}$ est totalement ordonné par $\leq$. On dit qu'un ensemble A est *totalement ordonné* par $\leq$ si et seulement si, pour tous $x, y \in A$, $x \leq y$ ou $y \leq x$.
- D'autre part, tout sous-ensemble X non vide de L a un élément minimal. Un élément x est *minimal* dans X si et seulement si le seul élément $y \in X$ tel que $y \leq x$ est x.

Un *arbre* est une forêt qui a un unique élément minimal, appelé la *racine* de l'arbre.

Pour tout x dans L, les *successeurs* de x sont les éléments minimaux de $\{y \in L | x \leq y, x \neq y\}$. On note $Succ(x)$ l'ensemble des successeurs de x. Si L est une forêt, soit x est maximal, et l'on dit que x est une *feuille* de la forêt L, soit x a au moins un successeur.

Une forêt L est à *branchement fini* si et seulement si, pour tout x, $Succ(x)$ est un ensemble fini. Une *branche* d'un arbre L est une suite finie ou infinie x_0, $x_1 \in Succ(x_0)$, ..., $x_{i+1} \in Succ(x_i)$, ... ; on dit qu'il s'agit d'une branche *issue de* x_0.

Théorème 25 (König) *Tout arbre à branchement fini dont toutes les branches sont finies est fini.*

Démonstration. Soit L un arbre infini, à branchement fini. On va construire une branche infinie de L. Pour ceci, il suffit de construire une branche finie $x_0, x_1, \ldots, x_i$ pour tout $i \in \mathbb{N}$, telle que $\{y \in L | y \geq x_i\}$ est infini, par récurrence sur i. Notons que pour tout x, $\{y \in L | y \geq x\}$ est un arbre, qu'on appellera le *sous-arbre* de L de racine x. Pour $i = 0$, on pose x_0 la racine de L. Sinon, on suppose $x_0, x_1, \ldots, x_i$ construits. Comme le sous-arbre de racine x_i est infini, et que x_i n'a qu'un nombre fini de successeurs, il en existe nécessairement un qui est racine d'un sous-arbre infini : soit x_{i+1} un de ces successeurs. □

Cet argument utilise, d'une façon assez cachée il est vrai, l'axiome du choix. Pour chaque $i \in \mathbb{N}$, nous choisissons un x_i ; nous pouvons effectuer ceci sans l'axiome du choix, en revanche dire que nous pouvons collecter tous ces choix en une unique suite est exactement ce que l'axiome du choix nous autorise. En fait, le lemme de König est équivalent à une forme faible d'axiome du choix, appelé axiome des choix dépendants, et qui exprime que pour toute relation binaire R, pour tout x_0, si pour tout x_i déjà construit, il existe x_{i+1} tel que $x_i \ R \ x_{i+1}$, alors il existe une suite $(x_i)_{i \in \mathbb{N}}$ telle que $x_i \ R \ x_{i+1}$ pour tout $i \in \mathbb{N}$.

A.3 Ordres bien fondés

Un *ordre strict* $\succ$ sur l'ensemble L est une relation binaire irréflexive ($x \not\succ x$) et transitive (si $x \succ y$ et $y \succ z$ alors $x \succ z$). Un ordre strict est *bien fondé* s'il n'admet aucune chaîne infinie décroissante $x_1 \succ x_2 \succ \ldots \succ x_k \succ \ldots$. Il s'agit d'un anglicisme [well-founded], que nous préférons pour des raisons de clarté au mot français *bon ordre*.

Par exemple, l'ordre $>$ sur les entiers naturels est bien fondé.

Une définition équivalente d'un ordre bien fondé est que tout ensemble X non vide contient un élément minimal pour $\prec$. En effet, si $\succ$ est bien fondé et X non vide, soit $x_1 \in X$ quelconque : si x_1 est minimal, alors c'est fini ; sinon, il existe $x_2 \prec x_1$, et si x_2 est minimal, alors on a encore gagné ; sinon, il existe $x_3 \prec x_2 \prec x_1$, etc. Ce processus doit terminer après un nombre fini d'étapes, fournissant un élément minimal de X. Réciproquement, si tout ensemble non vide X contient un élément minimal, alors soit X l'ensemble des x tels qu'il existe une chaîne décroissante infinie $x = x_1 \succ x_2 \succ \ldots \succ x_k \succ \ldots$; mais aucun $x = x_1$ de X ne peut être minimal, car x_2 est encore dans X, donc X est vide et donc $\succ$ est bien fondé.

Une caractéristique des ordres bien fondés est que l'on peut raisonner par *récurrence* sur ces ordres. Pour toute propriété P, pour montrer que $P(x)$ est vraie pour tout x, il suffit de montrer que $P(x)$ est vrai sous l'hypothèse que $P(y)$ est vrai pour tout $y \prec x$. En effet, soit X l'ensemble des x tels que $P(x)$ est fausse. Si $P(x)$ n'était pas vraie de tout x, X serait non vide, donc aurait un élément minimal x, qui serait donc tel que $x \in X$ mais $y \notin X$ pour tout $y \prec x$, c'est-à-dire $P(x)$ est fausse mais $P(y)$ est vraie pour tout $y \prec x$, contredisant l'hypothèse que tout x tel que $P(y)$ est vraie pour tout $y \prec x$ vérifie $P(x)$. Réciproquement, tout ordre qui admet un tel principe de récurrence est bien fondé (exercice).

Pour tout ordre strict $\succ$, notons $\succeq$ la relation définie par $x \succeq y$ si et seulement si $x \succ y$ ou $x = y$. Il s'agit d'une relation d'*ordre*, c'est-à-dire une relation binaire réflexive, antisymétrique, et transitive. On dit que $\succ$ est *total* si et seulement si $\succeq$ est total, autrement dit si et seulement si pour tous x, y, $x \prec y$ ou $x = y$ ou $x \succ y$.

Théorème 26 (Zermelo) *Pour tout ordre strict bien fondé $\succ$ sur L, il existe un ordre strict bien fondé* total $\overline{\succ}$ *sur L étendant $\succ$, c'est-à-dire tel que pour tous x, y, si $x \succ y$ alors $x \overline{\succ} y$.*

Démonstration. Pour tout sous-ensemble X non vide de L, comme $\succ$ est bien fondé, il existe un élément minimal de X. Par l'axiome du choix, il existe donc une fonction f qui à tout $X \subseteq L$ non vide associe un élément minimal de X.

Disons qu'un couple $(X, >)$ formé d'une partie X de L et d'un ordre strict $>$ sur X est *adapté* si et seulement si :

– $>$ est un ordre total bien fondé sur X ;

– $>$ étend la restriction de $\succ$ à X : pour tous $x, y \in X$, si $x \succ y$ alors $x > y$.

L'ensemble des couples adaptés est ordonné par : $(X, >) \sqsubseteq (X', >')$ si et seulement si :

1. $X \subseteq X'$;
2. $>'$ étend $>$ à X', c'est-à-dire que pour tout $x, y \in X$, si $x > y$ alors $x >' y$;
3. X est un segment initial de $(X', >')$: pour tous $x \in X$, $x' \in X'$, $x' >' x$.

Soit $(X_i, >_i)_{i \in I}$ une chaîne quelconque de couples adaptés, c'est-à-dire un ensemble totalement ordonné par $\sqsubseteq$. Montrons que les $(X_i, >_i)$ ont un majorant $(X, >)$. En fait, nous prétendons que l'on peut choisir $X = \bigcup_{i \in I} X_i$, et $x > y$ si et seulement s'il existe $i \in I$ tel que $x, y \in X_i$ et $x >_i y$. Clairement $X_i \subseteq X$ pour tout $i \in I$, $>$ étend $>_i$ à X, et X_i est un segment initial de $(X, >)$. Il ne reste qu'à montrer que $(X, >)$ est un couple adapté. Or :

– $>$ est irréflexive : si on avait $x > x$ on aurait $x >_i x$ pour un $i \in I$;

– $>$ est transitive : si $x > y$ et $y > z$, alors il existe $i, j \in I$ tels que $x, y \in X_i$, $y, z \in X_j$, $x >_i y$ et $y >_j z$. Comme les $(X_k, >_k)$ forment une chaîne, $(X_i, >_i) \sqsubseteq (X_j, >_j)$ ou $(X_j, >_j) \sqsubseteq (X_i, >_i)$. Dans les deux cas, par les propriétés 1 et 2 ci-dessus, on peut supposer qu'il existe $k \in I$ (tel que X_k est le plus grand de X_i, X_j) tel que $x, y, z \in X_k$, $x >_k y$ et $y >_k z$, donc $x >_k z$. Donc $x > z$.
– $>$ est total : pour tous $x, y \in X$, comme les $(X_k, >_k)$ forment une chaîne, il existe $k \in I$ tel que $x, y \in X_k$. Alors comme $>_k$ est total, $x >_k y$ ou $y >_k x$ ou $x = y$, donc $x > y$ ou $x < y$ ou $x = y$.
– $>$ est bien fondé : soit Y une partie quelconque non vide de X. En particulier, Y intersecte au moins un des X_i, $i \in I$, donc il existe un élément $y \in Y \cap X_i$ minimal pour $>_i$. Montrons que y est minimal dans Y pour $>$. Supposons au contraire qu'il existe z dans Y tel que $z < y$. Soit $j \in I$ tel que $z \in X_j$ et $z <_j y$. Si $(X_j, >_j) \sqsubseteq (X_i, >_i)$, alors $z \in X_i$ et $z <_i y$ par les propriétés 1 et 2, ce qui est impossible. Sinon, alors $(X_i, >_i) \sqsubseteq (X_j, >_j)$ par la propriété de chaîne, donc par la propriété 3, $y <_j z$, ce qui contredit $z <_j y$.
– $>$ étend la restriction de $\succ$ à X : soient $x, y \in X$ tels que $x \succ y$. Comme les $(X_k, >_k)$ forment une chaîne, il existe $k \in I$ tel que $x, y \in X_k$, donc $x >_k y$ puisque $(X_k, >_k)$ est un couple adapté ; donc $x > y$.

L'ordre $\sqsubseteq$ est donc un ordre *inductif* : toute chaîne a un majorant. Le lemme de Zorn (équivalent à l'axiome du choix) exprime que tout ensemble ordonné inductif a un élément maximal.

Soit donc $(X, >)$ un couple adapté maximal pour $\sqsubseteq$. Si on avait $X \neq L$, il existerait un élément $x_0 \in L \setminus X$, puisque $\succ$ est bien fondé. Posons $X' = X \cup \{x_0\}$, et définissons $>'$ par $x >' y$ si et seulement si $x = x_0$ et $y \in X$, ou $x, y \in X$ et $x > y$. Il est clair que $>'$ est un ordre total bien fondé sur X' qui étend $\succ$, donc $(X', >')$ est un couple adapté. D'autre part, il est clair aussi que $(X, >) \sqsubseteq (X', >')$, contredisant l'hypothèse de maximalité de $(X, >)$. Donc $X = L$, et $>$ est donc un ordre total bien fondé étendant $\succ$ sur L. □

Dans le cas particulier où $\succ$ est l'ordre strict vide ($x \not\succ x$, pour tout $x \in L$), le théorème 26 implique que tout ensemble peut être muni d'un ordre total bien fondé. Ce dernier résultat est ce qui est usuellement appelé le théorème de Zermelo. C'est une conséquence de l'axiome du choix, comme on l'a vu. Réciproquement, l'axiome du choix est conséquence du théorème de Zermelo : si on munit L d'un ordre total bien fondé $\overline{\succ}$, toute partie non vide X de L a un unique élément minimal $min(X)$, et la fonction $X \mapsto min(X)$ est alors une fonction de choix sur L.

Dans la suite, nous montrons quelques constructions permettant de combiner des ordres bien fondés à partir d'ordres bien fondés plus simples.

Ordre lexicographique. Si $\succ_1$ et $\succ_2$ sont deux ordres stricts, leur *produit lexicographique* $\succ_1 \times_{lex} \succ_2$ est défini par $(x_1, x_2)\ (\succ_1 \times_{lex} \succ_2)\ (y_1, y_2)$ si et seulement si

– $x_1 \succ_1 y_1$,
– ou $x_1 = y_1$ et $x_2 \succ_2 y_2$.

Si $\succ_1$ et $\succ_2$ sont deux ordres bien fondés, alors $\succ_1 \times_{lex} \succ_2$ est bien fondé. Soit en effet A un ensemble non vide de couples (x, y). Soit X l'ensemble $\{x | \exists y \cdot (x, y) \in A\}$. Comme X est non vide et $\succ_1$ est bien fondé, il existe un élément x_0 minimal pour $\succ_1$ dans X. Soit Y l'ensemble $\{y | (x_0, y) \in A\}$. Y est non vide par définition de $x_0 \in X$, et contient donc un élément minimal

y_0 pour $\succ_2$ dans Y. Alors (x_0, y_0) est dans A, et est minimal pour $\succ_1 \times_{lex} \succ_2$, finissant de prouver que $\succ_1 \times_{lex} \succ_2$ est bien fondé.

On peut aussi montrer que si $\succ_1$ et $\succ_2$ sont deux ordres stricts totaux, alors $\succ_1 \times_{lex} \succ_2$ est total lui aussi (exercice).

On peut généraliser et montrer que tout produit lexicographique d'un nombre *fini* d'ordres stricts bien fondés est bien fondé. Ce n'est plus vrai pour un produit infini. Par exemple, si $\gg$ est le produit lexicographique sur $\mathbb{N}^{\mathbb{N}}$, on a la suite infinie décroissante :

$$(1,0,0,0,\ldots) \gg (0,1,0,0,\ldots) \gg (0,0,1,0,\ldots) \gg \ldots$$

Ordre lexicographique sur suites finies triées. Soit L un ensemble muni d'un ordre strict bien fondé $\succ$. Soit $L^\bullet$ l'ensemble de toutes les suites $[x_1, \ldots, x_n]$ *triées*, c'est-à-dire telles que $x_1 \preceq x_2 \preceq \ldots \preceq x_n$.

On définit l'ordre $\succ^\bullet$ sur $L^\bullet$ par :

- $[x_1, \ldots, x_n] \succ^\bullet []$ si $n \geq 1$;
- $[x_1, \ldots, x_n] \succ^\bullet [y_1, \ldots, y_m]$ si $n, m \geq 1$ et :
  - soit $x_n \succ y_m$;
  - soit $x_n = y_m$ et $[x_1, \ldots, x_{n-1}] \succ^\bullet [y_1, \ldots, y_{m-1}]$.

La définition de $[x_1, \ldots, x_n] \succ^\bullet [y_1, \ldots, y_m]$ est bien formée, par récurrence sur $m + n$. On note qu'il s'agit essentiellement d'une forme infinie de produit lexicographique. (Nous comparons ici les éléments de droite à gauche, et non de gauche à droite comme pour $\succ_1 \times_{lex} \succ_2$, mais il est clair qu'il s'agit d'un point de détail.)

On notera parfois $\vec{x}$ la suite $[x_1, \ldots, x_n]$; sa *longueur* est n, son *max* est x_n.

Si $\succ$ est bien fondé sur L, alors $\succ^\bullet$ est bien fondé sur $L^\bullet$. En effet, supposons que $\succ^\bullet$ n'est pas bien fondé. Il existe alors une suite infinie $\vec{x}^0 \succ^\bullet \vec{x}^1 \succ^\bullet \ldots$; notons que $\vec{x}^0$ n'est pas la suite vide $[]$, et choisissons une telle suite infinie telle que le max x de $\vec{x}^0$ soit minimal, et ce max x étant fixé, telle que la longueur de $\vec{x}^0$ soit minimale. Par la minimalité du max x de $\vec{x}^0$, tous les $\vec{x}^i$ ont le même max x. On en déduit que $\vec{y}^0 \succ^\bullet \vec{y}^1 \succ^\bullet \ldots$ est une suite décroissante infinie, où $\vec{y}^i$ est la suite $\vec{x}^i$ dont on a retiré le dernier élément (égal à x). Mais comme $\vec{x}^0$ est une suite triée, le max de $\vec{y}^0$ est inférieur ou égal à x, et sa longueur est strictement inférieure à celle de $\vec{x}^0$, contredisant la minimalité de x, et à x fixé, la minimalité de la longueur de $\vec{x}^0$.

On peut aussi montrer que si $\succ$ est total, alors $\succ^\bullet$ est total sur $L^\bullet$.

Ordre multi-ensemble. Un *multi-ensemble* (fini) m sur un ensemble A est une application de A vers $\mathbb{N}$ telle que $m(x) = 0$ sauf pour un nombre fini d'éléments x de A. L'ensemble $\{x \in A | m(x) \neq 0\}$ est le *support* de m. Intuitivement, un multi-ensemble est un ensemble fini, où chaque élément peut apparaître une ou plusieurs fois. Par exemple, le multi-ensemble qui à x associe 1 et y associe 3 est celui où x apparaît une fois et y trois fois, et on le notera usuellement $\{x, y, y, y\}$. Une intuition utile est qu'un multi-ensemble est une liste, où l'ordre des éléments ne compte pas mais le nombre de leurs occurrences compte.

Si m et m' sont deux multi-ensembles, leur *union* $m \uplus m'$ est la fonction qui à tout x associe $m(x) + m'(x)$. Si m et m' sont représentés par des listes, alors $m \uplus m'$ est représenté par la concaténation des deux listes.

Si $\succ$ est un ordre strict sur A, son *extension multi-ensemble* $\succ_{mul}$ est le plus petit ordre strict tel que $m \uplus \{x\} \succ_{mul} m \uplus \{x_1, \ldots, x_n\}$ pour tout x, et tout multi-ensemble d'éléments $x_1, \ldots, x_n \prec x$.

Par exemple, si $>$ est l'ordre usuel sur les entiers naturels, on a

$$\{3\} >_{mul} \{2, 2, 2\} >_{mul} \{1, 1, 2, 2\} >_{mul} \{1, 1, 2\}$$

Dans l'inégalité de gauche, on a remplacé 3 par les éléments 2, 2, 2, qui sont plus petits que 3. Dans celle du milieu, on a remplacé un 2 par deux 1. Dans celle de droite, on a remplacé un 2 par zéro élément, autrement dit on a effacé un 2.

Le point important est que si $\succ$ est bien fondé, alors $\succ_{mul}$ est bien fondé. C'est ce que nous allons montrer maintenant. D'abord, par le théorème 26, soit $\overline{\succ}$ une extension totale bien fondée de $\succ$. Alors $\overline{\succ}_{mul}$ est une extension de $\succ_{mul}$, et il suffit de montrer que $\overline{\succ}_{mul}$ est bien fondé pour en déduire que $\succ_{mul}$ est bien fondé. Donc, sans perte de généralité, nous pouvons supposer que $\succ$ est un ordre strict bien fondé *total*.

Comme $\succ$ est total, on peut trier les éléments d'un multi-ensemble en ordre croissant : pour tout multi-ensemble m, de support l'ensemble fini X, soient $x_1 \prec \ldots \prec x_n$ les éléments de X, et posons $\widehat{m}$ la suite

$$\underbrace{x_1, \ldots, x_1}_{m(x_1)}, \underbrace{x_2, \ldots, x_2}_{m(x_2)}, \ldots, \underbrace{x_n, \ldots, x_n}_{m(x_n)}$$

L'application qui à tout multi-ensemble m associe la suite finie triée $\widehat{m}$ est clairement une bijection.

De plus, on vérifie aisément que $m \succ_{mul} m'$ si et seulement si $\widehat{m} \succ^{\bullet} \widehat{m}'$. Il s'ensuite que si $\succ$ est bien fondé, alors $\succ_{mul}$ est bien fondé.

Finalement, si $\succ$ est total, comme $\succ^{\bullet}$ est total sur $L^{\bullet}$, $\succ_{mul}$ est lui aussi total sur l'ensemble des multi-ensembles d'éléments de L.

B Classes de complexité

Nous faisons ici un rappel très informel, mais suffisant pour se faire une idée, de quelques notions de base en théorie de la complexité.

Un *problème de décision* est, brutalement parlant, un langage. En pratique, on décrit un problème de décision sous la forme :

ENTRÉE : une donnée x ;
QUESTION : la propriété $P(x)$ est-elle vraie ?

On verra souvent un problème de décision comme étant juste le prédicat P.

Le langage défini par ce format est l'ensemble des x telles que $P(x)$ est vraie. Par exemple, le problème HORNSAT est :

ENTRÉE : un ensemble S de clauses de Horn propositionnelles (c'est-à-dire, closes) ;
QUESTION : S est-il satisfiable ?

Le problème HORNSAT est décidable en *temps polynomial*, c'est-à-dire qu'il existe un algorithme qui, prenant en entrée un ensemble S de clauses de Horn propositionnelles, retourne "oui" si S et satisfiable et "non" si S est insatisfiable en un temps $O(|S|^n)$ pour un certain entier $n \geq 1$, et où $|S|$ dénote la taille de S. Il suffit en effet de saturer S par hyperrésolution positive, ce qui termine en temps $O(|S|^2)$. (On peut même résoudre le problème en temps [quasi-]linéaire.)

La classe de tous les problèmes de décision décidables en temps polynomial est notée P.

La classe NP ("Non-déterministe Polynomial") est la classe des problèmes de la forme :

ENTRÉE : une donnée x ;
QUESTION : existe-t-il une donnée y de taille $O(|x|^m)$ telle que $P(x, y)$ soit vraie ?

---

où m est un entier fixé supérieur ou égal à 1, et $P(x, y)$ est un problème dans P.

Par exemple, le problème SAT suivant est dans NP :

ENTRÉE : un ensemble S de clauses propositionnelles ;
QUESTION : S est-il satisfiable ?

On ne sait pas si SAT est dans P ou non, mais cela semble improbable. On peut montrer que SAT est *NP-complet*, c'est-à-dire que pour tout problème Q dans NP, il existe une fonction f calculable en temps polynomial telle que pour toute entrée x de Q, $Q(x)$ est vraie si et seulement si $f(x)$ est satisfiable. Ceci a comme conséquence que SAT est dans P si et seulement si P=NP.

Une façon plus classique de définir NP est d'utiliser un modèle de machines, les machines de Turing *non déterministes*. Ces machines ont la possibilité, en plus d'effectuer des étapes de calcul normales, de se dupliquer en autant de machines qu'il y a de choix pour la donnée y dans l'énoncé ci-dessus. La machine globale répond "oui" si l'une des machines filles répond "oui", et répond "non" si toutes les machines filles répondent "non". NP est alors la classe des problèmes résolubles sur une machine de Turing non déterministe en temps polynomial.

Clairement, P$\subseteq$NP.

Toujours plus haut, la classe PSPACE ("Polynomial SPACE") est la classe des problèmes P tels qu'on peut tester $P(x)$ en temps fini quelconque, mais en utilisant qu'un espace polynomial, c'est-à-dire une quantité de mémoire polynomiale. Il se trouve qu'ici, que la machine soit déterministe ou non n'a aucune importance : le théorème de Savitch dit que les problèmes décidables en espace $O(f(|x|))$ non-déterministe sont décidables en espace déterministe $O(f^2(|x|))$, et clairement tout problème décidable en espace déterministe $O(f(|x|))$ est décidable en espace non déterministe $O(f(|x|))$.

Une caractérisation équivalente, due à Wrathall, de PSPACE, est la suivante. Un problème de APTIME ("Alternating Polynomial Time") est un problème de la forme :

ENTRÉE : une donnée x ;
QUESTION : existe-t-il un entier $n = O(|x|^m)$, une donnée y_1 de taille $O(|x|^m)$ telle que pour toute donnée y_2 de taille $O(|x|^m)$, il existe y_3 de taille $O(|x|^m)$ telle que ..., il existe y_{2n-1} de taille $O(|x|^m)$ telle que pour tout y_{2n} de taille $O(|x|^m)$ la proposition $P(x, y_1, \ldots, y_{2n})$ soit vraie ?

Alors APTIME=PSPACE. On peut formaliser APTIME en utilisant la notion de machines de Turing *alternantes* [CKS81]. Ce sont des machines qui peuvent soit effectuer des étapes de calcul normales, soit des choix existentiels (se dupliquer en autant de machines filles qu'il y a

de valeurs possibles pour y_{2i-1}, et retourner "oui" si et seulement si au moins une des machines filles retourne "oui"), soit des choix universels (se dupliquer en autant de machines filles qu'il y a de valeurs possibles pour y_{2i-1}, et retourner "oui" si et seulement si toutes les machines filles retournent "oui"). APTIME est alors la classe des problèmes décidables en temps polynomial sur une machine alternante. Shapiro [Sha84] montre que les machines alternantes existent en pratique, en un sens : ce sont essentiellement les programmes Prolog, et la notion de temps d'un calcul alternant correspond à la profondeur d'une dérivation par hyperrésolution positive.

Le problème typique qui est complet pour PSPACE est QBF ("Quantified Boolean Formulae") :

ENTRÉE : une formule $F = Q_1 x_1 \cdot \ldots \cdot Q_n x_n \cdot S$, où S est un ensemble de clauses propositionnelles sur les variables $x_1, \ldots, x_n$, et les Q_i sont des quantificateurs $\forall$ ou $\exists$;
QUESTION : $\models F$?

On note que F est soit valide soit insatisfiable. On reconnaît l'alternance de quantificateurs caractéristique de APTIME. En particulier, NP$\subseteq$PSPACE.

Un autre problème PSPACE-complet typique est la vacuité de l'intersection d'un ensemble d'automates de mots (déterministes) [Koz77].

La classe DEXPTIME est celle des problèmes décidable en temps (déterministe) exponentiel. Par *exponentiel* on entend de la forme $O(2^{|x|^k})$ pour un certain $k \geq 1$, c'est-à-dire borné par l'exponentielle d'un polynôme en la taille de l'entrée. Un problème DEXPTIME-complet typique est la vacuité de l'intersection d'un ensemble d'automates d'arbres (déterministes) [Sei94].

On a PSPACE$\subseteq$DEXPTIME. De toutes les classes de complexité mentionnées ci-dessus, la seule inclusion stricte connue est P$\not\subseteq$DEXPTIME. On conjecture que toutes les autres inclusions sont strictes.

Un autre résultat remarquable de [CKS81] est que DEXPTIME=APSPACE ("Alternating Polynomial Space"), la classe des problèmes décidables sur une machine alternante en temps fini quelconque mais espace seulement polynomial.

En fait, [CKS81] montre que l'alternance transforme le temps en espace, et l'espace en une exponentielle du temps. On a vu que PSPACE=APTIME, mais on a par exemple aussi que la classe EXPSPACE des problèmes décidables en espace exponentiel sur une machine déterministe (ou non déterministe, de façon équivalente, par le théorème de Savitch) est identique à la classe AEXPTIME des problèmes décidables en temps alternant exponentiel. De même, DEXPTIME=APSPACE, mais on a aussi D2EXPTIME=AEXPSPACE, c'est-à-dire que la classe des problèmes décidables en temps doublement exponentiel (borné par $O(2^{2^{|x|^k}})$, $k \geq 1$) sur une machine déterministe est exactement la classe des problèmes décidables en espace exponentiel sur une machine alternante.

Il existe bien d'autres classes de complexité [Joh90]. La seule que nous utilisons dans ces notes et que nous n'avons pas encore défini est NEXPTIME, la classe des problèmes décidables en temps non déterministe exponentiel. Elle se définit exactement comme NP, en remplaçant les bornes polynomiales $O(|x|^m)$ par des bornes exponentielles $O(2^{|x|^m})$.

Les inclusions strictes connues entre les classes que nous avons mentionnées ici sont

P$\not\subseteq$DEXPTIME, NP$\not\subseteq$NEXPTIME, PSPACE$\not\subseteq$EXPSPACE, DEXPTIME$\not\subseteq$D2EXPTIME

Les autres sont conjecturées.

C Forme clausale définitionnelle

Pour convertir une formule F en un ensemble de clauses S, une façon intelligente est d'utiliser une mise en *forme clausale définitionnelle*. Une mise en forme clausale classique prend un temps en général exponentiel, une mise en forme clausale définitionnelle prend toujours un temps linéaire. On pourra consulter [FLHT01].

L'idée est de créer un nouveau symbole de prédicat servant d'abréviation pour chaque sous-formule de F, et d'écrire des clauses exprimant les relations entre ces prédicats.

On dit que F est une *sous-formule immédiate* de $F \wedge G$, $G \wedge F$, $F \vee G$, $G \vee F$, $\forall x \cdot F$, $\exists x \cdot F$. La relation *sous-formule* est la clôture réflexive transitive de la relation sous-formule immédiate.

Pour chaque sous-formule G de F, de variables libres y_1, ..., y_m, créons un nouveau symbole de prédicat m-aire P_G, posons $L_G = P_G(y_1, \ldots, y_m)$, et créons les *clauses structurelles* :

$-L_G \vee G$	si G est un littéral
$-L_G \vee +L_{G_1}, \quad -L_G \vee +L_{G_2}$	si $G = G_1 \wedge G_2$
$-L_G \vee +L_{G_1} \vee +L_{G_2}$	si $G = G_1 \vee G_2$
$-L_G \vee +L_{G_1}$	si $G = \forall x \cdot G_1$
$-L_G \vee +L_{G_1}$	si $G = \exists x \cdot G_1, x$ pas libre dans G_1
$-L_G \vee +P_{G_1}(y_1, \ldots, y_m, f_G(y_1, \ldots, y_m))$	si $G = \exists x \cdot G_1, x$ libre dans G_1

où, dans le dernier cas, $y_1, \ldots, y_m$ sont les variables libres de G, et f_G est un symbole de fonction. On suppose que si G et G' sont deux sous-formules distinctes, alors f_G et $f_{G'}$ sont deux symboles de fonctions distincts. (Il s'agit de fonctions de *Skolem*. On notera qu'on peut relaxer cette condition en demander à ce que f_G et $f_{G'}$ soient distincts si G et G' ne sont pas logiquement équivalentes.)

Soit $\mathcal{D}(F)$ l'ensemble des clauses ci-dessus, union la clause $+L_F$. Alors F est satisfiable si et seulement si $\mathcal{D}(F)$ l'est. En effet, si F est satisfiable, on peut étendre un modèle I, de domaine D, satisfaisant F en définissant :

- I_{f_G}, pour toute sous-formule $G = \exists x \cdot G_1$ telle que x est libre dans G_1, comme étant n'importe quelle fonction qui à $v_1, \ldots, v_n \in D$ associe une valeur v telle que $I, [x_1 := v_1, \ldots, x_n := v_n, x := v] \models G_1$ (ceci existe, par l'axiome du choix) ;
- I_{P_G} de sorte que $I, \rho \models L_G$ si et seulement si $I, \rho \models G$, pour toute sous-formule G de F ;

Cette structure étendue satisfait trivialement les clauses structurelles, et satisfait $+L_F$ puisqu'elle satisfait F.

Réciproquement, si $\mathcal{D}(F)$ est satisfiable, pour tout modèle I de $\mathcal{D}(F)$, pour toute sous-formule G de F, il est facile de voir que $I, \rho \models L_G$ implique $I, \rho \models G$, donc I est un modèle de F.