Automates d'arbres d'ordre deux

Rapport de stage de DEA

Version 1

17 novembre 2004

Benjamin Ratti

Table des matières

Première partie

Présentation du stage

Chapitre 1

Sujet du stage

Lieu : Laboratoire Spécification et Vérification
École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan CEDEX

Titre : Automates d'arbres d'ordre deux

Description du stage : Si vous avez suivi le cours d'option « Vérification de protocoles cryptographiques et démonstration automatique », vous êtes convaincu que la recherche de modèles d'ensembles de clauses du premier ordre est un outil fondamental pour la preuve de sécurité de protocoles cryptographiques. Des recherches récentes (Hubert Comon, Jean Goubault-Larrecq, Stéphanie Delaune, à paraître) suggèrent que certaines formes de logique du second ordre seraient utiles, notamment pour raisonner sur des attaques par dictionnaire. En l'occurrence, il semble que l'utilisation de formes clausales du premier ordre mais portant sur des termes du *second ordre* permette de représenter des modèles d'attaques intéressants.

L'intuition qui sous-tend ce genre de formes clausales est celle du papier [Gou02], où de telles formes clausales portaient sur des termes d'ordre supérieur — pas seulement d'ordre deux. Mais les formes clausales de [Gou02] semblent trop peu expressives pour coder les attaques par dictionnaire, car les termes d'ordre supérieur y sont restreints à être des *patterns* de Miller.

Le but du stage est de dégager une forme plus expressive de clauses, incluant des termes d'ordre supérieur, qui permette de modéliser de façon satisfaisante des attaques par dictionnaire dans le style évoqué plus haut, et dont la satisfiabilité soit décidable. Il semble que la restriction à l'ordre deux soit une restriction indispensable si l'on souhaite se libérer de la contrainte d'utiliser des patterns.

Ce stage demandera de bien étudier [Gou02] (une continuation naturelle du cours « Vérification de protocoles cryptographiques et démonstration automatique »), de se familiariser avec les techniques de résolution et l'unification du second ordre. Un guide fort sera d'utiliser le résultat de Prehofer selon lequel l'unification de patterns linéaires avec des termes du second ordre est décidable et finitaire [Pre94]. On pourra expérimenter l'algorithme proposé en se fondant sur l'outil MOP [Rog03].

Ce travail débouche naturellement sur un travail de thèse, qui pourra s'incurver vers l'extension de modèles à la Dolev-Yao, dans le cadre du projet RNTL Prouvé, ou vers l'exploration de classes décidables de logiques plus riches que la logique du premier ordre.

Bibliographie :

[Gou02] Jean Goubault-Larrecq. Higher-Order Positive Set Constraints. In *Proc. Conf. Computer Science Logic (CSL'02), Edinburgh, Scotland, September 2002*,

[Pre94] Christian Prehofer. Decidable Higher-Order Unification Problems. In *Proc. 12th Intl. Conf. Automated Deduction (CADE-12), Nancy, France, June 1994*, disponible sur CiteSeer (demander à Google : "Prehofer decidable higher-order").

[Rog03] Muriel Roger. L'outil modulaire de démonstration automatique MOP. LSV, 2003.

Personne(s) encadrant le stage :

Jean Goubault-Larrecq
Tél : 01 47 40 75 68
Email :

Chapitre 2

Présentation de l'équipe d'accueil

2.1 Le LSV

J'ai effectué mon stage au sein du Laboratoire Spécification et Vérification (LSV) de l'École Normale Supérieure de Cachan. La principale thématique du laboratoire est l'informatique de la vérification, c'est-à-dire l'ensemble des méthodes formelles permettant d'accroître la confiance dans les logiciels critiques. La recherche au sein du LSV est organisée autour de trois principaux axes :

- INFINI : Vérification symbolique des sytèmes complexes.
- SECSI : Sécurité des systèmes d'information.
- Vérification de systèmes temporisés.

2.2 L'axe SECSI

SECSI est un axe de recherche sur la sécurité des systèmes d'information. Traditionnellement, le premier problème de sécurité est la confidentialité : *certaines données doivent rester confinées à un cercle d'initiés*. Vu sous un autre angle, il s'agit de contrôler l'accès à certaines ressources.

Ce problème n'est pas nouveau et se pose depuis très longtemps dans les systèmes d'exploitation. Ce qui change essentiellement est qu'on s'intéresse à ces problèmes dans un environnement distribué dont les canaux de communication ne sont pas fiables.

Mais la confidentialité est loin d'être la seule propriété de sécurité intéressante. Tout aussi primordiale pour contrôler l'accès à certaines ressources est l'authentification. Des problèmes tels que l'anonymat, le délit d'initié, etc ..., sont aussi importants pour différentes transactions.

Les recherches au sein de cet axe sont organisées autour de trois thèmes principaux qui sont la *vérification automatique de protocoles cryptographiques*, *la détection d'intrusions* et *l'analyse statique de programmes.*

2.3 Vérification automatique de protocoles cryptographiques

La première étape de la vérification d'un protocole cryptographique étant sa spécification formelle, ils s'intéressent d'abord à des modèles formels décrivant les protocoles et leurs propriétés de sécurité, ainsi qu'à une définition rigoureuse de la relation de satisfaction entre protocoles et propriétés. Puis ils travaillent à la mise au point des techniques automatiques pour décider (ou semi-décider) cette relation de satisfaction, notamment des techniques basées sur les automates, les contraintes ensemblistes, et des techniques de déduction dans la logique du premier ordre. Finalement, il s'agit aussi du développement d'outils se fondant sur les méthodes développées.

2.4 Insertion du sujet de stage dans le thème de recherche

Le sujet de mon stage s'incrit dans le thème de recherche *Vérification automatique de protocoles cryptographiques*. En effet, l'objectif de mon stage est dégager une forme de clauses, incluant des termes d'ordre supérieur, qui permette de modéliser de façon satisfaisante des attaques par dictionnaire, et dont la satisfiabilité soit décidable.

Chapitre 3

Objectif

Dolev et Yao ont introduit dans [1], un modèle déduction de l'intrus qui permettait de modéliser certaines capacités d'un intrus comme le chiffrement, le déchiffrement, la composition et la décomposition. Notre idée est de lui permettre de pouvoir appliquer des fonctions quíl connait à des termes qu'il a pu récupérer lors de l'éxécution du protocole qu'il tente d'attaquer. Pour cela, nous souhaitons ajouter une règle de deduction à l'intrus de Dolev-Yao [1] qui permettrait de modiliser l'application d'une fonction :

$$\frac{E \vdash M : A \rightarrow B \qquad E \vdash N : A}{E \vdash MN : B}$$

Pour pouvoir appliquer un algorithme de résolution, il faut traduire toutes les règles de déductions de l'intrus sous forme de clauses de Horn. On obtient les clauses "classiques" comme $k(X) \Leftarrow k(\{X\}_Y), k(Y)$, ainsi qu'une nouvelle clause de la forme $k(MN) \Leftarrow k(M), k(N)$.

Le problème que cette nouvelle clause nous apporte est qu'elle n'appartient pas à une classe de clause où la satifiabilité est décidable. En se fixant des limites comme la restriction à des termes du second ordre (ce qui correspond bien à ce que l'on souhaite faire si l'on s'intéresse par exemple à des attaques par dictionnaire).

Nos objectifs sont :

1. Trouver une classe $\mathcal{C}$ pour laquelle l'unification soit décidable et qui accepte MN dans les têtes de ses clauses.
2. Trouver un ordre stable et une fonction de sélection pour que la résolution ordonnée par sélection termine.

Une fois que nous aurons atteint ces deux objectifs, il nous sera alors possible d'appliquer un algorithme de résolution ordonnée avec selection à notre ensemble de clauses obtenu à partir des règles de déduction de l'intrus, de la nouvelle règle et des clauses obtenues à partir du protocole à vérifier.

Chapitre 4

Prélimimaires

Dans ce chapitre, nous allons présenter les termes que nous manipulerons par la suite. Le calcul que nous utiliserons sera la λ-calcul typé. Notre objectif n'est pas de donner une définition détaillés du λ-calcul typé mais plutot d'en faire une présentation rapide.

4.1 λ-calcul typé

Par la suite, nous allons manipuler des λ-termes. Nous allons donc dans cette section définir les différents composants du λ-calcul typé, c'est-à-dire les types, les λ-termes, les règles de typage ainsi que trois règles de conversion.

4.1.1 Types

On considère un ensemble $\mathcal{S}$ de types de base. L'ensemble $\mathcal{T}_\mathcal{S}$ de types est l'ensemble des types que l'on peut construire à partir des types de bases $\mathcal{S}$ avec la règle suivante :

$$\mathcal{T}_\mathcal{S} := \mathcal{S} \mid \mathcal{T}_\mathcal{S} \rightarrow \mathcal{T}_\mathcal{S}$$

Une déclaration de type est une expression de la forme $\sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \sigma$ où $\sigma_1, \ldots, \sigma_n, \sigma$ sont des types. Etant donné que dans notre travail nous allons nous restreindre au second ordre, les types $\sigma_1, \ldots, \sigma_n, \sigma$ doivent être nécessairement des types de base.

4.1.2 Termes

Soit $\mathcal{F}$ un ensemble de symboles de fonctions auxquels sont associées une déclaration de type et $\mathcal{X}$ un ensemble de variable. On définit l'ensemble des λ-termes à l'aide de la grammaire suivante :

$$\mathcal{T} := \mathcal{X} \mid (\lambda\mathcal{X} \cdot \mathcal{T}) \mid (\mathcal{T}\ \mathcal{T}) \mid \mathcal{F}(\mathcal{T}, \ldots, \mathcal{T})$$

On définit trois ensembles associés à un λ-terme : Var, FV et BV qui correspondent respectivement à l'ensemble des variables, l'ensemble des variables libres et l'ensemble des variables liées.

On note explicitement $@(u, v)$, l'application de v à u. Par extension, on note $@(u, \overline{v})$ l'application $(\ldots(uv_1)\ldots v_n)$.

Définition 1 (left-flattening) *Le terme $@(u, \overline{v})$ est appelé le left-flattening de $s \equiv (\ldots(uv_1)\ldots v_n)$. Le left-flattening est partiel si l' application est partielle.*

Variables :

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

Fonctions :

$$\frac{\begin{array}{c} F : \sigma_1 \times \cdots \times \sigma_n \to \sigma \in \mathcal{F} \\ \Gamma \vdash t_1 : \sigma_1 \ \ldots \ \Gamma \vdash t_n : \sigma_n \end{array}}{\Gamma \vdash F(t_1, \ldots, t_n) : \sigma}$$

Abstraction :

$$\frac{\Gamma \ \cup \ \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash (\lambda x : \sigma \cdot t) : \sigma \to \tau}$$

Application :

$$\frac{\Gamma \vdash s : \sigma \to \tau \quad \Gamma \vdash t : \sigma}{\Gamma \vdash (s\ t) : \tau}$$

Congruence de type :

$$\frac{\Gamma \vdash s : \sigma}{\Gamma \vdash s : \tau} \text{ si } \sigma \equiv \tau$$

TAB. 4.1 – Règles de typage du λ-calcul typé

Exemples

- $@(\lambda x_1 x_2 \cdot x_1, X, Y)$ est le left-flattening de $(\lambda x_1 x_2 \cdot x_1) X\ Y$.
- $@(\lambda x_1 x_2 \cdot x_1, X)$ est un left-flattening partiel.

4.1.3 Règles de typage

Nous avons définie les λ-termes ainsi que les types. Nous allons maintenant définir ce qu'est le λ-calcul typé. Il s'agit en fait d'associer aux λ-termes des types et de restreindre le λ-calcul à l'ensemble des λ-termes correctement typés, c'est-à-dire qui respectent les règles de typage 4.1.

Un environnement Γ est un ensemble de paires $x : \sigma$ où x est une variable et σ est un type et tel que la même variable n'apparait qu'une fois au plus.

4.1.4 Règles de conversion

Le λ-calcul contient trois règles de conversion qui sont les suivantes :

$$\begin{array}{ll} (\lambda x \cdot v) u \to_\beta v\{x \mapsto u\} & \\ \lambda x \cdot (ux) \to_\eta u & \text{si } x \notin Var(u) \\ \lambda x \cdot v \to_\alpha \lambda y \cdot v\{x \mapsto y\} & \text{si } y \notin BV(v) \cup (Var(v) \setminus \{x\}) \end{array}$$

4.2 Notations

Dans toute cette partie, nous allons manipuler des λ-termes. Pour simplifier la tâche, nous allons utiliser un certain nombre de notations que nous allons définir maintenant.

Notation 1 *On note* $\lambda x_1 \dots x_n \cdot \alpha$ *le terme* $\lambda x_1 \cdot \lambda x_2 \cdot \dots \lambda x_n \cdot \alpha$.

Notation 2 *On note le* λ*-terme* $\lambda x_1 \dots x_n \cdot \alpha$, $\lambda \overline{x} \cdot \alpha$. *On peut aussi indicer* x *pour expliciter le nombre de variables.*

Exemple Le λ-terme $\lambda x_1 x_2 x_3 x_4 \cdot \alpha$ peut être noté $\lambda \overline{x} \cdot \alpha$ ou encore $\lambda \overline{x}_4 \cdot \alpha$.

Notation 3 *Soit* π *une fonction bijective de* $\{1, \dots, n\}$ *sur* $\{1, \dots, n\}$. *on note* $\overline{x}_{|\pi}$ *le terme* $x_{\pi(1)} \dots x_{\pi(i)} \dots x_{\pi(n)}$. *On peut étendre cette notation au cas ou* π *est injective.*

Exemple On définit deux fonctions $\pi_1 : \begin{array}{l} 1 \mapsto 2 \\ 2 \mapsto 3 \\ 3 \mapsto 1 \end{array}$ et $\pi_2 : \begin{array}{l} 1 \mapsto 2 \\ 2 \mapsto 3 \end{array}$.

$\overline{x}_{|\pi_1}$ représente $x_2\ x_3\ x_1$ et $\overline{x}_{|\pi_2}$ représente $x_2\ x_3$.

Remarque Dans tout le document, nous allons manipuler des λ-termes typés mais pour simplifier la lecture nous n'allons pas spécifier les types des termes dans la plupart des cas.

Deuxième partie

Unification d'ordre deux

Chapitre 5

L'état de l'art

5.1 Décidabilité de l'unification d'ordre deux

Le tableau 5.1 nous indique que dans le cas général, l'unification de terme d'ordre deux n'est pas décidable. Cependant, il existe tout de même certaines classes de termes d'ordre deux pour lesquelles l'unification est décidable. Notre objectif sera donc de trouver une classe de terme dont l'unification est décidable et qui contienne les termes dont on a besoin pour la résolution.

Cas général	Pattern de Miller	Termes Monadique
indécidable Goldfarb '81 [2] Farmer '91 [5]	décidable D. Miller '91 [3]	décidable Farmer '88 [4]

Tab. 5.1 – Décidabilité de l'unification pour la logique du second ordre, [6]

5.2 Pattern de Miller

D. Miller [3] a présenté une classe de termes dont l'unification est décidable pour des termes d'ordres supérieurs.

Définition 2 (Pattern de Miller) *Les* patterns de Miller *sont des λ-termes dont les variables libres sont appliquées uniquement à des variables liées distinctes.*

Exemples

$\lambda x_1, x_2, x_3 \cdot X x_3 x_1$	est un pattern
$\lambda x_1, x_2, x_3 \cdot X(X x_3 x_1)$	n'est pas un pattern
$\lambda x_1, x_2, x_3 \cdot X x_1 x_1 x_3$	n'est pas un pattern

Suppression		
$\{t = t\} \cup S$	$\Rightarrow$	S
Décomposition		
$\{\lambda\overline{x_k} \cdot f(\overline{t_n}) = \lambda\overline{x_k} \cdot f(\overline{t'_n})\} \cup S$	$\Rightarrow$	$\bigcup_{i=1,\ldots,n}\{\lambda\overline{x_k} \cdot t_i = \lambda\overline{x_k} \cdot t'_i\} \cup S,\ f \in \mathcal{F}$
Elimination		
$\{F = \lambda\overline{x_k} \cdot t\} \cup S$	$\Rightarrow$	$\{F = \lambda\overline{x_k} \cdot t\} \cup \{F \mapsto \lambda\overline{x_k} \cdot t\}S$ si $F \notin \mathcal{FV}(\lambda\overline{x_k} \cdot t)$ et $tete(t)$ n'est pas une variable libre
Imitation		
$\{\lambda\overline{x_k} \cdot F(\overline{t_n}) = \lambda\overline{x_k} \cdot f(\overline{t'_n})\} \cup S$	$\Rightarrow$	$\{F = \lambda\overline{x_k} \cdot f(\overline{H_m(\overline{x_n})})\} \cup$ $\{\lambda\overline{x_k} \cdot F(\overline{t_n}) = \lambda\overline{x_k} \cdot f(\overline{t'_n})\} \cup S$
Projection		
$\{\lambda\overline{x_k} \cdot F(\overline{t_n}) = \lambda\overline{x_k} \cdot v(\overline{t'_n})\} \cup S$	$\Rightarrow$	$\{F = \lambda\overline{x_n} \cdot x_i(\overline{H_m(\overline{x_n})})\} \cup$ $\{\lambda\overline{x_k} \cdot F(\overline{t_n}) = \lambda\overline{x_k} \cdot v(\overline{t'_n})\} \cup S$ où v est une constante ou une variable liée

TAB. 5.2 – Algorithme d'unification d'Huet [7]

5.3 Algorithme d'Unification d'Huet

Le tableau 5.2 présente l'algorithme d'unification d'Huet pour des λ-termes typés. Dans notre cas, comme nous nous sommes restreints à des termes d'ordre deux, nous pouvons simplifier la dernière règle (Projection) par la suivante :

$$\{\lambda\overline{x_k} \cdot F(\overline{t_n}) = \lambda\overline{x_k} \cdot v(\overline{t'_n})\} \cup S \quad \Rightarrow \quad \begin{array}{l} \{F = \lambda\overline{x_n} \cdot x_i\} \cup \\ \{\lambda\overline{x_k} \cdot F(\overline{t_n}) = \lambda\overline{x_k} \cdot v(\overline{t'_n})\} \cup S \\ \text{où } v \text{ est une constante ou une variable liée} \end{array}$$

Cela est possible car le terme $\lambda\overline{x_n} \cdot x_i(\overline{H_m(\overline{x_n})})$ doit être du second ordre, ce qui signifie que x_i est du permier ordre dont ne peut être appliqué à aucun terme.

Dans le cas où l'on souhaite unifier termes comme des patterns de Miller [3], l'algorithme de Huet n'est pas complet. Miller a montré la décidabilité de l'unification des patterns mais le même problème se pose lorsque l'on souhaite unifier des patterns avec de termes d'ordre deux quelconques. Par la suite, nous aurons justement besoin d'unifier des termes pour lesquels l'algorithme d'Huet ne décide pas l'unification. Il nous faudra donc essayer de trouver un algorithme qui décide l'unification de nos termes ou bien éviter d'avoir à unifier ces termes.

Chapitre 6

Unification

Lorsque l'on traduit la règle que l'on a ajoutée au système de déduction de l'intrus de Dolev-Yao sous forme de clause de Horn, on obtient la clause $P(M\ N) \Leftarrow P'(M), P(N)$. Si l'on souhaite savoir si un ensemble de clauses contenant cette dernière clause, on pourrait vouloir appliquer un algorithme de résolution comme nous le présenterons dans la partie suivante. Le problème qui se pose alors est la résolution de la règle suivante :

$$\frac{\begin{array}{ll} P(t) \Leftarrow & P_1(t_1), reste_1 \\ & P_1(M\ N) \Leftarrow reste_2 \end{array}}{P(t\sigma) \Leftarrow \quad reste_1\sigma, reste_2\sigma}$$

où $\sigma \in csu(t_1 \doteq M\ N)$. Le terme t_1 peut être un pattern de Miller. Pour pouvoir appliquer la règle, il faut tout d'abord pouvoir résoudre ce problème d'unification.

Dans ce chapitre, nous allons donc essayer d'unifier MN avec un terme de la forme $t := \lambda\overline{x} \cdot u$ avec $\begin{array}{lll} u & := & X\ \overline{x}_{|\pi} \\ & | & x_i \\ & | & f(u, \ldots, u) \end{array}$

Dans le cas où le terme que l'on souhaite a unifier avec MN n'est pas de la forme $\lambda\overline{x} \cdot X\ \overline{x}_{|\pi}$, il suffit simplement d'appliquer l'algorithme d'unification d'Huet et il n'y a pas de problème qui se pose.

- $MN = \lambda\overline{x} \cdot x_i$
  On applique la règle de projection et l'on obtient l'unificateur $\{M \mapsto \lambda y\overline{x} \cdot x_i\}$.
- $MN = \lambda\overline{x} \cdot f(t_1, \ldots, t_n)$
  Dans ce cas là, les règles d'imitation et de projection sont applicables
  Imitation Par application de la règle d'imitation, on obtient l'unificateur

  $$\sigma = \{M \mapsto \lambda y\overline{x} \cdot f(H_1 y\overline{x}, \ldots, H_n y\overline{x})\} \cup \bigcup_{i=1,\ldots,n} \sigma_i$$

  où σ_i est la solution du problème d'unification $H_i N = \lambda\overline{x} \cdot t_i$ et $H_1, \ldots, H_n$ sont des variables fraiches.
  Projection On peut appliquer la projection uniquement si $\overline{x} \notin \mathcal{FV}(f(t_1, \ldots, t_n))$. Dans ce cas là, on obtient l'unificateur $\sigma = \{M \mapsto \lambda y\overline{x} \cdot y, N \mapsto f(t_1, \ldots, t_n)\}$

Malheureusement, on ne peut pas appliquer l'algorithme d'Huet dans le cas où le terme a unifier à MN est de la forme $\lambda\overline{x} \cdot X\ \overline{x}_{|\pi}$

$$
\begin{array}{lll}
(1) & \{\lambda\overline{x}\cdot(M\overline{x})(N\overline{x}) = (\lambda\overline{x}\cdot X\ \overline{x}_{|\pi}\}\ \cup S & \Rightarrow \begin{array}[t]{l}\{M = \lambda\overline{x}y\cdot M'\overline{x}'y\}\cup \\ \{N = \lambda\overline{x}\cdot N'\overline{x}'\}\cup \\ \{\lambda\overline{x}'\cdot(M'\overline{x}')(N'\overline{x}') = \lambda\overline{x}'\cdot X\ \overline{x}_{|\pi}\}\cup S \\ \text{où } x_i\in\overline{x}\setminus\overline{x}_{|\pi} \text{ et } \overline{x}'=\overline{x}\setminus\{x_i\}\end{array} \\
(2) & \{\lambda\overline{x}\cdot(M\overline{x})(N\overline{x}) = \lambda\overline{x}\cdot X\ \overline{x}_{|\pi}\}\ \cup S & \Rightarrow \begin{array}[t]{l}\{M = \lambda\overline{x}y\cdot X\ \overline{x}_{|\pi}\}\ \cup S \\ \text{où } x_i\in\overline{x}\setminus\overline{x}_{|\pi} \text{ et } \overline{x}'=\overline{x}\setminus\{x_i\}\end{array} \\
(3) & \{\lambda\overline{x}\cdot(M\overline{x})(N\overline{x}) = (\lambda\overline{x}\cdot X\ \overline{x}_{|\pi}\}\ \cup S & \Rightarrow \{X = \lambda\overline{x}\cdot(M\overline{x}_{|\pi^{-1}})(N\overline{x}_{|\pi^{-1}})\}\ \cup S
\end{array}
$$

TAB. 6.1 – Règles d'unification

6.1 Unification de $\lambda\overline{x}\cdot(M\overline{x})(N\overline{x})$ avec un pattern de Miller

Je ne vais présenter ici qu'un seul problème d'unification que j'ai traité durant mon stage. Cela me permet en effet de présenter une preuve de complétude dont le principe de base est le même pour tous les autres problèmes de complétude que j'ai pu traiter par la suite. Nous allons donc nous intéresser d'unifier $\lambda\overline{x}\cdot(M\overline{x})(N\overline{x})$ avec un terme de la forme

$$
\begin{array}{rcl}
t & ::= & \lambda\overline{x}\cdot u \\
\\
u & ::= & X\overline{x}_{|\pi} \\
 & | & x_i \\
 & | & f(u,\ldots,u)
\end{array}
$$

Théorème 1 *Ce système de règles 6.1 est complet.*

Preuve Soit le problème d'unification $\lambda\overline{x}\cdot(M\overline{x})(N\overline{x}) = \lambda\overline{x}\cdot X\ \overline{x}_{|\pi}$.

Soit σ une substitution close qui est solution de ce problème. On a

$$
\sigma : \begin{array}[t]{rcl}\sigma(M) & = & \lambda\overline{x}y\cdot s \\ \sigma(N) & = & \lambda\overline{x}\cdot t \\ \sigma(X) & = & \lambda\overline{y}\cdot u\end{array}
$$

Comme σ est une substitution close, on a $FV(u[\overline{y := x_\pi}]) \subseteq \overline{x}_{|\pi}$.
Soit une variable z telle que $z\in\overline{x}$ et $z\notin\overline{x}_{|\pi}$. Supposons que $z\notin s[y:=t]^{\downarrow}$. On remarquera qu'au second ordre

$$
\begin{array}{rcl}
s[y:=t]^{\downarrow} & = & s^{\downarrow}[y:=t^{\downarrow}] \\
 & = & s[y:=t]
\end{array}
$$

– si $z\notin FV(s)$ et $z\notin FV(t)$, cela signifie que $\lambda\overline{x}\cdot t$ n'utilise pas l'argument z, On a donc $\sigma=\theta\sigma'$ avec

$$
\theta = \left\{\begin{array}{l} M\mapsto\lambda\overline{x}y\cdot M'(\overline{x}\setminus\{z\})y \\ N\mapsto\lambda\overline{x}\cdot N'(\overline{x}\setminus\{z\})\end{array}\right\}
$$

et σ' est l'unificateur de $\lambda\overline{x}'\cdot(M'\overline{x}')(N'\overline{x}') = \lambda\overline{x}'\cdot X\ \overline{x}_{|\pi}$ où $\overline{x}' = \overline{x}\setminus\overline{x}_{|\pi}$.

– si $z \notin FV(s)$ et $z \in FV(t)$, alors cela signife que $\lambda\overline{x}y \cdot s$ n'utilise pas son argument y. On a alors $\sigma = \theta\sigma'$ avec $\theta_1 = \{M \mapsto \lambda\overline{x}y \cdot M'\overline{x}\}$ et σ' est un unificateur de $\lambda\overline{x} \cdot M'\overline{x} = \lambda\overline{x} \cdot X\overline{x}_{|\pi}$.
  On remarquera que $csu(\lambda\overline{x} \cdot M'\overline{x}, \lambda\overline{x} \cdot X\overline{x}_{|\pi}) = \{\ \{M' \mapsto \lambda\overline{x} \cdot X\ \overline{x}_{|\pi}\}\ \}$. On a donc $\sigma' = \theta_2\sigma'$ avec $\theta_2 = \{M' \mapsto \lambda\overline{x} \cdot X\ \overline{x}_{|\pi}\}$. On a donc $\sigma = \theta_1\theta_2\sigma''$. En simplifiant $\theta_1\theta_2$, on obtient $\sigma = \theta\sigma''$ avec

$$\theta \ = \ \left\{\ M \mapsto \lambda\overline{x}y \cdot X\overline{x}_{|\pi}\ \right\}$$

  et σ'' est une substitution close.
– Le dernier cas est celui où π est une bijection. On a alors à résoudre

$$\lambda\overline{x} \cdot (M\ \overline{x})(N\ \overline{x}) = \lambda\overline{x} \cdot X\ \overline{x}_{|\pi}$$

  Cela revient à écrire que $X = \lambda\overline{x}_{|\pi^{-1}} \cdot (M\ \overline{x})(N\ \overline{x})$. On a donc $\sigma = \theta\sigma'$ avec

$$\theta = \{X \mapsto \lambda\overline{x} \cdot (M\ \overline{x}_{|\pi^{-1}})(N\ \overline{x}_{|\pi^{-1}})\}$$

  et σ' est une substitution close.

$\square$

Pour mieux comprendre à quoi correspond ces règles, nous allons les appliquer à un exemple assez simple.

Exemple On cherche à unifier les deux termes $\lambda x_1x_2 \cdot (Mx_1x_2)(Nx_1x_2)$ et $\lambda x_1x_2 \cdot Xx_2$.

Il est possible d'appliquer les deux premières règles. On a donc

$$\begin{array}{lcl}
\{\lambda x_1x_2 \cdot (Mx_1x_2)(Nx_1x_2) = \lambda x_1x_2 \cdot Xx_2\} & \rightarrow_{(2)} & \{M \mapsto \lambda x_1x_2y \cdot Xx_2\} \\
\downarrow_{(1)} & & \\
\left\{\begin{array}{l} M \mapsto \lambda x_1x_2y \cdot (M'x_2)y \\ N \mapsto \lambda x_1x_2 \cdot N'x_2 \\ \lambda x_2 \cdot (M'x_2)(N'x_2) = \lambda x_2 \cdot Xx_2 \end{array}\right\} & & \\
\downarrow_{(3)} & & \\
\left\{\begin{array}{l} M \mapsto \lambda x_1x_2y \cdot (M'x_2)y \\ N \mapsto \lambda x_1x_2 \cdot N'x_2 \\ X \mapsto \lambda x_2 \cdot (M'x_2)(N'x_2) \end{array}\right\} & &
\end{array}$$

On a donc comme résultat

$$csu(\lambda x_1x_2 \cdot (Mx_1x_2)(Nx_1x_2) \doteq \lambda x_1x_2 \cdot Xx_2) = \left\{\begin{array}{l} \left\{\begin{array}{l} M \mapsto \lambda x_1x_2y \cdot (M'x_2)y \\ N \mapsto \lambda x_1x_2 \cdot N'x_2 \\ X \mapsto \lambda x_2 \cdot (M'x_2)(N'x_2) \end{array}\right\}, \\ \{M \mapsto \lambda x_1x_2y \cdot Xx_2\} \end{array}\right\}$$

6.2 Changement de direction

En étudiant l'unification de ces termes, nous n'avons pas réussi à dégager de classe de termes telle qu'elle contiennent les termes qui nous intéressent mais qui soit fermée, c'est-à-dire que toutes applications d'une substitution donne un terme de cette classe à partir d'un format de clauses fixé. On souhaite garantir que le nombre de clause obtenu par résolution soit fini, ce qui n'est pas forcement évident avec les termes que l'on obtient après unification.

Dans un premier temps, on pourrait penser que nous sommes face à un problème qui nous empêche de résoudre notre problème de départ. Cependant, cela montre que lorsque nous appliquerons l'algorithme de résolution ordonnée avec sélection, nous devrons éviter d'avoir à unifier les termes $M\ N$ avec des termes de la forme $\lambda\overline{x}\cdot t$. Cela devra donc être une contrainte de notre fonction de sélection.

Troisième partie

Résolution ordonnée avec sélection

Chapitre 7

L'état de l'art

7.1 Algorithme de résolution

Une des techniques de démonstration automatique les plus connues et les plus efficaces pour détecter si un ensemble de clauses et insatisfiable, est la *résolution* inventée par J. Alan Robison en 1965. Cette méthode réside sur l'application successive de la règle de résolution.

7.2 Résolution binaire

La résolution binaire est composé de deux règles : la règle de résolution et la règle de factorisation.

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

$$\frac{C \vee +A \vee +A'}{C\sigma \vee +A\sigma} \; \sigma = csu(A \doteq A')$$

Théorème 2 *La résolution binaire avec factorisation est correcte et complète.*

Tab. 7.1 – Règle de résolution

La condition $\sigma = csu(A \doteq A')$ signifie que A et A' sont unifiables et que σ est leur mgu (unificateur le plus général).

7.3 Un raffinement de la résolution

Un raffinement de la résolution est la résolution ordonnée avec sélection. Nous allons tout d'abord présenter la résolution ordonnée puis la résolution ordonnée avec sélection.

7.3.1 Résolution ordonnée

La *résolution ordonnée* demande que :

- A soit maximal dans la clause $C \vee +A$, et A' maximal dans la clause $C' \vee -A'$, où $C \vee +A$ et $C' \vee -A'$ sont des prémisse de la règle de résolution binaire,
- A et A' soient tout deux maximaux dans la prémisse $C \vee +A \vee -A'$ de la règle de factorisation positive,

ceci pour un ordre 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 $B\sigma$ soient clos. Précisons qu'un atome A est maxiaml dans une clause C s'il n'existe pas de littéral $\pm B$ dans C tel que $B \succ C$.

7.3.2 Résolution ordonnée avec sélection

La *résolution ordonnée avec sélection* est similaire à la résolution ordonnée 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 une fonction de sélection pour déterminer sur quel littéral résoudre ; on n'utilise l'ordre que par défaut.

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

où :

1. $n \geq 1$
2. $\sigma \in csu(A_1 \doteq A', \ldots, A_n \doteq A')$
3. $\mathtt{sel}(C \vee +A_1 \cdots \vee +A_n) = \emptyset$ et $A_1, \ldots, A_n$ sont maximaux dans $C \vee +A_1 \vee \cdots \vee +A_n$
4. $-A' \in \mathtt{sel}(C' \vee -A')$, où bien $\mathtt{sel}(C' \vee -A') = \emptyset$ et A' est maximal dans $C' \vee -A'$.

TAB. 7.2 – Résolution ordonnée avec sélection au premier ordre

Dans [8], M. Roger a montrer que l'on pouvait étendre la résolution ordonnée avec sélection à des clauses contenant des termes d'ordre supérieurs de sorte à obtenir la complétude à condition que l'on possède un ordre stable sur les atomes.

7.4 Classes $\mathcal{H}_1$ et $\mathcal{H}_2$

Dans cette partie, nous allons étudier deux classes de clauses pour lesquelles la satisfiabilité est décidable.

7.4.1 La classe $\mathcal{H}_1$

Définition 3 (Classe $\mathcal{H}_1$) *La classe $\mathcal{H}_1$ est la classe de clauses de Horn de la forme :*

$$\begin{array}{rcl} \perp & \Leftarrow & P_1(t_1), \ldots, P_k(t_k) \\ P(X) & \Leftarrow & P_1(t_1), \ldots, P_k(t_k) \\ P(f(X_1, \ldots, X_n)) & \Leftarrow & P_1(t_1), \ldots, P_k(t_k) \end{array}$$

où dans le dernier cas, les variables $X_1, \ldots, X_n$ sont deux à deux disjointes.

Un résultat, présenté dans [9] qui reprends un résultat de [10], qui nous sera très utile par la suite est la décidabilité de la satisfiabilité de la classe $\mathcal{H}_1$.

Théorème 3 *La satisifabilité de la classe $\mathcal{H}_1$ est décidable et DEXPTIME-complète.*

7.4.2 La classe $\mathcal{H}_2$

Définition 4 (Classe $\mathcal{H}_2$) *On définie la classe $\mathcal{H}_2^k$, $k \geq 1$, comme la classe des clauses de $\mathcal{H}_1$ vérifiant les propriétés suivantes :*

(a) les variables libres de la tête apparaissent au plus une fois dans le corps de la clause.

(b) les variables libres du corps ont au plus k occurences.

Exemple $P(f(X_1, X_2)) \Leftarrow Q(X_1), R(g(X_2, X_3, X_3))$ est une clause de $\mathcal{H}_2^2$.

Théorème 4 *La satifiabilité de la classe $\mathcal{H}_2^k$ est décidable en temps polynomial.*

Chapitre 8

La classe $\mathcal{F}_1$

Dans ce chapitre, on va s'intéresser à un ensemble de clauses dont on s'attachera à démontrer la décidabilité de la satisfiabilité à l'aide de l'algorithme de résolution ordonnée avec sélection.

Définition 5 *On définie la classe $\mathcal{F}_1$ comme l'ensemble des clauses de la forme suivante :*
- $\perp \Leftarrow P_1(t_1), \ldots, P_k(t_k)$
- $P(X) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$
- $P(f(X_1, \ldots, X_n)) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$ *où* $X_1, \ldots, X_n$ *sont deux à deux distincts et* $f \in \mathcal{F}$.
- $P'(\lambda\overline{x} \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$, *où* $f \in \mathcal{F}$.
- $P(M\ N_1\ \ldots\ N_k) \Leftarrow P'(M), P_1(N_1), \ldots, P_k(N_k)$, *où* $N_1, \ldots, N_k$ *sont deux à deux distincts.*

où les prédicats P portent sur des termes du premier ordre et les prédicats P' sur des termes du second ordre.

La classe $\mathcal{F}_1$ a été construite comme un sur-ensemble de la classe $\mathcal{H}_1$. Par la suite, nous allons ainsi pouvoir nous appuyer sur la décidabilité de la classe $\mathcal{H}_1$ pour montrer que $\mathcal{F}_1$ est décidable.

Nous allons commencer par définir la fonction de sélection et l'ordre que nous allons utiliser sur dans l'algorithme de résolution par sélection.

8.1 Fonction de sélection

Nous allons maintenant definir la fonction de sélection que nous allons utiliser pour la résolution ordonnée avec sélection.

$$\mathbf{sel}(+A_0 \vee -A_1 \vee \cdots \vee -A_n) = \{-A_i \mid 1 \leq i \leq n, \delta(A_i) \geq \delta(A_0) \texttt{ et } \delta(A_i) \texttt{ est maximal}\}$$

avec δ qui est définie comme suit :

$$\delta(\perp) = (0, 0)$$
$$\delta(P(t)) = (\varphi(t), d(t) + 1)$$

La fonction d est la suivante :

$$\begin{aligned} d(\lambda x_1 \ldots x_n \cdot f(t_1, \ldots, t_m)) &= 1 + max_{i=1..m} d(t_i) \\ d(t_1 \ldots t_n) &= 1 + max_{i=1..m} d(t_i) \\ d(x) &= 0 \end{aligned}$$

La fonction φ prend un terme t et renvoie son ordre.

8.2 Ordre

Nous allons nous baser sur un ordre que Jouannaud et Rubio ont présenté dans [11], le RPO d'ordre supérieur ($\succ_{horpo}$). Cette ordre qui porte sur les termes d'ordre supérieur à les propriétés d'être décidable et stable à la substitution.

Définition 6 (The Higer-Order Recursive Path Ordering) *Soit $>_{\mathcal{F}}$ un ordre bien fondé sur $\mathcal{F}$.*

$s : \sigma \succ_{horpo} t : \tau$ si et seulement si $\sigma \equiv \tau$ et

1. *$s = f(\overline{s})$ avec $f \in \mathcal{F}$, et $s_i \succeq_{horpo} t$ pour un $s_i \in \overline{s}$*
2. *$f, g \in \{, f >_{\mathcal{F}} g$ et $t = g(\overline{t})$ et $\forall t_i \in \overline{t}$ $s \succ_{horpo} t_i$ or $s_j \succeq_{horpo} t_i$ pour un $s_j \in \overline{s}$*
3. *$f = g \in Mul$ et $\overline{s}(\succ_{horpo})_{mul}\overline{t}$*
4. *$f = g \in Lex$ et $\overline{s}(\succ_{horpo})_{lex}\overline{t}$, et $\forall t_i \in \overline{t}$ $s \succ_{horpo} t_i$ ou $s_j \succeq_{horpo} t_i$ ou $s_j \succeq_{horpo} t_i$ pour un $s_j \in \overline{s}$*
5. *$f \in \mathcal{F}, @(\overline{t})$ est un left-flattening partiel de t, et $\forall t_i \in \overline{t} s \succ_{horpo} t_i$ or $s_j \succeq_{horpo} t_i$ pour un $s_j \in \overline{s}$*
6. *$s = @(s_1, s_2)$, $t = @(t_1, t_2)$ et $\{s_1, s_2\}(\succ_{horpo})_{mul}\{t_1, t_2\}$*
7. *$s = \lambda x \cdot u$, $t = \lambda x \cdot v$ et $u \succ_{horpo} v$*

L'ordre dont nous avons besoin doit être un ordre sur les atomes. Nous allons tout simplement baser l'ordre des prédicats sur l'ordre des termes qu'ils contiennent.

Définition 7 (Ordre $\succ_\delta$) $P_1(t_1) \succ_\delta P_2(t_2) \Leftrightarrow t_1 \succ_{horpo} t_2$

Théorème 5 *L'ordre $\succ_\delta$ est décidable et stable par substitution.*

Preuve L'ordre $\succ_{horpo}$ est décidable et stable par substitution donc l'ordre $\succ_\delta$ est aussi décidable et stable par substitution.

$\square$

8.3 Fermeture

On peut remarquer tout de suite que les deux types de clauses qui ont été ajoutées pour former $\mathcal{F}_1$ ne permettent pas de générer des clauses qui ne seraient pas dans $\mathcal{H}_1$. Par la suite, on montrera que le nombre de clauses générés est fini et cela permettra de montrer la terminaison de l'algorithme de résolution ordonnée avec sélection.

Lemme 1 *Soit un ensemble $\mathcal{C}$ de clause de $\mathcal{F}_1$, Si l'on applique la règle de résolution à deux clauses de $\mathcal{C}$, alors la clause produite est dans $\mathcal{H}_1$.*

Preuve Soit un ensemble de clause $\mathscr{C}$ de $\mathcal{F}_1$. On peut déjà remarquer que les trois premiers types de clause décrivent la classe $\mathcal{H}_1$.

Dans [12], il a été montré que si l'on applique la règle de résolution à des clauses de $\mathcal{H}_1$, alors la clause produite est une clause $\mathcal{H}_1$. Il nous reste donc juste à montrer que l'on ne peut pas utiliser la règle de résolution avec une clause de $\mathcal{H}_1$ et une clause décrite par les deux derniers types de clauses et que si l'on applique la règle de résolution à des clauses de $\mathcal{F}_1 \setminus \mathcal{H}_1$, alors la clause produite est bien dans $\mathcal{H}_1$.

Soit C_1 une clause de $\mathcal{H}_1$ et C_2 une clause de $\mathcal{F}_1 \setminus \mathcal{H}_1$. Comme C_2 contient un terme d'ordre deux, alors la fonction de sélection aura sélectionné le littéral qui contient le terme d'ordre deux. Pour pouvoir appliquer la règle de résolution, il faut pouvoir appliquer le terme d'ordre deux avec un terme de C_1. Comme C_1 est une clause de $\mathcal{H}_1$, elle ne contient pas de terme d'ordre deux. L'unification est donc impossible, ce qui implique que la règle de résolution ne peut pas s'appliquer.

Nous devons maintenant montrer que si l'on applique la règle de résolution à deux clauses C_1 et C_2 de $\mathcal{F}_1 \setminus \mathcal{H}_1$, alors on n'obtient que des clauses de $\mathcal{H}_1$.

Soient deux clauses C_1 et C_2 de $\mathcal{F}_1 \setminus \mathcal{H}_1$. Pour que la règle de résolution s'applique, il faut que C_1 soit de la forme $P'(\lambda\overline{x} \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$ et C_2 de la forme $P(M\ N_1\ \ldots\ N_k) \Leftarrow P'(M), P(N_1), \ldots, P_k(t_k)$, ou inversement. La règle de résolution s'applique donc de la manière suivante :

$$\frac{\begin{array}{c} P(M\ N_1 \ldots N_k) \Leftarrow P'(M), P(N_1), \ldots, P(N_k) \\ P'(\lambda\overline{x}_k \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_m(t_m) \end{array}}{P(f(\overline{N}_{|\pi})) \Leftarrow P(N_1), \ldots, P(N_k), P_1(t_1), \ldots, P_m(t_m)}$$

La clause produite est alors dans $\mathcal{H}_1$.

□

8.4 Règles de résolution

Nous allons montrer que la résolution

Lemme 2 *La stratégie de résolution ordonnée avec sélection, avec splitting sans splitting, pour la classe $\mathcal{F}_1$ revient à utiliser les règles suivantes :*

Résolution (1)

$$\frac{P(f(X_1, \ldots, X_n)) \Leftarrow B_1(X_1), \ldots, B_n(X_n) \quad A \Leftarrow P(f(t_1, \ldots, t_n)), P_1(t'_1), \ldots, P_m(t'_m)}{A \Leftarrow B_1(t_2), \ldots, B_n(t_n), P'_1(t'_1), \ldots, P'_m(t'_m)}$$

Résolution (2)

$$\frac{P(f(X_1, \ldots, X_n)) \Leftarrow B_1(X_1), \ldots, B_n(X_n) \qquad A \Leftarrow P(X), B(X)}{A[X\ :=\ f(X_1, \ldots, X_n)] \Leftarrow B_1(X_2), \ldots, B_n(X_n), B(f(X_1, \ldots, X_n))}$$

Résolution (3)

$$\frac{q \quad A \Leftarrow P_1(t_1), \ldots, P_m(t_m), q, q_1, \ldots, q_j}{A \Leftarrow P_1(t_1), \ldots, P_m(t_m), q_1, \ldots, q_j}$$

Splitting sans splitting

$$\frac{A \Leftarrow P_1(t_1), \ldots, P_m(t_m), B(X), q_1, \ldots, q_j}{\begin{array}{c} A \Leftarrow P_1(t_1), \ldots, P_m(t_m), q, q_1, \ldots, q_j \\ q \Leftarrow B(X) \end{array}}$$

Résolution (4)

$$\frac{\begin{array}{c} P(M\ N_1 \ldots N_k) \Leftarrow P'(M), P(N_1), \ldots, P(N_k) \\ P'(\lambda \overline{x} \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_m(t_m) \end{array}}{P(f(N_1, \ldots, N_k)) \Leftarrow P(N_1), \ldots, P(N_k), P_1(t_1), \ldots, P_m(t_m)}$$

Preuve Dans la preuve du lemme précedent, on a vu que soit l'on appliquait la règle de résolution à deux clauses de $\mathcal{H}_1$, soit on l'appliquait à deux clauses de $\mathcal{F}_1 \setminus \mathcal{H}_1$. Dans le premier cas, [13] montre que la stratégie de résolution ordonnée avec sélection revient à utiliser les règles *Résolutions (1), (2) , (3) et Splitting sans splitting*. Dans le second cas, on voit dans la preuve du lemme precedent que si l'on applique la règle de résolution à des clauses de $\mathcal{F}_1 \setminus \mathcal{H}_1$, alors la règle est forcement de la forme suivante :

$$\frac{\begin{array}{c} P(M\ N_1 \ldots N_k) \Leftarrow P'(M), P(N_1), \ldots, P(N_k) \\ P'(\lambda \overline{x} \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_m(t_m) \end{array}}{P(f(\overline{N}_{|\pi})) \Leftarrow P(N_1), \ldots, P(N_k), P_1(t_1), \ldots, P_m(t_m)}$$

□

Théorème 6 *La satisfiabilité d'un ensemble fini de clauses de $\mathcal{F}_1$ est décidable.*

Preuve Considérons un ensemble S_0 de clauses de $\mathcal{F}_1$.

Notons $S_0 \rightarrow_{R_0} S_1 \rightarrow_{R_1} \cdots \rightarrow_{R_i} S_{i+1} \rightarrow_{R_{i+1}} \cdots$ la succession des application des règles de résolution. Nous allons montrer qu'il n'y a qu'un nombre fini de fois la règle de résolution (4).

Il est possible de décomposer la chaine en alternance d'application de règles de résolution de $\mathcal{H}_1$ [1]. On obtient alors la chaine suivante :

$$S_0 \rightarrow^*_{R\mathcal{H}_1} S'_0 \rightarrow_{(4)} S_1 \rightarrow^*_{R\mathcal{H}_1} S'_1 \cdots \rightarrow^*_{R\mathcal{H}_1} S'_{i-1} \rightarrow_{(4)} S_i \rightarrow^*_{R\mathcal{H}_1} S'_i \rightarrow_{R_{i+1}} \cdots$$

Considérons la règle de réécriture suivante $S_i \rightarrow_\alpha S_{i+1}$ qui correspond à la suite de réecriture $S_i \rightarrow^*_{R\mathcal{H}_1} S'_i \rightarrow_{(4)} S_{i+1}$.

Définissons maintenant un ordre sur les ensembles de clauses $\succ_{res}$ tel que $S_i \succ_{res} S_j \Leftrightarrow \gamma(S_i) > \gamma(S_j)$ où γ est la fonction qui renvoie le nombre de couple de clauses de $\mathcal{F}_1 \setminus \mathcal{H}_1$ non saturé.

On peut remarquer que $S_i \succ_{res} S_{i+1}$ car d'après le lemme 1, aucune clause de $\mathcal{F}_1 \setminus \mathcal{H}_1$ n'est générée et qu'un couple de $\mathcal{F}_1$ *setminus* $\mathcal{H}_1$ est résolu par l'application de la régle $\rightarrow_{(4)}$. Comme $\succ_{res}$ est bien fondé, et qu'il y a un nombre fini de clauses dans S_0 donc de couple de $\mathcal{F}_1 \setminus \mathcal{H}_1$, il y a un nombre fini d'application de $\rightarrow_\alpha$. La chaine $S_0 \rightarrow^*_{R\mathcal{H}_1} S'_0 \rightarrow_{(4)} S_1 \rightarrow^*_{R\mathcal{H}_1} S'_1 \cdots \rightarrow^*_{R\mathcal{H}_1} S'_{i-1} \rightarrow_{(4)} S_i \rightarrow^*_{R\mathcal{H}_1} S'_i \rightarrow_{R_{i+1}} \cdots$ est donc fini. Il existe donc un ensemble S_i

[1]Les règles de résolution de $\mathcal{H}_1$ sont les règles de résolution (1), (2) et (3) ainsi que la règle de Splitting sans splitting.

qui ne contient pas de couple $\mathcal{F}_1 \setminus \mathcal{H}_1$ de non saturé. La satisfiabilité de S_i revient donc à la satifiabilité d'un ensemble de clauses de $\mathcal{H}_1$ qui est décidable. La satisfiabilité de $\mathcal{F}_1$ est donc décidable.

□

8.5 Une extension décidable

Dans le cas de la classe $\mathcal{F}_1$ que l'on vient de présenté, une chose peut sembler assez étrange. Si l'on observe les clauses de la forme $P'(\lambda\overline{x} \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$, on remarque que le terme $\lambda\overline{x} \cdot f(\overline{x}_{|\pi})$ qui se trouve dans la tête est clos. Cela est plutôt restrictif et réduit l'expressivité de notre ensemble de clauses. On peut facilement étendre notre classe de manière à ce que ce problème ne se pose plus et que la satisfiabilité de la nouvelle classe soit décidable. Pour cela, il suffit d'autoriser la présence de variable libre dans les arguments de la fonction f mais sans pour autant autoriser les doublons.

Définition 8 *On définie la classe $\mathcal{F}_1'$ comme l'ensemble des clauses de la forme suivante :*
- $\perp \Leftarrow P_1(t_1), \ldots, P_k(t_k)$
- $P(X) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$
- $P(f(X_1, \ldots, X_n)) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$ *où $X_1, \ldots, X_n$ sont deux à deux distincts et $f \in \mathcal{F}$.*
- $P'(\lambda\overline{x} \cdot f(\overline{\alpha})) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$*, où $f \in \mathcal{F}$, oú α_i est soit un x_i, soit une variable libre X_i, les α_i étant deux à deux distincts.*
- $P(M\ N_1\ \ldots\ N_k) \Leftarrow P'(M), P_1(N_1), \ldots, P_k(N_k)$*, où $N_1, \ldots, N_k$ sont deux à deux distincts.*

où les prédicats P portent sur des termes du premier ordre et les prédicats P' sur des termes du second ordre.

Théorème 7 *La satisfiabilité de $\mathcal{F}_1'$ est décidable et DEXPTIME-complet.*

Preuve La preuve de décidabilité est sensiblement la même que celle de $\mathcal{F}_1$. La difficulté supplémentaire de la preuve réside dans le fait qu'il ne faille pas générer de clause qui ne soit pas dans $\mathcal{H}_1$. Comme pour la classe $\mathcal{F}_1$, la seule manière de générer des clauses qui ne soit pas dans $\mathcal{H}_1$ serait que la clause générée lors de l'application de la règle de résolution (4) ne soit pas dans $\mathcal{H}_1$.

$$\frac{\begin{array}{c} P(M\ N_1 \ldots N_k) \Leftarrow P'(M), P(N_1), \ldots, P(N_k) \\ P'(\lambda\overline{x} \cdot f(\overline{\alpha})) \Leftarrow P_1(t_1), \ldots, P_m(t_m) \end{array}}{P(f(\overline{\alpha'})) \Leftarrow P(N_1), \ldots, P(N_k), P_1(t_1), \ldots, P_m(t_m)}$$

où $\alpha_i' = \alpha_i[x_i := N_i]$. Pour que $P(f(\overline{\alpha'})) \Leftarrow P(N_1), \ldots, P(N_k), P_1(t_1), \ldots, P_m(t_m)$ soit dans $\mathcal{H}_1$, il faut simplement qu'il n'y ait pas de variables qui apparaissent deux fois dans $\overline{\alpha'}$. Cela se vérifie assez facilement. Les $N_1, \ldots, N_k$ sont deux à deux distincts ainsi que les variables libre de α que nous nommerons $X_1, \ldots, X_n$. L'intersection des variables libres des deux prémisses est vide, si ce n'est pas le cas, il suffit de raisonner modulo α-conversion. Il n'y a pas de variable en commun en les X_i et les N_i. Le seul moyen serait que l'a substitution $\alpha_i[x_i := N_i]$ ait généré des doublons, ce qui n'est pas le cas car les $\alpha_1, \ldots, \alpha_m$ sont deux à deux disjoint.

La clause générée est donc bien dans $\mathcal{H}_1$. La suite du raisonnement est la même que pour la classe $\mathcal{F}_1$.

$\square$

Ce théorème augmente l'expressivité de notre classe $\mathcal{F}_1$.

Chapitre 9

Extension de la classe $\mathcal{F}_1$

Dans ce chapitre, nous allons essayer d'étendre la classe $\mathcal{F}_1$ et nous montrerons que les différentes extensions que nous proposons seront indécidables.

9.1 Une première extension indécidable : les doublons

Nous allons apporter une petite modification à la clause $\mathcal{F}_1$ en modifiant le type de clause suivant $P'(\lambda\overline{x}_n{\cdot}f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1),\ldots,P_k(t_k)$. Nous allons modifier la tête pour que les doublons soient acceptés a l'intérieur de la fonction f. Pour cela, nous modifions la fonction π afin qu'elle ne soit plus linéaire. On la remplace par un fonction $\gamma : \{1,\ldots,m\} \rightarrow \{1,\cdots,n\}$ où m est un entier quelconque.

Il nous reste donc a définir notre nouvelle classe.

Définition 9 (Classe $\mathcal{F}_{1bis}$) *On définie la classe $\mathcal{F}_{1bis}$ comme l'ensemble des clauses de la forme suivante :*

- $\perp \Leftarrow P_1(t_1),\ldots,P_k(t_k)$
- $P(X) \Leftarrow P_1(t_1),\ldots,P_k(t_k)$
- $P(f(X_1,\ldots,X_n)) \Leftarrow P_1(t_1),\ldots,P_k(t_k)$
- $P'(\lambda\overline{x}\cdot f(\overline{x}_{|\gamma})) \Leftarrow P_1(t_1),\ldots,P_k(t_k)$*, où f est un symbole de fonction.*
- $P(M\ N_1\ \ldots\ N_k) \Leftarrow P'(M), P(N_1),\ldots,P_k(t_k)$

L'extension de cette classe est que l'on autorise maintenant qu'il y ait des doublons dans le terme $\lambda\overline{x}\cdot f(\overline{x}_{|\gamma})$.

Nous allons maintenant chercher à montrer que l'on peut ramener un problème indécidable à la satifiabilité de la classe $\mathcal{F}_{1bis}$. Pour cela, nous allons utiliser nous intéresser à un autre problème indécidable qui a été montré dans [9]

Théorème 8 *Soit C un ensemble de clause de $\mathcal{H}_1$. La satisfiabilité de l'ensemble de clauses $C \cup \{P(f(x,x)) \Leftarrow Q(x)\}$ est indécidable.*

Preuve La preuve de ce théorème est présenté dans [9]. L'idée de la preuve est de ramener le problème de Post, qui est connu pour être indécidable, à la satisfiabilité de notre nouvelle ensemble. Ce qui implique que la satisfiabilité de notre nouvel ensemble est indécidable.

□

Théorème 9 *La satisfiabilité de la classe $\mathcal{F}_{1bis}$ est indécidable.*

Preuve On peut facilement coder l'ensemble de clauses $C \cup \{P(f(x,x)) \Leftarrow Q(x)\}$ dans la classe $\mathcal{F}_{1bis}$. Soit l'ensemble $C \cup \{P'(\lambda x \cdot f(x,x)).,P(F\ X) \Leftarrow P'(F),Q(X)\}$.

$C \cup \{P'(\lambda x \cdot f(x,x).,P(F\ X) \Leftarrow P'(F),Q(X)\}$ est satisfiable si et seulement si $C \cup \{P(f(x,x)) \Leftarrow Q(x)\}$ est satisfiable. Cela est dû au fait qu'il suffit d'appliquer la règle de résolution sur les deux clauses $P'(\lambda x \cdot f(x,x))$. et $P(F\ X) \Leftarrow P'(F),Q(X)$.

9.2 La composition de fonctions variables

Une autre extension qui pourrait être intéressante est d'autoriser la composition de fonction. Cela permettrait ainsi d'apprendre de nouvelles fonction., Si l'on connait les fonctions F et G, il peut être naturel de dire que l'on connait aussi les fonctions $F \circ G$ ou les fonctions.

Pour permettre la composition de fonction, il suffit simplement d'autoriser le fait d'avoir plusieurs atomes avec des termes du second ordre dans le corps ou la tête des clauses pour que l'on puisse avoir la clause suivante qui code la composition de fonction.

$$P'(\lambda x \cdot X(Y x)) \Leftarrow Q'(X),R'(Y)$$

Théorème 10 *La satisfiabilité de la nouvelle classe est indécidable.*

Preuve Soit $\mathcal{F} = \{\texttt{symboles unaires}\ a_1,\ldots,a_n\}$. On remarquera qu'il est possible de coder les clauses de la forme $P'(\lambda x \cdot X(Y x)) \Leftarrow Q'(X),R'(Y)$ dans notre nouvelle classe de clauses. Ces nouvelles clauses réalise en fait la concaténation de X et Y. Cela permet alors de coder toute grammaire algébrique en forme normale de Greibach, c'est-à-dire de la forme $p \leftarrow p_1p_2$ ou $p \leftarrow terminal$. Ces grammaires ne permettent de reconnaitre uniquement les langages sans mot vide mais cela nous suffit. Le codage se fait de la manière suivante :

(1) $p \leftarrow p_1p_2$ se code en $P(\lambda x \cdot X(Y x)) \Leftarrow P_1(X),P_2(Y)$.

(2) $p \leftarrow a$, a est un terminal, se code en $P(\lambda x \cdot a(x))$.

Un problème connu pour être indécidable est la non-vacuité de l'intersection de deux langages algébriques. Si l'on considère la statifiabilité d'un ensemble de clause de la forme (1), (2) auquel on ajoute une clause de la forme $\perp \Leftarrow P(X),P'(X)$, on se rend compte que l'on a en fait codé le probème de la non-vacuité de deux langages algébriques. La satisfiabilité est donc indécidable.

□

9.3 Composition de fonctions avec une fonction fixée

Dans le paragraphe précédent, nous avons vous que l'on ne pouvait pas composer deux fonctions quelconque car nous tombions dans un cas indécidable. Nous allons maintenant essayer de restreindre ce cas composant une fonction quelconque X avec une fonction connue f. Il existe deux composition : $f \circ X$ et $X \circ f$. Cela se coderait par les formes de clauses

(1) $P'(\lambda x \cdot X(f(x))) \Leftarrow Q'(x)$ qui permettrait d'obtenir des compositions de la forme $X \circ f$.

(2) $P'(\lambda x \cdot f(X x)) \Leftarrow Q'(X)$ qui permettrait d'obtenir des compositions de la forme $f \circ X$.

Théorème 11 *La satisfiabilité de la classe de $\mathcal{F}_1$ étendu par les clauses de la forme* (1) *est insatisfiable.*

Preuve Si l'on autorise des clauses de la forme $P'(\lambda x \cdot X(f(x))) \Leftarrow Q'(X)$ et $P'(X) \Leftarrow Q'(\lambda x \cdot X(f(x)))$, il est alors possible de coder des règles de réécriture de la manière suivante :

$$l \rightarrow r \texttt{ se code par } Acc(X(r(x)) \Leftarrow Acc(X(l(x))$$

Avec cette règle, si l'on a le fait $Acc(t)$ pour t clos, on peut alors demander si t' clos est accéssible par la clause $\perp \Leftarrow Acc(t')$. Or ce problème est indécidable (cf. semi-Thuo systeme). L'ensemble des clauses dont on a besoin pour coder ce problème sont dans $\mathcal{H}_1$ ($Acc(t)$ et $\perp \Leftarrow Acc(t')$) ou dans $\mathcal{F}_1$ étendu. Si l'on souhaite avoir la composition de fonction par la gauche ($X \circ f$), on tombe dans un problème indécidable.

□

Par contre, dans le cas inverse ($f \circ X$), nous n'avons pas trouvé de preuve d'indécidabilité et par conjonction, il semblerait que l'on soit dans un cas décidable. En effet, lorsque nous appliquons la résolution ordonnée avec sélection, le nombre de clauses génées n'est pas fini mais leur forme nous permet de penser que nous somme dans un cas décidable. Cela est une question à laquelle il faudra répondre.

Chapitre 10

Conclusion

L'objectif de mon stage était de dégager une classe de clauses dont la satisfiabilité était décidable et dont qui accèpte des clauses de la forme $P(M\ N_1\ \ldots\ N_k) \Leftarrow P'(M), P(N_1), \ldots, P(N_k)$. La classe $\mathcal{F}_1$ que nous avons définie remplie exactement ces critères. Il serait donc possible de détecter de nouveaux types d'attaques comme les attaques par dictionnaire. Cependant, un problème qui reste à résoudre est la modèlisation de ces attaques. Nous avons permis à l'intrus d'appliquer des fonctions mais il faudrait maintenant savoir comment l'intrus peut apprendre de nouvelles fonctions. A partir des règles de déduction de l'intrus de Dolev Yav [1], auquelles on ajoute notre nouvelle règle, nous pouvons générer toutes les clauses de $\mathcal{F}_1$ hormis celle-ci $P'(\lambda\overline{x} \cdot f(\overline{x}_{|\pi})) \Leftarrow P_1(t_1), \ldots, P_k(t_k)$ qui permet de modéliser la connaissance d'une fonction par l'intrus.

Une autre approche consiste à réduire le modèle de déduction de Dolev Yao à notre unique règle en spécifiant qu'il connait les fonctions de chiffrement, de déchiffrement, de construction de paires et de décomposition. Cela permettrait de modéliser facilement l'ajout de nouvelles fonctions de l'intrus. Cependant, cet intérêt reste assez limité. L'apport de notre algorithme pour la vérification automatique de protocoles cryptographiques reste limité tant que l'on ne saura pas modéliser les fonctions que peut connaître l'intrus. Cela reste donc une piste à explorer dans l'avenir.

Le classe $\mathcal{F}_1$ est une extension de la classe $\mathcal{H}_1$ définie par Nielson, Nielson et Seidl dans [10] à des clauses portant sur des termes du second ordre. Nous avons montré que la décidabilité de satisfiabilité été conservé alors que de nombreuses extensions relativement proche sont indécidable.

Liste des tableaux

Bibliographie

[1] Dolev (D.) et Yao (A. C.), « On the security of public key protocols », p. 198–208. IEEE Transactions on Information Theory, 1983.

[2] Goldfarb (W. D.), « The undecidability of the second-order unification problem », *Theoretical Computer Science*, n° 13, 1981, p. 225–230.

[3] Miller (D.), « Unification of simply typed lambda-terms as logic programming », dans *Proc. 1991 Joint Int. Conf. Logic Programming*, p. 253–281, 1991.

[4] Farmer (W. M.), « A unification algorithm for second-order monadic terms », *Annals of Pure and Applied Logic*, n° 39, 1988, p. 131–174.

[5] Farmer (W. M.), « Simple second-order languages for which unification is undecideable », *Theoretical Computer Science*, n° 87, 1991, p. 25–41.

[6] Prehofer (C.), « Decidable higher-order unification problems », dans *Proceedings of the 12th International Conference on Automated Deduction*, p. 635–649, Nancy, France, 1994. Springer LNAI 814.

[7] Huet (G.), « A unification algorithm for typed lambda-calculus », *Journal of Theoretical Computer Science*, 1975, p. 27–58.

[8] Roger (M.), *Raffinements de la résolution et vérification de protocoles cryptographique*. Thèse de doctorat, Ecole Normale Supérieure de Cachan, 2003.

[9] Goubault-Larrecq (J.). « Une variante des automates d'arbres ». Examen 2003, Vérification Automatique de protocole cryptographique, DEA Programmation.

[10] Nielson (F.), Nielson (H. R.) et Seidl (H.). « Normalizable horn clauses, strongly recognizable relations and spi ».

[11] Jouannaud (J.-P.) et Rubio (A.), « The higher-order recursive path ordering », dans Longo (G.), éditeur, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, p. 402–411, Trento, Italy, 1999. IEEE Computer Society Press.

[12] Goubault-Larrecq (J.). « Spi-calcul et classes $\mathcal{H}_1$ et $\mathcal{H}_2$ ». Examen 2004, Vérification Automatique de protocole cryptographique, DEA Programmation.

[13] Goubault-Larrecq (J.). « Résolution ordonnée avec sélection et classes décidables de la logique du premier ordre ». Note de cours, Vérification Automatique de protocole cryptographique, DEA Programmation.