Utilisation de la sur-réduction pour le calcul du différentiel entre deux politiques de sécurité

Didier Villevalois

Laboratoire d'Informatique Fondamentale de Marseille

Encadrants

Clara Bertolissi (Maître de Conférence)

Jean-Marc Talbot (Professeur)

Résumé

Un aspect important des politiques de sécurité pour les systèmes d'information concerne le contrôle des droits d'accès aux ressources par leurs utilisateurs. Les politiques de contrôle d'accès peuvent être spécifiées formellement sous la forme de systèmes de réécriture. Cette représentation permet la vérification statique de certaines propriétés des politiques (consistance, complétude, terminaison, ...). Elle offre aussi un moyen opérationnel éprouvé pour résoudre et contrôler les requêtes d'accès, que l'on exprime alors sous la forme d'un terme clos.

Nous présentons la sur-réduction, une généralisation de la réécriture, et plus particulièrement la sur-réduction nécessaire la plus extérieure, une variante qui s'applique sur la classe des systèmes de réécriture inductivement séquentiels. Nous montrons comment elle peut être utilisée dans le cadre des politiques de sécurité pour résoudre des patrons de requêtes d'accès, cette fois exprimés sous la forme de termes avec variables.

Finalement, nous nous intéressons aux différences entre deux versions d'une même politique de contrôle d'accès. Nous formulons pour cela un problème nouveau, l'identification d'exemples pour lesquels deux systèmes de réécriture présentent un comportement différent. Pour répondre à ce problème, nous proposons une technique basée à la fois sur la transformation des systèmes de réécriture et sur la sur-réduction. Nous discutons de son applicabilité dans des contextes où l'un des deux (ou les deux) systèmes peut ne pas être terminant.

Mots-clés : Politiques de Sécurité, Contrôle d'Accès, Systèmes de Réécriture, Sur-réduction, Différentiel

1 Introduction

Assurer la confidentialité et l'intégrité des données est essentiel pour les systèmes d'information. Tous les utilisateurs ne peuvent visualiser ou modifier les mêmes ressources. Le contrôle des accès a pour objet de faire respecter ces différents privilèges.

On peut spécifier formellement les politiques de contrôle d'accès sous la forme de systèmes de réécriture [6]. Une telle représentation permet la vérification formelle de certaines propriétés importantes des politiques de contrôle d'accès : leur consistance, leur complétude, leur terminaison... [8] Par ailleurs, la réécriture peut être utilisée de manière opérationnelle pour contrôler une requête d'accès.

La sur-réduction est une généralisation de la réécriture. Elle offre à l'administrateur système un moyen plus sophistiqué pour interroger une politique de contrôle d'accès. Il peut par exemple poser des questions du type : — quels sont les utilisateurs pouvant effectuer telle action sur telles ressources ? — quels sont les rôles communs à tels utilisateurs ? — quel rôle dois-je ajouter à tel utilisateur pour qu'il puisse effectuer telle action ?

Les politiques de contrôle d'accès peuvent rapidement devenir complexes. Différents aspects d'une politique peuvent avoir un impact sur sa complexité comme, par exemple, le nombre d'utilisateurs, le nombre et la diversité des rôles des utilisateurs, et le nombre et la diversité des ressources utilisables, ... Une simple modification de la politique peut alors entraîner une transformation profonde de son comportement et sa maintenance devient une tâche coûteuse pour l'administrateur système.

Il serait possible, à chaque modification, d'à nouveau vérifier formellement l'ensemble des propriétés de la politique. Néanmoins, dans le cas où l'une des propriétés souhaitées est invalidée, l'administrateur ne dispose d'aucune indication pour en comprendre la raison et corriger le problème.

Dans l'optique de faciliter le travail de l'administrateur système lors de l'évolution d'une politique de sécurité, une approche alternative (et complémentaire) consiste à identifier des exemples pour lesquels l'ancienne et la nouvelle version de cette politique ont un comportement différent.

Nous discernons deux cas d'utilisation typiques où un tel outil serait utile : (i) l'administrateur souhaite modifier l'implémentation de la politique (pour par exemple la simplifier) sans en changer le comportement, (ii) l'administrateur souhaite modifier l'implémentation de la politique pour ajouter des utilisateurs, des rôles, des règles, ... sans en modifier le comportement sur la partie existante.

Dans un cadre où les politiques de contrôle d'accès sont spécifiées sous la forme de systèmes de réécriture, la question posée est alors, en toute généralité : étant donnés deux versions d'un système de réécriture quels sont les termes pour lesquels ces deux systèmes donnent des réductions différentes ?

Les contributions de ce mémoire sont les suivantes :

— nous montrons comment la sur-réduction peut être utilisée pour permettre à l'administrateur système d'interroger une politique de contrôle d'accès,

— afin de répondre au cas d'utilisation (i), nous développons la technique des différentiels de sur-réduction pour identifier les différences entre deux systèmes de réécriture définis sur un même ensemble de constructeurs, et

— afin de répondre au cas d'utilisation (ii), nous proposons une extension de cette technique pour identifier les différences entre deux systèmes de réécriture définis sur deux ensembles de constructeurs non-disjoints.

La suite de ce mémoire est organisée de la façon suivante. En section 2, nous rappelons les notions de base en réécriture, introduisons la sur-réduction et présentons la stratégie de sur-

réduction nécessaire la plus extérieure. En section 3, nous présentons les principes du contrôle d'accès et un modèle générique pour spécifier les politiques de contrôle d'accès. Nous montrons comment celles-ci peuvent être représentées sous la forme de systèmes de réécriture et être interrogées par sur-réduction. En section 4, nous développons les techniques de différentiels de sur-réduction permettant l'identification des différences, par l'exemple, entre deux systèmes de réécriture. Finalement, en section 5, nous concluons et suggérons des travaux complémentaires.

2 Réécriture et sur-réduction

Nous rappelons quelques notions et notations importantes sur la réécriture et la sur-réduction. Nous renvoyons le lecteur vers [9] et [13] pour plus de précisions sur la réécriture.

2.1 Termes

Soit S un ensemble de sortes. Une *signature* à plusieurs sortes (« many-sorted signature ») $\mathcal{F}$ est une famille d'ensembles de *fonctions* $\mathcal{F} = (\mathcal{F}_{a,s} \mid a \in S^* \text{ et } s \in S)$. Soit $\mathcal{X} = (\mathcal{X}_s \mid s \in S)$ une famille d'ensembles dénombrables de *variables*. Alors l'ensemble $\mathcal{T}(\mathcal{F}, \mathcal{X})_s$ de *termes* de sorte s est le plus petit ensemble contenant $\mathcal{X}_s$ tel que $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})_s$ si $f \in \mathcal{F}_{(s_1,\ldots,s_n),s}$ et, pour tout $1 \leqslant i \leqslant n$, $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})_{s_i}$. Si $f \in \mathcal{F}_{\epsilon,s}$, nous notons f au lieu de $f()$. $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \cup_{s \in S} \mathcal{T}(\mathcal{F}, \mathcal{X})_s$. Pour un terme t, on note $sort(t)$ la sorte de t.

$\mathcal{V}(t)$ dénote l'ensemble des variables apparaissant dans t. On dit d'un terme t qu'il est *clos* si $\mathcal{V}(t) = \emptyset$ et qu'il est *linéaire* si t ne contient pas plusieurs occurrences d'une même variable.

Une *position* est une chaîne d'entiers identifiant un sous-terme dans un terme. Pour tout terme t, ϵ identifie t lui-même. Pour tout terme $f(t_1, \ldots, t_n)$, la séquence $i \cdot p$, avec $1 \leqslant i \leqslant n$, identifie le sous-terme de t_i à la position p. On note $t|_p$ le sous-terme de t à la position p et $t[s]_p$ le terme t dans lequel le sous-terme à la position p est remplacé par le terme s.

Une *substitution* σ est une fonction de $\mathcal{X}$ vers $\mathcal{T}(\mathcal{F}, \mathcal{X})$ telle que $\sigma(x) \in \mathcal{T}(\mathcal{F}, \mathcal{X})_s$ pour tout $x \in \mathcal{X}_s$, et $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ est fini. Nous identifions une substitution σ avec l'ensemble $\{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$ et la substitution vide est dénotée par id. Une substitution σ est un renommage de variables si $\sigma(x) \in \mathcal{X}$ pour tout $x \in \mathcal{D}om(\sigma)$ et σ est bijective. Les substitutions sont étendues comme des morphismes de $\mathcal{T}(\mathcal{F}, \mathcal{X})$ vers $\mathcal{T}(\mathcal{F}, \mathcal{X})$, de telle façon que $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$ pour tout terme $f(t_1, \ldots, t_n)$.

On définit la *composition de deux substitutions* σ et τ par $(\sigma \circ \tau)(x) = \sigma(\tau(x))$ pour tout $x \in \mathcal{X}$. Une substitution σ est *moins spécifique* (ou *plus générale*) que σ', noté $\sigma \leqslant \sigma'$, s'il existe une substitution τ telle que $\tau \circ \sigma = \sigma'$.

On note $\sigma_{|V}$ la *restriction* d'une substitution σ à un ensemble de variables $V \subseteq \mathcal{X}$, définie telle que $\sigma_{|V}(x) = \sigma(x)$ si $x \in V$ et $\sigma_{|V}(x) = x$ sinon. On écrit $\sigma = \sigma'[V]$ si $\sigma_{|V} = \sigma'_{|V}$ et $\sigma \leqslant \sigma'[V]$ si $\sigma_{|V} \leqslant \sigma'_{|V}$.

Un terme t' est une *instance* de t, noté $t \leqslant t'$, s'il existe une substitution σ telle que $\sigma(t) = t'$. Il en est une *variante* si $t \leqslant t'$ et $t' \leqslant t$.

Un *unificateur* de deux termes s et t est une substitution σ telle que $\sigma(s) = \sigma(t)$. Un unificateur σ est appelé *l'unificateur le plus général* si $\sigma \leqslant \tau$ pour autre unificateur τ de s et t. Les unificateurs les plus généraux sont uniques à renommage de variable près. On note $mgu(s, t)$ l'unificateur le plus général de s et t.

Une substitution σ est *idempotente* si $\sigma \circ \sigma = \sigma$. Dans la suite, nous utilisons le fait qu'un unificateur le plus général σ de deux termes s et t est une substitution idempotente, et que $\mathcal{D}om(\sigma) \subseteq \mathcal{V}(s) \cup \mathcal{V}(t)$.

On dit d'une signature $\mathcal{F}$ qu'elle est *à base de constructeurs* si l'on peut la partitionner en deux ensembles disjoints $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$, où $\mathcal{C}$ est appelé l'ensemble des *constructeurs* et $\mathcal{D}$ l'ensemble des *opérations définies*. Les termes dans $\mathcal{T}(\mathcal{C}, \mathcal{X})$ sont appelés *termes de constructeurs*. Un terme $f(t_1, \ldots, t_n)$ est un *motif* si $f \in \mathcal{D}$ et $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, pour tout $1 \leqslant i \leqslant n$. Une *substitution de constructeurs* σ est une substitution telle que $\sigma(x) \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, pour tout $x \in \mathcal{D}om(\sigma)$.

2.2 Réécriture

Une *règle de réécriture* est une équation dirigée $l \to r$ telle que $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})_s \setminus \mathcal{X}$ et $r \in \mathcal{T}(\mathcal{F}, \mathcal{X})_s$, pour une sorte $s \in S$, et $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. Si une substitution σ est un renommage de variable, alors $\sigma(l) \to \sigma(r)$ est une *variante* de $l \to r$. Une règle $l \to r$ est *linéaire à gauche* (resp. *à droite*) si l (resp. r) est linéaire.

Un *système de réécriture de termes* (TRS) est une paire $\mathcal{R} = (\mathcal{F}, R)$ où $\mathcal{F}$ est une signature et R est un ensemble de règles de réécriture. Un TRS *linéaire à gauche* (resp. *à droite*) ne contient que des règles linéaires à gauche (resp. à droite).

Exemple 1. Le système de réécriture $\mathcal{RR}$ suivant définit les opérations « plus petit ou égal » et addition sur les nombres naturels, représentés par les termes construits avec $\mathsf{0}$ et s :

$$
\begin{array}{llcl@{\qquad}llcl}
(\mathrm{R}_1) & \mathsf{0} \leqslant x & \to & \mathsf{true} & (\mathrm{R}_4) & \mathsf{0} + x & \to & x \\
(\mathrm{R}_2) & \mathsf{s}(x) \leqslant \mathsf{0} & \to & \mathsf{false} & (\mathrm{R}_5) & \mathsf{s}(x) + y & \to & \mathsf{s}(x + y) \\
(\mathrm{R}_3) & \mathsf{s}(x) \leqslant \mathsf{s}(y) & \to & x \leqslant y & & & & \\
\end{array}
$$

Une *étape de réécriture* est l'application d'une règle d'un TRS $\mathcal{R} = (\mathcal{F}, R)$ sur un terme t, notée $t \to_R s$, $t \to_{p, l \to r} s$ ou encore $t \to_{p, l \to r, \sigma} s$, s'il existe une règle $l \to r$, une position p et une substitution σ telle que $t|_p = \sigma(l)$ et $s = t[\sigma(r)]_p$. On dit alors que $t|_p$ est un *rédex* et que t se *réécrit* en s en *contractant* $t|_p$. La relation $\overset{*}{\to}_R$ dénote la clôture transitive et réflexive de la relation induite par $\to_R$. Alors $t_0 \overset{*}{\to}_R t_n$ s'il existe une séquence de réductions $t_0 \to_R t_1 \to_R \cdots \to_R t_n$ (avec $n \geqslant 0$), et on dit que t_0 se *réduit* en t_n. Nous omettons les indices sur $\to$ et $\overset{*}{\to}$ lorsqu'aucune confusion n'est possible.

Exemple 2. Le terme $\mathsf{s}(\mathsf{s}(\mathsf{0})) \leqslant \mathsf{s}(\mathsf{s}(\mathsf{0})) + \mathsf{s}(\mathsf{0})$ se réduit en true par le système présenté dans l'exemple 1. (À chaque étape de réécriture, le rédex qui est contracté est souligné.)

$$
\begin{array}{lll}
\mathsf{s}(\mathsf{s}(\mathsf{0})) \leqslant \underline{\mathsf{s}(\mathsf{s}(\mathsf{0})) + \mathsf{s}(\mathsf{0})} & \to_{\mathrm{R}_5} & \underline{\mathsf{s}(\mathsf{s}(\mathsf{0})) \leqslant \mathsf{s}(\mathsf{s}(\mathsf{0}) + \mathsf{s}(\mathsf{0}))} \\
& \to_{\mathrm{R}_3} & \mathsf{s}(\mathsf{0}) \leqslant \underline{\mathsf{s}(\mathsf{0}) + \mathsf{s}(\mathsf{0})} \\
& \to_{\mathrm{R}_5} & \underline{\mathsf{s}(\mathsf{0}) \leqslant \mathsf{s}(\mathsf{0} + \mathsf{s}(\mathsf{0}))} \\
& \to_{\mathrm{R}_3} & \underline{\mathsf{0} \leqslant \mathsf{0} + \mathsf{s}(\mathsf{0})} \\
& \to_{\mathrm{R}_1} & \mathsf{true}
\end{array}
$$

Un terme t est en *forme normale* (par rapport à un TRS) s'il n'existe pas de terme s tel que $t \to s$. Un terme t *possède* une forme normale s'il existe s tel que $t \overset{*}{\to} s$ et s est en forme normale. Une substitution σ est *normalisée* (par rapport à un TRS) si $\sigma(x)$ est en forme normale, pour tout $x \in \mathcal{D}om(\sigma)$.

Deux termes t_1 et t_2 sont dits *joignables*, noté $t_1 \downarrow t_2$, s'il existe s tel que $t_1 \overset{*}{\to} s \overset{*}{\leftarrow} t_2$.

Un TRS est *faiblement normalisant* si tout terme possède une forme normale et *fortement normalisant* s'il n'existe pas de séquence de réductions infinie. Un TRS est *localement confluent*

si pour tous termes s, t_1, t_2 tels que $t_1 \leftarrow s \rightarrow t_2$ nous avons $t_1 \downarrow t_2$. Il est *confluent* (ou *Church-Rosser*) si pour tous termes s, t_1, t_2 tels que $t_1 \overset{*}{\leftarrow} s \overset{*}{\rightarrow} t_2$ nous avons $t_1 \downarrow t_2$.

Un TRS est dit *non-ambigu* (ou *non-superposable*) si les parties gauches de ses règles ne se superposent pas, c'est à dire, étant donné une règle $l \rightarrow r$, il n'existe pas de position non-variable p dans l et de règle de $l' \rightarrow r'$ telles que l' soit une instance de $l|_p$ (avec $p \neq \epsilon$ si $l \rightarrow r = l' \rightarrow r'$). Un TRS est dit *orthogonal* s'il est à la fois linéaire à gauche et non-ambigu. Un TRS orthogonal est confluent.

Un TRS est *à base de constructeurs* (en abrégé, CB-TRS [1]) si les parties gauches de ses règles sont des motifs. Par conséquent, un terme de constructeur t est nécessairement normalisé par rapport à tout CB-TRS $\mathcal{R}$ puisque aucune règle de $\mathcal{R}$ ne peut s'appliquer sur t. De même, une substitution de constructeurs est nécessairement normalisée par rapport à tout CB-TRS.

Pour un CB-TRS $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ tel que pour tout motif clos $t \in \mathcal{T}(\mathcal{C} \uplus \mathcal{D})$ une règle de R s'applique, les formes normales, par rapport à $\mathcal{R}$, de termes clos appartiennent à $\mathcal{T}(\mathcal{C})$, l'ensemble des termes de constructeurs clos sur $\mathcal{C}$.

Exemple 3. Le TRS $\mathcal{RR}$ présenté dans l'exemple 1 est donc un CB-TRS qui, de plus, est orthogonal. Par rapport à celui-ci, les termes $\mathsf{s}(\mathsf{s}(0))$ et $x + \mathsf{s}(0)$ sont tous deux en forme normale.

2.3 Sur-réduction

Soit un TRS $\mathcal{R} = (\mathcal{F}, R)$. Un terme t est sur-réductible en un terme s s'il existe une position non-variable p dans t, une variante $l \rightarrow r$ d'une règle de $\mathcal{R}$ (afin que les ensembles de variables de t et $l \rightarrow r$ soient disjoints) et un unificateur σ de $t|_p$ et l tels que $s = \sigma(t[r]_p)$. On note alors $t \rightsquigarrow_{p,l\rightarrow r,\sigma} s$, ou $t \rightsquigarrow_{R,\sigma} s$, ou simplement $t \rightsquigarrow_\sigma s$ si aucune confusion n'est possible.

On note $t_0 \overset{*}{\rightsquigarrow}_\sigma t_n$ s'il existe une séquence de sur-réductions $t_0 \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_n} t_n$ (appelée *dérivation* de sur-réduction) telle que $\sigma = \sigma_n \circ \cdots \circ \sigma_2 \circ \sigma_1$. Si $n = 0$ alors $\sigma = id$.

Dans une étape de réécriture $t \rightarrow_{p,l\rightarrow r,\sigma} s$, on suppose qu'il n'y a aucune variable en commun dans t et $l \rightarrow r$ et que σ est restreinte aux variables présentes dans l. σ est donc un unificateur le plus général [2] de $t|_p$ et l, et $s = t[\sigma(r)]_p = \sigma(t[r]_p)$. La réécriture peut donc être vue comme un cas particulier de la sur-réduction.

On peut montrer que si $t \rightsquigarrow_{p,R,\sigma} t'$ alors $\sigma(t) \rightarrow_{p,R} t'$. Et ceci peut être généralisé pour *abaisser* (lower) une dérivation de sur-réduction en dérivation de réécriture :

Lemme 1 (Abaissement)**.** *Soit $\mathcal{R}$ un système de réécriture. Si $t \overset{*}{\rightsquigarrow}_{\mathcal{R},\sigma} t'$ alors $\sigma(t) \overset{*}{\rightarrow}_{\mathcal{R}} t'$. De plus, on peut supposer que les deux dérivations utilisent les mêmes règles aux mêmes positions.*

Démonstration. Nous utilisons une induction sur la longueur de la dérivation $t \overset{*}{\rightsquigarrow}_{\mathcal{R},\sigma} t'$. Le cas de longueur zéro est trivial. Supposons $t \rightsquigarrow_{p,R,\sigma_1} t_1 \overset{*}{\rightsquigarrow}_{\mathcal{R},\sigma'} t'$ une dérivation de longueur $n+1$ telle que R est une règle de $\mathcal{R}$ et $\sigma' \circ \sigma_1 = \sigma$. Nous avons $\sigma_1(t) \rightarrow_{p,R} t_1$, avec p non-variable, et donc $\sigma'(\sigma_1(t)) \rightarrow_{p,R} \sigma'(t_1)$. L'hypothèse d'induction implique que $\sigma'(t_1) \overset{*}{\rightarrow}_{\mathcal{R}} t'$ et nous obtenons le résultat souhaité en concaténant les deux dérivations : $\sigma(t) = \sigma'(\sigma_1(t)) \rightarrow_{p,R} \sigma'(t_1) \overset{*}{\rightarrow}_{\mathcal{R}} t'$. □

Middledorp et Hamoen [15] montrent le lemme suivant qui lui permet d'*élever* (lift) une dérivation de réécriture en une dérivation de sur-réduction :

1. Les auteurs cités dans ce mémoire ont choisi « CS » comme abréviation pour dénoter ces systèmes.
2. Puisque $\mathcal{D}om(\sigma) = \mathcal{V}(l)$, $\sigma(t|_p) = t|_p = \sigma(l)$ et alors pour tout unificateur τ de $t|_p$ et l, $\tau_{|\mathcal{V}(l)} = \sigma$ et donc $\sigma \leqslant \tau$

Lemme 2 (Élévation)**.** *[15] Soit $\mathcal{R}$ un système de réécriture. On considère deux termes t et s, une substitution normalisée θ et un ensemble de variables V tels que $\mathcal{V}(s) \cup \mathcal{D}om(\theta) \subseteq V$ et $t = \theta(s)$. Si $t \xrightarrow{*}_{\mathcal{R}} t'$ alors il existe un terme s' et deux substitutions θ', σ tels que :*

- *$s \overset{*}{\leadsto}_{\mathcal{R},\sigma} s'$,*
- *$t' = \theta'(s')$,*
- *$\theta = \theta' \circ \sigma[V]$, et*
- *θ' est normalisée*

De plus, on peut supposer que les deux dérivations utilisent les mêmes règles aux mêmes positions.

Les deux lemmes précédents permettent d'obtenir une bonne intuition quant à la relation existant entre réécriture et sur-réduction. Dans la suite, nous les utiliserons dans nos preuves de correction et complétude.

On peut établir un lien entre sur-réduction et unification équationnelle. Considérons un système de réécriture $\mathcal{R}$. On introduit un symbole de fonction $\approx$ et une constante true (supposés non-existant dans la signature), ainsi qu'une nouvelle règle $x \approx x \to \mathsf{true}$.[3] Nous ne manipulons ici que des termes de la forme $t_1 \approx t_2$ ou true tels que t_1 et t_2 ne contiennent ni $\approx$ ni true. Si $t_1 \approx t_2 \overset{*}{\leadsto}_{\sigma} \mathsf{true}$ alors σ est un unificateur de t_1 et t_2 par rapport à $\mathcal{R}$ (autrement dit σ est une solution de l'équation $t_1 \approx t_2$ dans $\mathcal{R}$, c'est à dire $\sigma(t_1) \downarrow_{\mathcal{R}} \sigma(t_2)$).

En vue de préserver la linéarité à gauche d'un système orthogonal à base de constructeurs, on peut définir la sémantique de $\approx$ de la manière suivante :

$$(\approx) \qquad \mathsf{c}(X_1, \ldots, X_n) \approx \mathsf{c}(Y_1, \ldots, Y_n) \to \wedge_{i=1}^{n}(X_i \approx Y_i) \qquad \forall \mathsf{c} \in \mathcal{C} \text{ d'arité } n \geqslant 0$$

$$(\wedge) \qquad \mathsf{true} \wedge X \to X$$

Avec ces règles, un terme $t \approx t'$ ne se normalisera que pour t et t' des termes de constructeurs clos (cf. proposition 9 plus loin).

Afin de simplifier le parenthésage, nous utilisons le symbol $\wedge$ en notation infixe associative à droite. Ainsi, nous écrivons $t \wedge u \wedge v$ pour $t \wedge (u \wedge v)$. Dans le cas $n = 0$, $c(X_1, \ldots, X_n)$ signifie simplement c. De même, $\wedge_{i=1}^{0}(\ldots)$ signifie la constante true.

Exemple 4. En étendant le système $\mathcal{RR}$ présenté dans l'exemple 1 avec l'égalité, comme formulée ci-dessus, on obtient un TRS orthogonal. La dérivation de sur-réduction suivante permet de trouver l'unique solution $\{z \mapsto \mathsf{s}(\mathsf{0})\}$ de l'équation $z + \mathsf{s}(\mathsf{0}) \approx \mathsf{s}(\mathsf{s}(\mathsf{0}))$.

$$\begin{aligned}
\underline{z + \mathsf{s}(\mathsf{0})} \approx \mathsf{s}(\mathsf{s}(\mathsf{0})) \quad &\leadsto_{\mathrm{R}_5, \{z \mapsto \mathsf{s}(x_1)\}} \quad \underline{\mathsf{s}(x_1 + \mathsf{s}(\mathsf{0})) \approx \mathsf{s}(\mathsf{s}(\mathsf{0}))} \\
&\leadsto_{\approx, \{\}} \quad \underline{x_1 + \mathsf{s}(\mathsf{0})} \approx \mathsf{s}(\mathsf{0}) \\
&\leadsto_{\mathrm{R}_4, \{x_1 \mapsto \mathsf{0}\}} \quad \underline{\mathsf{s}(\mathsf{0}) \approx \mathsf{s}(\mathsf{0})} \\
&\leadsto_{\approx, \{\}} \quad \underline{\mathsf{0} \approx \mathsf{0}} \\
&\leadsto_{\approx, \{\}} \quad \mathsf{true}
\end{aligned}$$

Exemple 5. Les deux dérivations de sur-réduction suivantes permettent de trouver deux solutions ($\{z \mapsto \mathsf{s}(\mathsf{0}), y \mapsto \mathsf{0}\}$ et $\{z \mapsto \mathsf{s}(\mathsf{s}(\mathsf{0})), y \mapsto \mathsf{s}(\mathsf{0})\}$) parmi l'infinité de solutions de l'équation $z + \mathsf{s}(\mathsf{0}) \approx \mathsf{s}(\mathsf{s}(y))$, qui contient deux variables, une à droite et une à gauche.

3. L'utilisation de règles plus appropriées pour encoder l'égalité $\approx$ est nécessaire afin de préserver la linéarité à gauche. Nous présentons plus loin un tel jeu de règles.

$\underline{z + \mathsf{s}(\mathsf{0})} = \mathsf{s}(\mathsf{s}(y))$

$\leadsto_{\mathrm{R}_5,\{z \mapsto \mathsf{s}(x_1)\}}$ $\underline{\mathsf{s}(x_1 + \mathsf{s}(\mathsf{0})) = \mathsf{s}(\mathsf{s}(y))}$

$\leadsto_{\approx,\{\}}$ $\underline{x_1 + \mathsf{s}(\mathsf{0})} = \mathsf{s}(y)$

$\leadsto_{\mathrm{R}_4,\{x_1 \mapsto \mathsf{0}\}}$ $\underline{\mathsf{s}(\mathsf{0}) = \mathsf{s}(y)}$

$\leadsto_{\approx,\{y \mapsto x_2\}}$ $\underline{\mathsf{0} = x_2}$

$\leadsto_{\approx,\{x_2 \mapsto \mathsf{0}\}}$ true

$\underline{z + \mathsf{s}(\mathsf{0})} = \mathsf{s}(\mathsf{s}(y))$

$\leadsto_{\mathrm{R}_5,\{z \mapsto \mathsf{s}(x_1)\}}$ $\underline{\mathsf{s}(x_1 + \mathsf{s}(\mathsf{0})) = \mathsf{s}(\mathsf{s}(y))}$

$\leadsto_{\approx,\{\}}$ $\underline{x_1 + \mathsf{s}(\mathsf{0})} = \mathsf{s}(y)$

$\leadsto_{\mathrm{R}_5,\{x_1 \mapsto \mathsf{s}(x_2)\}}$ $\underline{\mathsf{s}(x_2 + \mathsf{s}(\mathsf{0})) = \mathsf{s}(y)}$

$\leadsto_{\approx,\{\}}$ $\underline{x_2 + \mathsf{s}(\mathsf{0})} = y$

$\leadsto_{\mathrm{R}_4,\{x_2 \mapsto \mathsf{0}\}}$ $\underline{\mathsf{s}(\mathsf{0}) = y}$

$\leadsto_{\approx,\{y \mapsto \mathsf{s}(x_3)\}}$ $\underline{\mathsf{0} = x_3}$

$\leadsto_{\approx,\{x_3 \mapsto \mathsf{0}\}}$ true

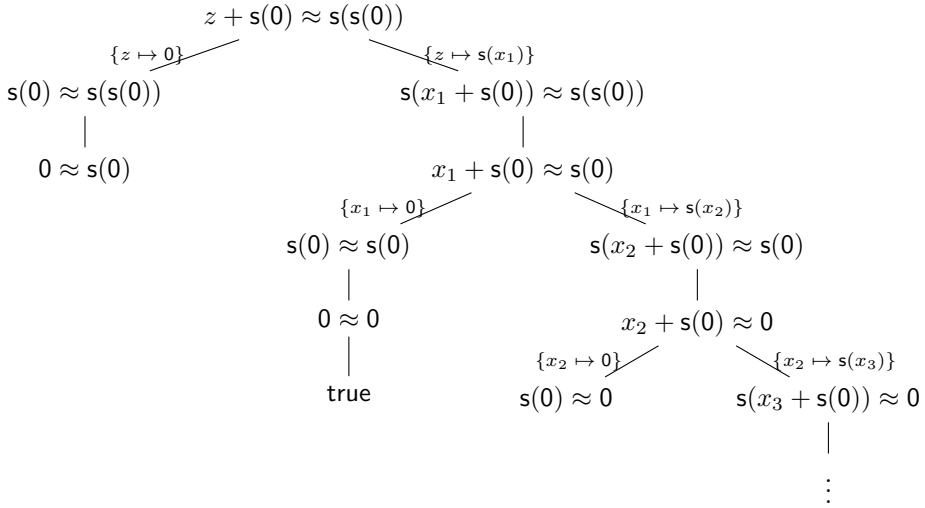
Il est possible de représenter sous la forme d'un arbre l'ensemble des dérivations de sur-réduction possibles pour un terme. Chaque nœud est une paire de substitution et terme obtenable par une étape de sur-réduction depuis son père.

Définition 1 (Arbre de sur-réduction)**.** $\mathcal{N}_\mathcal{R}(t)$ est un *arbre de sur-réduction* de t d'après le système de réécriture $\mathcal{R}$, défini inductivement :

— (id, t) est un nœud de $\mathcal{N}_\mathcal{R}(t)$

— (σ, s) est un nœud de $\mathcal{N}_\mathcal{R}(t)$ et $s \leadsto_{p,l\to r,\tau} s'$, alors $(\tau \circ \sigma, s')$ est un nœud de $\mathcal{N}_\mathcal{R}(t)$. On étiquette l'arête $((\sigma, s), (\tau \circ \sigma, s'))$ par le triplet $(p, l \to r, \tau)$. [4]

Exemple 6. L'arbre de sur-réduction suivant montre quelques dérivations de sur-réduction possibles pour le terme $z + \mathsf{s}(\mathsf{0}) \approx \mathsf{s}(\mathsf{s}(\mathsf{0}))$ dans le TRS $\mathcal{RR}$. (Pour améliorer la lisibilité, nous n'étiquetons les nœuds qu'avec les termes et les arêtes qu'avec les substitutions.)



Une *stratégie* de sur-réduction $\mathcal{S}$ pour un TRS $\mathcal{R}$ est une fonction qui associe à un terme t un ensemble des triplets $(p, l \to r, \sigma)$, où p est une position dans t, $l \to r$ une règle de $\mathcal{R}$ et σ une substitution, tels que $t \leadsto_{p,l\to r,\sigma} \sigma(t[r]_p)$ est une étape de sur-réduction de t dans $\mathcal{R}$.

À partir d'une stratégie de sur-réduction, il est donc possible de construire l'arbre de sur-réduction pour un terme donné. Pour le terme associé à un nœud de l'arbre, la stratégie nous

4. Cette structure est bien un arbre dont les nœuds sont des paires de substitution et terme. Ainsi, pour un nœud (σ, s), une position p, une règle $l \to r$, et une substitution τ, il existe au plus un fils $(\tau \circ \sigma, s')$ tel que l'arête $((\sigma, s), (\tau \circ \sigma, s'))$ est étiquetée par $(p, l \to r, \tau)$.

donne l'ensemble des triplets $(p, l \to r, \sigma)$ pour lesquels une sur-réduction est possible. Pour chacun de ces triplets, on peut alors calculer le nœud fils correspondant.

Définition 2 (Recherche en largeur d'abord)**.** L'ensemble des dérivations de sur-réductions d'un terme t de longueur exactement n par une stratégie $\mathcal{S}$ est définit inductivement :

$$bfs(t, \mathcal{S}, n) \ni \begin{cases} (id, t) & \text{si } n = 0 \\ (\tau \circ \sigma, \tau(s[r]_p)) & \text{si } n > 0 \text{ et } (\sigma, s) \in bfs(t, \mathcal{S}, n-1) \text{ et } (p, l \to r, \tau) \in \mathcal{S}(s) \end{cases}$$

Proposition 3. *Soit une stratégie $\mathcal{S}$ pour un TRS $\mathcal{R}$ et un terme t. S'il existe une substitution normalisée σ, un terme clos s et une dérivation de réécriture $\sigma(t) \to_{\mathcal{R}} s$ de longueur n alors $(\sigma, s) \in bfs(t, \mathcal{S}, n)$.*

Démonstration. Par application du lemme d'élévation (lemme 2) et par définition de *bfs*. □

Si, pour tout terme t, l'ensemble de triplets $\mathcal{S}(t)$ est fini, alors l'arbre de sur-réduction est à branchement fini et pour tout n, $bfs(t, \mathcal{S}, n)$ est fini. On peut alors effectuer un parcours en largeur d'abord de cet arbre, en maintenant l'ensemble des paires de substitution et termes obtenues à la profondeur courante. Si cette condition est respectée, on dit d'une étape de sur-réduction $t \leadsto_{p, l \to r, \sigma} s$ qu'elle est *calculée* par une stratégie $\mathcal{S}$ si $(p, l \to r, \sigma) \in \mathcal{S}(t)$. On dit d'une dérivation de sur-réduction qu'elle est *calculée* par une stratégie $\mathcal{S}$ si chaque étape de sur-réduction la composant est calculée par $\mathcal{S}$.

Une partie importante de la recherche sur la sur-réduction a porté sur la mise en exergue, en fonction de la classe de TRS, de stratégies de sur-réduction optimales, dans le sens où elles n'entraînent que des étapes inévitables.

Exemple 7. En utilisant le système de l'exemple 1, on cherche à sur-réduire le terme $x \leqslant y + z$. On pourrait sur-réduire ce terme en position 2 :

$$x \leqslant y + z \leadsto_{2, R_4, \{y \mapsto 0\}} x \leqslant z$$

Cette étape est nécessaire si x est éventuellement substitué par $\mathsf{s}(x')$ dans la suite de la dérivation. Elle est par contre évitable si x est substitué par $\mathsf{0}$ comme le montre la dérivation suivante qui calcule une solution plus générale :

$$x \leqslant y + z \leadsto_{\epsilon, R_1, \{x \mapsto 0\}} \mathsf{true}$$

Pour les TRS orthogonaux, la stratégie de réduction *nécessaire* (needed) [12, 14] se concentre sur les réductions qui seront appliquées dans n'importe quel dérivation menant à une forme normale, en ignorant les réductions inutiles.

2.4 Sur-réduction nécessaire la plus extérieure

Antoy et al. [2, 3] définissent la stratégie de sur-réduction *nécessaire la plus extérieure* (outermost-needed narrowing) que nous présentons dans cette section.

Définition 3 (Arbre définitionnel)**.** [2, 3] $\mathcal{T}$ est un *arbre définitionnel*, avec motif[5] π (noté $pattern(\mathcal{T})$), pour un système de réécriture $\mathcal{R}$ ssi un des cas suivants s'applique :

— $\mathcal{T} = branch(\pi, p, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, où π est un motif, p est la position d'une variable de π de sorte $s \in S$, $c_1, \ldots, c_k$ sont k constructeurs de sorte s, deux à deux distincts, et, pour tout $1 \leqslant i \leqslant k$, $\mathcal{T}_i$ est un arbre définitionnel avec motif $\pi[c_i(\overline{x})]_p$, où $\overline{x}$ sont des variables fraîches,

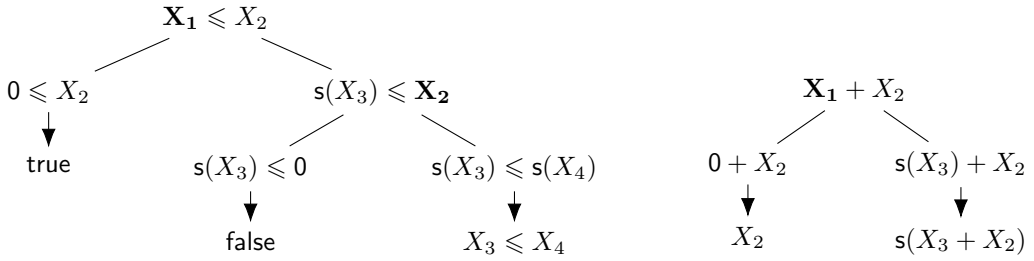
5. Nous rappelons qu'un motif est un terme $f(t_1, \ldots, t_n)$ tel que f est une opération et les $t_1, \ldots, t_n$ sont des termes de constructeurs.

— $\mathcal{T} = \mathit{leaf}(\pi, l \to r)$, où π est un motif et $l \to r$ est une variante de règle de $\mathcal{R}$ telle que $l = \pi$.

Définition 4 (Séquentialité inductive)**.** [2, 3] Une opération f d'un CB-TRS $\mathcal{R}$ est *inductivement séquentielle* ssi il existe un arbre définitionnel $\mathcal{T}$ de f tel que les règles contenues dans $\mathcal{T}$ sont *toutes et les seules* règles définissant f dans $\mathcal{R}$. Un CB-TRS $\mathcal{R}$ est *inductivement séquentiel* ssi toutes les opérations définies par $\mathcal{R}$ sont inductivement séquentielles.

Pour une opération, l'ensemble des feuilles de son arbre définitionnel est en bijection avec l'ensemble des règles la définissant. Les arbres définitionnels sont donc de taille finie.

Exemple 8. Nous présentons une représentation graphique de deux arbres définitionnels pour les opérations de $\mathcal{RR}$. Un nœud interne représente le motif du nœud correspondant de l'arbre définitionnel. Une feuille représente la partie droite de la règle contenue dans un nœud *leaf* de l'arbre. L'attribut de position des nœuds *branch* est représenté par la mise en gras du sous-terme dans le motif.



Définition 5 (Stratégie de sur-réduction nécessaire la plus extérieure)**.** [3] La fonction λ prend deux arguments, un terme t à racine opérationnelle et un arbre définitionnel $\mathcal{T}$, tels que $\mathit{pattern}(\mathcal{T})$ et t sont unifiables. Elle produit un ensemble de triplets $(p, l \to r, \sigma)$, où p est une position de t, $l \to r$ est une règle de réécriture et σ est un unificateur de $\mathit{pattern}(\mathcal{T})$ et t.

$$\lambda(t, \mathcal{T}) \ni \begin{cases} (\epsilon, l \to r, \sigma) & \text{si } \mathcal{T} = \mathit{leaf}(\pi, l \to r) \text{ et } \sigma = \mathit{mgu}(\pi, t) \\ (p, l \to r, \sigma) & \text{si } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k),\ \mathit{root}(t|_o) \in \mathcal{X}, \\ & \quad (p, l \to r, \sigma) \in \lambda(t, \mathcal{T}_i),\ \text{pour un } i \\ (p, l \to r, \sigma) & \text{si } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k),\ \mathit{root}(t|_o) \in \mathcal{C}, \\ & \quad \mathit{pattern}(\mathcal{T}_i) \leqslant t,\ \text{et } (p, l \to r, \sigma) \in \lambda(t, \mathcal{T}_i),\ \text{pour un } i \\ (o \cdot p, l \to r, \sigma \circ \tau) & \text{si } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k),\ \mathit{root}(t|_o) \in \mathcal{D}, \\ & \quad \tau = \mathit{mgu}(\pi, t) \text{ et } (p, l \to r, \sigma) \in \lambda(\tau(t|_o), \mathcal{T}_{\mathit{root}(t|_o)}) \end{cases}$$

Notons que le second cas ($\mathit{root}(t|_o) \in \mathcal{X}$) et troisième cas ($\mathit{root}(t|_o) \in \mathcal{C}$) de cette définition peuvent être rassemblés en un seul cas (i.e. $\mathit{root}(t|_o) \notin \mathcal{D}$) puisque la condition $\mathit{pattern}(\mathcal{T}_i) \leqslant t$ est trivialement vraie si $t|_o$ est une variable. Antoy et al. [3] le formule d'ailleurs de cette manière.

Un CB-TRS inductivement séquentiel peut être vu comme un ensemble d'arbres définitionnels $\{\mathcal{T}_f\}_{f \in \mathcal{D}}$ et l'on peut définir la stratégie de sur-réduction nécessaire la plus extérieure :

$$\lambda_{\mathcal{R}}(t) = \lambda(t, \mathcal{T}_{\mathit{root}(t)})$$

Exemple 9. Nous représentons la trace du calcul de λ pour l'étape initiale de la dérivation de sur-réduction de $x \leqslant y + z \leadsto_{2, R_4, \{y \mapsto 0\}} x \leqslant z$ (avec $R_4 = 0 + x \to x$), discutée dans l'exemple 7 :

$\lambda(x \leqslant y + z, \mathit{branch}(\mathbf{X_1} \leqslant X_2, 1, \ldots))$
$\quad \lambda(x \leqslant y + z, \mathit{branch}(\mathsf{s}(X_3) \leqslant \mathbf{X_2}, 2, \ldots))$

$$
\begin{array}{l}
\lambda(y+z, branch(\mathbf{Y_1}+Y_2, 1, \ldots)) \\
\quad \lambda(y+z, leaf(0+Y_2, 0+Y_2 \to Y_2)) \\
\quad \ni (\epsilon, R_4, \{y \mapsto 0, Y_2 \mapsto z\}) \\
\ni (\epsilon, R_4, \{y \mapsto 0, Y_2 \mapsto z\}) \\
\ni (2, R_4, \{x \mapsto \mathsf{s}(X_3), X_2 \mapsto 0+z, y \mapsto 0, Y_2 \mapsto z\}) \\
\ni (2, R_4, \{x \mapsto \mathsf{s}(X_3), X_2 \mapsto 0+z, y \mapsto 0, Y_2 \mapsto z\})
\end{array}
$$

La stratégie de sur-réduction nécessaire la plus extérieure est correcte et complète pour des substitutions de constructeurs dans la classe des CB-TRS inductivement séquentiels :

Theorème 4 (Correction et complétude de λ)**.** *[3] Soit $\mathcal{R}$ un CB-TRS inductivement séquentiel étendu avec les règles d'égalité. Alors, il existe une dérivation de sur-réduction $t \approx t' \overset{*}{\leadsto}_\sigma \mathsf{true}$ calculée par $\lambda_\mathcal{R}$ ssi σ est une substitution de constructeurs qui est une solution pour $t \approx t'$.*

Antoy et al [3] montrent que cette stratégie est optimale dans le sens où elle n'effectue pas de sur-réduction évitable, mais aussi en terme d'indépendance des solutions (une même substitution de constructeurs ne sera pas calculée dans deux dérivations différentes) et de longueur des dérivations (pour un système ne contenant que des règles linéaires à droite).

La stratégie de sur-réduction nécessaire la plus extérieure calcule toutes les substitutions de constructeurs qui sont solutions d'une équation. Les arbres définitionnels sont de taille finie car le nombre des règles du système est fini. Il en découle que λ, par définition, calcule un ensemble fini de triplets $(p, l \to r, \sigma)$. L'utilisation de la recherche en largeur d'abord, présentée en section 2.3, nous assure donc qu'une solution de taille finie, si elle existe, sera calculée en un temps fini. Nous exploitons cette idée dans la section suivante en l'appliquant aux politiques de contrôle d'accès.

Dans la suite, nous nous limitons à des systèmes de réécriture à base de constructeurs inductivement séquentiels et à l'utilisation de la stratégie de sur-réduction nécessaire la plus extérieure.

3 Contrôle d'accès

Nous présentons les principes des politiques de contrôle d'accès et comment elles peuvent être spécifiées formellement sous la forme de systèmes de réécriture. Enfin, nous montrons comment la sur-réduction peut être utilisée pour l'interrogation de politiques de contrôle d'accès.

3.1 Politiques de contrôle d'accès

Le contrôle d'accès est la partie d'un système d'information qui vérifie que ses utilisateurs sont autorisés à effectuer les actions qu'ils souhaitent sur les ressources du système. Une action peut être, par exemple, la lecture d'un fichier ou la suppression d'un élément d'une base donnée.

Il existe de nombreux modèles pour spécifier une politique de contrôle d'accès. Nous présentons les principaux ci-après. Afin d'en faciliter l'exposition, nous définissons un vocabulaire commun. Un *utilisateur* est une personne ou un programme interagissant avec le système. Une *ressource* est un élément du système d'information sur lequel les utilisateurs du système peuvent agir. Les *actions* définissent les manières dont les utilisateurs peuvent agir sur une ressource. Une *requête d'accès* est la demande d'un utilisateur pour effectuer une action particulière sur une ressource particulière. Finalement, les *autorisations* sont les réponses possibles à une requête d'accès, comme par exemple « acceptée » ou « rejetée ».

Le modèle standard Role-Based Access Control (RBAC) [1, 11] est centré autour de la notion de *rôle*. À un utilisateur sont assignés un ou plusieurs rôles. De plus, une hiérarchie entre rôles peut parfois être établie de telle manière qu'un rôle hérite des privilèges associés aux rôles qui lui sont subordonnés. La requête d'un utilisateur pour effectuer une certaine action sur une ressource

est alors autorisée si la permission d'effectuer cette action sur cette ressource est assignée à l'un des rôles de l'utilisateur.

Le modèle Mandatory Access Control (MAC) classifie les ressources avec des étiquettes, comme, par exemple, les niveaux d'accréditation militaires. Dans le modèle Discretionary Access Control (DAC), chaque ressource est possédée par un utilisateur et ce dernier peut gérer les droits d'accès des autres utilisateurs à cette ressource. Ces deux modèles limitent les actions possibles sur les ressources aux droits de lecture et écriture (ainsi qu'au droit d'exécution dans le cas des systèmes de fichier Unix). Il a été montré [16] que ces deux modèles peuvent être exprimés à l'aide de RBAC.

Dynamic Event-Based Access Control (DEBAC) [7] prend en compte l'historique des événements pour déterminer les droits d'un utilisateur. Par exemple, les actions passées de l'utilisateur peuvent influencer le calcul des autorisations d'accès de cet utilisateur.

Barker [5] a proposé le méta-model $\mathcal{M}$ permettant de représenter ces différentes politiques de contrôle d'accès dans un même et seul cadre, basé sur la notion de *catégorie*. Les catégories sont une généralisation du principe de rôles. Elles peuvent être utilisées pour classifier les utilisateurs mais aussi les autres entités utilisées pour modéliser la politique de contrôle d'accès comme les ressources, les permissions et, par exemple, les sites ou divisions d'une entreprise, ... D'une manière similaire aux hiérarchies de rôles, les catégories peuvent contenir des sous-catégories.

Les entités sont dénotées par des constantes dans un domaine à plusieurs sortes :

— Un ensemble dénombrable de catégories $\mathcal{C} = \{\mathsf{c}_1, \mathsf{c}_2, \ldots\}$,

— Un ensemble dénombrable d'utilisateurs $\mathcal{U} = \{\mathsf{u}_1, \mathsf{u}_2, \ldots\}$,

— Un ensemble dénombrable d'actions $\mathcal{A} = \{\mathsf{a}_1, \mathsf{a}_2, \ldots\}$,

— Un ensemble dénombrable de ressources $\mathcal{R} = \{\mathsf{r}_1, \mathsf{r}_2, \ldots\}$,

— Un ensemble fini $\mathcal{A}uth = \{\mathsf{grant}, \mathsf{deny}\}$ de réponses aux requêtes d'accès.

Dans $\mathcal{M}$, on considère les relations suivantes entre les entités :

— Assignation utilisateur-catégorie : $\mathcal{UC} \subseteq \mathcal{U} \times \mathcal{C}$, tel que $(u, c) \in \mathcal{UC}$ ssi à l'utilisateur u est assigné la catégorie c,

— Inclusion de catégories : $\mathcal{CC} \subseteq \mathcal{C} \times \mathcal{C}$, tel que $(c, c') \in \mathcal{CC}$ ssi la catégorie c inclue la catégorie c',

— Permissions : $\mathcal{CAR} \subseteq \mathcal{C} \times \mathcal{A} \times \mathcal{R}$, tel que $(c, a, r) \in \mathcal{CAR}$ ssi l'action a sur la ressource r peut être effectuée par un utilisateur assigné à la catégorie c,

— Autorisations : $\mathcal{UAR} \subseteq \mathcal{U} \times \mathcal{A} \times \mathcal{R} \times \mathcal{A}uth$, tel que $(u, a, r, auth) \in \mathcal{UAR}$ ssi à une requête par l'utilisateur u pour effectuer l'action a sur la ressource r est donnée l'autorisation $auth$.

La relation $\mathcal{UAR}$ satisfait l'axiome suivant :

$$\forall u \in \mathcal{U},\ \forall a \in \mathcal{A},\ \forall r \in \mathcal{R},\ \forall c \in \mathcal{C},$$
$$(u, c) \in \mathcal{UC} \wedge (\exists c', (c, c') \in \mathcal{CC} \wedge (c', a, r) \in \mathcal{CAR}) \Rightarrow (u, a, r, \mathsf{grant}) \in \mathcal{UAR}$$

Exemple 10. Nous présentons un exemple de spécification de politique de contrôle d'accès basé sur trois catégories : Administrative, Sales, Accounting. La catégorie Administrative contient les catégories Sales et Accounting. La politique définit deux permissions View et Edit ainsi que trois resources PasswdFile, SalesDB et AccountingDB.

La catégorie Administrative offre les permissions Edit et View sur la resource PasswdFile. Accounting offre les permissions Edit sur la resource AccountingDB et View sur la resource SalesDB. Finalement, la catégorie Sales offre les permission Edit sur SalesDB et View sur AccountingDB.

Aux trois utilisateurs Alice, Bob et Carol sont assignés les catégories[6] respectives Administrative, Sales et Accounting.

Ainsi, par exemple, Alice dispose de la permission Edit sur les trois ressources, alors que Bob ne peut éditer que la ressource SalesDB et Carol la ressource AccountingDB.

3.2 Spécifications formelles basées sur la réécriture

Bertolissi et Fernandez [6] ont défini une spécification formelle du méta-modèle $\mathcal{M}$ basée sur la réécriture de termes. La représentation sous la forme de système de réécriture donne une spécification formelle des politiques de contrôle d'accès. Il est alors possible d'en vérifier un certain nombre de propriétés de manière (semi-)automatique, telle la confluence et la terminaison.

Exemple 11. Il est possible de spécifier la politique de l'exemple 10 sous la forme d'un système de réécriture inductivement séquentiel $\mathcal{RP}$, comme présenté sur la figure 1. L'opération :: est l'opérateur *cons* de construction de listes, et ::: celui de concaténation de listes.

$$
\begin{array}{lrcl}
(\mathrm{car}_1) & car(\mathsf{Administrative}) & \to & (\mathsf{Edit}, \mathsf{PasswdFile}) :: (\mathsf{View}, \mathsf{PasswdFile}) :: \mathsf{nil} \\
(\mathrm{car}_2) & car(\mathsf{Accounting}) & \to & (\mathsf{Edit}, \mathsf{AccountingDB}) :: (\mathsf{View}, \mathsf{SalesDB}) :: \mathsf{nil} \\
(\mathrm{car}_3) & car(\mathsf{Sales}) & \to & (\mathsf{Edit}, \mathsf{SalesDB}) :: (\mathsf{View}, \mathsf{AccountingDB}) :: \mathsf{nil}
\end{array}
$$

$$
\begin{array}{lrcl@{\qquad}lrcl}
(\mathrm{cc}_1) & cc(\mathsf{Administrative}) & \to & \mathsf{Accounting} :: \mathsf{Sales} :: \mathsf{nil} & (\mathrm{uc}_1) & uc(\mathsf{Alice}) & \to & \mathsf{Administrative} :: \mathsf{nil} \\
(\mathrm{cc}_2) & cc(\mathsf{Accounting}) & \to & \mathsf{nil} & (\mathrm{uc}_2) & uc(\mathsf{Bob}) & \to & \mathsf{Sales} :: \mathsf{nil} \\
(\mathrm{cc}_3) & cc(\mathsf{Sales}) & \to & \mathsf{nil} & (\mathrm{uc}_3) & uc(\mathsf{Carol}) & \to & \mathsf{Accounting} :: \mathsf{nil}
\end{array}
$$

$$
(\mathrm{uar}) \quad uar(p, a, r) \to if((a, r) \in car^*(cc^*(uc(p))), \mathsf{grant}, \mathsf{deny})
$$

$$
\begin{array}{lrcl@{\qquad}lrcl}
(\mathrm{car}_1) & car^*(h :: t) & \to & car(h) ::: car^*(t) & (\mathrm{car}_2) & car^*(\mathsf{nil}) & \to & \mathsf{nil} \\
(\mathrm{cc}_1^*) & cc^*(h :: t) & \to & h :: cc(h) ::: cc^*(t) & (\mathrm{cc}_2^*) & cc^*(\mathsf{nil}) & \to & \mathsf{nil} \\
(\in_1) & x \in h :: t & \to & x \approx h \vee x \in t & (\in_2) & x \in \mathsf{nil} & \to & \mathsf{false} \\
(:::_1) & (h :: t) ::: l & \to & h :: (t ::: l) & (:::_2) & \mathsf{nil} ::: l & \to & l \\
(\mathrm{if}_1) & if(\mathsf{true}, x, y) & \to & x & (\mathrm{if}_2) & if(\mathsf{false}, x, y) & \to & y
\end{array}
$$

FIGURE 1 – $\mathcal{RP}$, une spécification formelle de la politique de l'exemple 10

Nous étendons l'opération d'égalité $\approx$ définie précédemment afin de pouvoir l'utiliser au sein de la spécification des politiques de contrôle d'accès. Nous la définissons pour qu'elle soit totale :

$$
\begin{array}{lll}
(\approx_1) & \mathsf{c}(X_1, \ldots, X_n) \approx \mathsf{c}(Y_1, \ldots, Y_n) \to \wedge_{i=1}^{n}(X_i \approx Y_i) & \forall \mathsf{c} \in \mathcal{C} \text{ d'arité } n \geqslant 0 \\
(\approx_2) & \mathsf{c}_1(X_1, \ldots, X_n) \approx \mathsf{c}_2(Y_1, \ldots, Y_m) \to \mathsf{false} & \forall \mathsf{c}_1, \mathsf{c}_2 \in \mathcal{C}_s \text{ d'arités } n, m \geqslant 0, \\
 & & \mathsf{c}_1 \neq \mathsf{c}_2, \text{ pour une sorte } s \in S
\end{array}
$$

$$
(\wedge_1) \quad \mathsf{true} \wedge X \to X \qquad (\wedge_2) \quad \mathsf{false} \wedge X \to \mathsf{false}
$$

Nous ajoutons les règles suivantes pour exprimer disjonction, négation et inégalité :

$$
(\vee_1) \quad \mathsf{true} \vee X \to \mathsf{true} \qquad (\vee_2) \quad \mathsf{false} \vee X \to X
$$

$$
(\neg_1) \quad \neg\mathsf{true} \to \mathsf{false} \qquad (\neg_2) \quad \neg\mathsf{false} \to \mathsf{true} \qquad (\not\approx) \quad X \not\approx Y \to \neg(X \approx Y)
$$

6. Par soucis de brièveté, cet exemple n'assigne qu'une catégorie à chaque utilisateur. Le modèle permet néanmoins d'associer plusieurs catégories à un utilisateur.

Les opérations définies par ces règles sont inductivement séquentielles. Leur ajout préserve donc la séquentialité inductive (et donc l'orthogonalité). On peut alors utiliser la stratégie de sur-réduction la plus nécessaire.

Spécifier une politique de sécurité comme un système de réécriture permet aussi d'exécuter directement la politique par simple réduction dans le système d'un terme représentant la demande d'accès.

Exemple 12. La figure 5 (en Annexe) montre une réduction pour le terme $uar(\mathsf{Alice}, \mathsf{Edit}, \mathsf{AccountingDB})$, permettant d'utiliser la politique pour savoir si Alice peut disposer de l'autorisation d'édition sur la ressource AccountingDB. Le résultat de la réduction grant donne une réponse positive à cette demande. (À chaque étape de réécriture, le redex qui est contracté est souligné.)

3.3 Interrogation d'une politique de contrôle d'accès

Il est possible d'utiliser la sur-réduction afin d'interroger, plus largement que par simple réécriture, une politique de contrôle d'accès spécifiée sous la forme d'un système de réécriture. La réécriture nous permettait de résoudre des demandes d'accès exprimées sous la forme de termes clos. La sur-réduction nous permet maintenant d'effectuer des requêtes sous la forme de termes contenant des variables et d'énumérer les solutions pour ces requêtes.

Exemple 13. Le système $\mathcal{RP}$ présenté dans l'exemple 11 est inductivement séquentiel. La figure 2 présente quelques uns des arbres définitionnels pour ses opérations.

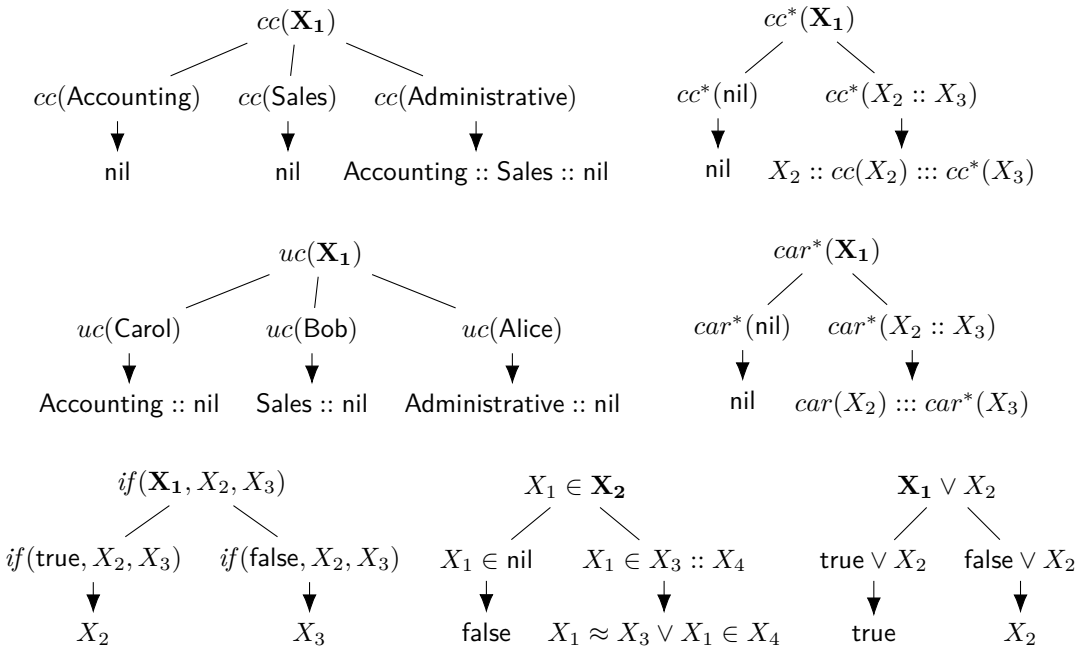


FIGURE 2 – Quelques arbres définitionnels pour le système de l'exemple 11

Exemple 14. La figure 6 (en Annexe) présente l'arbre de sur-réduction du terme $par(\mathsf{Alice}, \mathsf{Edit}, r) \approx \mathsf{grant}$, permettant d'interroger la politique définie dans l'exemple 11 pour connaître l'ensemble des

ressources pour lesquelles Alice peut avoir une autorisation d'édition. Les arêtes sont étiquetées avec les substitutions obtenues à chaque étape.

Nous avons présenté en section 2.3 une technique permettant d'énumérer les solutions d'une équation (ceci même si le système est non-terminant ou que l'équation présente un nombre non-borné de solutions). De plus, l'utilisation de la stratégie de sur-réduction nécessaire la plus extérieure permet que cette énumération soit efficace, en n'effectuant pas les étapes de sur-réduction évitables.

Comme nous venons de le voir, la classe des CB-TRS inductivement séquentiels est tout à fait adaptée à la spécification de politiques de contrôle d'accès et l'on peut aisément représenter une question que se pose un administrateur système sous la forme d'une équation à résoudre par sur-réduction. On peut de plus utiliser les opérations de conjonction et de négation pour répondre à des questions plus complexes, comme celles suggérées dans l'introduction.

4 Différentiels de sur-réduction

Dans cette section, nous développons une technique permettant d'identifier des exemples pour lesquelles deux systèmes de réécritures se comportent différemment. Les deux cas d'utilisation suggérés dans l'introduction correspondent à avoir (i) les deux systèmes définis entièrement sur le même ensemble de constructeurs, ou (ii) les deux systèmes ne partageant qu'un sous-ensemble de leurs constructeurs.

4.1 Différentiel sur un ensemble commun de constructeurs

Soient $\mathcal{C}$ un ensemble de constructeurs et $\mathcal{D}_1$, $\mathcal{D}_2$ deux ensembles d'opérations tels que $\mathcal{D}_1 \cap \mathcal{D}_2 \neq \emptyset$. Soient $\mathcal{F}_1 = (\mathcal{C} \uplus \mathcal{D}_1)$ et $\mathcal{F}_2 = (\mathcal{C} \uplus \mathcal{D}_2)$ deux signatures, et soient $\mathcal{R}_1 = (\mathcal{F}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{F}_2, R_2)$ deux TRS à base de constructeurs.

Pour un terme $t \in \mathcal{T}(\mathcal{F}_1 \cap \mathcal{F}_2, \mathcal{X})$ à racine opérationnelle, nous souhaitons caractériser les substitutions de constructeurs σ pour lesquelles $\sigma(t)$ n'est pas réductible dans $\mathcal{R}_1$ et dans $\mathcal{R}_2$ vers un même terme clos de constructeurs.

Définition 6 (Différentiel de sur-réduction)**.** Soit un terme $t \in \mathcal{T}(\mathcal{F}_1 \cap \mathcal{F}_2, \mathcal{X})$ à racine opérationnelle. On définit le *différentiel de sur-réduction* de t dans $\mathcal{R}_2$ par rapport à $\mathcal{R}_1$:

$$\mathbf{D}_{\mathcal{R}_1,\mathcal{R}_2}(t) \stackrel{\text{def}}{=} \{(\sigma, v_1, v_2) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}, \mathcal{X})) \times \mathcal{T}(\mathcal{C}) \times \mathcal{T}(\mathcal{C}) \mid \\ \sigma(t) \stackrel{*}{\to}_{\mathcal{R}_1} v_1 \text{ et } \sigma(t) \stackrel{*}{\to}_{\mathcal{R}_2} v_2 \text{ et } v_1 \neq v_2\}$$

Exemple 15. Nous considérons le CB-TRS $\mathcal{RP}$ et le CB-TRS $\mathcal{RP}'$ défini comme $\mathcal{RP}$ où l'on a remplacé la règle $cc(\mathsf{Administrative}) \to \mathsf{Accounting} :: \mathsf{Sales} :: \mathsf{nil}$ par $cc(\mathsf{Administrative}) \to \mathsf{Sales} :: \mathsf{nil}$. Cette modification du système revient à retirer la catégorie Accounting des sous-catégories de Administrative. On peut calculer le différentiel de sur-réduction $\mathbf{D}_{\mathcal{RP},\mathcal{RP}'}(uar(u,a,r))$ du terme $uar(u,a,r)$ dans $\mathcal{RP}'$ par rapport à $\mathcal{RP}$. Alors

$$\mathbf{D}_{\mathcal{RP},\mathcal{RP}'}(uar(u,a,r)) = \{\ (\{u \mapsto \mathsf{Alice}, a \mapsto \mathsf{Edit}, r \mapsto \mathsf{AccountingDB}\}, \mathsf{grant}, \mathsf{deny}), \\ (\{u \mapsto \mathsf{Alice}, a \mapsto \mathsf{View}, r \mapsto \mathsf{SalesDB}\}, \mathsf{grant}, \mathsf{deny})\ \}$$

Proposition 5.

$$\mathbf{D}_{\mathcal{R}_1,\mathcal{R}_2}(t) = \{(\sigma_{|\mathcal{V}(t)}, v_1, v_2) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}, \mathcal{X})) \times \mathcal{T}(\mathcal{C}) \times \mathcal{T}(\mathcal{C}) \mid \\ t \stackrel{*}{\leadsto}_{\mathcal{R}_1,\sigma} v_1 \ \textit{et}\ \sigma(t) \stackrel{*}{\to}_{\mathcal{R}_2} v_2 \ \textit{et}\ v_1 \neq v_2\}$$

Démonstration. Par les lemmes d'élévation et d'abaissement (lemme 2 et lemme 1). □

Cette formulation nous donne directement l'algorithme en figure 3.

Differential1

Entrée : $\mathcal{R}_1 = (\mathcal{F}_1, R_1)$, $\mathcal{R}_2 = (\mathcal{F}_2, R_2)$ deux CB-TRS et $t \in \mathcal{T}(\mathcal{F}_1 \cap \mathcal{F}_2, \mathcal{X})$ tels que $\mathcal{F}_1 = (\mathcal{C} \uplus \mathcal{D}_1)$, $\mathcal{F}_2 = (\mathcal{C} \uplus \mathcal{D}_2)$ et $\mathcal{D}_1 \cap \mathcal{D}_2 \neq \emptyset$.

Soit $\mathbf{D} = \emptyset$
pour σ, v tels que $t \overset{*}{\leadsto}_{\mathcal{R}_1,\sigma} v$ (en utilisant λ) **faire**
  si $\exists v' \in \mathcal{T}(\mathcal{C})$ tel que $\sigma(t) \overset{*}{\to}_{\mathcal{R}_2} v'$ et $v \neq v'$ **alors**
    $\mathbf{D} \leftarrow \mathbf{D} \cup \{(\sigma_{|\mathcal{V}(t)}, v, v')\}$
  fin si
fin pour
retourner **D**

FIGURE 3 – Différentiel de sur-réduction sur un ensemble commun de constructeurs

Il existe de nombreuses raisons pour lesquelles cet algorithme peut ne pas terminer :

— le nombre de solutions pour $t \overset{*}{\leadsto}_{\mathcal{R}_1,\sigma} v$ peut ne pas être borné,

— $\mathcal{R}_1$ peut ne pas être terminant,

— $\mathcal{R}_2$ peut ne pas être terminant.

Il serait possible de découper le travail de sur-réduction et de réécriture en morceaux (les étapes de sur-réduction et réécriture) afin d'exécuter en parallèle la recherche des différentes substitutions et les différentes réductions. Néanmoins, la formulation d'un tel algorithme serait nettement moins intelligible.

Pour palier ce problème d'une manière qui nous semble plus élégante, nous proposons la construction d'un système unique, représentant l'adjonction des deux systèmes. Dans ce système, nous pourrons directement sur-réduire un terme fabriqué adéquatement à partir du terme pour lequel nous souhaitons calculer le différentiel. La recherche en largeur d'abord des dérivations de sur-réduction, présentée en section 2.3, va nous permettre de ne pas souffrir du nombre potentiellement non-borné de solutions ou de la possible non-terminaison de nos systèmes, et ainsi de rendre le calcul du différentiel récursivement énumérable.

Nous définissons une transformation $\mathcal{R}_1 \# \mathcal{R}_2$ de deux CB-TRS $\mathcal{R}_1$ et $\mathcal{R}_2$ inductivement séquentiels. Nous montrons que le système obtenu est inductivement séquentiel et qu'il permet le calcul du différentiel de sur-réduction par simple sur-réduction.

Pour faciliter l'exposition de la technique, nous définissons trois opérations sur les systèmes :

— Deux opérations $\rho_k(\mathcal{R})$ ($k \in \{1, 2\}$) qui, pour un système $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$, créent une copie (par renommage) des opérations $o, p, q, \ldots \in \mathcal{D}$ en $o_k, p_k, q_k, \ldots$

— Une opération d'union disjointe de systèmes $\mathcal{R}_1 \uplus \mathcal{R}_2$

Définition 7 ($\rho_k(\mathcal{R})$)**.** Soit $\mathcal{R} = (\mathcal{F}, R)$ un CB-TRS inductivement séquentiel et $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$.

Pour $k \in \{1, 2\}$, ρ_k est une fonction bijective qui à $o \in \mathcal{D}$ associe o_k.

On étend naturellement ρ_k aux termes et règles :

— $x \in \mathcal{X}$, alors $\rho_k(x) = x$

— $t = c(t_1, \ldots, t_n)$ et $c \in \mathcal{C}$, alors $\rho_k(t) = c(\rho_k(t_1), \ldots, \rho_k(t_n))$

— $t = o(t_1, \ldots, t_n)$ et $o \in \mathcal{D}$, alors $\rho_k(t) = o_k(\rho_k(t_1), \ldots, \rho_k(t_n))$

— $l \to r \in R$, alors $\rho_k(l \to r) = \rho_k(l) \to \rho_k(r)$

Alors, $\rho_k(\mathcal{R}) = (\mathcal{C} \uplus \rho_k(\mathcal{D}), \rho_k(R))$.

Lemme 6. *Soit $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ un CB-TRS inductivement séquentiel, alors $\rho_k(\mathcal{R})$ est un CB-TRS inductivement séquentiel.*

Démonstration. ρ_k effectue un simple renommage des opérations. Pour toute opération $o \in \mathcal{D}$, il existe un arbre définitionnel $\mathcal{T}_o$. Alors, il existe un arbre définitionnel pour $\rho_k(o)$, identique à $\mathcal{T}_o$ à renommage près des opérations par ρ_k. □

Définition 8 ($\mathcal{R}_1 \uplus \mathcal{R}_2$)**.** Soit $\mathcal{R}_1 = (\mathcal{C} \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C} \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels tels que $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$, alors $\mathcal{R}_1 \uplus \mathcal{R}_2 = ((\mathcal{C}_1 \cup \mathcal{C}_2) \uplus (\mathcal{D}_1 \uplus \mathcal{D}_2), R_1 \uplus R_2)$

Lemme 7. *Soit $\mathcal{R}_1 = (\mathcal{C} \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C} \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels tels que $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$, alors $\mathcal{R}_1 \uplus \mathcal{R}_2$ est un CB-TRS inductivement séquentiel.*

Démonstration. Les ensembles d'opérations de $\mathcal{R}_1$ et de $\mathcal{R}_2$ sont disjoints. Toutes les opérations de ces systèmes sont inductivement séquentielles. Donc, l'union disjointe des systèmes est inductivement séquentielle. □

Définition 9 ($\mathcal{R}_1 \# \mathcal{R}_2$)**.** Soient $\mathcal{R}_1 = (\mathcal{C} \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C} \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels, alors $\mathcal{R}_1 \# \mathcal{R}_2$ est le CB-TRS obtenu à partir de $\rho_1(\mathcal{R}_1) \uplus \rho_2(\mathcal{R}_2)$ en l'étendant avec l'égalité ($\approx$, $\wedge$, $\neg$, et $\not\approx$).

Proposition 8. *Soient $\mathcal{R}_1 = (\mathcal{C} \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C} \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels, alors $\mathcal{R}_1 \# \mathcal{R}_2$ est un CB-TRS inductivement séquentiel.*

Démonstration. Par les lemmes 6 et 7, en observant que $\rho_1(\mathcal{D}_1) \cap \rho_2(\mathcal{D}_2) = \emptyset$, et le fait que l'égalité, la conjonction, la négation et l'inégalité, telles que définies précédemment, sont inductivement séquentielles. □

Antoy et al [3] montrent une proposition qui nous sera utile par la suite.

Proposition 9. *[3] Soit $\mathcal{R} = (\mathcal{F}, R)$ un CB-TRS inductivement séquentiel. Soit $\mathcal{R}'$ le CB-TRS obtenu à partir de $\mathcal{R}$ en l'étendant avec les règles d'égalité. (On considère que les symboles n'appartiennent pas déjà à la signature de $\mathcal{R}$.) Alors, pour tous termes t et u,*

$$t \approx u \text{ se réduit dans } \mathcal{R}' \text{ vers } \mathsf{true}$$
$$\iff t \text{ et } u \text{ se réduisent dans } \mathcal{R} \text{ vers le même terme clos de constructeurs}$$

Avant de prouver la correction et complétude de notre construction, nous nous assurons du bon comportement des opérations $\wedge$, $\neg$ et $\not\approx$, celui de l'égalité étant assuré par la proposition 9.

Lemme 10. *Soit $\mathcal{R} = (\mathcal{F}, R)$ un CB-TRS inductivement séquentiel. Soit $\mathcal{R}'$ le CB-TRS obtenu à partir de $\mathcal{R}$ en l'étendant avec la conjonction, la négation et l'inégalité. (On considère que les symboles n'appartiennent pas déjà à la signature de $\mathcal{R}$.) Alors, pour tous termes t, t' et t'',*

1. *$t \wedge t' \wedge t''$ se réduit dans $\mathcal{R}'$ vers* true *ssi t, t', t'' se réduisent dans $\mathcal{R}$ vers* true
2. *$\neg t$ se réduit dans $\mathcal{R}'$ vers* true *ssi t se réduit dans $\mathcal{R}$ vers* false
3. *$t \approx t'$ se réduit dans $\mathcal{R}'$ vers* false
   ssi t et t' se réduisent dans $\mathcal{R}$ vers deux formes normales différentes

Démonstration. Nous faisons un sketch de preuve. La preuve complète est disponible en Annexe.

Pour montrer les équivalences 1. et 2., nous procédons par une simple analyse de cas sur la dernière règle utilisée dans la réduction.

Pour l'équivalence 3., pour montrer que t et t' se réduisent vers deux formes normales différentes, nous procédons par induction sur le nombre d'application de règles (définissant) $\approx$ dans la dérivation $t \approx t' \xrightarrow{*} \mathsf{false}$. Pour montrer le sens opposé, nous procédons par induction sur la hauteur cumulée des deux formes normales de t et t'. □

Theorème 11. *(Correction) Soient $\mathcal{R}_1 = (\mathcal{C} \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C} \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels et $t \in \mathcal{T}(\mathcal{C} \uplus (\mathcal{D}_1 \cap \mathcal{D}_2), \mathcal{X})$.*

$$(\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \# \mathcal{R}_2, \sigma} \mathsf{true} \quad \Longrightarrow \quad (\sigma_{|\mathcal{V}(t)}, \sigma(x), \sigma(y)) \in \mathbf{D}_{\mathcal{R}_1, \mathcal{R}_2}(t)$$

Démonstration. Supposons $(\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \# \mathcal{R}_2, \sigma} \mathsf{true}$.

En abaissant la dérivation (lemme 1), on obtient

$$\sigma(\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$$

D'après le lemme 10,

$$\sigma(\rho_1(t) \approx x) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true} \text{ et } \sigma(\rho_2(t) \approx y) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true} \text{ et } \sigma(x \not\approx y) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$$
$$\Longrightarrow \quad \begin{cases} \sigma(\rho_1(t)) \approx \sigma(x) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true} \\ \text{et } \sigma(\rho_2(t)) \approx \sigma(y) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true} \\ \text{et } \sigma(x) \not\approx \sigma(y) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true} \end{cases}$$

Par la proposition 9, $\sigma(\rho_1(t)) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \sigma(x)$ et $\sigma(\rho_2(t)) \overset{*}{\to}_{\mathcal{R}_1 \# \mathcal{R}_2} \sigma(y)$, où $\sigma(x)$ et $\sigma(y)$ sont des termes clos de constructeurs. De plus, par le lemme 10, $\sigma(x) \neq \sigma(y)$

$\sigma(\rho_1(t))$, resp. $\sigma(\rho_2(t))$, ne peut être réécrit que par la partie $\rho_1(\mathcal{R}_1)$ de $\mathcal{R}_1 \# \mathcal{R}_2$, resp. $\rho_2(\mathcal{R}_2)$.

$$\sigma(\rho_1(t)) \overset{*}{\to}_{\rho_1(\mathcal{R}_1)} \sigma(x) \text{ et } \sigma(\rho_2(t)) \overset{*}{\to}_{\rho_2(\mathcal{R}_2)} \sigma(y) \text{ et } \sigma(x) \neq \sigma(y)$$

Par définition, ρ_1 et ρ_2 sont bijectives, et

$$\sigma(t) \overset{*}{\to}_{\mathcal{R}_1} \sigma(x) \text{ et } \sigma(t) \overset{*}{\to}_{\mathcal{R}_2} \sigma(y) \text{ et } \sigma(x) \neq \sigma(y)$$

Donc, par définition de $\mathbf{D}_{\mathcal{R}_1, \mathcal{R}_2}(t)$, $(\sigma_{|\mathcal{V}(t)}, \sigma(x), \sigma(y)) \in \mathbf{D}_{\mathcal{R}_1, \mathcal{R}_2}(t)$. □

Theorème 12. *(Complétude) Soient $\mathcal{R}_1 = (\mathcal{C} \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C} \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels et $t \in \mathcal{T}(\mathcal{C} \uplus (\mathcal{D}_1 \cap \mathcal{D}_2), \mathcal{X})$.*

$$(\sigma, v_1, v_2) \in \mathbf{D}_{\mathcal{R}_1, \mathcal{R}_2}(t) \quad \Longrightarrow \quad (\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \# \mathcal{R}_2, \sigma'} \mathsf{true},$$
où $\sigma' = \sigma \cup \{x \mapsto v_1, y \mapsto v_2\}[\mathcal{V}(t) \cup \{x, y\}]$ et x, y deux variables fraîches

Démonstration. Soit σ tel que $(\sigma, v_1, v_2) \in \mathbf{D}_{\mathcal{R}_1, \mathcal{R}_2}(t)$.

$\mathcal{R}_1$ et $\mathcal{R}_2$ sont inductivement séquentiels. D'après la proposition 8, $\mathcal{R}_1 \# \mathcal{R}_2$ est inductivement séquentiel, et donc confluent. Il nous suffit donc d'exhiber une dérivation de sur-réduction pour $(\rho_1(t) = x \wedge \rho_2(t) = y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \# \mathcal{R}_2, \sigma'} \mathsf{true}$.

Par définition de $\mathbf{D}_{\mathcal{R}_1, \mathcal{R}_2}(t)$,

$$v_1, v_2 \in \mathcal{T}(\mathcal{C}, \mathcal{X}),\ \sigma(t) \overset{*}{\to}_{\mathcal{R}_1} v_1 \text{ et } \sigma(t) \overset{*}{\to}_{\mathcal{R}_2} v_2 \text{ et } v_1 \neq v_2$$

ρ_1 et ρ_2 sont bijectives,

$\rho_1(\sigma(t)) \xrightarrow{*}_{\rho_1(\mathcal{R}_1)} v_1$ et $\rho_2(\sigma(t)) \xrightarrow{*}_{\rho_2(\mathcal{R}_2)} v_2$ et $v_1 \neq v_2$

ρ_1 et ρ_2 ne renomment que les opérations et σ est normalisée. Donc,

$\sigma(\rho_1(t)) \xrightarrow{*}_{\rho_1(\mathcal{R}_1)} v_1$ et $\sigma(\rho_2(t)) \xrightarrow{*}_{\rho_2(\mathcal{R}_2)} v_2$ et $v_1 \neq v_2$

$\sigma(\rho_1(t))$, resp. $\sigma(\rho_2(t))$, ne peut être réécrit que par la partie $\rho_1(\mathcal{R}_1)$ de $\mathcal{R}_1 \# \mathcal{R}_2$, resp. $\rho_2(\mathcal{R}_2)$.

$\sigma(\rho_1(t)) \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} v_1$ et $\sigma(\rho_2(t)) \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} v_2$ et $v_1 \neq v_2$

D'après la proposition 9 et le lemme 10,

$\sigma(\rho_1(t)) \approx v_1 \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$ et $\sigma(\rho_2(t)) \approx v_2 \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$ et $v_1 \not\approx v_2 \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$

$\implies (\sigma(\rho_1(t)) \approx v_1 \ \wedge \ \sigma(\rho_2(t)) \approx v_2 \ \wedge \ v_1 \not\approx v_2) \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$

$\implies \sigma(\rho_1(t) \approx v_1 \ \wedge \ \rho_2(t) \approx v_2 \ \wedge \ v_1 \not\approx v_2) \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$

Soit $\theta = \sigma \cup \{x \mapsto v_1, y \mapsto v_2\}$ avec x et y deux variables fraîches.

$\theta(\rho_1(t) \approx x \ \wedge \ \rho_2(t) \approx y \ \wedge \ x \not\approx y) \xrightarrow{*}_{\mathcal{R}_1 \# \mathcal{R}_2} \mathsf{true}$

θ est normalisée et l'on peut élever la dérivation (lemme 2)

$\exists \sigma',$ tel que $\theta = id \circ \sigma'[\mathcal{V}(t) \cup \{x, y\}]$

$(\rho_1(t) \approx x \ \wedge \ \rho_2(t) \approx y \ \wedge \ x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \# \mathcal{R}_2, \sigma'} \mathsf{true}$ □

En utilisant la recherche en largeur d'abord, présentée en section 2.3, et la construction $\mathcal{R}_1 \# \mathcal{R}_2$, on peut donc trouver un élément de taille fini de $\mathbf{D}_{\mathcal{R}_1,\mathcal{R}_2}(t)$, s'il en existe, en un temps fini et ceci même si l'un (ou les deux) des systèmes est non-terminant. $\mathbf{D}_{\mathcal{R}_1,\mathcal{R}_2}(t)$ est donc récursivement énumérable.

4.2 Différentiel sur deux ensembles de constructeurs

Soient $\mathcal{F}_1 = (\mathcal{C}_1 \uplus \mathcal{D}_1)$ et $\mathcal{F}_2 = (\mathcal{C}_2 \uplus \mathcal{D}_2)$ deux signatures telles que $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$ et $\mathcal{D}_1 \cap \mathcal{D}_2 \neq \emptyset$, et soient $\mathcal{R}_1 = (\mathcal{F}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{F}_2, R_2)$ deux TRS à base de constructeurs.

Pour un ensemble de constructeurs $\mathcal{C}$ sur un ensemble de sortes S, on définit une famille de constructeurs $\{\bot_s \mid s \in S\}$, où les $\bot_s$ sont des symboles n'appartenant pas déjà à $\mathcal{C}$. Nous notons $\mathcal{C}^\bot = \mathcal{C} \uplus \{\bot_s \mid s \in S\}$. Pour simplifier la présentation, nous utiliserons $\bot$ pour dénoter n'importe quelle instance $\bot_s$, quelque soit $s \in S$.

Pour un terme $t \in \mathcal{T}(\mathcal{F}_1 \cap \mathcal{F}_2, \mathcal{X})$ à racine opérationnelle, nous souhaitons caractériser les substitutions de constructeurs σ pour lesquelles $\sigma(t)$ n'est pas réductible dans $\mathcal{R}_1$ et dans $\mathcal{R}_2$ vers un même terme clos de constructeurs.

Définition 10 (Différentiel de sur-réduction)**.** Soit un terme $t \in \mathcal{T}(\mathcal{F}_1 \cap \mathcal{F}_2, \mathcal{X})$ à racine opérationnelle.

$$\mathbf{D}^{-}_{\mathcal{R}_1,\mathcal{R}_2}(t) \overset{\text{def}}{=} \{(\sigma, v_1, \bot) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}_1, \mathcal{X})) \times \mathcal{T}(\mathcal{C}_1) \times \{\bot\} \mid \sigma(t) \xrightarrow{*}_{\mathcal{R}_1} v_1 \text{ et } \sigma(t) \notin \mathcal{T}(\mathcal{C}_2 \uplus \mathcal{D}_2, \mathcal{X})\}$$

$$\mathbf{D}^{+}_{\mathcal{R}_1,\mathcal{R}_2}(t) \overset{\text{def}}{=} \{(\sigma, \bot, v_2) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}_2, \mathcal{X})) \times \{\bot\} \times \mathcal{T}(\mathcal{C}_2) \mid \sigma(t) \xrightarrow{*}_{\mathcal{R}_2} v_2 \text{ et } \sigma(t) \notin \mathcal{T}(\mathcal{C}_1 \uplus \mathcal{D}_1, \mathcal{X})\}$$

$$\mathbf{D}^{\neq}_{\mathcal{R}_1,\mathcal{R}_2}(t) \overset{\text{def}}{=} \{(\sigma, v_1, v_2) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}_1 \cap \mathcal{C}_2, \mathcal{X})) \times \mathcal{T}(\mathcal{C}_1) \times \mathcal{T}(\mathcal{C}_2) \mid \sigma(t) \xrightarrow{*}_{\mathcal{R}_1} v_1 \text{ et } \sigma(t) \xrightarrow{*}_{\mathcal{R}_2} v_2 \text{ et } v_1 \neq v_2\}$$

Alors le *différentiel de sur-réduction* de t dans $\mathcal{R}_2$ par rapport à $\mathcal{R}_1$ est

$$\mathbf{D}^{\perp}_{\mathcal{R}_1,\mathcal{R}_2}(t) \subseteq (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{X})) \times (\mathcal{T}(\mathcal{C}_1) \cup \perp) \times (\mathcal{T}(\mathcal{C}_2) \cup \perp)$$
$$\mathbf{D}^{\perp}_{\mathcal{R}_1,\mathcal{R}_2}(t) \stackrel{\text{def}}{=} \mathbf{D}^{-}_{\mathcal{R}_1,\mathcal{R}_2}(t) \cup \mathbf{D}^{\neq}_{\mathcal{R}_1,\mathcal{R}_2}(t) \cup \mathbf{D}^{+}_{\mathcal{R}_1,\mathcal{R}_2}(t)$$

Exemple 16. Nous considérons le TRS $\mathcal{RP}$ et le TRS $\mathcal{RP}''$ comme $\mathcal{RP}$ où l'on a ajouté la constante Dan et la règle $pca(\mathsf{Dan}) \to \mathsf{Sales} :: \mathsf{nil}$. Cette modification du système revient à ajouter l'utilisateur Dan en lui assignant la catégorie Sales. On peut calculer le différentiel de sur-réduction $\mathbf{D}^{\perp}_{\mathcal{RP},\mathcal{RP}''}(uar(u,a,r))$ du terme $uar(u,a,r)$ dans $\mathcal{RP}''$ par rapport à $\mathcal{RP}$. Alors

$$\begin{aligned}\mathbf{D}^{\perp}_{\mathcal{RP},\mathcal{RP}''}(uar(u,a,r)) = \{\ & (\{u \mapsto \mathsf{Dan}; a \mapsto \mathsf{Edit}; r \mapsto \mathsf{SalesDB}\}, \perp, \mathsf{grant}),\\ & (\{u \mapsto \mathsf{Dan}; a \mapsto \mathsf{View}; r \mapsto \mathsf{AccountingDB}\}, \perp, \mathsf{grant}),\\ & (\{u \mapsto \mathsf{Dan}; a \mapsto \mathsf{Edit}; r \mapsto \mathsf{PasswdFile}\}, \perp, \mathsf{deny}),\\ & (\{u \mapsto \mathsf{Dan}; a \mapsto \mathsf{Edit}; r \mapsto \mathsf{AccountingDB}\}, \perp, \mathsf{deny}),\\ & (\{u \mapsto \mathsf{Dan}; a \mapsto \mathsf{View}; r \mapsto \mathsf{PasswdFile}\}, \perp, \mathsf{deny}),\\ & (\{u \mapsto \mathsf{Dan}; a \mapsto \mathsf{View}; r \mapsto \mathsf{SalesDB}\}, \perp, \mathsf{deny})\ \}\end{aligned}$$

Dans cette section, nous définissons une transformation $\mathcal{R}_1 \#_{\perp} \mathcal{R}_2$ de deux CB-TRS $\mathcal{R}_1$ et $\mathcal{R}_2$ inductivement séquentiels. Nous montrons que le système obtenu est inductivement séquentiel et qu'il permet le calcul du différentiel de sur-réduction par simple sur-réduction.

Pour cela, nous supposons qu'il existe une transformation $\tau_{\mathcal{C}}(\mathcal{R})$, qui, pour un système $\mathcal{R} = (\mathcal{C}_0 \uplus \mathcal{D}, R)$ inductivement séquentiel et un ensemble de constructeurs $\mathcal{C} \supseteq \mathcal{C}_0$, construit un système inductivement séquentiel vérifiant les propriétés suivantes :

$$\forall t \in \mathcal{T}(\mathcal{C}^{\perp} \uplus \mathcal{D}, \mathcal{X}), \quad t \stackrel{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R})} v \implies v \in \mathcal{T}(\mathcal{C}_0) \cup \{\perp\} \tag{1}$$
$$\forall t \in \mathcal{T}(\mathcal{C}^{\perp} \uplus \mathcal{D}, \mathcal{X}), \quad t \stackrel{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R})} v \text{ et } v \in \mathcal{T}(\mathcal{C}_0) \implies t \in \mathcal{T}(\mathcal{C}_0 \uplus \mathcal{D}, \mathcal{X}) \text{ et } t \stackrel{*}{\to}_{\mathcal{R}} v \tag{2}$$
$$\forall t \in \mathcal{T}(\mathcal{C}^{\perp} \uplus \mathcal{D}, \mathcal{X}), \quad t \stackrel{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R})} \perp \implies t \notin \mathcal{T}(\mathcal{C}_0 \uplus \mathcal{D}, \mathcal{X}) \tag{3}$$
$$\forall t \in \mathcal{T}(\mathcal{C}_0 \uplus \mathcal{D}, \mathcal{X}), \quad t \stackrel{*}{\to}_{\mathcal{R}} v \text{ et } v \in \mathcal{T}(\mathcal{C}_0) \implies t \stackrel{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R})} v \tag{4}$$
$$\forall t \notin \mathcal{T}(\mathcal{C}_0 \uplus \mathcal{D}, \mathcal{X}), \quad t \stackrel{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R})} \perp \tag{5}$$

Définition 11 ($\mathcal{R}_1 \#_{\perp} \mathcal{R}_2$)**.** Soient $\mathcal{R}_1 = (\mathcal{C}_1 \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C}_2 \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels et $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$, alors $\mathcal{R}_1 \#_{\perp} \mathcal{R}_2 = \tau_{\mathcal{C}}(\mathcal{R}_1) \# \tau_{\mathcal{C}}(\mathcal{R}_2)$.

Proposition 13. *Soient $\mathcal{R}_1 = (\mathcal{C}_1 \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C}_2 \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels, alors $\mathcal{R}_1 \#_{\perp} \mathcal{R}_2$ est un CB-TRS inductivement séquentiel.*

Démonstration. Soit $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Notons que $\tau_{\mathcal{C}}(\mathcal{R}_1)$ et $\tau_{\mathcal{C}}(\mathcal{R}_2)$ sont définis sur le même ensemble de constructeurs $\mathcal{C}^{\perp}$. Donc $\tau_{\mathcal{C}}(\mathcal{R}_1) \# \tau_{\mathcal{C}}(\mathcal{R}_2)$ est correctement défini. Par hypothèse, $\tau_{\mathcal{C}}(\mathcal{R}_1)$ et $\tau_{\mathcal{C}}(\mathcal{R}_2)$ sont inductivement séquentiels. Alors, par la proposition 8, $\mathcal{R}_1 \#_{\perp} \mathcal{R}_2$ est inductivement séquentiel. □

Theorème 14. *(Correction) Soient $\mathcal{R}_1 = (\mathcal{C}_1 \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C}_2 \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels et $t \in \mathcal{T}((\mathcal{C}_1 \cap \mathcal{C}_2) \uplus (\mathcal{D}_1 \cap \mathcal{D}_2), \mathcal{X})$.*

$$(\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \stackrel{*}{\rightsquigarrow}_{\mathcal{R}_1 \#_{\perp} \mathcal{R}_2, \sigma} \mathsf{true} \quad \implies \quad (\sigma_{|\mathcal{V}(t)}, \sigma(x), \sigma(y)) \in \mathbf{D}^{\perp}_{\mathcal{R}_1,\mathcal{R}_2}(t)$$

Démonstration. Soit $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Par le théorème 11,

$$\begin{aligned}
&(\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \#_\perp \mathcal{R}_2, \sigma} \mathsf{true} \\
&\quad\Longrightarrow\quad (\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\tau_{\mathcal{C}}(\mathcal{R}_1) \# \tau_{\mathcal{C}}(\mathcal{R}_2), \sigma} \mathsf{true} \\
&\quad\Longrightarrow\quad (\sigma_{|\mathcal{V}(t)}, \sigma(x), \sigma(y)) \in \mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t)
\end{aligned}$$

Il nous suffit donc de montrer que $\mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t) \subseteq \mathbf{D}^{\perp}_{\mathcal{R}_1, \mathcal{R}_2}(t)$. Par définition, nous avons

$$\begin{aligned}
\mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t) = \{(\sigma, v_1, v_2) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}^{\perp}, \mathcal{X})) \times \mathcal{T}(\mathcal{C}^{\perp}) \times \mathcal{T}(\mathcal{C}^{\perp}) \mid \\
\sigma(t) \overset{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R}_1)} v_1 \text{ et } \sigma(t) \overset{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R}_2)} v_2 \text{ et } v_1 \neq v_2\}
\end{aligned}$$

Soit $(\sigma, v_1, v_2) \in \mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t)$. Nous éliminons d'abord les cas impossibles.

D'après la propriété (1), $v_1 \in \mathcal{T}(\mathcal{C}_1) \cup \{\perp\}$ et $v_2 \in \mathcal{T}(\mathcal{C}_2) \cup \{\perp\}$. De plus, v_1 et v_2 ont la même sorte. Or $v_1 \neq v_2$ et donc v_1 et v_2 ne peuvent appartenir en même temps à $\{\perp_s \mid s \in S\}$.

Il subsiste donc trois cas :

$$\begin{cases}
v_1 \in \mathcal{T}(\mathcal{C}_1) \text{ et } v_2 \in \mathcal{T}(\mathcal{C}_2) \\
\text{ou } v_1 \in \mathcal{T}(\mathcal{C}_1) \text{ et } v_2 = \perp \\
\text{ou } v_1 = \perp \text{ et } v_2 \in \mathcal{T}(\mathcal{C}_2)
\end{cases}$$

En utilisant les propriétés (2) et (3), nous ramenons $\sigma(t) \overset{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R}_1)} v_1$ et $\sigma(t) \overset{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R}_2)} v_2$ à des dérivations dans $\mathcal{R}_1$ et $\mathcal{R}_2$:

$$\begin{cases}
v_1 \in \mathcal{T}(\mathcal{C}_1) \text{ et } v_2 \in \mathcal{T}(\mathcal{C}_2) \text{ et } \sigma(t) \overset{*}{\to}_{\mathcal{R}_1} v_1 \text{ et } \sigma(t) \overset{*}{\to}_{\mathcal{R}_2} v_2 \\
\text{ou } v_1 \in \mathcal{T}(\mathcal{C}_1) \text{ et } v_2 = \perp \text{ et } \sigma(t) \overset{*}{\to}_{\mathcal{R}_1} v_1 \text{ et } \sigma(t) \notin \mathcal{T}(\mathcal{C}_2 \uplus \mathcal{D}_2, \mathcal{X}) \\
\text{ou } v_1 = \perp \text{ et } v_2 \in \mathcal{T}(\mathcal{C}_2) \text{ et } \sigma(t) \overset{*}{\to}_{\mathcal{R}_2} v_2 \text{ et } \sigma(t) \notin \mathcal{T}(\mathcal{C}_1 \uplus \mathcal{D}_1, \mathcal{X})
\end{cases}$$

C'est à dire,

$$(\sigma, v_1, v_2) \in \mathbf{D}^{\neq}_{\mathcal{R}_1, \mathcal{R}_2}(t) \text{ ou } (\sigma, v_1, v_2) \in \mathbf{D}^{-}_{\mathcal{R}_1, \mathcal{R}_2}(t) \text{ ou } (\sigma, v_1, v_2) \in \mathbf{D}^{+}_{\mathcal{R}_1, \mathcal{R}_2}(t) \qquad \square$$

Theorème 15. *(Complétude) Soient $\mathcal{R}_1 = (\mathcal{C}_1 \uplus \mathcal{D}_1, R_1)$ et $\mathcal{R}_2 = (\mathcal{C}_2 \uplus \mathcal{D}_2, R_2)$ deux CB-TRS inductivement séquentiels et $t \in \mathcal{T}((\mathcal{C}_1 \cap \mathcal{C}_2) \uplus (\mathcal{D}_1 \cap \mathcal{D}_2), \mathcal{X})$.*

$$\begin{aligned}
&(\sigma, v_1, v_2) \in \mathbf{D}^{\perp}_{\mathcal{R}_1, \mathcal{R}_2}(t) \quad\Longrightarrow\quad (\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \#_\perp \mathcal{R}_2, \sigma'} \mathsf{true}, \\
&\textit{où } \sigma' = \sigma \cup \{x \mapsto v_1, y \mapsto v_2\}[\mathcal{V}(t) \cup \{x, y\}] \textit{ et } x, y \textit{ deux variables fraîches}
\end{aligned}$$

Démonstration. Soit $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Par le théorème 12,

$$\begin{aligned}
&(\sigma, v_1, v_2) \in \mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t) \\
&\quad\Longrightarrow\quad (\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\tau_{\mathcal{C}}(\mathcal{R}_1) \# \tau_{\mathcal{C}}(\mathcal{R}_2), \sigma'} \mathsf{true} \\
&\qquad \text{où } \sigma' = \sigma \cup \{x \mapsto v_1, y \mapsto v_2\}[\mathcal{V}(t) \cup \{x, y\}] \text{ et } x, y \text{ deux variables fraîches} \\
&\quad\Longrightarrow\quad (\rho_1(t) \approx x \wedge \rho_2(t) \approx y \wedge x \not\approx y) \overset{*}{\rightsquigarrow}_{\mathcal{R}_1 \#_\perp \mathcal{R}_2, \sigma'} \mathsf{true}
\end{aligned}$$

Il nous suffit donc de montrer que $\mathbf{D}^{\perp}_{\mathcal{R}_1, \mathcal{R}_2}(t) \subseteq \mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t)$. Par définition, nous avons

$$\begin{aligned}
\mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1), \tau_{\mathcal{C}}(\mathcal{R}_2)}(t) = \{(\sigma, v_1, v_2) \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}^{\perp}, \mathcal{X})) \times \mathcal{T}(\mathcal{C}^{\perp}) \times \mathcal{T}(\mathcal{C}^{\perp}) \mid \\
\sigma(t) \overset{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R}_1)} v_1 \text{ et } \sigma(t) \overset{*}{\to}_{\tau_{\mathcal{C}}(\mathcal{R}_2)} v_2 \text{ et } v_1 \neq v_2\}
\end{aligned}$$

Nous observons trois cas :

$(\sigma, v_1, v_2) \in \mathbf{D}^{\neq}_{\mathcal{R}_1,\mathcal{R}_2}(t)$ Nous avons $\sigma(t) \xrightarrow{*}_{\mathcal{R}_1} v_1$ et $\sigma(t) \xrightarrow{*}_{\mathcal{R}_2} v_2$ et $v_1 \neq v_2$.
De plus, $\sigma \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}_1 \cap \mathcal{C}_2, \mathcal{X}))$ et $v_1 \in \mathcal{T}(\mathcal{C}_1)$ et $v_2 \in \mathcal{T}(\mathcal{C}_2)$.
Donc $\sigma(t) \in \mathcal{T}((\mathcal{C}_1 \cap \mathcal{C}_2) \uplus (\mathcal{D}_1 \cap \mathcal{D}_2), \mathcal{X})$.
Ainsi, d'après la propriété (4), $\sigma(t) \xrightarrow{*}_{\tau_{\mathcal{C}}(\mathcal{R}_1)} v_1$ et $\sigma(t) \xrightarrow{*}_{\tau_{\mathcal{C}}(\mathcal{R}_2)} v_2$.
Nous obtenons $(\sigma, v_1, v_2) \in \mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1),\tau_{\mathcal{C}}(\mathcal{R}_2)}(t)$

$(\sigma, v_1, \bot) \in \mathbf{D}^{-}_{\mathcal{R}_1,\mathcal{R}_2}(t)$ Nous avons $\sigma(t) \xrightarrow{*}_{\mathcal{R}_1} v_1$ et $\sigma(t) \notin \mathcal{T}(\mathcal{C}_2 \uplus \mathcal{D}_2, \mathcal{X})$.
De plus, $\sigma \in (\mathcal{V}(t) \to \mathcal{T}(\mathcal{C}_1, \mathcal{X}))$ et $v_1 \in \mathcal{T}(\mathcal{C}_1)$.
Donc $\sigma(t) \in \mathcal{T}(\mathcal{C}_1 \uplus (\mathcal{D}_1 \cap \mathcal{D}_2), \mathcal{X})$.
Ainsi, d'après la propriété (4), $\sigma(t) \xrightarrow{*}_{\tau_{\mathcal{C}}(\mathcal{R}_1)} v_1$.
Et, d'après la propriété (5), $\sigma(t) \xrightarrow{*}_{\tau_{\mathcal{C}}(\mathcal{R}_2)} \bot$.
Nous obtenons $(\sigma, v_1, \bot) \in \mathbf{D}_{\tau_{\mathcal{C}}(\mathcal{R}_1),\tau_{\mathcal{C}}(\mathcal{R}_2)}(t)$

$(\sigma, \bot, v_2) \in \mathbf{D}^{+}_{\mathcal{R}_1,\mathcal{R}_2}(t)$ Similaire au cas précédent. □

Nous proposons la transformation $\tau_{\mathcal{C}}(\mathcal{R})$ suivante qui, intuitivement, réécrit en $\bot$ tous les termes n'utilisant pas l'ensemble initial de constructeurs :

Définition 12 ($\tau_{\mathcal{C}}(\mathcal{R})$)**.** Soit $\mathcal{R} = (\mathcal{C}_0 \uplus \mathcal{D}, R)$ un CB-TRS inductivement séquentiel. Soit $\mathcal{C} \supseteq \mathcal{C}_0$ un ensemble de constructeurs. Alors, $\tau_{\mathcal{C}}(\mathcal{R}) = (\mathcal{C}^{\bot} \uplus \mathcal{D}', R')$, où $\mathcal{D}'$ et R' sont construits de la manière suivante :
$\forall s \in S$,
$\quad valued_s \in \mathcal{D}'$ avec $valued_s$ d'arité 1
$\quad (valued_s(\bot_s) \to \mathsf{false}) \in R'$
$\quad \forall c \in (\mathcal{C}_0)_s$ d'arité n, $(valued_s(c(x_1, \ldots, x_n)) \to \wedge_{i=1}^{n} valued_{sort(x_i)}(x_i)) \in R'$
$\quad \forall c \in (\mathcal{C} \setminus \mathcal{C}_0)_s$ d'arité n, $(valued_s(c(x_1, \ldots, x_n)) \to \mathsf{false}) \in R'$
$\forall o \in \mathcal{D}$ d'arité n,
$\quad o, o_0 \in \mathcal{D}'$ avec o et o_0 d'arité n
$\quad (o(x_1, \ldots, x_n) \to \mathit{if}(\wedge_{i=1}^{n} valued_{sort(x_i)}(x_i), o_0(x_1, \ldots, x_n), \bot_{sort(o)})) \in R'$
$\quad \forall (o(a_1, \ldots, a_n) \to r_i) \in R$, avec $a_i \in \mathcal{T}(\mathcal{C}_0, \mathcal{X})$ pour tout $i \in \{1, \ldots, n\}$,
$\quad\quad (o_0(a_1, \ldots, a_n) \to r_i) \in R'$

Conjecture. *Soient $\mathcal{R} = (\mathcal{C}_0 \uplus \mathcal{D}, R)$ un CB-TRS inductivement séquentiel et $\mathcal{C} \supseteq \mathcal{C}_0$ un ensemble de constructeurs. $\tau_{\mathcal{C}}(\mathcal{R})$ est un CB-TRS inductivement séquentiel et vérifie les propriétés (1-5).*

Sous réserve que la transformation $\tau_{\mathcal{C}}(\mathcal{R})$ n'introduise pas de nouvelles causes de non-terminaison indépendantes de $\mathcal{R}$, nous profitons des résultats relatifs à $\mathbf{D}_{\mathcal{R}_1,\mathcal{R}_2}$. Ceux-ci nous permettent d'affirmer que $\mathbf{D}^{\bot}_{\mathcal{R}_1,\mathcal{R}_2}(t)$ est lui-aussi récursivement énumérable.

5 Mise en œuvre

Nous avons mis en œuvre l'ensemble des techniques décrites dans ce mémoire sous la forme d'une librairie écrite dans le langage Scala et fonctionnant sur machine virtuelle Java (JVM).

Le moteur de sur-réduction construit tout d'abord les arbres définitionnels pour le système de réécriture donné en entrée. Si les arbres définitionnels ne sont pas constructibles ou bien ne prennent pas en compte toutes les règles alors une exception est émise pour signifier à l'utilisateur que le système n'est pas inductivement séquentiel.

Ensuite, nous utilisons la stratégie de sur-réduction nécessaire la plus extérieure via la technique de recherche en largeur d'abord. Un paramètre (optionnel) peut être donné à la procédure pour indiquer une profondeur maximum de recherche.

Deux implémentations de la procédure de recherche sont proposées : la première permet la collecte des résultats (substitution restreinte aux variables du terme de départ et forme normale obtenue) et la seconde permet la construction explicite de l'arbre de sur-réduction.

Nous avons implémenté le calcul de différentiels de sur-réduction à la fois avec l'algorithme initial de la figure 3 et avec les transformations de systèmes.

Nous avons défini un DSL (Domain Specific Language) pour l'expression des règles des systèmes de réécriture. Celui-ci utilise directement les variables du langage hôte en permettant de définir une règle comme une fonction anonyme dont les arguments sont les variables de la règle. La figure 4 montre la représentation en Scala du système de réécriture $\mathcal{RR}$.

```
val leq1 = Rule((x) ⇒ (0 ⩽ x) → true)
val leq2 = Rule((x) ⇒ (s(x) ⩽ 0) → false)
val leq3 = Rule((x, y) ⇒ (s(x) ⩽ s(y)) → (x ⩽ y))

val plus1 = Rule((x) ⇒ (0 + x) → x)
val plus2 = Rule((x, y) ⇒ (s(x) + y) → s(x + y))

val rules = Set(leq1, leq2, leq3, plus1, plus2)
```

FIGURE 4 – Représentation en Scala du système $\mathcal{RR}$

Finalement, nous avons réalisé au sein de la librairie un export LaTeX pour les arbres définitionnels ainsi que pour les arbres de sur-réduction (soit sous la forme de plusieurs dérivations de sur-réduction soit sous la forme d'un arbre). Le code LaTeX généré pour les arbres utilise le package LaTeX `forest`. L'ensemble des exemples de ce document a été produit par ce biais.

6 Discussion et perspectives

Nous avons montré comment la sur-réduction peut être utilisée pour réaliser des interrogations complexes d'une politique de contrôle d'accès spécifiée sous la forme d'un système de réécriture.

Nous avons présenté une technique utilisant la sur-réduction et la transformation de systèmes afin d'identifier des exemples pour lesquels deux politiques de contrôle d'accès, spécifiées sous la forme de systèmes de réécriture, se comportent différemment. À notre connaissance, ce problème et la solution que nous y apportons sont nouveaux.

Notre prochain travail concernant ces développements consistera à prouver l'adéquation de la transformation $\tau_{\mathcal{C}}$ proposée pour le calcul de différentiel sur deux ensembles de constructeurs. Néanmoins, nous pouvons d'ores et déjà identifier un certain nombre de pistes de recherche additionnelles :

Antoy et al [3] suggèrent que la restriction aux termes à racine opérationnelle concernant l'utilisation de la stratégie de sur-réduction nécessaire la plus extérieure peut être levée. De plus, ils proposent d'utiliser les techniques de Term Graph Rewriting [4] pour le partage de sous-termes et éviter la duplication de calculs en présence de règles non-linéaire à droite. Il nous semble que les techniques développées dans ce mémoire peuvent aisément être adaptées dans ces deux directions.

Escobar [10] propose une généralisation de la stratégie nécessaire la plus extérieure, nommée Natural Narrowing, qui s'applique même à des TRS simplement linéaires à gauche. Ils définissent une sous-classe de TRS sur laquelle cette stratégie est optimale. Il serait intéressant d'étudier si nos transformations peuvent être améliorées pour fonctionner sur cette classe de systèmes.

Références

[1] ANSI. RBAC, 2004. INCITS 359-2004., 2004.

[2] S. Antoy. Definitional Trees. In *In Proc. of the 3rd International Conference on Algebraic and Logic Programming*, pages 143–157. Springer LNCS, 1992.

[3] S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. In *Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 268–279. ACM, 1994. 00415.

[4] H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe*, number 259 in Lecture Notes in Computer Science, pages 141–158. Springer Berlin Heidelberg, 1987.

[5] S. Barker. The next 700 access control models or a unifying meta-model ? In *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 187–196, 2009.

[6] C. Bertolissi and M. Fernández. Category-Based Authorisation Models : Operational Semantics and Expressive Power. pages 140–156, 2010. 00002.

[7] C. Bertolissi, M. Fernández, and S. Barker. Dynamic Event-based Access Control as Term Rewriting. In *In Proc. DBSEC 2007, LNCS*. Springer, 2007.

[8] C. Bertolissi and W. Uttha. Automated Analysis of Rule-based Access Control Policies. In *Proceedings of the 7th Workshop on Programming Languages Meets Program Verification*, PLPV '13, pages 47–56, New York, NY, USA, 2013. ACM.

[9] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pages 243–320. 1990. 00010.

[10] S. Escobar. Refining Weakly Outermost-Needed Rewriting and Narrowing. In *Proc. of 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'03*, pages 113–123. ACM Press, 2003. 00021.

[11] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and Ramaswamy Chandramouli. *Proposed NIST Standard for Role-Based Access Control*. 2001.

[12] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, I and II. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic : Essays in Honor of Alan Robinson*, pages 395–443 and 415–443. The MIT Press, Cambridge, MA, 1992.

[13] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaurn, editors, *Handbook of Logic in Computer Science*, pages 1–116. Oxford University Press, 1992.

[14] J. W. Klop and A. Middeldorp. Sequentiality in Orthogonal Term Rewriting Systems. *Journal of Symbolic Computation*, 12 :161–195, 1991.

[15] A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5(3-4) :213–253, 1994. 00138.

[16] R. Sandhu and Q. Munawer. How to do Discretionary Access Control Using Roles. In *Proceedings of the third ACM workshop on Role-based access control*, pages 47–54. ACM, 1998.

Annexe

$\underline{uar(\mathsf{Alice}, \mathsf{Edit}, \mathsf{AccountingDB})}$

$\to_{uar}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in car^*(cc^*(\underline{uc(\mathsf{Alice})})), \mathsf{grant}, \mathsf{deny})$

$\to_{uc_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in car^*(\underline{cc^*(\mathsf{Administrative}::\mathsf{nil})}), \mathsf{grant}, \mathsf{deny})$

$\to_{cc^*_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{car^*(\mathsf{Administrative}::cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{car^*_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{car(\mathsf{Administrative})}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{car_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{(\mathsf{Edit}, \mathsf{PasswdFile})::(\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{:::_1}$ $if(\underline{(\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{Edit}, \mathsf{PasswdFile})::(\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{\in_1}$ $if(\underline{(\mathsf{Edit}, \mathsf{AccountingDB}) \approx (\mathsf{Edit}, \mathsf{PasswdFile})} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{Edit} \approx \mathsf{Edit}} \wedge \mathsf{AccountingDB} \approx \mathsf{PasswdFile} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{true} \wedge \mathsf{AccountingDB} \approx \mathsf{PasswdFile}} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\wedge_1}$ $if(\underline{\mathsf{AccountingDB} \approx \mathsf{PasswdFile}} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{false} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{\vee_2}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{(\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{:::_1}$ $if(\underline{(\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{PasswdFile})::\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{\in_1}$ $if(\underline{(\mathsf{Edit}, \mathsf{AccountingDB}) \approx (\mathsf{View}, \mathsf{PasswdFile})} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in \mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{Edit} \approx \mathsf{View}} \wedge \mathsf{AccountingDB} \approx \mathsf{PasswdFile} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in \mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{false} \wedge \mathsf{AccountingDB} \approx \mathsf{PasswdFile}} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in \mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\wedge_2}$ $if(\underline{\mathsf{false} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in \mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{\vee_2}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{\mathsf{nil}:::car^*(cc(\mathsf{Administrative}):::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{:::_2}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in car^*(\underline{cc(\mathsf{Administrative})}:::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{cc_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in car^*(\underline{\mathsf{Accounting}::\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil})}), \mathsf{grant}, \mathsf{deny})$

$\to_{:::_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{car^*(\mathsf{Accounting}::\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{car^*_1}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{car(\mathsf{Accounting})}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{car_2}$ $if((\mathsf{Edit}, \mathsf{AccountingDB}) \in \underline{(\mathsf{Edit}, \mathsf{AccountingDB})::(\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{:::_1}$ $if(\underline{(\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{Edit}, \mathsf{AccountingDB})::(\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{\in_1}$ $if(\underline{(\mathsf{Edit}, \mathsf{AccountingDB}) \approx (\mathsf{Edit}, \mathsf{AccountingDB})} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{Edit} \approx \mathsf{Edit}} \wedge \mathsf{AccountingDB} \approx \mathsf{AccountingDB} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{true} \wedge \mathsf{AccountingDB} \approx \mathsf{AccountingDB}} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\wedge_1}$ $if(\underline{\mathsf{AccountingDB} \approx \mathsf{AccountingDB}} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil})), \mathsf{grant}, \mathsf{deny})$

$\to_{\approx}$ $if(\underline{\mathsf{true} \vee (\mathsf{Edit}, \mathsf{AccountingDB}) \in (\mathsf{View}, \mathsf{SalesDB})::\mathsf{nil}:::car^*(\mathsf{Sales}::\mathsf{nil}:::cc^*(\mathsf{nil}))}, \mathsf{grant}, \mathsf{deny})$

$\to_{\vee_1}$ $\underline{if(\mathsf{true}, \mathsf{grant}, \mathsf{deny})}$

$\to_{if_1}$ grant

FIGURE 5 – Une réduction pour $uar(\mathsf{Alice}, \mathsf{Edit}, \mathsf{AccountingDB})$

FIGURE 6 – Un arbre de sur-réduction pour le but $uar(\mathsf{Alice}, \mathsf{Edit}, r) \approx \mathsf{grant}$

Preuve du Lemme 10.

1. $\Rightarrow$ Considérons d'abord une conjonction simple.
Si $t \wedge t' \xrightarrow{*} \mathsf{true}$, alors il existe une application de la règle $\mathsf{true} \wedge x \to x$, et
il existe un terme u tel que $t \wedge t' \xrightarrow{*} \mathsf{true} \wedge u \to u \xrightarrow{*} \mathsf{true}$
Donc, $t \xrightarrow{*} \mathsf{true}$ et $t' \xrightarrow{*} u \xrightarrow{*} \mathsf{true}$. $(*)$

Considérons maintenant une conjonction double.
Si $t \wedge (t' \wedge t'') \xrightarrow{*} \mathsf{true}$, alors il existe une application de la règle $\mathsf{true} \wedge x \to x$, et
il existe un terme u tel que $t \wedge (t' \wedge t'') \xrightarrow{*} \mathsf{true} \wedge u \to u \xrightarrow{*} \mathsf{true}$
Donc, $t \xrightarrow{*} \mathsf{true}$ et $t' \wedge t'' \xrightarrow{*} u \xrightarrow{*} \mathsf{true}$, et donc par $(*)$, $t' \xrightarrow{*} \mathsf{true}$ et $t'' \xrightarrow{*} \mathsf{true}$.

1. $\Leftarrow$ $t \xrightarrow{*} \mathsf{true}$, $t' \xrightarrow{*} \mathsf{true}$ et $t'' \xrightarrow{*} \mathsf{true}$
Donc, $t \wedge t' \wedge t'' \xrightarrow{*} \mathsf{true} \wedge \mathsf{true} \wedge \mathsf{true} \to \mathsf{true} \wedge \mathsf{true} \to \mathsf{true}$.

2. $\Leftrightarrow$ De manière similaire à 1.

3. $\Rightarrow$ Considérons une dérivation de réécriture $t \approx t' \xrightarrow{*}_{\mathcal{R}'} \mathsf{false}$. Nous procédons par induction forte sur le nombre k d'applications de règles (définissant) $\approx$. Pour $k = 1$, il y a exactement une application de règle $\approx$:

$$t \approx t' \xrightarrow{*} s \approx s' \to r \xrightarrow{*} \mathsf{false}$$

r ne peut pas avoir le symbole $\wedge$ à la racine, sinon il devrait y avoir plus d'applications de règles $\approx$ dans la dérivation $r \xrightarrow{*} \mathsf{false}$. Donc la règle appliquée est $c_1(\ldots) \approx c_2(\ldots) \to \mathsf{false}$. On peut obtenir les dérivations dans $\mathcal{R}$:

$$t \xrightarrow{*} c_1(\ldots) \xrightarrow{*} u \qquad\qquad t' \xrightarrow{*} c_2(\ldots) \xrightarrow{*} u'$$

Comme $\mathcal{R}$ est à base de constructeurs, u et u' sont deux formes normales ayant pour racine c_1 et c_2 respectivement. Et nous obtenons le résultat souhaité.

Pour le cas d'induction ($k > 1$), il y a une première application de règle $\approx$:

$$t \approx t' \xrightarrow{*} s \approx s' \to r \xrightarrow{*} \mathsf{false}$$

r ne peut être false, sinon il ne pourrait y avoir qu'une application de règle $\approx$ dans $r \xrightarrow{*} \mathsf{false}$. Donc, $s = c(t_1, \ldots, t_n)$, $s' = c(t'_1, \ldots, t'_n)$ et $r = (t_1 \approx t'_1) \wedge \ldots \wedge (t_n \approx t'_n)$.

Comme $r \xrightarrow{*} \mathsf{false}$, il doit exister dans cette dérivation une application d'une règle $\wedge$ à la racine, $r \xrightarrow{*} \mathsf{false} \wedge r' \xrightarrow{*} \mathsf{false}$. Et donc, il existe un i tel que $(t_i \approx t'_i) \xrightarrow{*} \mathsf{false}$, avec au plus $k - 1$ applications de règles $\approx$. Par l'hypothèse d'induction, t_i et t'_i se réduisent dans $\mathcal{R}$ vers deux formes normales différents u_i et u'_i.

On peut obtenir alors les dérivations dans $\mathcal{R}$:

$$t \xrightarrow{*} c(t_1, \ldots, t_i, \ldots, t_n) \xrightarrow{*} c(s_1, \ldots, u_i, \ldots, s_n) \xrightarrow{*} u$$
$$t' \xrightarrow{*} c(t'_1, \ldots, t'_i, \ldots, t'_n) \xrightarrow{*} c(s'_1, \ldots, u'_i, \ldots, s'_n) \xrightarrow{*} u' \qquad \text{où } u_i \neq u'_i$$

Comme $\mathcal{R}$ est à base de constructeurs, $c(s_1, \ldots, u_i, \ldots, s_n)$ et $c(s'_1, \ldots, u'_i, \ldots, s'_n)$ ne peuvent être réécrits à la racine. u et u' sont donc deux formes normales différentes. Et nous obtenons le résultat souhaité.

3. $\Leftarrow$ t et t' se réduisent dans $\mathcal{R}$ en deux formes normales différentes u et u'. Donc, dans $\mathcal{R}'$, $t \approx t' \xrightarrow{*} u \approx u'$. Il nous suffit donc de montrer que $u \approx u' \xrightarrow{*} \mathsf{false}$. Nous procédons par induction forte sur la hauteur k cumulée de u et u'. Pour $k = 2$, puisque $u \neq u'$, $u = c_1$ et $u' = c_2$, et $u \approx u'$ se réduit directement par la règle $c_1(\ldots) \approx c_2(\ldots) \to \mathsf{false}$.

Pour $k > 2$, nous observons deux cas. Si $u = c_1(\ldots)$ et $u' = c_2(\ldots)$ ont pour racine deux constructeurs différents, $u \approx u'$ se réduit directement par la règle $c_1(\ldots) \approx c_2(\ldots) \to \mathsf{false}$. Si $u = c(t_1, \ldots, t_n)$ et $u' = c(t'_1, \ldots, t'_n)$ ont pour racine le même constructeur :

$$(u \approx u') \to (t_1 \approx t'_1) \wedge \ldots \wedge (t_n \approx t'_n)$$

u et u' étant différents, soit i le premier indice pour lequel t_i et t'_i sont différents. Leur hauteur cumulée est strictement inférieure à k et, par l'hypothèse d'induction, $t_i \approx t'_i \xrightarrow{*} \mathsf{false}$. D'après la proposition 9, pour tout $j < i$, $t_j \approx t'_j \xrightarrow{*} \mathsf{true}$. On peut montrer par induction sur j que $(t_1 \approx t'_1) \wedge \ldots \wedge (t_{i-1} \approx t'_{i-1}) \xrightarrow{*} \mathsf{true}$ et donc

$$\begin{aligned} \mathsf{true} \wedge (t_i \approx t'_i) \wedge \ldots \wedge (t_n \approx t'_n) &\to (t_i \approx t'_i) \wedge \ldots \wedge (t_n \approx t'_n) \\ &\to \mathsf{false} \wedge (t_{i+1} \approx t'_{i+1}) \wedge \ldots \wedge (t_n \approx t'_n) \\ &\to \mathsf{false} \end{aligned}$$

□