# Ensemble de termes accessibles en réécriture : aux frontières de la régularité

Yann Salmon

**Rapport de stage**

# Ensemble de termes accessibles en réécriture : aux frontières de la régularité

Yann Salmon
*Département de Mathématiques*
*Antenne de Bretagne de*
*l'École normale supérieure de Cachan*

sous la direction de Thomas Genet
*Équipe Celtique*
*IRISA — INRIA Rennes Bretagne Atlantique*

2 juin 2011

**Résumé.** Avant d'exploiter ou de diffuser un logiciel, il convient de s'assurer de sa correction. Lorsque la taille du programme est grande, l'automatisation de la preuve est à rechercher, mais elle ne peut être une panacée (théorème de Rice). Ce rapport s'inscrit dans une étude utilisant les systèmes de réécriture (TRS) comme modèles formels des programmes : la correction s'exprime par le fait que l'ensemble des termes accessibles par le TRS considéré ($R^*(L)$) est disjoint d'un ensemble de termes représentant les mauvais comportements ($L_m$). On sait décider une telle propriété si les langages en jeu sont réguliers (au sens des automates d'arbres). Certaines classes de TRS préservant la régularité sont connues, mais ne couvrent pas tous les cas. Ici, nous nous intéressons à la complétion d'automate : ce procédé vise une sur-approximation régulière de $R^*(L)$, disjointe de $L_m$, en ajoutant des transitions à l'automate jusqu'à obtention d'un point fixe. La complétion équationnelle permet de paramétrer la sur-approximation par un ensemble $E$ d'équations entre termes afin de favoriser l'obtention d'un point fixe. Nous exposons cette méthode et tentons de l'exprimer dans le vocabulaire de l'interprétation abstraite, une autre méthode d'élaboration d'analyses statiques.

# Table des matières

# 1 Introduction

Il est important de s'assurer de la correction des logiciels que l'on écrit, en particulier lorsqu'on envisage de les diffuser. Pour des programmes courts, il est possible d'effectuer une preuve de correction à la main, mais dès que le programme à étudier est long, établir une telle preuve devient très fastidieux.

Les méthodes de vérification ont pour objet d'automatiser l'assurance de correction des programmes. Il en existe de fort nombreuses, qu'on peut répartir en deux grandes classes. Les méthodes de vérification dynamique sont actives au moment de l'exécution ; il peut s'agir par exemple d'instructions de test qui auront été rajoutées au programme initial. Ces méthodes ne sont pas étudiées ici.

La présente étude s'inscrit dans le cadre de la vérification statique. Le programme est analysé sans qu'il soit exécuté. On souhaite assurer la sureté du programme, c'est-à-dire vérifier qu'aucune de ses exécutions possibles n'atteint un état incorrect. Bien entendu, le théorème de Rice interdit d'espérer obtenir une preuve de correction dans tous les cas : on admet qu'une analyse statique réponde que l'absence d'erreur n'est pas garantie sans qu'elle assure la présence d'une erreur.

Une analyse statique utilise un modèle formel du programme étudié. Ce modèle doit être correct, suffisamment grossier pour que l'analyse termine à coup sûr mais suffisamment précis pour que le résultat de l'analyse soit intéressant.

Les modèles formels utilisés par la complétion équationnelle, qui est l'objet principal de notre étude, sont basés sur les termes et la réécriture. Les termes représentent les états de la machine. L'exécution d'une instruction correspond alors à la réécriture du terme représentant l'état courant en le terme représentant l'état suivant. L'exemple suivant vise à représenter la fonction factorielle comme un système de réécriture.

*Exemple 1.*

Le programme Caml suivant calcule la factorielle d'un entier naturel.

```
1 lec rec factorielle = function
2   | 0 -> 1
3   | n -> n * factorielle (n-1) ;;
```

Ici, on peut obtenir une représentation très simple du programme sous la forme d'un système de réécriture, en utilisant la représentation unaire des entiers (disposant de la multiplication `mult`) : on se donne les deux règles $\texttt{factorielle}(0) \to \texttt{S}(0)$ et $\texttt{factorielle}(\texttt{S}(x)) \to \texttt{mult}(\texttt{S}(x), \texttt{factorielle}(x))$. 1 ◁

Il existe d'autres cadres théoriques permettant de concevoir des analyses statiques. L'un d'eux est l'interprétation abstraite. Il consiste à exécuter le programme dans une sémantique suffisamment grossière pour que le problème de l'arrêt soit contourné.

Au cours de ce stage effectué sous la direction de Thomas Genet au sein de l'équipe Celtique commune à l'IRISA et à l'INRIA Rennes, nous avons tenté d'acquérir une compréhension de la complétion équationnelle et de l'exprimer dans le vocabulaire de l'interprétation abstraite. Ce stage fait suite à notre étude bibliographique [Sal11], [1] et certaines parties du présent rapport (la présente introduction, les sections 2 et 3 ainsi que quelques phrases) en sont issues, après augmentations, remaniements, adaptations ou corrections. [2]

Afin que notre tentative de traduction de la complétion équationnelle en le vocabulaire de l'interprétation abstraite soit lisible par les connaisseurs de chacun des deux domaines, il fallait prendre le temps de définir chacun d'entre eux.

1. Si vous consultez le fichier PDF original de ce rapport, les citations sont « clicables » ainsi que les références internes.
2. Nous remercions les relecteurs de [Sal11] pour leurs remarques.

C'est pourquoi nous commençons par définir un cadre théorique comprenant les langages d'arbres, les automates d'arbres et les systèmes de réécriture (section 2). Cette section est classique pour celui qui connait ce domaine. Toutefois, des notations non standard mais adaptées à notre étude sont présentées en section 2.2.2. On pourra également lire la remarque 4 et la définition d'un « remplacement » (11).

Ce cadre nous permet d'énoncer le problème de l'atteignabilité et de mentionner rapidement celui de la préservation de la régularité (section 3).

La section 4 présente la complétion équationnelle d'une manière que nous souhaitons accessible. Nous comptons cette présentation d'une notion ardue au nombre de nos contributions.

La section 5 est une présentation résumée mais classique de l'interprétation abstraite. Le connaisseur de ces notions peut ignorer cette section.

Nos principales contributions sont présentées en section 6 : une tentative de traduction de la complétion équationnelle en interprétation abstraite.

### 1.1 Notations générales

Nous notons $\mathbb{Z}$ l'ensemble des entiers (relatifs), $\mathbb{N}$ l'ensemble des entiers naturels et $\mathbb{N}_*$ l'ensemble des entiers naturels non nuls. Pour $i, j \in \mathbb{N}$, nous notons $[\![i\,;j]\!]$ l'ensemble des entiers naturels $x$ qui vérifient $i \leqslant x \leqslant j$.

Pour $E$ un ensemble, nous notons $\mathcal{P}(E)$ l'ensemble des parties de $E$ et $\mathrm{Id}_E$ l'application identité

$$\mathrm{Id}_E : \begin{array}{ccc} E & \longrightarrow & E \\ x & \longmapsto & x \end{array}.$$

Nous notons $\{x \in E \mid \mathscr{P}(x)\}$ l'ensemble des éléments $x$ de $E$ qui vérifient la propriété $\mathscr{P}(x)$.

## 2 Réécriture

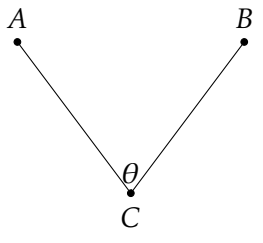
### 2.1 Termes et arbres

Cette section s'appuie sur [Com+08] et [BN98].

La première étude des termes et leur représentation sous forme d'arbre semble être due à Axel Thue dans [Thu10]. Il énonce qu'en certaines circonstances, on peut, à partir d'un concept $A$ et d'un concept $B$ et en utilisant une opération (binaire) $\theta$, construire de manière univoque un nouveau concept $C$, qu'il propose de noter

$$[(A)\theta(B)]$$

ou encore




.

Il s'agit là de la notation infixe d'un opérateur binaire et de sa représentation sous forme d'arbre. Comme nous allons considérer aussi des opérateurs unaires, ternaires, etc., nous utiliserons la notation préfixe.

### 2.1.1 Généralités

Pour définir un langage de termes, on a besoin de symboles de fonctions (chacun ayant une arité, ou « nombre d'arguments »), de symboles de constantes et de symboles de variables. Les termes les plus simples sont les constantes et les variables. On construit tous les autres termes en appliquant formellement un symbole de fonction d'arité $k$ à $k$ termes déjà construits.

**Définition 2 (Signature).**
On appelle signature la donnée, pour chaque $k \in \mathbb{N}$, d'un ensemble fini $\Sigma_k$ dont les éléments sont appelés « symboles de fonctions d'arité $k$ ». Ces ensembles sont deux à deux disjoints. Les symboles de fonctions d'arité 0 sont appelés « symboles de constantes ». Une telle signature est notée $\Sigma$.

On peut adjoindre à une telle signature un ensemble $\mathcal{X}$ dénombrable et disjoint des précédents dont les éléments sont appelés « symboles de variables ». On note une telle signature $(\Sigma, \mathcal{X})$. 2◀

*Remarque 3.* On dit qu'une signature est finie si le nombre total de symboles de fonctions est fini, c'est-à-dire si les ensembles $\Sigma_k$ sont vides à partir d'un certain rang. Dans ce document, nous ne considérons que des signatures finies.

*Remarque 4.* Dans les exemples, nous convenons d'indiquer l'arité de chaque symbole de fonction en indice à gauche : ${}_2f$, ${}_0c$, ... Les variables sont alors distinguées des constantes par le fait qu'elles ne portent pas d'indice. L'exemple 7 ci-dessous sera l'occasion d'inaugurer cette convention de notation.

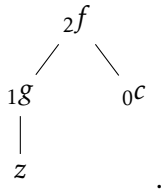
**Définition 5 (Langage de termes).**
Soit $(\Sigma, \mathcal{X})$ une signature. Le langage des termes sur cette signature, noté $T(\Sigma, \mathcal{X})$, est défini par induction. À la base, $\Sigma_0 \subseteq T(\Sigma, \mathcal{X})$ et $\mathcal{X} \subseteq T(\Sigma, \mathcal{X})$ ; et inductivement, pour tout $k \in \mathbb{N}_*$, tout $f \in \Sigma_k$ et tous $t_1, \ldots, t_k \in T(\Sigma, \mathcal{X})$, on a $f(t_1, \ldots, t_k) \in T(\Sigma, \mathcal{X})$. 5◀

**Définition 6 (Langage de termes clos).**
Lorsqu'on utilise le procédé inductif décrit ci-dessus à une signature $\Sigma$ dépourvue d'ensemble de variables, on obtient un langage de termes dits « clos » : $T(\Sigma)$.

Étant donné une signature avec ensemble de variables $(\Sigma, \mathcal{X})$, on dit d'un terme $t \in T(\Sigma, \mathcal{X})$ qu'il est clos s'il ne contient pas de variable, c'est-à-dire si $t \in T(\Sigma)$. 6◀

Chaque terme peut être considéré comme un arbre écrit dans la notation préfixe, de sorte que les langages de termes peuvent être vus comme des langages d'arbres. Cette analogie permet de définir les positions dans un terme (chaque position est un mot sur $\mathbb{N}_*$) et les sous-termes.

*Exemple 7.*
Le terme $t = {}_2f({}_1g(z), {}_0c)$, écrit sur une signature appropriée où ${}_0c$ est une constante et $z$ est une variable, peut être représenté par l'arbre




.

${}_2f$ est à la position $\lambda$ (le mot vide), ${}_1g$ est à la position 1, $z$ à la position 1.1, ${}_0c$ à la position 2. Le sous-terme à la position 1 est le sous-arbre dont la racine est le nœud situé à cette position : il s'agit de ${}_1g(z)$. 7◁

*Remarque 8.* On peut très facilement composer plusieurs fois à suivre un même symbole de fonction unaire. Nous le ferons souvent dans les discussions ultérieures. Afin d'alléger les notations, nous convenons d'écrire, pour ${}_1f$ un symbole de fonction unaire, $t$ un terme et $n$ un entier naturel, ${}_1f^n(t)$ pour $\underbrace{{}_1f({}_1f(\ldots({}_1f}_{n \text{ fois}}(t)\underbrace{))\ldots)}_{n \text{ parenthèses}}$.

**Définition 9 (Substitution).**
Une substitution sur $T(\Sigma,\mathcal{X})$ est une application $\sigma$ de $\mathcal{X}$ dans $T(\Sigma,\mathcal{X})$ tel que l'ensemble $\{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ est fini.

Une substitution peut être appliquée à un terme $t \in T(\Sigma,\mathcal{X})$ pour produire un autre terme, noté $t\sigma$, grâce au procédé d'induction suivant. Pour $x \in \mathcal{X}$, $x\sigma = \sigma(x)$, pour $c \in \Sigma 0$, $c\sigma = c$ ; pour $k \in \mathbb{N}$, $f \in \Sigma_k$ et $t_1,\ldots,t_k \in T(\Sigma,\mathcal{X})$, notant $t = f(t_1,\ldots,t_k)$, on a

$$t\sigma = f(t_1\sigma,\ldots,t_k\sigma).$$ 9◀

*Exemple 10.*
Reprenons l'exemple précédent. Soit $s$ un terme. Le terme $t[z \mapsto s]$ est ${}_2f({}_1g(s),{}_0c)$. Le terme $t[z \mapsto t]$ est ${}_2f({}_1g({}_2f({}_1g(z),{}_0c)),{}_0c)$. 10◁

Nous définissons également une notion analogue, mais plus générale que la substitution. Elle sera utile en complétion équationnelle pour faire un lien entre des règles de réécriture et des automates.

**Définition 11 (Remplacement).**
Soit $(\Sigma,\mathcal{X})$ une signature et $Q$ un ensemble. On appelle remplacement de $\mathcal{X}$ vers $Q$ toute application de $\mathcal{X}$ dans $Q$. Étant donné un remplacement $\sigma : \mathcal{X} \to Q$, on définit son application aux termes comme pour les substitutions : $x\sigma = \sigma(x)$ pour $x \in \mathcal{X}$ et pour $t = f(t_1,\ldots,t_k)$, $t\sigma = f(t_1\sigma,\ldots,t_k\sigma)$. 11◀

*Remarque 12.* En pratique, on ne définit pas un remplacement $\sigma$ sur $\mathcal{X}$ tout entier, mais seulement sur les variables qui apparaissent dans les termes auxquels on envisage d'appliquer $\sigma$.

### 2.1.2 Propriétés des termes, Contextes

**Définition 13 (Terme linéaire).**
On dit qu'un terme $t$ est linéaire si chaque variable apparait au plus une fois dans $t$. 13◀

*Exemple 14.*
Les termes $x$, ${}_0c$, ${}_1f(x)$, ${}_1f({}_1f(x))$, ${}_2g(x,y)$, ${}_2g({}_0c,{}_0c)$ sont linéaires. Les termes ${}_2g(x,x)$, ${}_3h({}_1f(x),{}_0c,x)$ ne sont pas linéaires. 14◁

*Remarque 15.* Tout terme clos est linéaire.

La notion de contexte est importante car elle va nous servir à définir la réécriture et les automates d'arbres.

**Définition 16 (Contexte monovariable).**
On appelle contexte monovariable (ou contexte) sur la signature $\Sigma$ tout terme linéaire de $T(\Sigma,\{\diamond\})$, où $\diamond \notin \Sigma$. Pour $C$ un contexte sur $\Sigma$ et $t \in T(\Sigma,\mathcal{X})$, on note $C[t]$ le terme $C[\diamond \mapsto t]$, où l'on a appliqué la substitution qui envoie $\diamond$ sur $t$. 16◀

## 2.2 Automates d'arbres

Nous verrons en section 3 que le problème de vérification que nous étudions se traduit en un problème d'intersection de langages de termes. Ce problème n'est pas décidable en général, mais il l'est si l'on se restreint à des langages d'arbres réguliers.

Cette section s'appuie largement sur [Com+08], mais nous introduisons des notations adaptées à notre étude. Nous ne considérons que des termes clos sur une signature $\Sigma$ donnée, que nous identifions aux arbres qui les représentent.

### 2.2.1 Présentation informelle

Les automates d'arbres sont l'analogue des automates de mots, souvent appelé « automates » sans plus de précision. Ainsi, les langages réguliers dont il est question dans ce rapport sont les langages d'arbres reconnus par des automates d'arbres.

Nous supposons que le lecteur est familier de la notion d'automate de mots, et nous présentons les automates d'arbres par analogie. Commençons par dire qu'on n'aura pas, comme c'est le cas pour les automates de mots, de représentation agréable de l'automate sous la forme d'un graphe. Mais voyons une façon d'appréhender le fonctionnement d'un automate de mots :

*Exemple 17.*

Soit $\mathcal{A}$ un automate de mots : on en connait l'ensemble des états et les transitions. Soit un mot $u = u_1 u_2 \dots u_n$ et notons $q_i$ l'état initial d'où commence une exécution de $\mathcal{A}$ sur $u$. Nous écrivons cette situation

$$q_i u_1 u_2 \dots u_n.$$

Supposons que l'automate possède une transition $q_i \xrightarrow{u_1} r$. Sur la représentation ci-dessus, cette transition peut se noter $q_i u_1 \to r$, de sorte qu'en la suivant, on obtient :

$$r u_2 \dots u_n.$$ 17 ◁

Généralisons ce comportement aux arbres. Contrairement à un mot, un arbre n'a pas un début et une fin, mais une racine et des feuilles (étiquetées par des symboles de constantes). Les automates que nous considérerons ici « commencent » à agir sur les feuilles. Contrairement au cas des mots, où n'importe quelle lettre peut être le début d'un mot à traiter, les premières étapes de l'exécution d'un automate d'arbre ont nécessairement lieu sur des symboles de constantes. On n'a donc pas besoin de notion d'état initial : il suffit d'avoir des transitions initiales qui s'appliquent aux symboles de constantes.

*Exemple 18.*

Construisons un automate d'arbre qui reconnait le langage

$$L = \{{}_2f({}_1g^m({}_0c), {}_1g^n({}_0d)) \mid m \in \mathbb{N}, n \in \mathbb{N}_*\}.$$

Sont à construire un ensemble d'états $Q$, une partie $Q_f$ de $Q$ pour les états terminaux et une relation de transition $\delta$.

Il nous faut un état pour reconnaitre chacune des deux constantes (il ne peut s'agir du même, sinon on ne pourrait pas distinguer ${}_2f({}_1g({}_0c), {}_1g({}_0c))$ et ${}_2f({}_1g({}_0c), {}_1g({}_0d))$). Soit donc les états et les transitions ${}_0c \to q_{{}_0c}$, ${}_0d \to q_{{}_0d}$. Une fois qu'on a lu un ${}_0c$, composer formellement par ${}_1g$ ne change rien vis-à-vis de $L$ : prenons donc ${}_1g(q_{{}_0c}) \to q_{{}_0c}$. En revanche, les mots que nous voulons accepter doivent avoir au moins un ${}_1g(\dots)$ dans le deuxième argument de ${}_2f$ : il faut distinguer la présence d'un ${}_1g$, soit donc l'état $q_{{}_1g({}_0d)}$ et les transitions ${}_1g(q_{{}_0d}) \to q_{{}_1g({}_0d)}$ et ${}_1g(q_{{}_1g({}_0d)}) \to q_{{}_1g({}_0d)}$. Enfin, soit ${}_2f(q_{{}_0c}, q_{{}_1g({}_0d)}) \to q_{{}_2f}$. On prend donc $Q = \{q_{{}_0c}, q_{{}_0d}, q_{{}_1g({}_0d)}, q_{{}_2f}\}$, $Q_f = \{q_{{}_2f}\}$ et $\delta$ comme indiqué ci-dessus. Les transitions initiales sont ${}_0c \to q_{{}_0c}$ et ${}_0d \to q_{{}_0d}$.

Une exécution de cet automate sur l'arbre ${}_2f({}_1g({}_0c), {}_1g({}_0d))$ est représentée en figure 1. 18 ◁

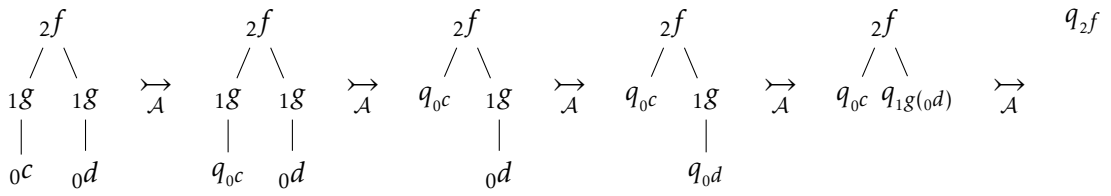



FIGURE 1 : Une exécution de l'automate de l'exemple 18

*Remarque 19.* Les transitions qui apparaissent dans l'exemple précédent respectent un certain schéma : à gauche, un symbole de fonction prenant en « arguments » des états ; à droite, un état. Les transitions d'un automate d'arbres doivent être de cette forme. On peut voir l'exécution d'un automate comme une opération de typage : le type des constantes est d'abord établi, puis remonte en utilisant les transitions indiquant quel type est renvoyé par une fonction selon les types des arguments auxquels on l'applique.

Du reste, au travers de cette interprétation, la notion d'automate d'arbres apparait implicitement dans [Thu10] : les « catégories de concepts » se propagent suivant des règles de cette forme.

### 2.2.2 Définitions et intérêt

Formalisons la notion d'automate d'arbre, en commençant par une définition que nous n'utiliserons pas, mais qui sera familière au lecteur, car elle est la formalisation de la sous-section précédente.

**Définition 20 (Automate d'arbre avec états terminaux).**
Un automate d'arbre (avec états terminaux) sur la signature $\Sigma$ est la donnée d'un ensemble fini d'états $Q$, d'une partie $Q_f$ de $Q$ dont les éléments sont appelés états terminaux et d'un ensemble $\delta$ de transitions de la forme, pour $k \in \mathbb{N}$, $f \in \Sigma_k$, $q_1, \ldots, q_k, q \in Q$,

$$f(q_1, \ldots, q_k) \to q.$$

Lorsque $k = 0$, on parle de transition initiale. 20 ◀

Cependant, dans notre étude, nous serons amené à nous intéresser aux termes reconnus par chacun des états d'un automate, si bien qu'il n'apparait pas pertinent de faire jouer un rôle particulier d'états terminaux à une partie $Q_f$ de l'ensemble des états. Cela nous amène à définir une notion d'automate dépourvu d'état terminal : c'est le plus souvent ce que nous entendrons par la suite par « automate ». Nous autorisons également la mise en place, comme dans le cas des mots, d'epsilon-transitions.

**Définition 21 (Automate d'arbre).**
Un automate d'arbre sur la signature $\Sigma$ est la donnée d'un ensemble fini d'états $Q$, d'un ensemble $\delta$ de transitions de la forme, pour $k \in \mathbb{N}$, $f \in \Sigma_k$, $q_1, \ldots, q_k, q \in Q$,

$$f(q_1, \ldots, q_k) \to q$$

et d'un ensemble $\delta_\varepsilon$ d'epsilon-transitions de la forme $q \to q'$ pour $q, q' \in Q$.

On note un tel automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$. On note $\Delta = \delta \cup \delta_\varepsilon$. 21 ◀

**Définition 22 (Automate déterministe).**
On dit d'un automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ qu'il est déterministe si $\delta_\varepsilon = \varnothing$ et $\delta$ est une relation fonctionnelle, c'est-à-dire qu'il n'existe pas de $c \to q_1 \in \delta$ et $c \to q_2 \in \delta$ avec $q_1 \neq q_2$. 22 ◀

*Remarque 23.* Nous serons amenés à relaxer légèrement la condition de finitude de l'ensemble des états $Q$ : nous autoriserons que $Q$ soit infini, à condition que la relation de transition $\Delta$ soit finie. Ainsi, l'ensemble des états utiles au fonctionnement de l'automate reste fini : tous les résultats de la présente section resteront donc valables.

Nous allons à présent formaliser l'exécution d'un automate sur un arbre. Nous devons d'abord désigner les « arbres en cours de traitement » qui apparaissent dans les étapes intermédiaires de l'exécution, comme les cinq derniers arbres de l'exemple 18 : ce sont des configurations.

**Définition 24 (Configuration).**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate. On enrichit $\Sigma$ en traitant les états comme des nouveaux symboles de constantes s'ajoutant à ceux de $\Sigma_0$. On note cette nouvelle signature $\Sigma \cup Q$.

On appelle configuration de l'automate $\mathcal{A}$ tout terme $c \in T(\Sigma \cup Q)$. 24 ◀

Une étape d'exécution emprunte une transition de $\Delta$ pour passer d'une configuration à une autre. La transition est le plus souvent appliquée à un sous-arbre, comme on le voit dans les quatre premières étapes de l'exemple 18. De manière duale, cela signifie que les transitions peuvent être utilisées dans un contexte.

**Définition 25 (Exécution d'un automate sur un arbre).**
Soit $t, t'$ deux configurations. On dit que l'automate $\mathcal{A}$ peut passer de $t$ à $t'$ s'il existe un contexte $C$ sur $\Sigma \cup Q$ et une transition $c \to q \in \Delta$ tels que

$$t = C[c] \qquad \text{et} \qquad t' = C[q].$$

Cette relation entre $t$ et $t'$ est notée $t \underset{\mathcal{A}}{\rightarrowtail} t'$. On note $\overset{+}{\underset{\mathcal{A}}{\rightarrowtail}}$ la clôture transitive de $\underset{\mathcal{A}}{\rightarrowtail}$ et $\overset{*}{\underset{\mathcal{A}}{\rightarrowtail}}$ sa clôture réflexive et transitive. 25 ◀

*Exemple 26.*
Pour passer du quatrième au cinquième arbre de l'exemple 18, on a utilisé la transition ${}_1g(q_{0d}) \to q_{{}_1g({}_0d)}$ avec le contexte ${}_2f(q_{0c}, \Diamond)$. 26 ◁

Nous pouvons à présent définir la reconnaissance par un automate. Plus précisément, par des états d'un automate, puisque que nous avons renoncé à la notion d'état terminal.

**Définition 27 (Langage reconnu par un automate, $\mathscr{L}(\cdot,\cdot)$).**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate et $X$ une partie de $Q$. Le langage reconnu par $\mathcal{A}$ en la partie $X$ est l'ensemble des arbres pour lesquels il existe une exécution aboutissant à un état de $X$ :

$$\mathscr{L}(\mathcal{A}, X) = \left\{ t \in T(\Sigma) \;\middle|\; \exists q \in X, t \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q \right\}.$$ 27 ◀

*Remarque 28.* On aurait pu se contenter de parler de la reconnaissance en un état. En effet, $\mathscr{L}(\mathcal{A}, X) = \bigcup_{q \in X} \mathscr{L}(\mathcal{A}, \{q\})$.

Dans notre étude, nous aurons besoin de considérer non seulement quels termes sont reconnus en un état, mais aussi quelles sont les configurations qui aboutissent à un état. Nous introduisons donc la notion, plus large, d'ensemble des configurations reconnues.

**Définition 29 (Ensemble des configurations reconnues par un automate, $\mathscr{C}(\cdot,\cdot)$).**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate et $X$ une partie de $Q$. L'ensemble des configurations reconnues par $\mathcal{A}$ en la partie $X$ est l'ensemble des configurations de $\mathcal{A}$ pour lesquelles il existe une exécution aboutissant à un état de $X$ :

$$\mathscr{C}(\mathcal{A}, X) = \left\{ c \in T(\Sigma \cup Q) \;\middle|\; \exists q \in X, c \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q \right\}.$$ 29 ◀

*Remarque 30.* Les termes sont des configurations, donc $\mathscr{C}(\mathcal{A}, X) \supseteq \mathscr{L}(\mathcal{A}, X)$. Les états sont des configurations, donc $\mathscr{C}(\mathcal{A}, X) \supseteq X$.

*Exemple 31.*
Avec les notations de l'exemple 18, on a

$$\mathscr{L}(\mathcal{A}, \{q_{{}_1g({}_0d)}\}) = \{{}_1g^n({}_0d) \mid n \in \mathbb{N}_*\}$$

et

$$\mathscr{C}(\mathcal{A}, \{q_{{}_1g({}_0d)}\}) = \{{}_1g^n({}_0d) \mid n \in \mathbb{N}_*\} \cup \left\{{}_1g^n(q_{{}_0d}) \;\middle|\; n \in \mathbb{N}_*\right\} \cup \left\{{}_1g^n(q_{{}_1g({}_0d)}) \;\middle|\; n \in \mathbb{N}\right\}.$$ 31 ◁

Notre choix de ne pas utiliser d'états terminaux nous conduit à une présentation de la notion de régularité légèrement différente de celle qui est utilisée habituellement.

**Définition 32 (Langage régulier).**
Soit $\Sigma$ une signature et $L$ un langage sur $\Sigma$. On dit que $L$ est régulier s'il existe un automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ et une partie $Q_f$ de $Q$ telle que

$$L = \mathscr{L}(\mathcal{A}, Q_f).$$ 32 ◀

**Définition 33 (Automate universel).**
On dit d'un automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ qu'il est universel si $\mathscr{L}(\mathcal{A}, Q) = T(\Sigma)$. 33 ◀

Comme dit, les automates d'arbres sont analogues aux automates de mots. On trouve dans [Com+08] de nombreux résultats renforçant cette analogie : déterminisation d'automate non déterministe, lemme de pompage, stabilité de la reconnaissabilité par union, passage au complémentaire et intersection. Comme nous l'annoncions dans l'introduction de cette section 2.2, l'utilisation du type d'automates d'arbres présenté ci-dessus dans notre étude est motivée par le résultat suivant :

**Théorème 34 (Intersection de langages réguliers).**
Le problème de la vacuité de l'intersection de deux langages réguliers est décidable. 34 ■

## 2.3 Systèmes de réécriture

Cette section s'appuie sur [BN98], [Com+08] et [GT95].

### 2.3.1 Définitions

Considérons la représentation unaire des entiers naturels munie de l'addition. Nous savons déduire des axiomes de neutralité additive de zéro ($x + 0 = x$) et de compatibilité de l'addition avec le successeur ($x + S(y) = S(x + y)$) que $1 + 2 = 3$, c'est-à-dire que $+(S(0), S(S(0)))$ et $S(S(S(0)))$ désignent le même entier. Pour avoir un espoir d'obtenir un tel résultat automatiquement, il faut indiquer à l'ordinateur que l'égalité de chacun des deux axiomes est utilisée dans le sens de gauche à droite. Voilà ce qu'est une règle de réécriture : une égalité orientée.

**Définition 35 (Règle de réécriture).**
Une règle de réécriture sur $T(\Sigma, \mathcal{X})$ est un couple de termes $(\ell, r) \in T(\Sigma, \mathcal{X}) \times T(\Sigma, \mathcal{X})$ qu'on note $\ell \to r$. On impose, pour cette notion, que $\ell$ ne soit pas une variable et que toute variable présente dans $r$ soit également présente dans $\ell$. 35 ◀

**Définition 36 (Système de réécriture de termes).**
On appelle système de réécriture (de termes) sur $T(\Sigma, \mathcal{X})$ tout ensemble de règles de réécriture sur ce langage. 36 ◀

Dans toute la suite de cette section, on considère un système de réécriture $R$ sur le langage de termes $T(\Sigma, \mathcal{X})$. Les règles de réécriture contiennent (en général) des variables : lorsqu'on applique une règle pour réécrire un terme, une substitution est appliquée pour associer chaque variable à un morceau du terme réécrit. Enfin, la règle peut s'appliquer à un sous-terme au lieu de la racine. De manière duale, cela signifie qu'une règle peut s'appliquer dans un contexte.

**Définition 37 (Réécriture d'un terme).**
Soit $t, s$ deux termes de $T(\Sigma, \mathcal{X})$. On dit que le terme $t$ peut se réécrire en le terme $s$ grâce au système de réécriture $R$, et on note $t \to_R s$, s'il existe une règle $\ell \to r \in R$, un contexte $C$ sur $\Sigma$ et une substitution $\sigma$ sur $T(\Sigma, \mathcal{X})$ tels que

$$t = C[\ell\sigma] \qquad \text{et} \qquad s = C[r\sigma].$$

On désigne par $\to_R$ la relation de réécriture engendrée par $R$. On note $\to_R^*$ la clôture réflexive et transitive de $\to_R$. 37 ◀

*Remarque 38.* Lorsqu'il n'y a pas d'ambigüité, on note $\to$ au lieu de $\to_R$.

**Définition 39 ($R$, $R^*$).**
Soit $L$ une partie de $T(\Sigma, \mathcal{X})$. On note $R(L)$ l'ensemble des termes qui peuvent être obtenus à partir des termes de $L$ en une étape de réécriture par $R$ :

$$R(L) = \{s \in T(\Sigma, \mathcal{X}) \mid \exists t \in L, t \to_R s\}.$$

De même, on note $R^*(L)$ l'ensemble des termes obtenus après un nombre fini quelconque d'étapes de réécriture :

$$R^*(L) = \left\{s \in T(\Sigma, \mathcal{X}) \;\middle|\; \exists t \in L, t \to_R^* s\right\}.$$ 39 ◀

*Remarque 40.* En particulier, $R^*(L) \supseteq L$.

*Exemple 41.*
Reprenons l'exemple 1 de la factorielle. Nous nous étions donné deux règles

$$\begin{array}{lrcl} \rho_1 : & \texttt{factorielle}(\texttt{0}) & \to & \texttt{S}(\texttt{0}) \\ \rho_2 : & \texttt{factorielle}(\texttt{S}(x)) & \to & \texttt{mult}(\texttt{S}(x), \texttt{factorielle}(x)) \end{array}$$

auxquelles il faudrait rajouter des règles traitant de la multiplication. Avec ces règles, on peut exprimer 2! sous la forme d'un produit $(2 \times 1 \times 1)$. Nous partons du terme $\texttt{factorielle}(\texttt{S}(\texttt{S}(\texttt{0})))$. Nous appliquons la règle $\rho_2$ avec la substitution $\sigma : x \mapsto \texttt{S}(\texttt{0})$ et le contexte $C = \Diamond$ (on réécrit à la racine) : nous obtenons $\texttt{mult}(\texttt{S}(\texttt{S}(\texttt{0})), \texttt{factorielle}(\texttt{S}(\texttt{0})))$. Nous allons à nouveau appliquer la règle $\rho_2$, mais cette fois avec la substitution $\sigma' : x \mapsto \texttt{0}$ et le contexte $C' = \texttt{mult}(\texttt{S}(\texttt{S}(\texttt{0})), \Diamond)$. Nous obtenons $\texttt{mult}(\texttt{S}(\texttt{S}(\texttt{0})), \texttt{mult}(\texttt{S}(\texttt{0}), \texttt{factorielle}(\texttt{0})))$. On applique cette fois $\rho_1$ avec une substitution quelconque (il n'y a pas de variable dans $\rho_1$) et un contexte approprié. 41 ◁

*Exemple 42.*
L'exemple 1 était de nature fonctionnelle. Voici comment on peut utiliser le formalisme de la réécriture d'une manière différente afin de simuler l'effet d'une instruction impérative, par exemple `if`. On considère un symbole de fonction ternaire `if`, deux constantes `true` et `false`. On se donne les règles

$$\begin{array}{rcl} \texttt{if}(\texttt{true}, x, y) & \to & x \\ \texttt{if}(\texttt{false}, x, y) & \to & y \end{array}$$

L'exécution de **if**(Condition, TruePart, FalsePart) consiste d'abord à réduire Condition en l'une des deux constantes **true** ou **false**, puis en l'application de l'une des deux règles ci-dessus.

Dans le cas d'une comparaison numérique <=, on peut se donner, en utilisant la représentation unaire des entiers naturels : <=(0, $x$) → **true** ; <=(S($x$), 0) → **false** et <=(S($x$), S($y$)) → <=($x$, $y$).

Nous pouvons traduire les boucles des langages impératifs :

**while**(Condition, Instr) → **if**(Condition, **Sequence**(Instr, **while**(Condition, Instr)), **skip**).

42 ◁

Sur l'exemple du **while** précédent, on constate qu'il est possible d'appliquer à nouveau la règle dans le membre de droite. En pratique bien sûr, une bonne stratégie de réécriture consiste à réduire Condition puis **if**, de sorte que lorsque la condition n'est plus vérifiée, on ne rencontre plus le **while**. Toutefois, la possibilité de développer le **while** a pour conséquence que le langage $M = R^*(\{$**while**(**false**, **skip**)$\})$ est infini. En effet, en appliquant plusieurs fois la règle ci-dessus, on observe successivement que chacun des termes qui suivent est dans $M$ :

**while**(Condition, Instr)

**if**(Condition, **Sequence**(Instr, **while**(Condition, Instr)), **skip**)

**if**(Condition, **Sequence**(Instr, **if**(Condition, **Sequence**(Instr, **while**(Condition, Instr)), **skip**), **skip**))

**if**(Condition, **Sequence**(Instr, **if**(Condition, **Sequence**(Instr, **if**(Condition, **Sequence**(Instr, **if**(Condition, **Sequence**(Instr, **while**(Condition, Instr)), **skip**), **skip**), **skip**), **skip**))))

...

Les systèmes de réécriture sont un formalisme Turing-complet : on a donc les mêmes résultats d'indécidabilité que pour les machines de Turing [BN98]. En particulier, on ne peut pas décider $R^*(L)$, même lorsque $L$ est décidable. C'est pourquoi nous avons introduit les langages réguliers.

### 2.3.2 Propriétés des systèmes de réécriture

Nous indiquons ici quelques propriétés que les systèmes de réécriture peuvent vérifier. Nous ne détaillons pas ces notions, qui sont expliquées dans [BN98].

**Définition 43 (Forme normale).**
On dit qu'un terme $t$ est une forme normale du système de réécriture $R$ si ce terme ne peut pas être réécrit par $R$, c'est-à-dire

$$R(\{t\}) = \emptyset.$$

Un système $R$ étant considéré, on dit que le terme $t$ est une forme normale du terme $s$ si $t$ est une forme normale au sens précédent et si $t$ est accessible depuis $s$. Autrement dit, si

$$t \in R^*(\{s\}) \qquad \text{et} \qquad R(\{t\}) = \emptyset.$$

On note

$$R^!(L)$$

l'ensemble des formes normales des termes du langage $L \subseteq T(\Sigma)$. 43 ◀

**Définition 44 (Caractère normalisant).**
On dit qu'un système de réécriture $R$ est normalisant si tout terme admet une forme normale, c'est-à-dire

$$\forall t \in T(\Sigma), \quad R^!(\{t\}) \neq \emptyset.$$

44 ◀

**Définition 45 (Caractère terminant).**
On dit qu'un système de réécriture est terminant (ou nœtherien) s'il n'existe pas de suite de réécritures infinie. 45 ◀

*Remarque 46.* Tout système terminant est normalisant, mais la réciproque n'est pas vraie.

**Définition 47 (Caractère confluent).**
On dit qu'un système de réécriture $R$ est confluent si pour tout terme $s$, pour tous $t, t' \in R^*(\{s\})$, on a $R^*(\{t\}) \cap R^*(\{t'\}) \neq \emptyset$. Autrement dit, si les traits pleins de la figure 2 peuvent être complétés par les traits pointillés. 47 ◀

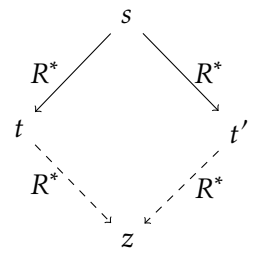



FIGURE 2 : Situation de confluence

Nous avons exposé la notion de linéarité d'un terme dans la section précédente. Cette notion est également utilisée pour parler de systèmes de réécriture. Nous donnons une définition et quelques exemples avant d'expliquer en quoi cette notion est importante.

**Définition 48 (Système de réécriture linéaire, à gauche, à droite).**
On dit qu'un système de réécriture $R$ est linéaire à gauche si le membre de gauche de chacune de ses règles est linéaire. On dit que $R$ est linéaire à droite si le membre de droite de chacune de ses règles est linéaire. On dit que $R$ est linéaire s'il est linéaire à gauche et linéaire à droite. 48 ◀

*Exemple 49.*
Le système de réécriture suivant, noté $R$, est linéaire à gauche, mais pas à droite :

$$\begin{aligned} {}_3h({}_0c, x, y) &\to {}_0c \\ {}_2g({}_0d, x) &\to {}_3h(x, {}_1f(x), {}_0c) \\ {}_2g({}_0e, {}_0e) &\to {}_0e. \end{aligned}$$

Le système de réécriture suivant, noté $S$, est linéaire à droite, mais pas à gauche :

$$\begin{aligned} {}_3h(x, x, y) &\to {}_1f(x) \\ {}_4i(x, x, y, z) &\to {}_2k(y, z). \end{aligned}$$

49 ◁

Pour comprendre l'importance de la linéarité, observons ce que permet la non-linéarité. La deuxième règle du système $R$ de l'exemple 49 n'est pas linéaire à droite : le terme ${}_3h(x, {}_1f(x), {}_0c)$ contient deux fois la variable $x$. Ainsi, lors de l'application de cette règle de réécriture, l'information représentée par $x$ au travers d'une substitution dans ${}_2g({}_0d, x)$ va être dupliquée.

*Exemple 50.*
Reprenons les notations de l'exemple 49. Soit $L$ le langage $\left\{ {}_2g({}_0d, {}_1f^n({}_0e)) \;\middle|\; n \in \mathbb{N} \right\}$. On a

$$R^*(L) = \left\{ {}_3h({}_1f^n({}_0e), {}_1f^{n+1}({}_0e), {}_0c) \;\middle|\; n \in \mathbb{N} \right\},$$

qui n'est pas régulier : les automates d'arbres ne permettent pas de tester des relations entre sous-termes. 50 ◁

### 2.3.3 Théories équationnelles

Certaines égalités ne peuvent pas être orientées pour définir des règles de réécriture, comme par exemple les égalités traduisant la commutativité d'un opérateur binaire : il n'y a pas de sens à privilégier, et donner les deux orientations conduit à un système de réécriture qui « boucle » en passant d'une forme à l'autre. La réécriture modulo équations a été introduite pour résoudre ce problème : on renonce à orienter les égalités en cause, mais l'on considère équivalents des termes qui sont en relation par ces égalités, et l'on réécrit les classes d'équivalence.

Mais ce n'est pas pour résoudre ce problème que nous introduirons des équations : nous nous en servirons pour paramétrer la sur-approximation de la complétion équationnelle. Nous allons définir des relations d'équivalence, mais il faut qu'elles soient « compatibles avec l'application des fonctions », c'est-à-dire que ce soient des congruences.

**Définition 51 (Congruence).**
On appelle relation de congruence sur un langage de termes toute relation d'équivalence $\sim$ sur ce langage qui est de plus compatible avec l'application formelle des symboles de fonctions : pour tout $k \in \mathbb{N}$, pour tout $f \in \Sigma_k$, pour tous termes $t_1, \ldots, t_k$ et $s_1, \ldots, s_k$, si pour tout $i \in [\![1\,;k]\!]$, $t_i \sim s_i$, alors $f(t_1, \ldots, t_k) \sim f(s_1, \ldots, s_k)$. 51 ◀

**Définition 52 (Équation).**
On appelle équation tout couple de termes. On dit que deux termes clos $t$ et $t'$ sont en relation modulo un ensemble d'équations $E$ s'il existe une équation $(\ell, r) \in E$ et une substitution $\sigma$ sur $T(\Sigma)$ tels que $\ell\sigma = t$ et $r\sigma = t'$.

On note $\equiv_E$ la plus petite relation de congruence sur $T(\Sigma)$ qui contient toutes les paires de termes $(t, t')$ qui sont en relation modulo $E$ au sens précédent. 52 ◀

*Exemple 53.*
Considérons l'ensemble d'équations $E = \{{}_2f(x,y) = {}_2f(y,x)\}$, qui exprime la commutativité de ${}_2f$. Montrons que ${}_1g({}_2f({}_0c, {}_0d)) \equiv_E {}_1g({}_2f({}_0d, {}_0c))$. En utilisant la substitution $[x \mapsto {}_0c, y \mapsto {}_0d]$ avec l'unique équation de $E$, on obtient que ${}_2f({}_0c, {}_0d)$ et ${}_2f({}_0d, {}_0c)$ sont en relation modulo $E$ au sens de la première partie de la définition précédente, et donc que ${}_2f({}_0c, {}_0d) \equiv_E {}_2f({}_0d, {}_0c)$. Mais $\equiv_E$ est une congruence, donc ceci entraine que ${}_1g({}_2f({}_0c, {}_0d)) \equiv_E {}_1g({}_2f({}_0d, {}_0c))$. 53 ◁

**Définition 54 (Réécriture modulo équations).**
Soit $R$ un système de réécriture et $E$ un ensemble d'équations. On dit que le terme clos $t \in T(\Sigma)$ se réécrit par $R$ en le terme clos $s \in T(\Sigma)$ modulo $E$ s'il existe $t', s' \in T(\Sigma)$ tels que $t \equiv_E t'$, $s \equiv_E s'$ et $t' \to_R s'$. En ce cas, on note $t \to_{R/E} s$. On adopte également les notations analogues $(R/E)(L)$, $(R/E)^*(L)$. 54 ◀

*Remarque 55.* Pour $t$ un terme, on note $[t]_E$ la classe d'équivalence des termes $s$ qui sont en relation modulo $E$ avec $t$ :

$$[t]_E = \{s \in T(\Sigma) \mid s \equiv_E t\}.$$

Ainsi, $R/E$ peut être vu comme une relation de réécriture sur les classes d'équivalence ; on peut noter

$$[t]_E \to_{R/E} [s]_E$$

lorsque la situation de la définition précédente se présente.

*Exemple 56.*
Reprenons l'ensemble $E$ de l'exemple 53 et considérons le système de réécriture $R$ ayant pour unique règle ${}_2f({}_0c, x) \to {}_1g(x)$. Le terme ${}_2f({}_0d, {}_0c)$ est une forme normale pour $R$, mais pas pour $R/E$. En effet, on a montré à l'exemple 53 que ${}_2f({}_0d, {}_0c) \equiv_E {}_2f({}_0c, {}_0d)$, et d'autre part, on a ${}_2f({}_0c, {}_0d) \to_R {}_1g({}_0d)$, donc ${}_2f({}_0d, {}_0c) \to_{R/E} {}_1g({}_0d)$. 56 ◁

# 3 Le problème de l'atteignabilité

Le problème d'atteignabilité évoqué en introduction se formalise dans notre cadre théorique de la manière suivante :

**Problème 57 (Atteignabilité).**
Soit $R$ un système de réécriture et $L_0$ un langage de termes. Soit $L_{\text{bad}}$ un langage de termes. A-t-on $R^*(L_0) \cap L_{\text{bad}} \neq \emptyset$ ? 57 ◀

Compte tenu de la signification que nous attachons aux objets en jeu, nous cherchons une réponse négative à ce problème.

Ce problème n'est pas décidable en général. Toutefois, si $R^*(L_0)$ et $L_{\text{bad}}$ sont réguliers, on peut appliquer le théorème 34. Il n'est pas difficile de savoir si le langage $L_{\text{bad}}$ que l'on considère pour une analyse donnée est régulier, car ce langage est connu.

*Exemple 58.*
On peut vouloir vérifier que le programme de l'exemple 1 ne renvoie jamais la valeur 0. En ce cas, compte tenu de la traduction en système de réécriture retenue, $L_{\text{bad}} = \{0\}$. 58 ◁

En revanche, on ne connait pas extensivement $R^*(L_0)$ : cette connaissance reviendrait à connaitre toutes les exécutions possibles du programme étudié. Toutefois, pour les mêmes raisons que pour $L_{\text{bad}}$, il est aisé de savoir si $L_0$, qui dénote l'état initial, est régulier. Ces considérations ont amené les chercheurs à s'intéresser au problème suivant :

**Problème 59.**
Soit $R$ un système de réécriture. Est-il vrai que pour tout langage régulier $L$, $R^*(L)$ est un langage régulier ? 59 ◀

Les auteurs de [GT95] ont montré que le problème 59 n'est pas décidable.

Néanmoins, plusieurs classes de systèmes de réécriture préservant la régularité sont connues. Une liste hiérarchisée de telles classes connues se trouve dans [Gen09, chap. 2]. Nous n'en disons pas plus à ce sujet, qui n'est pas le cœur de l'étude menée pendant le stage.

# 4 Complétion équationnelle

## 4.1 Introduction

Pour résoudre le problème d'atteignabilité, on peut essayer de trouver un langage régulier $K$ contenant $R^*(L_0)$ et disjoint de $L_{\text{bad}}$ :

**Problème 60.**
Soit $R$ un système de réécriture et $L_0$ un langage de termes. Soit $L_{\text{bad}}$ un langage de termes. Trouver un langage régulier $K$ tel que $R^*(L_0) \subseteq K$ et $K \cap L_{\text{bad}} = \emptyset$. 60 ◀

En effet, dès lors qu'on a résolu le problème 60, la réponse au problème 57 d'atteignabilité est non.

Il est démontré dans [BH08] qu'il n'est pas toujours possible d'intercaler ainsi un langage régulier dans l'étude :

**Théorème 61.**
Il existe un système de réécriture $R$ et un langage régulier $L_0$ tel qu'il existe $t \in T(\Sigma)$ tel que
– pour tout $K$ régulier contenant $L_0$ et stable par $R$, $t \in K$ ;
– et pourtant, $t \notin R^*(L_0)$. 61 ■

Néanmoins, cette idée est féconde car elle fournit de nouvelles classes de systèmes de réécriture pour lesquels on dispose d'un critère de résolution du problème d'atteignabilité. Un moyen de répondre au problème 60 de manière effective est de procéder par complétion d'automate d'arbre.

Au surplus, pour certaines classes de systèmes de réécriture préservant la régularité et pour certains langages, la complétion n'effectue en réalité aucune sur-approximation : elle calcule exactement $R^*(L_0)$.

Il existe plusieurs méthodes de complétion. Dans notre rapport de bibliographie [Sal11], nous présentions de manière résumée la complétion dite « standard » proposée par [Gen97]. Dans le présent document, nous présentons la complétion équationnelle de [GR10], qui est l'objet de notre étude.

Le contenu de la présente section reprend largement [Gen09, sect. 3.2], avec une présentation et une organisation différente. Nous y renvoyons pour les preuves omises ici.

## 4.2 Présentation informelle

Dans toute cette section, on fixe une signature $(\Sigma, \mathcal{X})$. Tous les objets considérés portent sur cette signature.

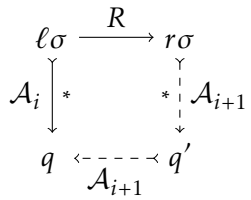
On considère un système de réécriture $R$ fini, qui représente un programme à analyser. On se donne aussi un langage $L_0$, régulier, qui représente l'état initial de la machine destinée à exécuter le programme. Enfin, on se donne un ensemble fini d'équations $E$, qui va permettre d'effectuer des sur-approximations : nous allons identifier les paires de termes qui sont en relation modulo $E$.

La complétion équationnelle est une méthode itérative de point fixe. Chaque itération se décompose en deux étapes : la complétion proprement dite, que nous appelons « complétion exacte », et la simplification par les équations.

Nous ne travaillons ici qu'avec des langages réguliers, et nous allons exploiter ce fait en ne parlant que des automates reconnaissant les langages manipulés. Ainsi, nous notons $\mathcal{A}_0$ un automate d'arbre (avec états terminaux $Q_f$) reconnaissant $L_0$. Une étape de complétion exacte consiste à ajouter certaines transitions à l'automate $\mathcal{A}_i$ pour obtenir un automate $\mathcal{A}_{i+1}$ tel que $\mathscr{L}(\mathcal{A}_{i+1}, Q_f) \supseteq \mathscr{L}(\mathcal{A}_i, Q_f) \cup R(\mathscr{L}(\mathcal{A}_i, Q_f))$. L'intérêt d'obtenir un point fixe est le suivant :

**Énoncé 62.**
S'il existe un automate $\mathcal{A}_{i_{\text{fix}}}$ tel que $\mathcal{A}_{i_{\text{fix}}+1} = \mathcal{A}_{i_{\text{fix}}}$, alors $\mathscr{L}(\mathcal{A}_{i_{\text{fix}}}, Q_f) \supseteq R^*(L_0)$. 62 ■

L'étape de complétion exacte consiste à voir l'automate $\mathcal{A}_i$ comme un système de réécriture et à joindre les paires critiques comme indiqué en figure 3, en ajoutant un nouvel état $q'$.




*($\sigma : \mathcal{X} \to Q$ est un remplacement,*
*$q'$ est un nouvel état)*

FIGURE 3 : Une paire critique et sa jonction par complétion exacte

*Exemple 63.*
Reprenons l'automate de l'exemple 18. Considérons un système de réécriture comprenant la règle ${}_1g(x) \to {}_2f({}_0d, {}_1g(x))$ et $\sigma : x \mapsto q_{{}_0d}$. Alors $\ell\sigma = {}_1g(q_{{}_0d})$ et on a $\ell\sigma \xrightarrow[\mathcal{A}_0]{*} q_{{}_1g({}_0d)}$, tandis que

$r\sigma = {}_2f({}_0d, {}_1g(q_{{}_0d}))$, et on a $r\sigma \overset{*}{\underset{\mathcal{A}_0}{\not\rightarrow}} q_{{}_1g({}_0d)}$. Pour résoudre la paire critique, on doit joindre ${}_2f({}_0d, {}_1g(q_{{}_0d}))$ à $q'$ dans $\mathcal{A}_1$. Cette situation est décrite par la figure 4.

La jonction ne peut être effectuée brutalement car la définition 21 n'autorise pas des transitions aussi compliquées. Nous allons donc devoir détailler les transitions successives. L'automate dispose déjà de transitions ${}_0d \to q_{{}_0d}$ et ${}_1g(q_{{}_0d}) \to q_{{}_1g({}_0d)}$, de sorte qu'on a déjà ${}_2f({}_0d, {}_1g(q_{{}_0d})) \overset{*}{\underset{\mathcal{A}_0}{\rightarrow}} {}_2f(q_{{}_0d}, q_{{}_1g({}_0d)})$. Il suffit donc d'ajouter la transition de cette dernière configuration vers $q'$ pour avoir ${}_2f({}_0d, {}_1g(q_{{}_0d})) \overset{*}{\underset{\mathcal{A}_1}{\rightarrow}} q'$. 63 ◁

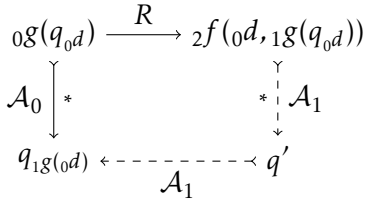



Figure 4 : La paire critique de l'exemple 63

Souvenons-nous que nous cherchons à obtenir un point fixe. La complétion exacte aboutit à un point fixe lorsqu'elle ne trouve plus de paire critique à joindre, mais rien ne garantit que cette situation se produise, car une étape de complétion exacte peut créer de nouvelles paires critiques. Afin de favoriser la terminaison, nous sur-approximons le langage reconnu en fusionnant les états qui reconnaissent des termes équivalents modulo $E$ : c'est la simplification de l'automate par les équations.

*Exemple 64.*
Reprenons l'automate de l'exemple 18 et considérons $E = \{{}_1g(x) = x\}$. La simplification équationnelle par $E$ va fusionner les états $q_{{}_0d}$ et $q_{{}_1g({}_0d)}$, car ces états reconnaissent respectivement les termes ${}_0d$ et ${}_1g({}_0d)$, qui sont égaux modulo $E$. Cet exemple sera détaillé dans les exemples 77 et 84. 64 ◁

## 4.3 Définition et première étude

La complétion exacte crée de nouveaux états, la simplification équationnelle en supprime. Afin de ne faire varier que les relations de transitions, nous allons faire usage de la remarque 23 et prendre un ensemble d'états infini, qui sera commun à tous les automates que nous allons construire.

Dans la complétion équationnelle, les epsilon-transitions jouent un rôle particulier : elles vont correspondre à des étapes de réécriture. C'est pourquoi nous partirons d'un automate initial sans epsilon-transition : $\mathcal{A}_0 = (\Sigma, Q, \delta, \varnothing)$.

Pour la même raison, nous introduisons la notion suivante, qui permet de considérer la « reconnaissance sans epsilon-transition ».

**Définition 65 (Automate dépouillé).**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate. On appelle automate dépouillé de $\mathcal{A}$, et on note $\mathcal{A}^\not\varepsilon$, l'automate obtenu en retirant les epsilon-transitions : $\mathcal{A}^\not\varepsilon = (\Sigma, Q, \delta, \varnothing)$. 65 ◀

*Remarque 66.* Il faut comprendre que l'on retire les transitions « brutalement » et sans modifier les transitions de $\delta$, si bien que $\mathcal{A}^\not\varepsilon$ n'a en général pas le même comportement que $\mathcal{A}$. Pour des raisons de clarté, on notera parfois $\overset{\not\varepsilon}{\underset{\mathcal{A}}{\rightarrow}}$ et $\overset{\not\varepsilon,*}{\underset{\mathcal{A}}{\rightarrow}}$ au lieu de $\underset{\mathcal{A}^\not\varepsilon}{\rightarrow}$ et $\overset{*}{\underset{\mathcal{A}^\not\varepsilon}{\rightarrow}}$.

### 4.3.1 Complétion exacte

Commençons par formaliser la notion de nouvel état. Notons qu'en vertu de la remarque 23, il y a toujours des nouveaux états au sens de la définition suivante.

**Définition 67 (Nouvel état).**
Soit $Q$ un ensemble infini et $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate. On appelle nouvel état pour $\mathcal{A}$ tout état de $Q$ qui n'apparait pas dans $\Delta$. 67 ◀

Dans toute la suite, nous notons $R$ un système de réécriture fini et $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate dont nous considérerons qu'il a été obtenu par de précédentes étapes de complétion exacte à partir d'un automate sans epsilon-transition. Cela a pour conséquence (nous le montrerons dans le lemme 74) que les seules epsilon-transitions sont celles qui ont été délibérément ajoutées par complétion.

Nous formalisons à présent la notion de paire critique.

**Définition 68 (Paire critique).**
Une paire critique relativement à $R$ et $\mathcal{A}$ est la donnée d'une règle $\ell \to r$ de $R$, d'un remplacement $\sigma : \mathcal{X} \to Q$ et d'un état $q \in Q$ vérifiant

$$\ell\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q \qquad \text{et} \qquad r\sigma \overset{*}{\underset{\mathcal{A}}{\not\rightarrowtail}} q.$$ 68 ◀

Le lecteur aura reconnu la partie en trait plein de la figure 3. La complétion consiste à ajouter la partie en pointillés de cette figure. Il suffit en fait de considérer les paires critiques non triviales au sens suivant. [3]

**Définition 69 (Paire critique triviale).**
On dit qu'une paire critique $(\ell \to r, \sigma, q)$ est triviale si pour chacune des dérivations $\ell\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q$, il existe un remplacement $\tau : \mathcal{X} \to Q$ qui s'insère dans la dérivation : $\ell\sigma \overset{+}{\underset{\mathcal{A}}{\rightarrowtail}} \ell\tau \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q$. 69 ◀

Dans la situation considérée, il n'est pas nécessaire d'étudier la paire critique triviale $(\ell \to r, \sigma, q)$ : on peut se contenter de $(\ell \to r, \tau, q)$.

**Définition 70 (Paire critique non triviale).**
On appelle paire critique non triviale relativement à $R$ et $\mathcal{A}$ toute paire critique de ce système qui n'est pas triviale au sens précédent.

On note $\mathcal{PC}(R, \mathcal{A})$ l'ensemble de ces paires critiques non triviales. 70 ◀

Nous souhaitons joindre les paires critiques (non triviales). À cause de la forme imposée des transitions de la définition des automates (21), il faut décomposer la jonction : c'est l'opération de normalisation, qui est définie inductivement. La normalisation consiste à simplifier une configuration $c$ en un état $q'$, de sorte que la transition $q' \to q$ est licite. L'état $q'$ peut être nouveau : c'est ainsi que la procédure de complétion exacte ajoute de nouveaux états. Enfin, la procédure de normalisation doit également fournir l'ensemble des transitions qui permettent de passer de la configuration $c$ à l'état $q'$.

**Définition 71 (Normalisation).**
Étant donné $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$, la normalisation prend une configuration $c$ et retourne un couple $(q', N)$ composé d'un état et d'un ensemble de transitions, noté $Norm_{\mathcal{A}}(c)$.

Si $c$ est un état ($c \in Q$), on n'a rien à faire : $q' = c$ et $N = \varnothing$.

Sinon, $c$ est de la forme $f(c_1, \ldots, c_k)$, avec $k \in \mathbb{N}$, $f \in \Sigma_k$ et $c_1, \ldots, c_n$ des configurations. On commence par traiter chacune des sous-configurations $c_i$, en construisant pour chacune un couple $(q_i', N_i)$ comme suit :

3. Il s'agit là d'une optimisation utile dans les implémentations mais dont la compréhension n'est pas indispensable à la poursuite de la lecture de ce rapport.

– s'il existe déjà un état $q'_i$ tel que $c_i \overset{\not\varepsilon,*}{\underset{\mathcal{A}}{\rightarrow}} q'_i$, on prend un tel $q'_i$ et $N_i = \varnothing$ ;

– sinon, on prend $(q'_i, N_i) = Norm_{\mathcal{A}}(c_i)$.

Enfin, s'il existe déjà un état $q'$ tel que la transition $f(q'_1, \ldots, q'_k) \to q'$ soit dans $\delta$, on choisit un tel $q'$ et on prend $N = \bigcup_{i=1}^{k} N_i$ ; tandis que s'il n'existe pas un tel état, on prend un nouvel état $q'$ et on prend $N = \{f(q'_1, \ldots, q'_k) \to q'\} \cup \bigcup_{i=1}^{k} N_i$. 71 ◀

Forts de cette procédure, nous pouvons définir la complétion exacte proprement dite.

**Définition 72 (Complétion exacte).**
Une étape de complétion exacte transforme l'automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ en l'automate

$$\mathcal{C}_R(\mathcal{A}) = (\Sigma, Q, \delta', \delta'_\varepsilon)$$

défini comme suit. Initialement, $\delta' = \delta$ et $\delta'_\varepsilon = \delta_\varepsilon$. Pour chaque paire critique non triviale $(\ell \to r, \sigma, q) \in \mathcal{PC}(R, \mathcal{A})$, notant $(q', N) = Norm_{\mathcal{A}}(r\sigma)$, on ajoute les transitions de $N$ à $\delta'$ et on ajoute $q' \to q$ à $\delta'_\varepsilon$.

Si bien que finalement,

$$\delta' = \delta \cup \bigcup_{(\ell \to r, \sigma, q) \in \mathcal{PC}(R, \mathcal{A})} Norm_{\mathcal{A}}(r\sigma).N \ ;$$

$$\delta'_\varepsilon = \delta_\varepsilon \cup \bigcup_{(\ell \to r, \sigma, q) \in \mathcal{PC}(R, \mathcal{A})} \{Norm_{\mathcal{A}}(r\sigma).q' \to q\}.$$ 72 ◀

*Remarque 73.* La procédure de normalisation n'étant pas déterministe, l'automate $\mathcal{C}_R(\mathcal{A})$ n'est pas défini de manière unique. De plus, les transitions ajoutées dépendent de l'ordre dans lequel est parcouru $\mathcal{PC}(R, \mathcal{A})$.

Pour la bonne définition de $\delta'$, il convient de vérifier le lemme suivant, qui traduit également le rôle particulier joué par les epsilon-transitions dans les automates que nous considérons.

**Lemme 74.**
Soit $c$ une configuration et $(q', N) = Norm_{\mathcal{A}}(c)$. L'ensemble $N$ ne possède aucune epsilon-transition. 74 ■

*Démonstration.*
On procède par induction. Si $c$ est un état, $N = \varnothing$. Supposons maintenant que $c = f(c_1, \ldots, c_k)$ avec les $c_i$ vérifiant l'hypothèse d'induction. Pour tout $i \in [\![1\,;k]\!]$, $N_i = \varnothing$ ou $N_i = Norm_{\mathcal{A}}(c_i)$, qui par hypothèse d'induction ne possède pas d'epsilon-transition. Enfin, $N$ est la réunion de ces $N_i$, à laquelle on ajoute éventuellement une transition de la forme $f(q'_1, \ldots, q'_k) \to q'$, qui n'est pas une epsilon-transition. □

### 4.3.2 Simplification équationnelle

Cette étape est conceptuellement indépendante de la précédente. On ne trouvera donc pas ici de question relative à des paires critiques.

On se donne un ensemble fini $E$ d'équations sur $(\Sigma, \mathcal{X})$. Considérons deux termes $s$ et $t$ vérifiant $s \equiv_E t$. La simplification équationnelle a pour but de faire en sorte que ces deux termes soient reconnus par un unique état de $\mathcal{A}$, et aucun autre. Cet objectif est atteint en fusionnant tous les états reconnaissant $s$ et $t$. La fusion est effectuée par renommage.

**Définition 75 (Renommage).**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate et $q_1, q_2 \in Q$. On appelle renommage de $q_1$ en $q_2$ dans l'automate $\mathcal{A}$ l'opération consistant à remplacer toutes les occurrences de $q_1$ par des occurrences de $q_2$ dans $\delta$ et $\delta_\varepsilon$. L'automate résultant de cette opération est noté $\mathcal{A}\{q_1 \mapsto q_2\}$. On peut également effectuer le renommage simultané d'un $n$-uplet d'états en un autre $n$-uplet d'états : on note un tel renommage $\{(q_1, \dots, q_n) \mapsto (r_1, \dots, r_n)\}$. 75 ◀

*Remarque 76.* Pour $c$ une configuration et $X$ un ensemble de configurations, on note aussi $c\{q_1 \mapsto q_2\}$ et $X\{q_1 \mapsto q_2\}$ pour signifier qu'on a remplacé $q_1$ par $q_2$. Ainsi, pour un état $q$, $q\{q_1 \mapsto q_2\}$ désigne $q_2$ si $q = q_1$ et $q$ dans tout autre cas.

*Exemple 77.*
Reprenons l'exemple 18 et notons $\mathcal{A}' = \mathcal{A}\{q_{1d(0d)} \mapsto q_{0d}\}$. Le renommage a un effet sur les transitions suivantes.

| Avant | Après |
|---|---|
| ${}_1g(q_{0d}) \to q_{1g(0d)}$ | ${}_1g(q_{0d}) \to q_{0d}$ |
| ${}_1g(q_{1g(0d)}) \to q_{1g(0d)}$ | ${}_1g(q_{0d}) \to q_{0d}$ |
| ${}_2f(q_{0c}, q_{1g(0d)}) \to q_{2f}$ | ${}_2f(q_{0c}, q_{0d}) \to q_{2f}$ |

Si bien qu'on a $\mathscr{L}(\mathcal{A}', Q_f) = \{{}_2f({}_1g^m({}_0c), {}_1g^n({}_0d)) \mid m, n \in \mathbb{N}\}$. Le renommage considéré revient à négliger la différence entre la présence d'au moins un ${}_1g$ et son absence : l'automate $\mathcal{A}'$ reconnait donc en $Q_f$ les mots de la forme ${}_2f({}_1g^m({}_0c), {}_0d)$, ce qui n'était pas le cas de $\mathcal{A}$. 77 ◁

*Remarque 78.* Le renommage ne change pas le type des transitions, de sorte que l'on a $(\mathcal{A}\{q_1 \mapsto q_2\})^\not{\varepsilon} = \mathcal{A}^\not{\varepsilon}\{q_1 \mapsto q_2\}$. Ainsi, les propriétés que nous allons énoncer sur le renommage restent vraies si l'on y remplace $\mathcal{A}$ par $\mathcal{A}^\not{\varepsilon}$.

On a vu sur l'exemple ci-dessus que le renommage provoque une sur-approximation des langages reconnus. C'est vrai en général : c'est en effectuant de telles opérations de renommage que la complétion équationnelle calcule une sur-approximation de $R^*(L_0)$.

**Lemme 79.**
Soit $\mathcal{A}$ un automate et $q_1, q_2$ deux états. Pour tout état $q$, on a

$$\mathscr{C}(\mathcal{A}\{q_1 \mapsto q_2\}, q\{q_1 \mapsto q_2\}) \supseteq \mathscr{C}(\mathcal{A}, q)\{q_1 \mapsto q_2\}.$$

En particulier, on a

$$\mathscr{L}(\mathcal{A}\{q_1 \mapsto q_2\}, q\{q_1 \mapsto q_2\}) \supseteq \mathscr{L}(\mathcal{A}, q).$$ 79 ■

Nous avons dit que le renommage avait pour objet de réaliser la fusion de deux états. Toutefois, ce mode opératoire brise la symétrie de la fusion : $\mathcal{A}\{q_1 \mapsto q_2\}$ et $\mathcal{A}\{q_2 \mapsto q_1\}$ ne mettent pas en jeu les mêmes états. Ils ont cependant le même comportement, et nous les considérerons comme identiques modulo la relation suivante, dont nous laissons le lecteur se convaincre qu'elle est une relation d'équivalence.

**Définition 80 (Équivalence modulo renommage).**
Soit $\mathcal{A}, \mathcal{A}'$ deux automates sur la même signature $\Sigma$ et le même ensemble d'états $Q$. On dit que $\mathcal{A}$ et $\mathcal{A}'$ sont équivalents à renommage près s'il existe un entier naturel $n$ et deux parties de $Q$ à $n$ éléments $\{q_1, \dots, q_n\}$ et $\{q'_1, \dots, q'_n\}$ tels que $\mathcal{A}' = \mathcal{A}\{(q_1, \dots, q_n) \mapsto (q'_1, \dots, q'_n)\}$. 80 ◀

Le renommage permet d'effectuer des sur-approximations. Il reste à choisir lesquelles. Nous allons à présent définir la simplification équationnelle. Elle consiste à déterminer quels sont les renommages qui vont être effectués, en fonction de l'ensemble d'équations que l'on s'est donné initialement.

**Définition 81 (Relation de simplification).**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate et $E$ un ensemble d'équations sur $\Sigma$. Soit $s = t$ une équation de $E$ ($s, t \in T(\Sigma, \mathcal{X})$) et $\sigma : \mathcal{X} \to Q$ un remplacement. Soit $q_1$, $q_2$ deux états. On dit que la donnée de ces objets constitue une situation d'application d'une équation de $E$ si $s\sigma \overset{\not\varepsilon,*}{\underset{\mathcal{A}}{\rightarrow}} q_1$ et $t\sigma \overset{\not\varepsilon,*}{\underset{\mathcal{A}}{\rightarrow}} q_2$ avec $q_1 \neq q_2$ (voir la figure 5).

En ce cas, l'automate $\mathcal{A}$ est en relation de simplification par $E$ avec l'automate $\mathcal{A}\{q_1 \mapsto q_2\}$. 81 ◀

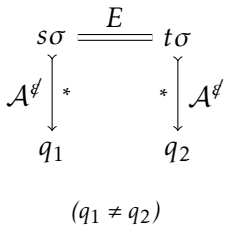



Figure 5 : Situation d'application d'une équation

Dans un même automate, des équations peuvent être appliquées de plusieurs manières et dans des ordres différents, mais après un nombre fini d'opérations, il n'est plus possible d'appliquer aucune équation. De plus, ce résultat final est indépendant de l'ordre des applications effectuées, à renommage près. Cela s'exprime par le théorème suivant, prouvé dans [Gen09]. [4]

**Théorème 82.**
Soit $E$ un ensemble d'équations sur $(\Sigma, \mathcal{X})$. La relation de simplification par $E$, vue comme une relation de réécriture, est terminante et confluente à renommage près. 82 ■

Partant d'un automate $\mathcal{A}$, il existe donc une unique (à renommage près) forme normale pour cette relation : c'est l'automate simplifié $\mathcal{S}_E^!(\mathcal{A})$. Voilà ce qui nous permet de définir l'étape de simplification par les équations, au sens de la complétion équationnelle.

**Définition 83 (Simplification équationnelle).**
Une étape de simplification équationnelle transforme l'automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ en l'automate $\mathcal{S}_E^!(\mathcal{A})$ obtenu par applications successives de la relation de simplification par $E$. 83 ◀

*Exemple 84.*
Reprenons, comme dans l'exemple 64, l'automate de l'exemple 18 et l'équation $E = \{{}_1g(x) = x\}$. On a une situation d'application d'une équation de $E$ en prenant $s = {}_1g(x)$, $t = x$, $\sigma : x \mapsto q_{0d}$, $q_1 = q_{0d}$, $q_2 = q_{1g(0d)}$ (la reconnaissance sana epsilon-transition s'effectue dans chacun des deux cas en zéro étape). On est donc conduit à effectuer le renommage décrit dans l'exemple 77. On vérifie, par énumération, que dans l'automate ainsi obtenu, il n'y a plus de situation d'application : nous avons obtenu $\mathcal{S}_E^!(\mathcal{A})$. 84 ◁

### 4.3.3 Complétion équationnelle

Nous pouvons à présent définir la complétion équationnelle, qui est la composée des deux procédés précédents.

**Définition 85 (Complétion équationnelle).**
Une étape de complétion équationnelle transforme l'automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ en l'automate $\mathcal{CE}_{R,E}(\mathcal{A}) = \mathcal{S}_E^!(\mathcal{C}_R(\mathcal{A}))$. 85 ◀

4. L'auteur impose dès la définition de la relation de simplification que les équations soient linéaires, mais cela ne semble pas nécessaire à ce stade de l'étude.

L'automate $\mathcal{CE}_{R,E}(\mathcal{A})$ n'est pas défini de manière unique car $\mathcal{C}_R(\mathcal{A})$ ne l'est pas.

*Remarque 86.* La procédure de complétion équationnelle ne termine pas toujours. Il s'agit d'un semi-algorithme, comme l'est par exemple la procédure de complétion de MM. Knuth et Bendix [KB70] qui vise à rendre un système de réécriture confluent.

*Exemple 87.*

Considérons $R = \{{}_2f(x,y) \to {}_2f({}_1s(x), {}_1s(y))\}$, $\mathcal{A}_0$ ayant les transitions ${}_0c \to q_c$ et ${}_2f(q_c, q_c) \to q_f$ et $E = \varnothing$. Nous avons une paire critique :

$$\begin{array}{ccc} {}_2f(q_c,q_c) & \xrightarrow{R} & {}_2f({}_1s(q_c), {}_1s(q_c)) \\ \mathcal{A}_0 \downarrow * & & * \vdots \mathcal{A}_1 \\ q_f & \xleftarrow[\mathcal{A}_1]{} & q'_2 \end{array}$$

La complétion exacte introduit les transitions ${}_1s(q_c) \to q'_1$ et ${}_2f(q'_1, q'_1) \to q'_2$ dans l'automate $\mathcal{A}_1 = \mathcal{CE}_{R,E}(\mathcal{A}_0)$. Plus généralement, notant $\mathcal{A}_{p+1} = \mathcal{CE}_{R,E}\left(\mathcal{A}_p\right)$, le fait que ${}_2f(q'_{2p-1}, q'_{2p-1}) \underset{\mathcal{A}_p}{\rightarrowtail} q'_{2p}$ crée une nouvelle paire critique qui conduit à rajouter les transitions ${}_1s(q'_{2p-1}) \to q'_{2p+1}$ et ${}_2f(q'_{2p+1}, q'_{2p+1}) \to q'_{2p+2}$ dans $\mathcal{A}_{p+1}$, de sorte qu'on a de nouveau ${}_2f(q'_{2(p+1)-1}, q'_{2(p+1)-1}) \underset{\mathcal{A}_{p+1}}{\rightarrowtail} q'_{2(p+1)}$.

Une nouvelle paire critique est générée à chaque étape, ce qui empêche l'obtention d'un point fixe. 87 ◁

Dans la suite, nous nous placerons dans le cas où la complétion termine.

**Définition 88 ($\mathcal{A}^!_{R,E}$).**

Étant donné un système de réécriture $R$, un ensemble d'équations $E$ et un automate $\mathcal{A}_0$, nous notons $\mathcal{A}_1 = \mathcal{CE}_{R,E}(\mathcal{A}_0)$, $\mathcal{A}_2 = \mathcal{CE}_{R,E}(\mathcal{A}_1)$, … Dans le cas où la procédure de complétion équationnelle aboutit à un point fixe, nous notons l'automate ainsi construit

$$\mathcal{A}^!_{R,E}.$$ 88 ◀

### 4.3.4 $R/E$-cohérence

Nous définissons et étudions brièvement la notion de $R/E$-cohérence, qui sera utile pour le théorème de précision de la section suivante. Dans cette section, nous nous donnons un système de réécriture $R$, un ensemble d'équations $E$ et un automate $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$. Nous rappelons que $[s]_E$ désigne la classe d'équivalence du terme $s$ par $E$.

Commençons par deux notions essentiellement techniques.

**Définition 89 (Automate rempli).**

On dit que l'automate $\mathcal{A}$ est rempli si pour tout état $q \in Q$, $\mathscr{L}(\mathcal{A}, \{q\}) \neq \varnothing$. 89 ◀

*Remarque 90.* Cette notion n'est pas conciliable avec notre remarque 23. Il faut la considérer comme un artefact discursif destiné à mettre de côté les états inutiles pour $\Delta$.

**Définition 91 (Automate séparant).**

On dit que l'automate $\mathcal{A}$ sépare les classes de $E$ si pour tout état $q \in Q$, il existe un terme $s \in T(\Sigma)$ tel que

$$\mathscr{L}(\mathcal{A}, \{q\}) \subseteq [s]_E.$$ 91 ◀

*Remarque 92.* On n'a pas ici besoin de parler d'automate rempli : si $q \in Q$ est tel que $\mathscr{L}(\mathcal{A}, \{q\}) = \varnothing$, n'importe quel $s$ convient. La notion définie ci-dessus signifie simplement que l'automate $\mathcal{A}$ est « compatible » avec $E$. Notons que l'on n'impose qu'une inclusion : il est possible qu'un état ne reconnaisse pas la totalité d'une classe.

Voici la notion importante. Elle exprime le fait que le langage reconnu par chaque état $q$ de $\mathcal{A}$ admet un ancêtre commun $s_0$ pour la réécriture $R/E$.

**Définition 93 (Automate $R/E$-cohérent).**
On dit que $\mathcal{A}$ est $R/E$-cohérent si

(i) $\mathcal{A}^\sharp$ est rempli ;

(ii) $\mathcal{A}^\sharp$ sépare les classes de $E$ et

(iii) pour tout $q \in Q$, il existe $s_0 \in \mathscr{L}(\mathcal{A}^\sharp, \{q\})$ tel que $\mathscr{L}(\mathcal{A}, \{q\}) \subseteq (R/E)^*(\{s_0\})$. 93 ◀

*Remarque 94.* Dans le (iii) de la définition précédente, le $s_0 \in \mathscr{L}(\mathcal{A}^\sharp, \{q\})$ ne joue pas de rôle particulier : n'importe quel élément de $\mathscr{L}(\mathcal{A}^\sharp, \{q\})$ convient dès lors que l'un d'entre eux convient. En effet, à cause du point (ii), les éléments $\mathscr{L}(\mathcal{A}^\sharp, \{q\})$ sont équivalents modulo $E$, donc indistincts pour $(R/E)^*$. Pour cette raison, lorsque $\mathcal{A}$ est $R/E$-cohérent, on appelle représentant de l'état $q$ tout élément de $\mathscr{L}(\mathcal{A}^\sharp, \{q\})$.

Il faut remarquer que contrairement à l'idée que l'on pourrait avoir, il est plus facile d'être $R/E$-cohérent lorsque $R$ et $E$ sont « gros ».

**Lemme 95.**
Soit $R'$ un système de réécriture et $E'$ un ensemble d'équations tels que

$$R \subseteq R' \qquad \text{et} \qquad E \subseteq E'.$$

Si $\mathcal{A}$ est $R/E$-cohérent, alors $\mathcal{A}$ est aussi $R'/E$-, $R/E'$- et $R'/E'$-cohérent. 95 ■

*Démonstration.*
Le point (i) n'est pas affecté par $R$ et $E$. L'équivalence modulo $E$ implique l'équivalence modulo $E'$, donc chaque classe de $E$ est incluse dans une classe de $E'$. Ainsi, $\mathcal{A}^\sharp$ sépare les classes de $E'$, soit le point (ii).

Vérifions le point (iii). Pour tout terme $s_0$, on a $(R/E)^*(\{s_0\}) \subseteq (R'/E)^*(\{s_0\})$ car toute réécriture faite par $R$ peut être faite par $R'$. Ainsi, $\mathcal{A}$ est $R'/E$-cohérent. De même, $(R/E)^*(\{s_0\}) \subseteq (R/E')^*(\{s_0\})$, donc $\mathcal{A}$ est $R/E'$-cohérent.

Par composition, $\mathcal{A}$ est $R'/E'$-cohérent. □

*Remarque 96.* La $R/\varnothing$-cohérence est très difficile. En effet, si $E = \varnothing$, la relation $\equiv_E$ est l'égalité syntaxique. Un automate séparant ces classes doit ne reconnaitre qu'au plus un terme en chacun de ses états !

La $R/E$-cohérence est préservée par la complétion équationnelle, ce qui permettra de ne faire des hypothèses que sur l'automate initial.

**Théorème 97.**
Si $\mathcal{A}$ est $R/E$-cohérent, alors $\mathcal{C}_R(\mathcal{A})$ aussi et $\mathcal{S}^!_E(\mathcal{A})$ également. Ainsi, $\mathcal{CE}_{R,E}(\mathcal{A})$ est $R/E$-cohérent. 97 ■

## 4.4 Correction et précision

Nous allons à présent étudier la correction et la précision de la complétion équationnelle *dans le cas où elle aboutit à un point fixe*. En effet, ce procédé itératif ne termine pas en général, comme nous l'avons dit en remarque 86.

Dans toute la suite, nous considérons un langage initial $L_0 = \mathscr{L}(\mathcal{A}_0, Q_f)$, où $\mathcal{A}_0 = (\Sigma, Q, \delta, \varnothing)$ est un automate sans epsilon-transition et $Q_f$ une partie de $Q$. Nous considérons également un système de réécriture $R$ fini et linéaire à gauche et un ensemble fini d'équations linéaires $E$. Ces restrictions de linéarité s'expliquent par le fait que le défaut de linéarité empêche le bon fonctionnement de la complétion en présence d'un automate non déterministe.

*Exemple 98.*

Nous empruntons cet exemple à [Gen09, sect. 4.4.1]. Considérons $R = \{{}_2f(x,x) \to {}_1g(x)\}$ et $\mathcal{A}_0$ avec les transitions ${}_0c \to q_0$, ${}_0c \to q_1$ et ${}_2f(q_0,q_1) \to q_3$. Nous voyons bien que $\mathscr{L}(\mathcal{A}_0, \{q_3\}) = \{{}_2f({}_0c, {}_0c)\}$ et que $R^*(\mathscr{L}(\mathcal{A}_0, \{q_3\})) = \{{}_2f({}_0c, {}_0c), {}_1g({}_0c)\}$, mais l'algorithme de complétion ne peut pas découvrir la pseudo-paire critique issue de ${}_2f(q_0,q_1) \to q_3$, car il faudrait pour cela disposer d'un remplacement $\sigma$ associant à la variable $x$ à la fois l'état $q_0$ et l'état $q_1$.

La procédure de complétion s'arrête immédiatement, et l'automate point-fixe est $\mathcal{A}_0$ lui-même. Nous n'avons donc pas construit une sur-approximation du langage des termes accessibles. 98 ◁

Nous appliquons la procédure de complétion équationnelle, qui produit successivement les automates $\mathcal{A}_1 = \mathcal{CE}_{R,E}(\mathcal{A}_0)$, $\mathcal{A}_2 = \mathcal{CE}_{R,E}(\mathcal{A}_1)$, … Comme indiqué, nous supposons que ces étapes de complétion équationnelle terminent, c'est-à-dire qu'il existe un automate point-fixe $\mathcal{A}^!_{R,E}$.

Il y a un grand absent dans ce tableau : le langage des mauvais termes $L_{\text{bad}}$. Le lecteur ne doit pas s'en étonner. En effet, nous ne cherchons pas à évaluer la correction de la complétion équationnelle dans une situation concrète particulière, mais dans sa généralité. Nous dirons que la méthode est correcte si le fait que $\mathscr{L}(\mathcal{A}^!_{R,E}, Q_f) \cap L_{\text{bad}} = \varnothing$ entraine $R^*(L_0) \cap L_{\text{bad}} = \varnothing$, et ce quelque soit $L_{\text{bad}}$. Cela nous conduit à énoncer le théorème de correction comme suit.

**Théorème 99 (Correction de la complétion équationnelle).**

Soit $R$ un système de réécriture fini et linéaire à gauche, $E$ un ensemble fini d'équations linéaires, $\mathcal{A}_0$ un automate sans epsilon-transition avec états terminaux $Q_f$. Si la complétion équationnelle produit un point fixe $\mathcal{A}^!_{R,E}$, alors

$$\mathscr{L}(\mathcal{A}^!_{R,E}, Q_f) \supseteq R^*(\mathscr{L}(\mathcal{A}_0, Q_f)).$$

99 ■

La notion de $R/E$-cohérence a été introduite pour obtenir le résultat de précision suivant.

**Théorème 100 (Précision de la complétion équationnelle).**

Soit $R$ un système de réécriture fini et linéaire à gauche, $E$ un ensemble fini d'équations linéaires, $\mathcal{A}_0$ un automate sans epsilon-transition avec états terminaux $Q_f$ tels que la complétion équationnelle produit un point fixe $\mathcal{A}^!_{R,E}$. Si $\mathcal{A}_0$ est $R/E$-cohérent, alors

$$\mathscr{L}(\mathcal{A}^!_{R,E}, Q_f) \subseteq (R/E)^*(\mathscr{L}(\mathcal{A}_0, Q_f)).$$

100 ■

# 5 Interprétation abstraite

L'interprétation abstraite est une méthode d'élaboration d'analyses statiques inventée par Patrick Cousot dans [CC77]. Une présentation rapide est proposée par M. Cousot lui-même dans [Cou]. L'interprétation abstraite fait l'objet d'un chapitre dans [NNH05]. Nous avions déjà écrit une introduction à l'interprétation abstraite dans [Sal09], dont nous reprenons ici certains passages.

## 5.1 Présentation informelle

Considérons le programme représenté en figure 6. Le langage dans lequel il est écrit possède une sémantique standard qui donne la signification de chaque expression et de chaque instruction. Nous supposons que le langage ne traite que des entiers. Ainsi, la division par 2 d'un entier impair est interdite. Ici, nous devons donc nous assurer qu'au moment de réaliser la division de la ligne 10, la variable a contient une valeur paire, et cela dans toute exécution du programme.

```
1   var a,b,c,d,e;
2   input c;
3   input d;
4   b = 0;
5   a = 2 * d;
6   while (b < c) do {
7     a = a + 2;
8     b = b + 1;
9   }
10  e = a / 2;
11  output e;
```

Figure 6 : Un programme exemple

Bien entendu, il est aisé de le vérifier à la main en faisant une preuve. Mais voyons comment aborder ce problème par le biais de l'interprétation abstraite.

Dans le cas présent, l'état de la mémoire dans laquelle s'exécute le programme est entièrement déterminé par la donnée de la valeur de chacune des variables qui apparaissent dans le texte du programme. Pour des raisons de simplicité d'exposition, nous considérons que toute variable est initialisée à zéro dès sa déclaration. En prenant les variables dans l'ordre alphabétique, l'espace des environnements est donc identifiable au produit cartésien $\mathbb{Z}^5$ :

$$Env = \mathbb{Z}^5.$$

Nous voulons nous assurer qu'au moment d'exécuter l'instruction de la ligne 10, la première variable contient une valeur paire, c'est-à-dire que l'environnement courant est dans $2\mathbb{Z} \times \mathbb{Z}^4$. Nous ne pouvons pas faire cette analyse statique avec la sémantique standard, car nous ne pouvons parcourir toutes les valeurs possibles pour c et d.

L'interprétation abstraite consiste à définir une sémantique plus grossière de ce langage et à exécuter le programme considéré en utilisant cette sémantique grossière, appelée sémantique abstraite. Ici, nous allons considérer un espace de valeurs abstrait constitué des 3 symboles P, I et $\top$ (qui dénoteront les valeurs paires, impaires, et celles dont la parité est inconnue). Il est fréquent de signaler par un dièse ce qui relève du domaine abstrait. L'espace des environnements abstraits est donc

$$Env^\sharp = \{\mathsf{P}, \mathsf{I}, \top\}^5.$$

Dans cette sémantique abstraite, l'instruction **input** c; affecte simplement la valeur $\top$ à la variable c. Les constantes 0 et 2 représentent la valeur P, la constante 1 représente la valeur I. Nous devons maintenant donner la sémantique abstraite des opérateurs + et $*$, ce que nous faisons en figure 7. Notons que ces tables ont été dressées à la main, en utilisant des connaissances d'arithmétique élémentaire.

La sémantique abstraite des opérateurs nous permet de déterminer que juste après la ligne 5, la variable a contient la valeur abstraite P.

| $[\![+]\!]^\sharp$ | P | I | $\top$ |
|---|---|---|---|
| P | P | I | $\top$ |
| I | I | P | $\top$ |
| $\top$ | $\top$ | $\top$ | $\top$ |

| $[\![*]\!]^\sharp$ | P | I | $\top$ |
|---|---|---|---|
| P | P | P | P |
| I | P | I | $\top$ |
| $\top$ | P | $\top$ | $\top$ |

FIGURE 7 : Sémantique abstraite des opérateurs

Le point délicat est bien sûr la boucle : nous ne savons pas si elle va être empruntée, ni combien de fois. Nous devons donc envisager plusieurs cas possibles.

Supposons que la boucle est empruntée une première fois : à la ligne 7, l'expression `a + 2` vaut P, et donc l'affectation donne à `a` la valeur P (qu'elle avait déjà) ; à la ligne 8, l'expression `b + 1` vaut I (car `b` vaut P depuis la ligne 4), et l'affectation courante donne donc à `b` la valeur I.

Considérons maintenant un passage quelconque dans la boucle. Par propagation, nous pouvons affirmer que la valeur de `a` est toujours P (elle n'est pas modifiée par le passage dans le boucle). En revanche, selon que nous sommes, par exemple, dans le premier ou dans le deuxième passage de la boucle, nous ne savons pas si `b` contient la valeur P ou la valeur I. Nous n'avons pas d'autre choix que de considérer cette valeur comme inconnue : $\top$.

Mais qu'importe : au début de la ligne 10, nous ne savons pas si la boucle a été exécutée ni combien de fois, donc nous n'avons pour `b` pas de valeur plus précise que $\top$. En revanche, nous avons vu que quelque soit la situation, `a` contient la valeur P. C'est précisément ce que nous voulions vérifier.

Notons que si la ligne 5 du programme avait été ainsi rédigée : `a = d + d;`, notre analyse aurait échoué, car elle n'aurait pas pu déterminer que cette instruction affecte nécessairement une valeur paire à `a`. En effet, à la ligne 5, `d` contient la valeur abstraite $\top$, si bien que la sémantique abstraite de + donne le résultat $\top$.

## 5.2 Cadre formel confortable

### 5.2.1 Univers et connexions de Galois

Les valeurs abstraites envisagées pour une analyse sont appelées propriétés abstraites, leur ensemble forme l'univers abstrait. De même que dans la section précédente, nous considérions $\top$ comme moins précis que P, nous avons besoin d'une relation d'ordre sur l'univers abstrait pour dénoter la précision relative des propriétés abstraites. Étant donné deux propriétés $p$ et $q$, nous souhaitons que l'univers abstrait possède une propriété qui puisse tenir le rôle de « $p$ ou $q$ ». Cette propriété doit être moins précise que $p$ (au-dessus de $p$, pour notre relation d'ordre), moins précise que $q$ (au-dessus de $q$), mais plus précise que toute autre propriété qui est à la fois au-dessus de $p$ et de $q$.

Finalement, il est souhaitable que toute paire de propriétés $\{p,q\}$ possède une borne supérieure pour la relation d'ordre. De même, il faut qu'elle possède une borne inférieure, qui joue le rôle de « $p$ et $q$ ». Ces considérations nous conduisent à imposer que l'univers abstrait soit un treillis.

**Définition 101 (Treillis).**
Soit $(T,\sqsubseteq)$ un ensemble ordonné. On dit que $(T,\sqsubseteq)$ est un treillis si pour tous $p,q \in T$, l'ensemble $\{p,q\}$ admet une borne supérieure et une borne inférieure pour $\sqsubseteq$. Autrement dit, s'il existe $m,M \in T$ tels que

$$\begin{array}{lll} m \sqsubseteq p, & m \sqsubseteq q, & \forall x \in T, (x \sqsubseteq p \text{ et } x \sqsubseteq q) \Rightarrow x \sqsubseteq m, \\ M \sqsupseteq p, & M \sqsupseteq q, & \forall x \in T, (x \sqsupseteq p \text{ et } x \sqsupseteq q) \Rightarrow x \sqsupseteq M. \end{array}$$ 101 ◀

Dans un treillis, la borne supérieure (respectivement inférieure) d'une paire est unique, ce qui nous permet d'introduire l'opérateur borne supérieure (respectivement inférieure).

**Définition 102 (Opérateur borne supérieure, inférieure).**
Soit $(T, \sqsubseteq)$ un treillis. On note $\sqcup$ l'opérateur infixe qui à deux éléments $p, q \in T$ associe $p \sqcup q$, la borne supérieure de $\{p, q\}$. De même, on note $\sqcap$ l'opérateur borne inférieure. 102 ◀

Dans l'univers concret, on identifie chaque propriété (concrète) à l'ensemble des environnements qui vérifient cette propriété. Dans l'exemple de la section précédente, $2\mathbb{Z} \times \mathbb{Z}^4$ est une propriété concrète. Plus généralement, les propriétés concrètes sont des parties de $Env$. Ainsi, l'univers concret est l'ensemble des parties de $Env$, $\mathcal{P}(Env)$.

*Remarque 103.* Ainsi défini, un univers concret est un treillis : il suffit de le munir de l'inclusion. L'opérateur borne supérieure est l'union $\cup$, l'opérateur borne inférieure est l'intersection $\cap$.

En fait, des raisons théoriques que nous n'explicitons pas, mais qui sont indiquées dans les références que nous mentionnons, nous conduisent à imposer que les treillis considérés soient complets, c'est-à-dire que n'importe quelle partie non vide admette une borne inférieure, et pas seulement les parties à deux éléments.

**Définition 104 (Treillis complet).**
Soit $(T, \sqsubseteq)$ un treillis. On dit qu'il est complet si toute partie non vide $X$ de $T$ admet une borne inférieure et une borne supérieure, notées respectivement $\bigsqcap X$ et $\bigsqcup X$. 104 ◀

*Remarque 105.* Un treillis complet non vide $T$ admet nécessairement un plus petit élément, noté

$$\bot = \bigsqcap T,$$

et un plus grand élément, noté

$$\top = \bigsqcup T.$$

Deux treillis complets étant pris comme univers, l'un concret $\mathcal{U}$, l'autre abstrait $\mathcal{U}^\sharp$, il faut mettre en place une connexion entre eux, afin de pouvoir concrétiser les propriétés abstraites et abstraire les propriétés concrètes. On se donne à cette fin une fonction de concrétisation $\gamma : \mathcal{U}^\sharp \to \mathcal{U}$ et une fonction d'abstraction $\alpha : \mathcal{U} \to \mathcal{U}^\sharp$.

L'univers concret $\mathcal{U}$ est l'ensemble des parties $\mathcal{P}(Env)$ d'un certain ensemble d'environnements possibles, muni de la relation d'ordre de l'inclusion. C'est pourquoi on utilise des opérateurs « ronds » $(\cup, \subseteq)$ pour les objets concrets, et des opérateurs « carrés » $(\sqcup, \sqsubseteq)$ pour les objets abstraits.

La connexion entre $\mathcal{U}$ et $\mathcal{U}^\sharp$ ne peut pas être quelconque : elle doit respecter les relations d'ordre qui représentent la précision relative des différentes propriétés.

Dans l'univers concret, une petite propriété $P$ (au sens de l'inclusion) est plus fine qu'une propriété $Q$ qui la contient. Il est raisonnable de demander que, dans ce cas, le représentant de $P$ dans l'univers abstrait soit plus petit que celui de $Q$. Autrement dit, il est souhaitable que $\alpha$ soit croissante.

De même, une propriété abstraite fine doit se concrétiser en une propriété concrète fine. Ainsi, $\gamma$ aussi doit être croissante.

On souhaite que l'approximation effectuée soit correcte. Cela requiert notamment que, pour toute propriété concrète $P$, $\gamma \circ \alpha(P)$ ne doit pas contredire $P$. Ainsi, si un objet vérifie la propriété concrète $P$, il doit aussi vérifier $\gamma \circ \alpha(P)$. On doit donc avoir $\gamma \circ \alpha(P) \supseteq P$. Cette propriété, $\forall P \in \mathcal{U}, \gamma \circ \alpha(P) \supseteq P$, est appelée extensivité de $\gamma \circ \alpha$. On la note $\gamma \circ \alpha \dot{\supseteq} \mathrm{Id}_{\mathcal{U}}$.

On souhaite que $\alpha$ fournisse la meilleure approximation possible de chaque propriété concrète. Soit $y$ une propriété abstraite et $P = \gamma(y)$ sa concrétisation. Par construction, $y$ est une approximation possible de $P$, donc on doit avoir $\alpha(P) \sqsubseteq y$, *ie.* $\alpha \circ \gamma(y) \sqsubseteq y$. Cette propriété est appelée rétractivité de $\alpha \circ \gamma$.

Ces contraintes sont rassemblées dans la notion de connexion de Galois :

**Définition 106 (Connexion de Galois).**
On dit d'un couple $(\alpha, \gamma)$ formé d'une fonction d'abstraction et d'une fonction de concrétisation entre un univers concret $\mathcal{U}$ et un univers abstrait $\mathcal{U}^\sharp$ qu'il est une connexion de Galois si

(i) $\alpha$ est croissante,

(ii) $\gamma$ est croissante,

(iii) $\gamma \circ \alpha \dot{\supseteq} \mathrm{Id}_{\mathcal{U}}$ (extensivité) et

(iv) $\alpha \circ \gamma \dot{\sqsubseteq} \mathrm{Id}_{\mathcal{U}^\sharp}$ (rétractivité). 106 ◀

*Remarque 107.* La notion de connexion de Galois est exprimée de manière équivalente comme suit. Soit $(T_1, \leqslant_1)$ et $(T_2, \leqslant_2)$ deux treillis complets. Soit $\alpha : T_1 \longrightarrow T_2$ et $\gamma : T_2 \longrightarrow T_1$. On dit que $(\alpha, \gamma)$ est une connexion de Galois entre $T_1$ et $T_2$ si pour tout $x_1 \in T_1$ et pour tout $x_2 \in T_2$, on a $\alpha(x_1) \leqslant_2 x_2 \Leftrightarrow x_1 \leqslant_1 \gamma(x_2)$.

La notion de connexion de Galois est assez contraignante : la définition de l'une des deux fonctions en jeu impose l'autre, comme l'énonce le lemme suivant.

**Lemme 108.**
Si $(\alpha, \gamma)$ forme une connexion de Galois entre $\mathcal{U}$ et $\mathcal{U}^\sharp$, alors pour tout $P \in \mathcal{U}$ (respectivement pour tout $p \in \mathcal{U}^\sharp$), on a

$$\alpha(P) = \bigsqcap \left\{ x \in \mathcal{U}^\sharp \;\middle|\; P \subseteq \gamma(x) \right\}$$

et

$$\gamma(p) = \bigcup \{ X \in \mathcal{U} \mid \alpha(X) \sqsubseteq p \}.$$ 108 ■

### 5.2.2 Opérateurs et leurs abstractions

Après avoir abstrait les objets manipulés par les programmes, il faut abstraire les manipulations. Chaque étape d'exécution d'un programme transforme l'environnement courant. On s'intéresse donc aux fonctions de $\mathcal{U}$ dans $\mathcal{U}$, qu'on appelle « opérateurs ». Pour chaque opérateur concret agissant sur les environnements, il faut construire un opérateur abstrait agissant sur les abstractions de ces environnements.

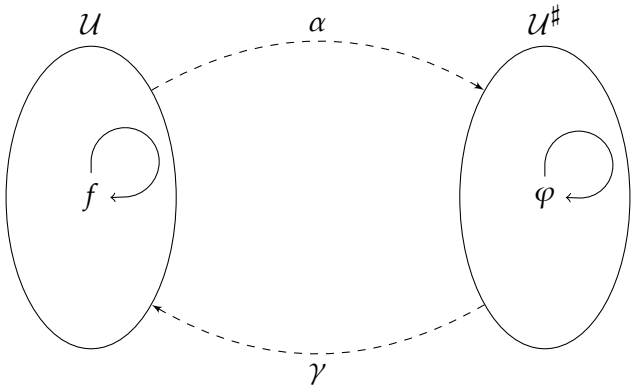



Figure 8 : Domaines, opérateurs concrets et abstraits

Soit $f : \mathcal{U} \longrightarrow \mathcal{U}$ un opérateur concret. On souhaite construire un opérateur abstrait $\varphi : \mathcal{U}^\sharp \longrightarrow \mathcal{U}^\sharp$ tel que pour toute approximation correcte $x$ de $P$, $\varphi(x)$ soit une approximation correcte de $f(P)$.

**Définition 109 (Approximation correcte).**
Soit $f$ un opérateur concret et $\varphi$ un opérateur abstrait. On dit que $\varphi$ est une approximation correcte de $f$ si pour tout $P \in \mathcal{U}$ et pour tout $x \in \mathcal{U}^\sharp$ tel que $\alpha(P) \sqsubseteq x$, on a $\alpha(f(P)) \sqsubseteq \varphi(x)$ (ou, de manière équivalente : si $P \subseteq \gamma(x)$ alors $f(P) \subseteq \gamma(\varphi(x))$). 109 ◀

On dispose d'un théorème de caractérisation des approximations correctes :

**Théorème 110.**
Soit $f$ un opérateur concret croissant et $\varphi$ un opérateur abstrait croissant. On a équivalence entre

(i) $\varphi$ est une approximation correcte de $f$,

(ii) $\alpha \circ f \mathrel{\dot{\sqsubseteq}} \varphi \circ \alpha$,

(iii) $\alpha \circ f \circ \gamma \mathrel{\dot{\sqsubseteq}} \varphi$ et

(iv) $f \circ \gamma \mathrel{\dot{\subseteq}} \gamma \circ \varphi$. 110 ■

**Corolaire 111.**
Soit $f$ un opérateur concret croissant. $\alpha \circ f \circ \gamma$ est une approximation correcte de $f$. De plus, c'est la meilleure approximation croissante de $f$.

En pratique, les opérateurs concrets que nous manipulerons seront croissants. Il est donc intéressant de définir la notion d'approximation optimale en se fondant sur le corolaire précédent.

**Définition 112 (Approximation optimale).**
Soit $f$ un opérateur concret croissant. On dit que $\alpha \circ f \circ \gamma$ est l'approximation optimale de $f$, et on la note $f^{\mathcal{U}^\sharp}$. 112 ◀

# 6 Complétion en interprétation abstraite

Après avoir présenté la complétion équationnelle et l'interprétation abstraite, nous allons tenter d'exprimer la première dans le vocabulaire de la seconde.

## 6.1 Préliminaires

Ces préliminaires ont pour but d'étudier l'action de la simplification équationnelle sur le caractère déterministe des automates.

Commençons par remarquer que le déterminisme des automates est compatible avec l'équivalence à renommage près. Ce résultat nous permettra de parler du déterminisme d'automates qui ne sont définis qu'à renommage bijectif près, ce qui est le cas des $\mathcal{S}^!_E(\mathcal{A})$.

**Lemme 113.**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \varnothing)$ un automate déterministe, $\alpha : Q \to Q'$ un renommage bijectif et $\mathcal{A}' = \mathcal{A}\alpha$. Alors $\mathcal{A}'$ est déterministe. 113 ■

*Démonstration.*
Le résultat est clair si $Q$ et $Q'$ sont disjoints. Dans le cas contraire, introduisons un ensemble $Q''$ disjoint de $Q$ et $Q'$, en bijection avec $Q$, et $\alpha_1 : Q \to Q''$ un renommage bijectif quelconque. Notons $\alpha_2 = \alpha \circ \alpha_1^{-1}$. On a un renommage bijectif $\alpha_2 : Q'' \to Q$ tel que $\alpha = \alpha_2 \circ \alpha_1$. Les ensembles de départ et d'arrivée étant disjoints, on obtient que $\mathcal{A}\alpha_1$ est déterministe, puis que $(\mathcal{A}\alpha_1)\alpha_2$ est déterministe, c'est-à-dire que $\mathcal{A}'$ est déterministe. □

Le déterminisme est créé par un certain type d'équations : les tautologies.

**Définition 114 (Tautologie).**
Soit $(\Sigma, \mathcal{X})$ une signature. On appelle tautologie sur $(\Sigma, \mathcal{X})$ toute équation de la forme

$$f(x_1, \ldots, x_k) = f(x_1, \ldots x_k)$$

où $k \in \mathbb{N}$, $f \in \Sigma_k$ et $x_1, \ldots, x_k$ sont des variables *deux à deux distinctes*. 114 ◀

*Remarque 115.* Les tautologies sont des équations linéaires.

*Remarque 116.* Les signatures considérées étant finies, on peut restreindre l'usage des variables de manière à n'obtenir qu'un ensemble fini de tautologies. C'est ce que nous faisons pour la suite.

**Lemme 117.**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \varnothing)$ un automate sans epsilon-transition et $E$ un ensemble fini d'équations (linéaires) sur $(\Sigma, \mathcal{X})$. On suppose que $E$ possède toutes les tautologies sur $(\Sigma, \mathcal{X})$. Alors $\mathcal{S}_E^!(\mathcal{A})$ est déterministe. 117 ■

*Démonstration.*
Notons $\mathcal{A}' = \mathcal{S}_E^!(\mathcal{A})$ et montrons par induction sur $t \in T(\Sigma)$ que si $t \xrightarrow[\mathcal{A}']{*} q$ et $t \xrightarrow[\mathcal{A}']{*} q'$ avec $q, q' \in Q$, alors $q = q'$.

Si $t \in \Sigma_0$, comme $\mathcal{A}'$ est sans epsilon-transition, on a $t \underset{\mathcal{A}'}{\rightarrowtail} q$ et $t \underset{\mathcal{A}'}{\rightarrowtail} q'$. Notant $\ell = r = t$ et $\sigma$ le remplacement vide, on a une situation d'application de l'équation $(\ell = t) \in E$ dès lors que $q \neq q'$. Or, par construction, $\mathcal{A}'$ est une forme normale pour la relation de simplification par $E$, donc $q = q'$.

Si $t$ est de la forme $f(t_1, \ldots t_k)$ avec $k \in \mathbb{N}$, $f \in \Sigma_k$ et $t_1, \ldots, t_k \in T(\Sigma)$, alors il existe $q_1, \ldots, q_n \in Q$ tels que pour tout $i \in [\![1\,;k]\!]$, $t_i \xrightarrow[\mathcal{A}']{*} q_i$. Par hypothèse d'induction, chacun des $q_i$ est unique, de sorte que, $\mathcal{A}'$ étant sans epsilon-transition, on a $f(q_1, \ldots, q_k) \underset{\mathcal{A}'}{\rightarrowtail} q$ et $f(q_1, \ldots, q_k) \underset{\mathcal{A}'}{\rightarrowtail} q'$. En utilisant la tautologie $f(x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$ et le remplacement $\sigma : x_i \mapsto q_i$, on obtient une situation d'application d'une équation de $E$ dès lors que $q \neq q'$. L'automate $\mathcal{A}'$ étant une forme normale pour la relation de simplification, on a $q = q'$. □

*Remarque 118.* Ainsi, la « simplification par les tautologies » rend l'automate considéré déterministe. Bien entendu, cette opération n'est pas la déterminisation classique des automates : dans notre cas, une sur-approximation est effectuée. En fait, l'opération que nous avons envisagée ici est analogue à la fusion pour déterminisation que l'on trouve dans certains algorithmes d'apprentissage automatique utilisant des automates (de mots), comme RPNI [OG92].

## 6.2 Première étude

### 6.2.1 Univers et connexion

Nous nous plaçons dans la situation du théorème 99 : nous considérons $R$ un système de réécriture fini et linéaire à gauche, $E$ un ensemble fini d'équations linéaires, $\mathcal{A}_0$ un automate sans epsilon-transition avec états terminaux $Q_f$ et nous supposons que la complétion équationnelle produit un point fixe $\mathcal{A}_{R,E}^!$. Nous notons

$$\mathcal{A}_{\text{fix}} = \mathcal{A}_{R,E}^! = (\Sigma, Q, \delta, \delta_\varepsilon)$$

et

$$\mathcal{A}_{\text{fix}}^{\not\varepsilon} = \mathcal{A}_{R,E}^{!\,\not\varepsilon} = (\Sigma, Q, \delta, \varnothing).$$

Rappelons que le système de réécriture $R$ modélise un programme et que les termes modélisent les environnements. Ainsi, l'univers concret que nous considérons est

**Définition 119.**

$$\mathcal{U} = \mathcal{P}(T(\Sigma)) \tag{119}$$

et les opérateurs de la définition 39 s'interprètent naturellement comme des opérateurs concrets au sens de l'interprétation abstraite.

**Définition 120.**

$$R : \begin{array}{l} \mathcal{U} \longrightarrow \mathcal{U} \\ L \longmapsto \{t \in T(\Sigma) \mid \exists s \in L, s \to_R t\} \end{array},$$

$$R^* : \begin{array}{l} \mathcal{U} \longrightarrow \mathcal{U} \\ L \longmapsto \left\{t \in T(\Sigma) \;\middle|\; \exists s \in L, s \to_R^* t\right\} \end{array}.$$

120

Déterminons quel est le domaine abstrait envisagé au travers de la complétion équationnelle. Nous procédons à une abstraction en ne considérant que des langages réguliers : une première idée de domaine abstrait serait donc l'ensemble des langages réguliers. Alternativement, nous pourrions considérer l'ensemble des automates qui reconnaissent ces langages.

Mais en fait, les seuls langages réguliers auxquels nous avons accès sont ceux reconnus par chacun des états de l'automate $\mathcal{A}_{\text{fix}}$. Le domaine abstrait que nous retenons est donc

**Définition 121.**

$$\mathcal{U}^\sharp = \mathcal{P}(Q). \tag{121}$$

*Remarque 122.* Il est suffisant de ne considérer que les états de $Q$ qui apparaissent dans $\Delta$, ce que nous faisons. Ainsi, notre domaine abstrait est fini.

Ainsi, chacun des deux univers que nous considérons est un ensemble de parties. Bien entendu, nous munissons chacun de ces deux ensembles de la relation d'inclusion. Pour cette raison, nous allons utiliser les symboles « ronds » ($\subseteq$, $\cap$), y compris pour l'univers abstrait (nous abandonnons les symboles « carrés » de la section 5). Le fait d'utiliser des univers qui sont des ensembles de parties munis de la relation d'inclusion fournit le lemme suivant gratuitement.

**Lemme 123.**
Les univers concret $\mathcal{U}$ et abstrait $\mathcal{U}^\sharp$ considérés sont des treillis complets. 123 ■

Une fonction de concrétisation évidente vient à l'esprit : une partie de $Q$ se concrétise en le langage qu'elle permet de reconnaitre. Toutefois, nous nous restreignons à la reconnaissance sans epsilon-transition, car ces dernières jouent le rôle des étapes de réécriture.

**Définition 124 ($\gamma$).**

$$\gamma : \begin{array}{l} \mathcal{U}^\sharp \longrightarrow \mathcal{U} \\ Q_f \longmapsto \mathscr{L}(\mathcal{A}^\sharp_{\text{fix}}, Q_f) \end{array}. \tag{124}$$

Si nous voulons espérer obtenir une connexion de Galois, la fonction d'abstraction est alors imposée par le lemme 108. Pour un langage $L \in \mathcal{U}$, nous devons considérer toutes les abstractions possibles compatibles avec notre choix de $\gamma$, c'est-à-dire l'ensemble $\mathscr{Q}_L \in \mathcal{P}(\mathcal{P}(Q))$ défini ci-dessous puis choisir la plus petite de ces abstractions possibles, c'est-à-dire l'intersection de cet ensemble.

**Définition 125 ($\mathscr{Q}_L$).**
Nous définissons en toute généralité, pour un automate $\mathcal{A}$ donné,

$$\mathscr{Q}_L = \{X \in \mathcal{P}(Q) \mid L \subseteq \mathscr{L}(\mathcal{A}, X)\}. \qquad 125 \blacktriangleleft$$

**Définition 126 ($\alpha$).**
En considérant l'automate $\mathcal{A}^{\not\epsilon}_{\text{fix}}$, nous définissons

$$\alpha : \begin{array}{ccc} \mathcal{U} & \longrightarrow & \mathcal{U}^\sharp \\ L & \longmapsto & \bigcap \mathscr{Q}_L \end{array}. \qquad 126 \blacktriangleleft$$

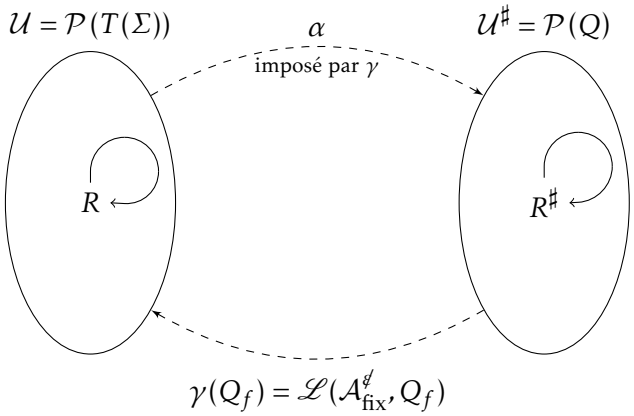



Figure 9 : Domaines, connexion et opérateurs de la complétion équationnelle

Il reste à vérifier que $(\alpha, \gamma)$ réalise une connexion de Galois. Nous devons établir les quatre points de la définition 106. Deux d'entre eux ne posent pas de problème.

**Lemme 127.**
Les fonctions $\alpha$ et $\gamma$ considérées vérifient :

(ii) $\gamma$ est croissante et

(iv) $\alpha \circ \gamma$ est rétractive ($\alpha \circ \gamma \dot{\subseteq} \mathrm{Id}_{\mathcal{U}^\sharp}$). $\qquad 127 \blacksquare$

*Démonstration.*
Le point (ii) est évident.

Pour le point (iv), soit $Q_f \in \mathcal{U}^\sharp$ et $L = \gamma(Q_f)$. On a $L \subseteq \mathscr{L}(\mathcal{A}^{\not\epsilon}_{\text{fix}}, Q_f)$ (il y a égalité), donc, par construction de $\alpha$, $\alpha(L) \subseteq Q_f$. Ainsi, $\alpha \circ \gamma(Q_f) \subseteq Q_f$. □

Les deux autres points ne sont pas vrais en général.

*Exemple 128.*
Si les transitions de $\mathcal{A}^{\not\epsilon}_{\text{fix}}$ sont ${}_0c \to q_1$ et ${}_0c \to q_2$, alors

$$\mathscr{Q}_{\{{}_0c\}} = \{\{q_1\}, \{q_2\}, \{q_1, q_2\}\},$$

si bien que $\alpha(\{{}_0c\}) = \varnothing$. Or $\gamma(\varnothing) = \varnothing$, donc $\gamma \circ \alpha(\{{}_0c\}) = \varnothing$, ce qui viole l'extensivité de $\gamma \circ \alpha$. $\qquad 128 \blacktriangleleft$

Pour la clarté du discours, nous donnons immédiatement une condition suffisante pour que $(\alpha, \gamma)$ soit une connexion de Galois : le déterminisme et l'universalité de $\mathcal{A}^{\not\epsilon}_{\text{fix}}$. La nécessité de cette hypothèse sera discutée dans la section 6.2.3.

**Lemme 129.**
Si $\mathcal{A}^{\not\varepsilon}_{\text{fix}}$ est déterministe et universel, alors

(i) $\alpha$ est croissante et

(iii) $\gamma \circ \alpha$ est extensive ($\gamma \circ \alpha \dot{\supseteq} \mathrm{Id}_{\mathcal{U}}$). 129 ■

La démonstration de ce lemme repose sur le résultat suivant. [5]

**Lemme 130.**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \varnothing)$ un automate déterministe et universel. Pour tout langage $L \subseteq T(\Sigma)$, l'ensemble

$$\mathcal{Q}_L = \{X \in \mathcal{P}(Q) \mid L \subseteq \mathscr{L}(\mathcal{A}, X)\}$$

est non vide et stable par intersection. 130 ■

*Démonstration du lemme 130.*
Soit $L \subseteq T(\Sigma)$. Par universalité de $\mathcal{A}$, l'ensemble $\mathcal{Q}_L$ est non vide. Si $L = \varnothing$, alors $\mathcal{Q}_L = \mathcal{P}(Q)$, qui est stable par intersection.

Supposons $L \neq \varnothing$. L'ensemble $\mathcal{Q}_L$ est alors fini : il suffit de vérifier la stabilité par intersection de deux éléments, c'est-à-dire $\forall X, Y \in \mathcal{Q}_L$, $X \cap Y \in \mathcal{Q}_L$. Procédons par l'absurde : soit $X, Y \subseteq Q$ tels que $L \subseteq \mathscr{L}(\mathcal{A}, X)$, $L \subseteq \mathscr{L}(\mathcal{A}, Y)$ mais $L \not\subseteq \mathscr{L}(\mathcal{A}, X \cap Y)$. Il existe donc $t \in L$, $q_X \in X \setminus Y$ et $q_Y \in Y \setminus X$ tels que $t \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q_X$ et $t \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q_Y$. Mais nécessairement, $q_X \neq q_Y$, ce qui contredit le déterminisme de $\mathcal{A}$. Dém. de 130 □

*Démonstration du lemme 129.*
Le lemme 130 a pour conséquence que pour tout $L \in \mathcal{U}$, $\alpha(L) \in \mathcal{Q}_L$.

Pour le point (i), soit $L \subseteq L' \in \mathcal{U}$. Par le lemme 130, $\alpha(L')$ vérifie $\mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, \alpha(L')) \supseteq L'$. Par transitivité de l'inclusion, $\mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, \alpha(L')) \supseteq L$, c'est-à-dire que $\alpha(L') \in \mathcal{Q}_L$. Par définition de $\alpha$ et par propriétés de l'intersection, $\alpha(L) \subseteq \alpha(L')$.

Pour le point (iii), soit $L \in \mathcal{U}$. Par le lemme 130, on a $\mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, \alpha(L)) \supseteq L$, c'est-à-dire que $\gamma \circ \alpha(L) \supseteq L$. Dém. de 129 □

Les résultats précédents peuvent être résumés comme suit.

**Théorème 131.**
Si $\mathcal{A}^{\not\varepsilon}_{\text{fix}}$ est déterministe et universel, alors $(\alpha, \gamma)$ réalise une connexion de Galois entre $\mathcal{U}$ et $\mathcal{U}^\sharp$. 131 ■

Ce théorème, combiné avec le lemme 117, permet d'énoncer une condition suffisante en termes de complétion équationnelle :

**Théorème 132.**
Si l'ensemble $E$ d'équations considéré pour la complétion équationnelle possède toutes les tautologies sur $(\Sigma, \mathcal{X})$, alors $(\alpha, \gamma)$ réalise une connexion de Galois entre $\mathcal{U}$ et $\mathcal{U}^\sharp$. 132 ■

*Démonstration.*
Considérons la dernière étape de complétion équationnelle. D'un automate $\mathcal{A}$ obtenu précédemment, la complétion exacte produit un automate $\mathcal{A}' = \mathcal{C}_R(\mathcal{A})$, puis il est procédé à la simplification équationnelle : $\mathcal{A}_{\text{fix}} = \mathcal{S}^!_E(\mathcal{A}')$ et $\mathcal{A}^{\not\varepsilon}_{\text{fix}} = \mathcal{S}^!_E\left(\mathcal{A}'^{\not\varepsilon}\right)$. Comme $\mathcal{A}'^{\not\varepsilon}$ est sans epsilon-transition et que $E$ est supposé posséder toutes les tautologies, par le lemme 117, $\mathcal{A}'^{\not\varepsilon}_{\text{fix}}$ est déterministe. Il suffit d'appliquer le théorème 131. □

5. Le connaisseur de l'interprétation abstraite reconnaitra dans la propriété étudiée la traduction dans notre cadre du fait (général pour les connexions de Galois) que $\gamma$ doit être un morphisme d'intersections.

### 6.2.2 Opérateur abstrait : difficulté

Nous nous plaçons dans la situation du théorème 132.

Nous avons écrit plusieurs fois que dans la complétion équationnelle, les epsilon-transitions jouent le rôle des étapes de réécriture. Il parait donc raisonnable de définir $R^\sharp$ à l'aide des epsilon-prédecesseurs dans $\mathcal{A}_{\text{fix}}$. Nous utilisons les prédecesseurs et non les successeurs car dans la figure de complétion d'une paire critique (figure 3), le sens de parcours du côté automate est inversé par rapport au système de réécriture. Ainsi, nous définissons

**Définition 133 ($R^\sharp$).**

$$R^\sharp : \begin{array}{ccc} \mathcal{U}^\sharp & \longrightarrow & \mathcal{U}^\sharp \\ X & \longmapsto & \left\{ q \in Q \;\middle|\; \exists x \in X, q \underset{\mathcal{A}_{\text{fix}}}{\rightarrowtail} x \right\} \end{array}. \qquad 133 \blacktriangleleft$$

Remarquons que l'on peut définir cet opérateur « état par état » par

$$R^\sharp(x) = \left\{ q \in Q \;\middle|\; q \underset{\mathcal{A}_{\text{fix}}}{\rightarrowtail} x \right\}.$$

*Remarque 134.* La relation $\underset{\mathcal{A}_{\text{fix}}}{\rightarrowtail}$ n'impose pas en elle-même de n'utiliser que des epsilon-transitions. C'est le fait que les réductions envisagées débutent par un état qui met cette contrainte.

Nous devons vérifier que l'opérateur $R^\sharp$ proposé est une approximation correcte de $R$. Nous utilisons pour cela le théorème de caractérisation 110, et plus particulièrement son point (iv). Il faut donc montrer l'énoncé suivant.

**Énoncé 135.**
Dans la situation du théorème 132, l'opérateur $R^\sharp$ proposé est une approximation correcte de $R$ : pour tout $X \in \mathcal{U}^\sharp$,

$$R(\mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, X)) \subseteq \mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, R^\sharp(X)). \qquad 135 \blacksquare$$

Mais cet énoncé n'est pas vrai, comme nous l'expliquons maintenant. Pour cela, considérons $X = \{q\}$, $s \in \mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, X)$ et $t \in R(\{s\}) \subseteq R(\mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, X))$, c'est-à-dire la situation en traits pleins de la figure 10. Nous devons démontrer qu'il existe un prédécesseur $q'$ de $q$ tel que $t \in \mathscr{L}(\mathcal{A}^{\not\varepsilon}_{\text{fix}}, \{q'\})$, c'est-à-dire la partie en pointillés de cette même figure.

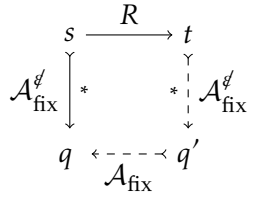



Figure 10 : Résultat à établir pour démontrer l'énoncé 135

Mais le fait que $\mathcal{A}_{\text{fix}}$ soit un point fixe de la complétion équationnelle ne nous fournit pas une telle situation. En effet, les situations possibles sont celles de la figure 11.

Remarquons que la figure 11(a) n'est qu'un cas particulier de la figure 11(b). La figure 11(a) est d'un abord plus favorable, mais elle ne se présente pas si l'étape de réécriture considérée a été effectuée dans un sous-terme. En effet, en un tel cas, la paire critique correspondant à l'application de la règle a


(a) Réécriture à la racine

(b) Réécriture dans un sous-terme

FIGURE 11 : Situations possibles dans $\mathcal{A}_{\text{fix}}$

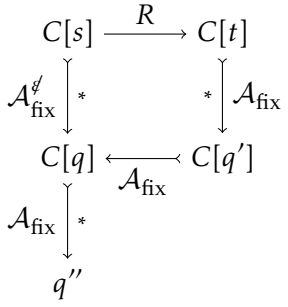



FIGURE 12 : Reconnaissance d'un terme avec réécriture en un sous-terme

produit une situation semblable à la figure 11(a), mais au niveau du sous terme. La reconnaissance du terme complet correspond à la figure 12, qui n'est autre qu'une version particulière de la figure 11(b).

Néanmoins, la distinction effectuée au sein de la figure 11 montre que nous ne devons pas considérer pour $R^\sharp(\{q\})$ seulement les prédécesseurs de $q$ mais aussi l'état $q$ lui-même.

Cela n'est toutefois pas suffisant : même dans le cas de la figure 11(a), il n'y a pas de raison en général que le terme $t$ soit reconnu en l'état $q'$ sans epsilon-transition. En effet, le critère utilisé en complétion exacte pour déterminer si une paire critique doit être jointe est « $r\sigma \overset{*}{\not\underset{\mathcal{A}}{\to}} q$ » et non « $r\sigma \overset{\not\varepsilon,*}{\not\underset{\mathcal{A}}{\to}} q$ » (voir la définition 68).

Ces réflexions nous incitent à envisager d'utiliser l'automate $\mathcal{A}_{\text{fix}}$ au lieu de l'automate $\mathcal{A}_{\text{fix}}^{\not\varepsilon}$ dans la définition de la fonction de concrétisation $\gamma$. Si nous faisons ce choix, l'énoncé 135 exprimant la correction de $R^\sharp$ devient :

**Énoncé 136.**
Pour tout $X \in \mathcal{U}^\sharp$,

$$R(\mathscr{L}(\mathcal{A}_{\text{fix}}, X)) \subseteq \mathscr{L}(\mathcal{A}_{\text{fix}}, R^\sharp(X)). \qquad 136 \blacksquare$$

Hélas, comme nous allons maintenant le montrer, l'hypothèse de déterminisme que nous avons faite est nécessaire pour obtenir une connexion de Galois. Or $\mathcal{A}_{\text{fix}}$ n'est pas déterministe : travailler avec cet automate impose d'abandonner le cadre confortable de la connexion de Galois. Nous aborderons cette possibilité dans la section 6.3.

### 6.2.3 Nécessité des hypothèses

Nous avons montré en section 6.2.1 que l'hypothèse de déterminisme de $\mathcal{A}_{\text{fix}}^{\not\varepsilon}$ est suffisante pour que les fonctions $\alpha$ et $\gamma$ considérées forment une connexion de Galois. Nous étudions maintenant la nécessité de cette hypothèse.

**Lemme 137.**
Si $\gamma \circ \alpha$ est extensive, alors pour tout langage $L \in \mathcal{U}$, $\mathcal{Q}_L$ est non vide et stable par intersection. 137 ■

*Démonstration.*
Soit $L \in \mathcal{U}$. On a $\gamma \circ \alpha(L) \supseteq L$, donc $\alpha(L) \in \mathcal{Q}_L$, qui est ainsi non vide. Si $L = \varnothing$, alors $\mathcal{Q}_L = \mathcal{P}(Q)$, qui est stable par intersection.

Supposons $L \neq \varnothing$ : $\mathcal{Q}_L$ est fini ; vérifions la stabilité par intersection de deux éléments. Soit donc $X, Y \in \mathcal{Q}_L$. Par définition, on a $\gamma(X) \supseteq L$ et $\gamma(Y) \supseteq L$. Par associativité de l'intersection, $X \cap Y \supseteq \bigcap \mathcal{Q}_L$, c'est-à-dire $X \cap Y \supseteq \alpha(L)$. Par croissance de $\gamma$ (qui est toujours vérifiée), $\gamma(X \cap Y) \supseteq \gamma \circ \alpha(L)$. Or, par hypothèse de ce lemme, $\gamma \circ \alpha$ est extensive, donc, par transitivité de l'inclusion, $\gamma(X \cap Y) \supseteq L$. Par définition, cela signifie que $X \cap Y \in \mathcal{Q}_L$. □

Le lemme suivant est énoncé pour un automate $\mathcal{A}$ quelconque. Dans le cadre de notre étude, il s'applique à l'automate $\mathcal{A}^{\not\varepsilon}_{\text{fix}}$.

**Lemme 138.**
Soit $\mathcal{A} = (\Sigma, Q, \delta, \delta_\varepsilon)$ un automate tel que pour tout $L \subseteq T(\Sigma)$, $\mathcal{Q}_L$ est stable par intersection. Alors $\mathcal{A}$ est effectivement déterministe, c'est-à-dire que toute configuration accessible de $\mathcal{A}$ a au plus un successeur par la relation $\underset{\mathcal{A}}{\rightarrowtail}$. 138 ■

*Démonstration.*
Soit $c$ une configuration accessible : il existe un terme $t \in T(\Sigma)$ tel que $t \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} c$. Supposons qu'il existe $q, q' \in Q$ tels que $c \underset{\mathcal{A}}{\rightarrowtail} q$ et $c \underset{\mathcal{A}}{\rightarrowtail} q'$ : montrons $q = q'$.

L'ensemble $\mathcal{Q}_{\{t\}}$ possède $\{q\}$ et $\{q'\}$, donc, comme $\mathcal{Q}_{\{t\}}$ est stable par intersection, $\{q\} \cap \{q'\} \in \mathcal{Q}_{\{t\}}$. Or $\varnothing \notin \mathcal{Q}_{\{t\}}$ (il faut au moins un état pour reconnaitre un terme !), donc $\{q\} \cap \{q'\} \neq \varnothing$, puis $q = q'$. □

La condition de déterminisme énoncée en section 6.2.1 est donc presque nécessaire : l'extensivité de $\gamma \circ \alpha$ l'impose, sauf éventuellement pour des configurations de l'automate qui ne reconnaissent aucun terme !

Si $\mathcal{A}^{\not\varepsilon}_{\text{fix}}$ permet de construire une connexion de Galois, ce ne sera donc pas le cas de $\mathcal{A}_{\text{fix}}$.

## 6.3 Deuxième étude

Comme annoncé, cette deuxième étude vise à construire des fonctions $\alpha$ et $\gamma$ qui permettent à un opérateur $R^\sharp$ défini comme suit d'être une bonne approximation de $R$. Nous abandonnons donc les définitions 124, 126 et 133. Les théorèmes 131 et 132 ne trouveront pas à s'appliquer. Nous conservons toutefois le domaine abstrait $\mathcal{U}^\sharp = \mathcal{P}(Q)$.

**Définition 139 ($R^\sharp$, nouvelle définition).**

$$R^\sharp : \begin{array}{c} \mathcal{U}^\sharp \longrightarrow \mathcal{U}^\sharp \\ X \longmapsto X \cup \left\{ q \in Q \;\middle|\; \exists x \in X, q \underset{\mathcal{A}_{\text{fix}}}{\rightarrowtail} x \right\} \end{array}.$$

139 ◀

Comme dit en section 6.2.2, nous définissons $\gamma$ de sorte que vérifier la correction de $R^\sharp$ revienne à démontrer l'énoncé 136.

**Définition 140 ($\gamma$, nouvelle définition).**

$$\gamma : \begin{array}{c} \mathcal{U}^\sharp \longrightarrow \mathcal{U} \\ X \longmapsto \mathscr{L}(\mathcal{A}_{\text{fix}}, X) \end{array}.$$

140 ◀

Il faut encore définir $\alpha$, ce qui n'est pas sans difficulté. Mais avant, étudions l'impact de notre nouvelle définition de $\gamma$ sur l'objectif fixé.

Mais que signifie le théorème 99 compte tenu de notre nouvelle définition de $\gamma$ ? Tout simplement que pour tout $X \in \mathcal{U}^\sharp$,

$$R^* \circ \gamma(X) \subseteq \gamma(X),$$

c'est-à-dire que $\mathrm{Id}_{\mathcal{U}^\sharp}$ est une approximation correcte de $R^*$, et donc à fortirori de $R$.

Cette approximation correcte est évidemment meilleure (plus précise et plus simple à calculer) que le $R^\sharp$ que nous proposions. Il ne parait donc pas pertinent de poursuivre l'étude de cet opérateur abstrait.

# 7 Conclusion

L'objectif du stage était d'acquérir une compréhension de la complétion équationnelle, de l'exposer d'une manière accessible (section 4) et de tenter d'étendre son efficacité. Au cours du stage, il est apparu souhaitable d'essayer d'exprimer la complétion équationnelle dans le vocabulaire de l'interprétation abstraite (section 6).

L'étude menée conduit à un résultat négatif. La première étude (section 6.2) a donné une condition nécessaire de déterminisme et d'universalité contraignante, si bien que nous ne voyons pas dans l'immédiat d'abstraction correcte pertinente de $R$. La deuxième étude (section 6.3), qui abandonne le cadre confortable des connexions de Galois, fournit une réponse triviale : l'identité est une abstraction correcte de $R^*$ et de $R$.

Toutefois, des pistes peuvent être explorées d'ici la fin du stage ou à plus lointaine échéance. L'opérateur abstrait de la première étude semble pouvoir être rapproché de la « réécriture à la racine » de [Boy10]. De plus, d'autres domaines abstraits peuvent éventuellement être envisagés. Par ailleurs, l'étude des tautologies (section 6.1) a permis d'entrevoir, comme dit en remarque 118, une analogie avec des procédés d'apprentissage automatique. Des stratégies d'apprentissage avec exemples et contre-exemples seraient à étudier, le rôle des exemples étant tenu par $L_0$ et celui des contre-exemples par $L_{\mathrm{bad}}$.

> Thues Arbeit ist ein Musterbeispiel dafür, wie weit die Zitationsrate und die bahnbrechende Originalität eines wissenschaftlichen Artikels auseinanderklaffen können. [...] Und so fällt sie nach allen gängigen Maßstäben der Bibliometrie in die Kategorie „bedeutungslos". Die Historie gibt dieser Einschätzung in gewissem Sinne auch recht, denn die Theorie der Termersetzung hat sich ja unabhängig von Thues Arbeit entwickelt. Dennoch wird jeder, der wissenschaftliche Originalität und Substanz zu schätzen weiß, Thues Arbeit von 1910 zu den besten Juwelen der Wissenschaft zählen. [6]
>
> *(Wolfgang Thomas [Tho10], à propos de [Thu10])*

6. « Cet article de Thue montre de manière exemplaire à quel point un article original sortant des sentiers battus peut rester très faiblement cité. [...] Ainsi, il est considéré comme „insignifiant" à l'aune de tous les critères bibliométriques actuels. L'Histoire donne en partie raison à ce jugement, car la théorie de la réécriture s'est effectivement développée indépendamment de l'œuvre de Thue. Néanmoins, quiconque sait apprécier la substance et l'originalité scientifiques comptera l'article écrit par Thue en 1910 au nombre des plus beaux joyaux de la science. » — Traduction personnelle

## Références


[BH08] Yohan Boichut et Pierre-Cyrille Héam. « A theoretical limit for the safety verification techniques with resular fix-point compuations ». Dans : *Information Processing Letters* 108 (2008), p. 1–2.

[BN98] Franz Baader et Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Boy10] Benoît Boyer. « Réécriture d'automates certifiée pour la vérification de modèles ». Thèse de doctorat. Université de Rennes 1, 2010.

[CC77] Patrick Cousot et Radhia Cousot. « Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints ». Dans : *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. POPL '77. Los Angeles, California : ACM, 1977, p. 238–252. URL : http://doi.acm.org/10.1145/512950.512973.

[Com+08] Hubert Comon et al. *Tree Automata Techniques and Applications*. 2008.

[Cou] Patrick Cousot. *Abstract Interpretation in a Nutshell*. URL : http://www.di.ens.fr/~cousot/AI/IntroAbsInt.html.

[GR10] Thomas Genet et Vlad Rusu. « Equational Tree Automata Completion ». Dans : *Journal of Symbolic Computation* 45 (2010), p. 574–597.

[GT95] Rémy Gilleron et Sophie Tison. « Regular Tree Languages and Rewrite Systems ». Dans : *Fundamenta informaticae* 24 (1995), p. 157–175.

[Gen09] Thomas Genet. « Reachability analysis of rewriting for Software verification ». Thèse d'habilitation à diriger des recherches. Université de Rennes 1, 2009.

[Gen97] Thomas Genet. « Decidable Approximations of Sets of Descendants and Sets of Normal Forms ». Dans : *Proc. 9th RTA Conf., Tsukuba (Japan), volume 1379 of LNCS*. Springer-Verlag, 1997, p. 151–165.

[KB70] D. E. Knuth et P. B. Bendix. « Simple word problems in universal algebras ». Dans : *Computational Problems in Abstract Algebra*. Sous la dir. de J. Leech. New York : Pergamon, 1970, p. 263–267.

[NNH05] Flemming Nielson, Hanne Riis Nielson et Chris Hankin. *Principles of Program Analysis*. Springer, 2005. Chap. 4, p. 209–280.

[OG92] Jose Oncina et Pedro Garcia. « Inferring regular languages in polynomial update time ». Dans : *Pattern Recognition and Image Analysis*. 1992, p. 49–61.

[Sal09] Yann Salmon. « Analyse d'alias par interprétation abstraite ». Rapport de stage. Université de Rennes 1, École normale supérieure de Cachan, 2009.

[Sal11] Yann Salmon. « Ensemble de termes accessibles en réécriture : aux frontières de la régularité ». Rapport de bibliographie. École normale supérieure de Cachan, Université de Rennes 1, jan. 2011.

[TS77] Axel Thue et Carl Ludwig Siegel. *Selected Mathematical Papers of Axel Thue*. Sous la dir. de Trygve Nagell et al. Universitetsforlaget, 1977.

[Tho10] Wolfgang Thomas. « „When nobody else dreamed of these things" — Axel Thue und die Termersetzung ». Dans : *Informatik-Spektrum* 33 (5 2010), p. 504–508. ISSN : 0170-6012. URL : http://dx.doi.org/10.1007/s00287-010-0468-9.

[Thu10] Axel Thue. « Die Lösung eines Spezialfalles eines generellen logischen Problems ». Dans : *Kristiania Videnskabs-Selskabets Skrifter, I. Matematisk-Naturvidenskabelig klasse* 8 (1910). Réimprimé dans [TS77].