# Remerciements

Je tiens à exprimer toute ma gratitude aux membres du jury :

- S. Krakowiak et F. Ouabdesselam qui me font l'honneur de présider ce jury.
- D. Bert et A. Lux, qui ont eu la gentillesse d'accepter mon invitation à juger ce travail.
- R. Echahed, mon maitre de stage, pour son aide précieuse, et les encouragements appréciés qu'il n'a cessé de me prodiguer.

Je remercie également tous les membres de l'équipe SCOP pour m'avoir accepté dans leur rang, et pour les discussions intéressantes que nous avons pu avoir, au fil des jours. Je tiens à remercier plus généralement tous les gens qui ont indirectement contribué à la réalisation de ce travail, en particulier ceux qui se languissent de me voir réapparaître.

# Contents

# Introduction

Les langages de programmation jouent un rôle déterminant dans les phases de conception, d'implantation et de maintenance des logiciels. Aussi, l'étude des langages de programmation occupe une place importante dans la recherche en informatique. Traditionnellement, on distingue plusieurs classes de langages "généralistes" :

- les langages impératifs, comme Fortran, C, Ada, ...
- les langages à objet, comme SmallTalk, Eiffel, ...
- les langages fonctionnels, comme Lisp, ML, Miranda, ...
- les langages logiques, ceux issus de la famille Prolog

Cette distinction est parfois factice, car de nombreux langages utilisent des notions provenant de plusieurs classes. On pensera par exemple à C++, à Lisp++, à LogLisp, ou à LIFE. Dans ce projet, nous nous intéressons à une classe hybride de langages, à savoir les langages logico-fonctionnels. Un langage logico-fonctionnel allie, dans un cadre unifié, à la fois la programmation logique et la programmation fonctionnelle.

## Le monde des langages logico-fonctionnels

Il y a plusieurs raisons qui plaident pour l'utilisation des langages logico-fonctionnels. D'abord leur expressivité : les fonctions et les prédicats y sont définies de manière naturelle. Ce qui n'est pas le cas des langages fonctionnels où les prédicats sont codés par leurs fonctions caractéristiques, ni des langages logiques où les fonctions sont définies par leurs graphes. D'autre part, les langages logico-fonctionnels bénéficient des meilleurs atouts des deux paradigmes de programmation logique et fonctionnel. Ainsi ils offrent l'évaluation déterministe des fonctions, trait propre des langages fonctionnels ; ils sont donc plus efficaces que les langages logiques qui, eux, nécessitent des évaluations non-déterministes (donc couteuses). Par ailleurs, les langages logico-fonctionnels offrent la possibilité d'utiliser des variables dites "logiques", ce qui est le fort des langages du même nom ; ils permettent par exemple de calculer l'ensemble des $x$ et $y$ tel que $x + y == 2$, ce qu'un programme fonctionnel n'est pas capable de faire[1].

Il existe plusieurs langages logico-fonctionnels dans la littérature, voir par exemple [1] et [2] pour une compilation significative des propositions qui ont été faites. On peut y distinguer cinq groupes de langages :

- les langages combinant le $\lambda$-calcul et les clauses de Horn (dont LogLisp, le tout premier langage logico-fonctionnel, né en 1981)

[1] On parle de bidirectionalité dans l'évaluation des arguments d'une fonction ou d'un prédicat.

- les langages basés sur la logique des clauses de Horn avec égalité (EqLog, LPG)
- les langages basés sur la logique conditionnelle (SLog, Babel, Curry)
- les langages logiques avec contraintes (Prolog 3)
- les autres langages (Uniform, Life, Oz)

Dans notre travail, nous nous intéressons aux langages du second groupe.

De manière subjective, la syntaxe de la logique des clauses de Horn avec égalité est certainement l'une des plus naturelles qui soit lorsqu'on programme avec un langage logico-fonctionnel. En effet, elle permet de définir à la fois des fonctions et des prédicats, et ce, dans un formalisme unifié. Nous présentons ci-après un exemple de programme. Nous y définissons les entiers naturels à partir de la constante `0` et de la fonction successeur `s` ; nous y définissons aussi l'opération d'addition `+`, le prédicat `Pair` qui réussit si son argument est un naturel pair, et le prédicat `Meme_parite` qui réussit si ses deux arguments ont la même parité :

```
0 + x == x
s(x) + y == s(x + y)

Pair(x + x)

Meme_parite(x, y) <== Pair(x + y)
```

D'une part, on constate que l'addition est effectivement prise comme une opération, et la parité, comme un prédicat. D'autre part, la définition de `Pair` et de `Meme_parite` se fait à l'aide d'une opération (`+`) que nous avons nous-mêmes définie. C'est une des caractéritiques de ce groupe de langages logico-fonctionnels. Elle leur confère une grande souplesse.

Pour permettre la comparaison avec un langage fonctionnel, nous reprenons le même exemple, écrit dans un langage algébrique (comme OBJ3 [3]). Celà donne :

```
0 + x == x
s(x) + y == s(x + y)

pair(0) == vrai
pair(s(0)) == faux
pair(s(s(x))) == pair(x)

meme_parite(x, y) == pair(x + y)
```

Nous sommes obligé de coder les relations `pair` et `meme_parite` par leurs fonctions caractéristiques.

Enfin, pour permettre la comparaison avec un langage logique, nous réécrivons le même programme à l'aide de clauses de Horn. Celà donne :

```
Add(0, x, x)
Add(s(x), y, s(z)) <== Add(x, y, z)

Pair(0)
Pair(s(s(x)) <== Pair(x)

Meme_parite(x, y) <== Add(x, y, z), Pair(z)
```

Dans ce cas, on est obligé de coder l'addition par son graphe !

Durant les dix dernières années, des efforts importants ont été entrepris afin de rendre opérationnels les langages logico-fonctionnels. Ainsi, plusieurs améliorations de la sémantique opérationnelle ont été proposées qui ont abouti finalement à un calcul optimal [4]. D'autre part, plusieurs machines abstraites accompagnées de techniques de compilations ont vu le jour conférant ainsi aux langages logico-fonctionnels, le statut de langages de programmation à part entière.

Dans notre travail, nous partons du constat suivant:
Aucun langage logico-fonctionnel actuel n'offre la possibilité de définir des types de données abstraits avec invariants (sous-types) permettant d'utiliser à la fois des termes de premier ordre (classiques) et des termes rationnels (graphes), en présence de sémantiques dénotationnelle et opérationnelle adéquates.

## Objectifs du projet

Nous proposons dans ce projet une nouvelle approche des types de données. Force est de constater l'utilité du typage dans les langages. Du point de vue du programmeur, il impose une rigueur parfois pesante dans la conception des programmes. Mais d'une part, il facilite la structuration des programmes, et donc leur lisibilité. Et d'autre part, il permet de raisonner plus facilement sur un programme. Du point de vue de la machine maintenant, le typage permet de détecter un grand nombre d'erreurs à la compilation. Il augmente ainsi la fiabilité, la maintenabilité, la rapidité avec laquelle on développe les logiciels.

### Les graphes

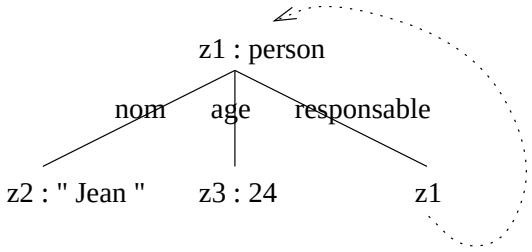
Les graphes sont omniprésent en informatique, parce qu'ils le sont dans notre vie de tous les jours. D'une part, nous les utilisons naturellement pour modéliser. Ainsi, une molécule, le plan d'une maison, ou un schéma électrique sont des graphes. D'autre part, les graphes sont des objets mathématiques définis de manière précise ; ils nous permettent donc de raisonner facilement. Ainsi, utiliser une carte routière pour se déplacer d'un point $A$ vers un point $B$, c'est appliquer un algorithme de plus court chemin. Compte tenu de ces remarques, les graphes sont forcément des objets fondamentaux de l'informatique.

La puissance actuelle des ordinateurs nous permet de les utiliser avec des performances raisonnables [5]. C'est par exemple le cas en bases de données objets ou déductives, en systèmes de recherche d'informations et en systèmes multimédias, en traitement des langues naturelles, en vérification des systèmes finis (preuve d'une spécification écrite en Lotos, ou avec un réseau de Pétri par exemple) ...

Les langages impératifs ont été les premiers à permettre de représenter les graphes : on peut y manipuler directement des pointeurs. Dans les langages fondés sur le lambda-calcul, l'utilisation d'un symbole d'affectation (le `setq` de Lisp) a aussi pu permettre de "régler" le problème. Quant aux langages logiques, la manipulation de graphes est rendue possible par l'utilisation des termes infinis rationnels ([6]) dans le langage Prolog 2 ([7]). Citons enfin le cas assez particulier de Life ([8]), un langage qui tente d'allier *tous* les paradigmes de la programmation, et qui utilise des $\Psi$-termes, des sortes de graphes très appréciés en IA ([9], [10]).

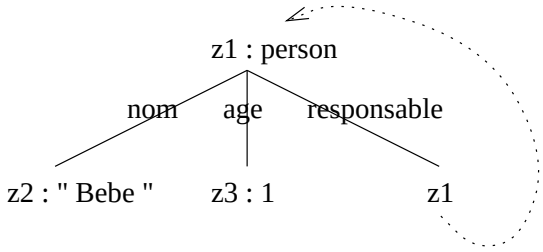
Néanmoins, à notre connaissance, aucun langage algébrique n'est doté de tels structures. Le premier objectif de notre projet est de définir un langage fondé sur la logique des clauses de Horn avec égalité, qui permette de les manipuler. Nous donnons ci-dessous un exemple de graphe que

nous pourrons manipuler. Informellement, il "signifie" que Jean, 24 ans, est responsable de lui-même (parce que majeur). Ce graphe peut être vu comme un élément du type de données *humain* :




### Les sous-types

D'un autre coté, il existe des graphes qui ne représentent pas des éléments du type *humain.* Par exemple, considérons le graphe suivant :




Ce graphe représente un bébé de 1 an responsable de lui-même. Celà ne correspond plus à l'idée intuitive qu'on a du type *humain.* En effet, seul les personnes de plus de 18 ans peuvent être responsables d'elles-mêmes. Formellement, cet élément ne satisfait pas l'invariant qui caractérise les éléments du type *humain* :

$$\text{Is\_}humain(person(n : nom\ , a : age\ , r : humain\ )) \iff \text{Is\_}adulte(r)$$

Le type *adulte* est un sous-type des *humains* qui caractérise ceux dont l'âge est supérieur à 18 ans. Ceci s'écrit formellement :

$$\text{Is\_}adulte(r : person(n : nom\ , a : age\ , r)) \iff a \geq 18$$

Cet exemple montre que les invariants sont nécessaires pour définir de manière précise les éléments d'un type de données (ici, le type des *humains* et celui des *adultes*). Dans la littérature, on trouve d'autres exemples classiques qui nécessitent des invariants, comme par exemple le type des *listes triées*, celui des *arbres de recherche*, celui des *arbres équilibrés* ...

Clairement, la notion de sous-type est utile dans les langages de programmation. Celà permet, entre autres, de mieux structurer les types de données, et de mieux définir les domaines de définitions de fonctions.

Dans notre travail, nous proposons une syntaxe de programme logico-fonctionnel permettant la définition de sous-types (sous-ensembles) à l'aide d'invariants (ou de prédicats) . Nous définissons ensuite les sémantiques dénotationnelle et opérationnelle des programmes avec sous-types

en fonction des programmes écrits dans la logique multi-sortée des clauses de Horn avec égalité. Notons que la définition de sous-types à l'aide de prédicats est une notion classique en mathématiques. Son utilisation dans des langages de programmation a déja été utilisée. L'approche des sous-types telle qu'elle a été esquissée dans EQLOG [11] est celle qui se rapproche le plus de notre proposition.

## Plan du rapport

Le premier chapitre traite de la syntaxe de notre langage. Nous y définissons les éléments syntaxiques de bases tels que les termes (ou graphes), les clauses de Horn et les programmes.

Le chapitre 2 donne les définitions fondamentales qui nous sont nécessaires pour manipuler les graphes. En particulier, nous travaillons sur une relation, dite de bisimulation, qui identifie des graphes dénotant la même information. Nous nous intéressons aussi au problème de l'unification.

Le chapitre 3 présente d'abord la sémantique dénotationnelle d'un programme, soit la définition mathématique de son sens ; elle est donnée par le E-modèle standard d'un programme. Ce chapitre présente ensuite une première sémantique opérationnelle. Informellement, il s'agit de décrire comment doit s'exécuter un programme, de manière adéquate avec le sémantique dénotationnelle de celui-ci. Nous proposons d'utiliser la SLDEI-résolution, comme calcul cohérent et complet.

Le chapitre 4 traite d'un algorithme de résolution d'équations, fondé sur une extension de la surréduction. Nous montrons la cohérence et la complétude de ce calcul dans le cadre des systèmes de réécriture de graphe.

Le chapitre 5 définit la syntaxe et la sémantique dénotationnelle et opérationnelle des programmes avec sous-sortes, à partir de celles données aux programmes sans sous-sorte.

Le dernier chapitre définit les perspectives de nos futurs travaux.

# Chapter 1

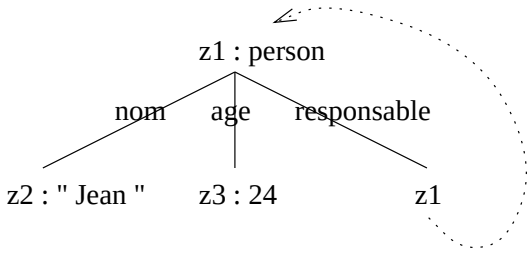
# Syntaxe

Dans ce chapitre, nous définissons la syntaxe de notre langage, soit la manière d'écrire un programme.

Dans une première partie, nous définissons les signatures. Elles correspondent approximativement à la partie déclaration d'un programme en Ada. Elles sont composées de noms de sortes, de noms de fonctions, et de noms de prédicats. Notez que :

- Les sortes sont de nature finie ou rationnelle. Une sorte rationnelle supporte les graphes, alors qu'une sorte finie ne les supporte pas.
- Les noms de fonctions regroupent à la fois les noms de constructeurs et d'opérateurs : comme son nom l'indique, un constructeur permet de construire des données (c'est le *cons* de Lisp) ; un opérateur permet de calculer sur des données (c'est le *car* et le *cdr* de Lisp). Dans l'étude théorique que nous proposons, nous n'avons pas besoin de faire cette distinction. Par contre, elle serait utile pour une implémentation [12].

Dans une seconde partie, nous définissons les termes, soit les objets qu'on manipule dans le langage, c'est-à-dire des graphes. Pour les construire, nous avons besoin de nouveaux éléments syntaxiques : les étiquettes. L'exemple suivant éclairsira notre propos :




Ce graphe "dit que" Jean, 24 ans, est responsable de lui-même (puisqu'il est majeur). Les étiquettes sont $z_1, z_2, z_3$. Le terme est représenté en trait plein, et le graphe sous-entendu, en pointillé. Les étiquettes servent à faire des références. Elles ressemblent aux pointeurs utilisés dans les langages impératifs.

Dans une dernière partie, nous définissons les atomes, les clauses et les programmes. Les clauses permettent de définir le code associé aux fonctions et aux prédicats, ce qui correspond approximativement à la partie implémentation d'un programme en Ada. Enfin, un programme

dans notre syntaxe, consiste en une partie déclaration (la signature) et une partie implémentation (les clauses).

## 1.1 Signature

**Définition 1** (SIGNATURE) : Soient $\Im$ et $\Re$ deux symboles distingués. Une **signature** $\Sigma$ est un triplet

$$\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$$

tel que

- $\mathcal{S}$ soit un ensemble fini de **sortes**, union disjointe d'un **ensemble de sortes $\Im$inies** $\mathcal{S}_\Im$ et d'un **ensemble de sortes $\Re$ationnelles** $\mathcal{S}_\Re$ :

$$\mathcal{S} = \mathcal{S}_\Im \uplus \mathcal{S}_\Re$$

- $\Omega$ soit une famille $\mathcal{S}^\star \times \mathcal{S}$-indicée (de noms) de **fonctions** :

$$\Omega = \{\Omega_{w,s} \mid (w,s) \in \mathcal{S}^\star \times \mathcal{S}\}$$

- $\Pi$ soit une famille $\mathcal{S}^+$-indicée (de noms) de **prédicats** :

$$\Pi = \{\Pi_w \mid w \in \mathcal{S}^+\}$$

Un symbole de fonction $f \in \Omega_{w,s}$ est dit d'**arité** $w$, de **sorte** $s$, et de **profil** $(w,s)$. Si $f \in \Omega_{\lambda,s}$, on le note $f : \rightarrow s$, et si $f \in \Omega_{s_1 \ldots s_n, s}$, on le note $f : s_1 \ldots s_n \rightarrow s$. De même, un symbole de prédicat $P \in \Pi_{s_1 \ldots s_n}$ est dit d'**arité** $s_1 \ldots s_n$, et on le note $P : s_1 \ldots s_n$.

**Définition 2** (SIGNATURE ÉGALITAIRE) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature. On dit que $\Sigma$ est **égalitaire** si :

$$\forall s \in \mathcal{S}, =_{ss} \in \Pi_{ss}$$

.

**Exemple** : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ la signature égalitaire telle que :

- $\mathcal{S} = \mathcal{S}_\Im \uplus \mathcal{S}_\Re$ avec $\mathcal{S}_\Im = \{nat, string\}$ et $\mathcal{S}_\Re = \{human\}$.
- $\Omega = \{\Omega_{\lambda,nat}\ ,\ \Omega_{nat,nat}\ ,\ \Omega_{nat\ nat,\ nat}\ ,\ \Omega_{\lambda,string}\ ,\ \Omega_{string\ nat\ human,\ human}\}$ avec :

  $0 : \rightarrow nat \in \Omega_{\lambda,nat}$
  $s : nat \rightarrow nat \in \Omega_{nat,nat}$
  $+ : nat\ nat \rightarrow nat \in \Omega_{nat\ nat,\ nat}$

  "Jean" $: \rightarrow string \in \Omega_{\lambda,string}$

  person $: string\ nat\ human \rightarrow human \in \Omega_{string\ nat\ human\ ,\ human}$

- $\Pi = \{\Pi_{nat}\ ,\ \Pi_{nat\ nat}\ ,\ \Pi_{string\ string}\ ,\ \Pi_{human\ human}\}$ avec :

  Pair $: nat \in \Pi_{nat}$
  Impair $: nat \in \Pi_{nat}$
  $=_{nat\ nat} : nat\ nat \in \Pi_{nat\ nat}$

  $=_{string\ string} : string\ string \in \Pi_{string\ string}$

  $=_{human\ human} : human\ human \in \Pi_{human\ human}$

$\Diamond$

**Remarque** : En définissant des familles de fonctions et de prédicats, on bénéficie du polymorphisme dit *ad hoc* : un nom de fonction peut avoir plusieurs profils, et donc appartenir à plusieurs familles. Par exemple, on peut définir + sur les entiers et sur les chaines :

- $+$ : *nat nat* $\rightarrow$ *nat*
- $+$ : *string string* $\rightarrow$ *string*

Si nous avions seulement considéré un *ensemble* de noms de fonctions (resp. de prédicats), un nom n'aurait pu apparaître qu'une seule fois, et donc, il n'aurait pu avoir qu'un seul profil. D'autres types de polymorphismes existent ([13]), et nous en reparlerons dans le chapitre sur les sous-sortes.
$\Diamond$

## 1.2 Termes

Nous allons définir les termes (nos graphes) en deux temps. Dans le premier paragraphe, on parle de pré-termes qui vont fixer la structure syntaxique des termes. Dans le second paragraphe, nous donnons des conditions sur ces pré-termes pour qu'ils définissent les termes.

### 1.2.1 Pré-termes

On considère un ensemble de **variables** $\mathcal{X} = \biguplus_{s\in\mathcal{S}} \mathcal{X}_s$ et un ensemble d'**étiquettes** $\mathcal{L} = \biguplus_{s\in\mathcal{S}} \mathcal{L}_s$.

**Définition 3** (PRÉ-TERMES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi\rangle$ une signature, $\mathcal{X} = \biguplus_{s\in\mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s\in\mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On définit récursivement l'ensemble des **pré-termes** $\breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X}) = \biguplus_{s\in\mathcal{S}} \breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X})_s$ en posant :

1. $z \in \mathcal{L}_s \Longrightarrow \boxed{z \in \breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X})_s}$
2. $x \in \mathcal{X}_s, z \in \mathcal{L}_s \Longrightarrow \boxed{z : x \in \breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X})_s}$
3. $c \in \Omega_{\lambda,s}, z \in \mathcal{L}_s \Longrightarrow \boxed{z : c \in \breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X})_s}$
4. $$\left.\begin{array}{l} f \in \Omega_{s_1\ldots s_n,s} \\ t_i \in \breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X})_{s_i}, (i \in 1\ldots n) \\ z \in \mathcal{L}_s \end{array}\right\} \Longrightarrow \boxed{z : f(t_1,\ldots,t_n) \in \breve{\mathcal{T}}(\Sigma,\mathcal{L},\mathcal{X})_s}$$

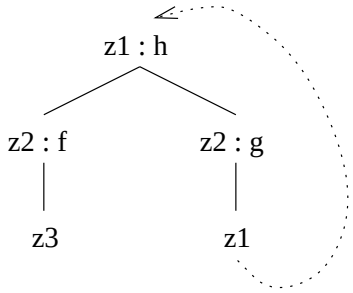
Nous donnons immédiatement quelques définitions supplémentaires sur les pré-termes :

1. $\mathrm{Var}(t) = \biguplus_{s\in S} \mathrm{Var}_s(t)$ est l'ensemble des **variables** (libres) du pré-terme $t$
2. $\mathrm{Label}(t) = \biguplus_{s\in S} \mathrm{Label}_s(t)$ est l'ensemble des **étiquettes** du pré-terme $t$
3. $\mathrm{Rac}(t)$ désigne la **racine** du pré-terme $t$

| $t$ | $z \in \mathcal{L}_s$ | $z : x$ | $z : c$ | $z : f(t_1,\ldots,t_n)$ |
|---|---|---|---|---|
| $\mathrm{Var}(t)$ | $\emptyset$ | $\{x\}$ | $\emptyset$ | $\bigcup_{i\in1\ldots n} \mathrm{Var}(t_i)$ |
| $\mathrm{Label}(t)$ | $\{z\}$ | $\{z\}$ | $\{z\}$ | $\{z\} \cup \bigcup_{i\in1\ldots n} \mathrm{Label}(t_i)$ |
| $\mathrm{Rac}(t)$ | $\{z\}$ | $\{z\}$ | $\{z\}$ | $\{z\}$ |

### 1.2.2 Termes

Il s'agit maintenant de définir les termes (bien formés) à partir des pré-termes. Visualisons les problèmes avec un exemple :




Ce pré-terme a pour expression littérale $t_0 = z_1 : h(z_2 : f(z_3), z_2 : g(z_1))$.

Informellement, on dit qu'une étiquette est définie dans un pré-terme lorsqu'elle référence une variable, une constante, ou un symbole de fonction dans ce pré-terme. Dans cet exemple, l'étiquette $z_2$ est définie deux fois ; ensuite, l'étiquette $z_3$ n'est pas définie (elle ne référence rien) ; enfin, il est possible que $g$ soit une fonction de sorte finie, alors qu'on "boucle" infiniment dessus, du fait de l'étiquette $z_1$.

Nous commençons par caractériser les étiquettes définies dans un pré-terme, en définissant la notion de position. Puis nous introduisons la relation d'accessibilité, qui relie les étiquettes d'un pré-terme entre elles.

**Définition 4** (Position) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On définit récursivement l'ensemble des **positions** d'un pré-terme $t$, noté $\mathrm{Pos}(t) = \biguplus_{s \in S} \mathrm{Pos}_s(t)$ en posant :

1. $z \in \mathcal{L} \Longrightarrow \mathrm{Pos}(z) = \emptyset$
2. $\mathrm{Pos}(z : x) = \{z\}$
3. $\mathrm{Pos}(z : c) = \{z\}$
4. $\mathrm{Pos}(z : f(t_1, \ldots, t_n)) = \{z\} \cup \bigcup_{i=1}^{n} \mathrm{Pos}(t_i)$

On note $\mathrm{Pos}^\star(t)$ l'ensemble des positions qui ne référencent pas de variables.

**Exemple** : Nous reprenons le pré-terme ci-dessus : $t_0 = z_1 : h(z_2 : f(z_3), z_2 : g(z_1))$. Nous avons :

- $\mathrm{Label}(t_0) = \{z_1, z_2, z_3\}$
- $\mathrm{Pos}(t_0) = \{z_1, z_2\}$

$\diamondsuit$

**Définition 5** (Relation d'accessibilité) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On définit pour tout pré-terme $t$ la **relation d'accessibilité** $\rightsquigarrow_t$ sur $\mathrm{Label}(t)$ en posant :

1. $z \in \mathcal{L} \Longrightarrow \rightsquigarrow_z = \emptyset$
2. $\rightsquigarrow_{z:x} = \emptyset$
3. $\rightsquigarrow_{z:c} = \emptyset$
4. $\rightsquigarrow_{z:f(t_1,\ldots,t_n)} = \bigcup_{i=1}^{n}(\{(z, \mathrm{Rac}(t_i))\} \cup \rightsquigarrow_{t_i})$

On note ${\rightsquigarrow_t}^+$ la fermeture transitive de $\rightsquigarrow_t$.

**Exemple** : Nous reprenons encore le pré-terme $t_0 = z_1 : h(z_2 : f(z_3), z_2 : g(z_1))$. Nous avons : $\rightsquigarrow_{t_0} = \{(z_1, z_2), (z_1, z_1)\}$ $\diamondsuit$

**Définition 6** (TERMES) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Un pré-terme $t$ est un **terme** (bien formé) si :

1. Toutes ses étiquettes sont définies :

$$\mathrm{Pos}(t) = \mathrm{Label}(t)$$

2. Toutes ses étiquettes ne sont définies qu'une seule fois :

$$t = z : f(t_1, \ldots, t_n) \implies \begin{cases} \forall i, j \in 1 \ldots n, i \neq j \implies \mathrm{Pos}(t_i) \cap \mathrm{Pos}(t_j) = \emptyset \\ \forall i \in 1 \ldots n, z \notin \mathrm{Pos}(t_i) \end{cases}$$

3. On ne "boucle" pas sur les étiquettes de sorte finie :

$$\forall s \in \mathcal{S}_{\mathbb{S}} \ , \forall z \in \mathrm{Label}_s(t) \ , \neg(z \ {\rightsquigarrow_t}^+ \ z)$$

On note l'ensemble des termes $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) = \biguplus_{s \in \mathcal{S}} \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s$.

**Définition 7** (TERME FERMÉ) : On dit qu'un terme $t$ est **fermé** si il n'a pas de variable libre : $\mathrm{Var}(t) = \emptyset$. On note $\mathcal{T}(\Sigma, \mathcal{L}) = \biguplus_{s \in S} \mathcal{T}(\Sigma, \mathcal{L})_s$ leur ensemble. On a : $\mathcal{T}(\Sigma, \mathcal{L}) = \mathcal{T}(\Sigma, \mathcal{L}, \emptyset)$.

**Exemple** : Le pré-terme précédent $t_0 = z_1 : h(z_2 : f(z_3), z_2 : g(z_1))$ n'est pas un terme bien formé puisque $\mathrm{Label}(t_0) \neq \mathrm{Pos}(t_0)$. Par contre, le pré-terme donné p. 9, d'expression littérale $t_1 = z_1 : \mathrm{person}(z_2 : \text{"Jean"}, z_3 : 24, z_1)$ est un terme bien formé. $\diamondsuit$

## 1.3 Atomes, Clauses, Spécifications, Programmes

Pour finir, nous définissons les atomes, les clauses et les spécifications.

**Définition 8** (ATOMES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes.

- On définit l'ensemble des **atomes** $\mathcal{AT}(\Sigma, \mathcal{L}, \mathcal{X})$ par :

$$\left.\begin{array}{l} P \in \Pi_{s_1 \ldots s_n} \\ t_i \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_{s_i}, (i \in 1 \ldots n) \end{array}\right\} \Longrightarrow P(t_1, \ldots, t_n) \in \mathcal{AT}(\Sigma, \mathcal{L}, \mathcal{X})$$

- Si $\Sigma$ est une signature égalitaire, on appelle **équation** tout atome où le symbole de prédicat est le symbole d'égalité : $=_{ss}\ (t_1, t_2)$, que l'on notera aussi $t_1 == t_2$ lorsqu'aucune confusion n'est possible.

**Définition 9** (CLAUSES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $Q_0, \ldots, Q_n$ des atomes (de $\mathcal{AT}(\Sigma, \mathcal{L}, \mathcal{X})$), où $n \geq 0$.

- On appelle **clause de Horn définie** la formule :

$$Q_0 \Longleftarrow Q_1, \ldots, Q_n \text{ avec } n \geq 0$$

  Lorsque $n = 0$, on dit que $Q_0$ est un **fait**. Lorsque $n \neq 0$, on dit que $Q_0$ est la **tête** de la clause et que $Q_1, \ldots, Q_n$ est le **corps** de la clause.
- On définit aussi les **clauses de Horn** générales qui nous servirons dans les raisonnements par réfutations. En plus des clauses définies, on admet les **clauses de but**, soit les formules : $\Longleftarrow\ A_1 \ldots A_n$. On note $\Box$ la clause (de but) vide, symbole de la contradiction dans les raisonnements par réfutation.

**Définition 10** (SPÉCIFICATION, PROGRAMME) : Une **spécification** dans la logique des clauses de Horn avec égalité ou un **programme** est un couple

$$\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$$

où $\Sigma$ est une signature égalitaire, et $\mathcal{HC}$, un ensemble de clauses de Horn définies.

**Exemple** : On considère de nouveau la signature $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ (p. 10). On définit la spécification $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ en posant :

$$\begin{array}{ll} \mathcal{HC} = \{ & z_0 : +(z_1 : x, z_2 : 0) == z_3 : x \\ & z_0 : +(z_1 : x, z_2 : s(z_3 : y)) == z_4 : s(z_5 : +(z_6 : x, z_7 : y)) \\ & \text{Pair}(z_0 : 0) \\ & \text{Pair}(z_0 : s(z_1 : x)) \Longleftarrow \text{Impair}(z_2 : x) \\ & \text{Impair}(z_0 : s(z_1 : x)) \Longleftarrow \text{Pair}(z_2 : x)\} \end{array}$$

En omettant les étiquettes "inutiles", on obtient :

$$\begin{array}{ll} \mathcal{HC} = \{ & +(x, 0) == x \\ & +(x, s(y)) == s(+(x, y)) \\ & \text{Pair}(0) \\ & \text{Pair}(s(x)) \Longleftarrow \text{Impair}(x) \\ & \text{Impair}(s(x)) \Longleftarrow \text{Pair}(x)\} \end{array}$$

$\Diamond$

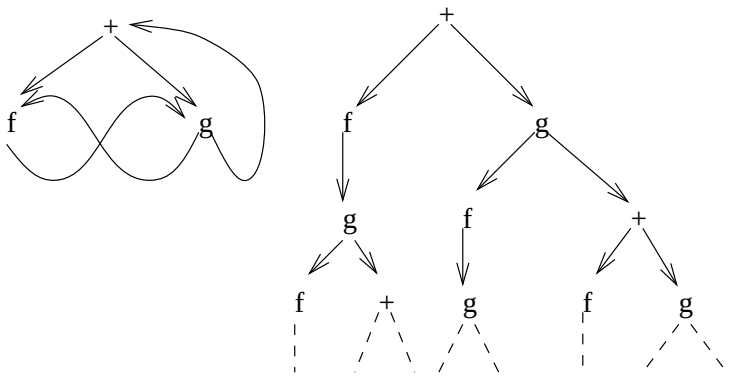
# Chapter 2

# Graphes

Dans ce chapitre, nous nous concentrons sur les objets syntaxiques élémentaire de notre langage, à savoir les graphes, soit les termes de notre langage. Puisque nous ne travaillons plus dans un cadre classique, il s'agit d'une part, de préciser les définitions traditionnellement données sur des termes finis, et d'autre part, de travailler sur de nouvelles notions.

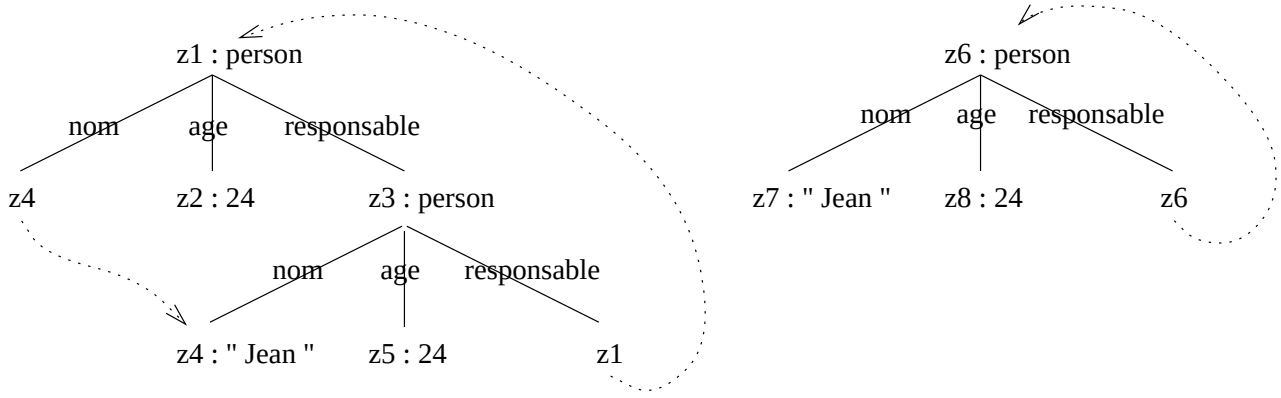
Nous commençons par définir ce qu'est une relation de congruence sur les termes. Puis nous étudions la première d'entre elle, dite $\alpha$-convertibilité. Elle identifie des termes égaux au renommage de leurs étiquettes près. Le même nom est utilisé en $\lambda$-calcul, où il correspond au renommage des variables liées des $\lambda$-termes. Nous profitons de cette partie pour parler d'$\alpha$-conversion, de renommage des variables libres et de variantes.

Nous développons ensuite une seconde représentation des graphes, utilisant cette fois des systèmes d'équations. Ceux-ci facilitent l'écriture d'algorithmes. Ces systèmes sont capitaux dans les travaux faits sur les arbres infinis ([14]). Et les graphes sont, d'une certaine manière, des arbres infinis "repliés", comme le montre l'exemple ci-dessous :




Nous profitons aussi des systèmes pour définir les sous-termes et la notion de remplacement à une position dans un terme.

Dans la section suivante, nous définissons une nouvelle congruence sur les termes, qui vise à identifier des graphes dénotant un même objet (un même arbre infini, en particulier) :




Nous appelons cette relation de congruence, relation de bisimulation, en référence à la théorie des automates. Nous nous servons des systèmes pour la définir.

Dans une dernière section, nous nous intéressons au problème dit d'unification, c'est-à-dire à la résolution d'équations syntaxiques sur les graphes. Nous commençons par définir les substitutions, puis nous posons le problème d'unification de deux termes, et nous donnons un algorithme qui calcule l'unificateur minimal au renommage près.

## 2.1 Congruence

**Définition 11** (CONGRUENCE SUR LES TERMES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On appelle **congruence** sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ toute famille $\mathcal{S}$-indicée $\rho = \{\rho_s \mid s \in \mathcal{S}\}$ de relations $\rho_s \subseteq \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s \times \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s$ telle que :

1. $t \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s \implies \boxed{t \ \rho_s \ t}$
2. $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s \implies \boxed{t_1 \ \rho_s \ t_2 \implies t_2 \ \rho_s \ t_1}$
3. $t_1, t_2, t_3 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s \implies \boxed{t_1 \ \rho_s \ t_2 \ , \ t_2 \ \rho_s \ t_3 \implies t_1 \ \rho_s \ t_3}$
4. $$\left.\begin{array}{l} f \in \Omega_{s_1 \ldots s_n, s} \\ t_i, t'_i \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_{s_i}, (i \in 1 \ldots n) \\ z, z' \in \mathcal{L}_s \\ \\ \boxed{t_i \ \rho_{s_i} \ t'_i}, (i \in 1 \ldots n) \end{array}\right\} \implies \boxed{z : f(t_1, \ldots, t_n) \ \rho_s \ z' : f(t'_1, \ldots, t'_n)}$$

Les trois premières clauses définissent une **relation d'équivalence**.

Il va de soit que $z : f(t_1, \ldots, t_n)$ et $z' : f(t'_1, \ldots, t'_n)$ doivent définir des termes, et donc que les conditions de bonnes définitions énoncées précédemment doivent être vérifiées. Nous le supposerons toujours dans les démonstrations. Ces relations de congruence sont dites partielles [15].

## 2.2 $\alpha$-convertibilité , $\alpha$-conversion , renommage, variante

### 2.2.1 $\alpha$-convertibilité

Informellement, deux termes sont $\alpha$-convertis si on peut obtenir le second à partir du premier en changeant de manière bijective les noms de toutes ses étiquettes. Ce paragraphe donne un premier exemple de congruence sur les termes.

**Définition 12** (RELATION D'$\alpha$-CONVERTIBILITÉ) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t_1$ et $t_2$ deux termes. On dit que $t_1$ et $t_2$ sont **$\alpha$-convertis**, et on note $t_1 =_\alpha t_2$, s'il existe une bijection $\phi : \text{Label}(t_1) \to \text{Label}(t_2)$ telle que $\phi^\flat(t_1) = t_2$, où $\phi^\flat$ est définie par :

1. $\phi^\flat(z) = \phi(z)$
2. $\phi^\flat(z : x) = \phi(z) : x$
3. $\phi^\flat(z : c) = \phi(z) : c$
4. $\phi^\flat(z : f(t_1, \ldots, t_n)) = \phi(z) : f(\phi^\flat(t_1), \ldots, \phi^\flat(t_n))$

**Proposition 13** : $=_\alpha$ est une congruence sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$.

**Preuve** : $=_\alpha$ est une relation d'équivalence (trivial, après avoir souligné que les $\phi$ utilisées sont des bijections). Soient $t_1 =_\alpha t'_1, \ldots, t_n =_\alpha t'_n$ des termes $\alpha$-convertis, et $z$, $z'$ des étiquettes. Nous supposons que tout est tel que $z : f(t_1, \ldots, t_n)$ et $z' : f(t'_1, \ldots, t'_n)$ soient bien formés : toutes leurs étiquettes sont définies une et une seule fois, et on ne "boucle" pas sur les étiquettes de sortes finies. Par hypothèse, il existe $n$ bijections $\phi_i : \text{Label}(t_i) \to \text{Label}(t'_i)$. Soit $\phi : \{z\} \cup \bigcup_{i=1}^n \text{Label}(t_i) \to \{z'\} \cup \bigcup_{i=1}^n \text{Label}(t'_i)$ telle que $\phi(z) = z'$, et $\phi_{\uparrow \text{Label}(t_i)} = \phi_i$. Clairement, $\phi$ est bijective, et on a : $\phi^\flat(z : f(t_1, \ldots, t_n)) = z' : f(t'_1, \ldots, t'_n)$. Donc $z : f(t_1, \ldots, t_n) =_\alpha z' : f(t'_1, \ldots, t'_n)$. $\diamondsuit$

### 2.2.2 $\alpha$-conversion , renommage , variante

Une $\alpha$-conversion est une application qui change toutes les étiquettes d'un terme. Un renommage est une application qui change toutes les variables libres d'un terme. Par abus de langage, nous utiliserons les mêmes mots pour parler de l'application et de son effet sur un terme. Pour finir, nous donnons la définition de variante, correspondant à la fois aux $\alpha$-conversions et aux renommages.

**Définition 14** ($\alpha$-CONVERSION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t$ un terme, et $M$ un ensemble d'étiquettes contenant $L = \text{Label}(t)$. Soient $A, B \subseteq \mathcal{L}$. On appelle **$\alpha$-conversion de $t$ en dehors de $M$** toute application $\phi : A \to B$ telle que :

1. $\phi$ soit injective
2. $L \subseteq A$
3. $\phi(A) \cap M = \emptyset$

On définit l'**extension** de $\phi$ à l'ensemble des termes $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$, notée $\phi^\flat$, en posant :

1. $\phi^\flat(z) = \phi(z)$
2. $\phi^\flat(z : x) = \phi(z) : x$
3. $\phi^\flat(z : c) = \phi(z) : c$
4. $\phi^\flat(z : f(t_1, \ldots, t_n)) = \phi(z) : f(\phi^\flat(t_1), \ldots, \phi^\flat(t_n))$

Nous étendons cette notion aux atomes et aux clauses.

**Remarque** : Clairement, si $\phi$ est une $\alpha$-conversion de $t$, alors $\phi^\flat(t) =_\alpha t$, parce que l'application

$$\phi' \left| \begin{array}{l} \mathrm{Label}(t) \to \phi(\mathrm{Label}(t)) \\ z \mapsto \phi(z) \end{array} \right.$$

est une bijection. Par conséquent, on a : ${\phi'}^{\flat^{-1}}(\phi^\flat(t)) = t$, et donc $\phi^\flat(t) =_\alpha t$. Autre conséquence, si deux termes sont $\alpha$-convertis, toutes leurs $\alpha$-conversions le sont encore. $\diamondsuit$

**Définition 15** (Renommage) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t$ un terme, et $W$ un ensemble de variables contenant $V = \mathrm{Var}(t)$. Soient $X, Y \subseteq \mathcal{X}$. On appelle **renommage de $t$ en dehors de $W$** toute application $\psi : X \to Y$ telle que :

1. $\psi$ soit injective
2. $V \subseteq A$
3. $\psi(A) \cap W = \emptyset$

On définit l'**extension** de $\psi$ à l'ensemble des termes $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$, notée $\psi^\sharp$, en posant :

1. $\psi^\sharp(z) = z$
2. $\psi^\sharp(z : x) = z : \psi(x)$
3. $\psi^\sharp(z : c) = z : c$
4. $\psi^\sharp(z : f(t_1, \ldots, t_n)) = z : f(\psi^\sharp(t_1), \ldots, \psi^\sharp(t_n))$

Nous étendons cette notion aux atomes et aux clauses.

**Définition 16** (Variante) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soit $E$ une entité syntaxique quelconque : terme, atome ou clause. On dit qu'une entité $E'$ est une **variante** de $E$ si $E'$ est obtenue à partir de $E$ par $\alpha$-conversion de toutes les étiquettes de $E$ et par renommage de toutes les variables libres de $E$.

## 2.3 Représentation des termes avec des systèmes d'équations

### 2.3.1 Des termes aux systèmes

Dans ce paragraphe, nous développons la représentation des termes sous forme de systèmes.

**Définition 17** (Équations et Systèmes) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On appelle **équation sylvestre** toute expression de la forme :

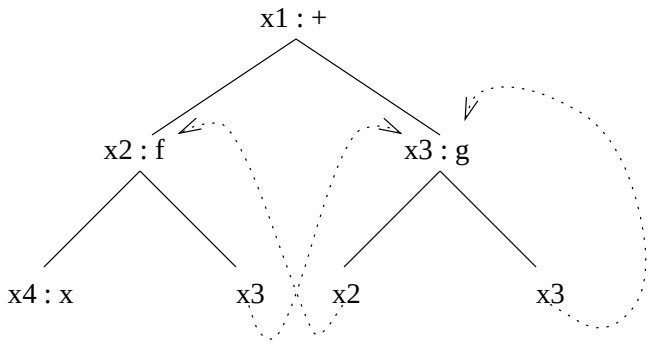
1. $z = x$ où $x \in \mathcal{X}_s, z \in \mathcal{L}_s$
2. $z = c$ où $c \in \Omega_{\lambda,s}, z \in \mathcal{L}_s$
3. $z = f(z_1, \ldots, z_n)$ où $f \in \Omega_{s_1 \ldots s_n, s}, z \in \mathcal{L}_s$ et $z_i \in \mathcal{L}_{s_i}$ pour $i \in 1 \ldots n$

Nous noterons plus généralement $z = \phi$ n'importe laquelle de ces expressions. On appelle **système** tout ensemble fini $\xi$ d'équations (sylvestres).

**Définition 18** (Système associé à un terme) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On définit récursivement le **système associé à un terme** $t$ noté System($t$) en posant :

1. $\text{System}(z) = \emptyset$
2. $\text{System}(z : x) = \{z = x\}$
3. $\text{System}(z : c) = \{z = c\}$
4. $\text{System}(z : f(t_1, \ldots, t_n)) = \{z = f(\text{Rac}(t_1), \ldots, \text{Rac}(t_n))\} \cup \bigcup_{i=1}^{n} \text{System}(t_i)$

**Exemple** : Nous donnons ici le terme $t_0$ qui nous servira dans toute la suite :




Littéralement, $t_0 = x_1 : +(x_2 : f(x_4 : x, x_3), x_3 : g(x_2, x_3))$

Son système associé, $\xi_0$, est :

$$\xi_0 = \text{System}(t_0) = \begin{cases} x_1 = +(x_2, x_3) \\ x_2 = f(x_4, x_3) \\ x_3 = g(x_2, x_3) \\ x_4 = x \end{cases}$$

$\Diamond$

### 2.3.2 Des systèmes au termes

Nous traitons maintenant le problème inverse : le passage d'un système à un terme. Il est clair que ce n'est pas toujours possible, que seuls des systèmes bien formés peuvent représenter des termes. Pour s'en convaincre, considérons le système suivant :

$$\begin{cases} x_1 = +(x_2, x_2) \\ x_2 = f(x_3) \\ x_2 = g(x_1) \end{cases}$$

Dans ce système, $x_2$ est définie deux fois, et $x_3$ n'est pas définie. C'est le même problème qu'avec les pré-termes. Le système $\xi_0$ (p. 19), ne pose pas ces problèmes. De plus, comme dans le cas des pré-termes, il se peut que $g$ soit une fonction de sorte finie, alors qu'on "boucle" dessus par l'intermédiaire de $x_1$.

Pour reconstruire un terme à partir d'un système bien formé, nous commençons par définir les positions, les étiquettes et la relation d'accessibilité, comme dans le cas des pré-termes. Nous définissons ensuite ce qu'est un système bien formé. Puis nous parlons de sous-système, soit des systèmes dont toutes les équations sont "accessibles" depuis une étiquette donnée. Nous finissons par donner un algorithme de construction d'un terme à partir d'un système.

**Définition 19** (Position, Étiquette, Accessibilité) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soit $\xi$ un système. Par analogie avec les termes, on définit :

1. l'ensemble des **positions** de $\xi$ :

$$\mathrm{Pos}(\xi) = \{z \in \mathcal{L} \mid z = \phi \in \xi\}$$

2. l'ensemble des **étiquettes** de $\xi$ :

$$\mathrm{Label}(\xi) = \mathrm{Pos}(\xi) \cup \{z_1, \ldots, z_n \mid z = f(z_1, \ldots, z_n) \in \xi\}$$

3. la **relation d'accessibilité** sur $\xi$ :

$$\rightsquigarrow_\xi = \{(z, z_i) \mid z = f(z_1, \ldots, z_n) \in \xi\}$$

On note ${\rightsquigarrow_\xi}^+$ la fermeture transitive de $\rightsquigarrow_\xi$ et ${\rightsquigarrow_\xi}^\star$, la fermeture réflexive de ${\rightsquigarrow_\xi}^+$.

**Définition 20** (Système bien formé) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soit $\xi$ un système. Par analogie avec la définition des termes, on dit que $\xi$ est un **système bien formé** si :

1. Toutes ses étiquettes sont définies :

$$\mathrm{Pos}(\xi) = \mathrm{Label}(\xi)$$

2. Toutes ses étiquettes ne sont définies qu'une seule fois :

$$\left.\begin{array}{l} x_1 = \phi_1 \in \xi \\ x_2 = \phi_2 \in \xi \\ \phi_1 \neq \phi_2 \end{array}\right\} \Longrightarrow x_1 \neq x_2$$

3. On ne "boucle" pas sur les étiquettes de sorte finie :

$$\forall s \in \mathcal{S}_\mathbb{S} \ , \forall z \in \mathrm{Label}_s(\xi) \ , \neg(z \ {\rightsquigarrow_\xi}^+ \ z)$$

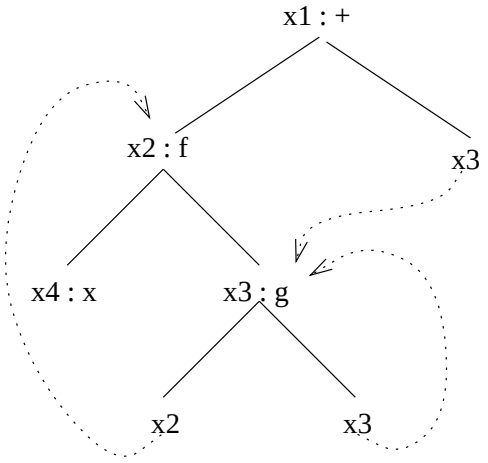
**Définition 21** (Terme associé à une position dans un système) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\xi$ un système bien formé, et $z \in \mathrm{Pos}(\xi)$ une de ses positions. On construit le **terme associé à la position $z$ dans le système** $\xi$, noté $\mathrm{Term}(z, \xi)$, en posant $(z, \xi)$ et en appliquant l'algorithme suivant :

1. $(z, E) \vdash (z, E)$ si $z \notin \mathrm{Pos}(E)$
2. $(z, \{z = x\} \cup E) \vdash (z : x, E)$
3. $(z, \{z = c\} \cup E) \vdash (z : c, E)$
4. $$(z, \{z = f(z_1, \ldots, z_n)\} \cup E) \vdash (z : f(t_1, \ldots, t_n)\} \cup E_n) \text{ si } \begin{array}{l} (z_1, E) \vdash (t_1, E_1) \\ (z_2, E_1) \vdash (t_2, E_2) \\ \ldots \\ (z_n, E_{n-1}) \vdash (t_n, E_n) \end{array}$$

**Exemple** : Nous utilisons le système $\xi_0$ :

$$\xi_0 = \mathrm{System}(t_0) = \begin{cases} x_1 = +(x_2, x_3) \\ x_2 = f(x_4, x_3) \\ x_3 = g(x_2, x_3) \\ x_4 = x \end{cases}$$

Construisons le terme associé à la position $x_0$, racine du terme initial $t_0$. L'algorithme donne :




Littéralement, ce terme a pour expression : $x_1 : +(x_2 : f(x_4 : x, x_3 : g(x_2, x_3)), x_3)$. Le terme initial $t_0$ avait pour expression : $x_1 : +(x_2 : f(x_4 : x, x_3), x_3 : g(x_2, x_3))$ $\diamondsuit$

On constate que nous n'obtenons pas le même terme que $t_0$, duquel nous sommes partis. Autrement dit, Term et System ne sont pas réciproque l'une de l'autre :

- En général, $\mathrm{System}(\mathrm{Term}(z, \xi)) \neq \xi$ parce qu'on oublie des équations en construisant le terme associé à $z$.
- Et de même : $\mathrm{Term}(\mathrm{Rac}(t), \mathrm{System}(t)) \neq t$ parce que le passage par le système casse l'ordre dans la définition des sous-termes.

Nous règlerons ce problème en définissant la prochaine congruence (la bisimulation).

### 2.3.3 Sous-termes et remplacement

Dans ce paragraphe, nous profitons des systèmes pour définir la notion de sous-terme et celle de remplacement.

**Définition 22** (SOUS-SYSTÈME) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\xi$ un système bien formé, et $z \in \mathrm{Pos}(\xi)$, une de ses positions. On définit le **sous-système de $\xi$ associé à** $z$, noté $\xi_z$, en posant :

$$\xi_z = \{x = \phi \in \xi \mid z \rightsquigarrow_\xi^\star x\}$$

**Remarque** : Tout sous-système (d'un système bien formé) reste un système bien formé. ♢

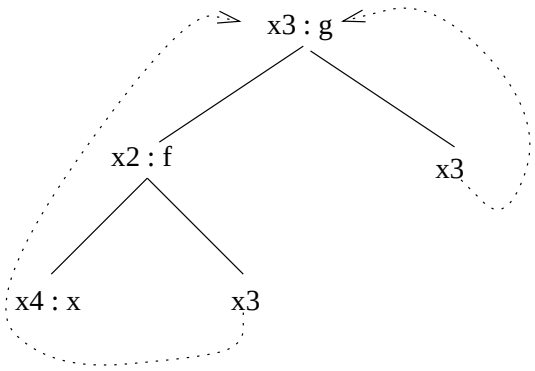
**Définition 23** (SOUS-TERME) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t$ un terme, et $z \in \mathrm{Pos}(t)$ une de ses positions. On appelle **sous-terme de $t$ à la position** $z$, noté $t_{\uparrow z}$, le terme associé à $z$ dans le système associé à $t$ :

$$t_{\uparrow z} = \mathrm{Term}(z, \mathrm{System}(t))$$

**Exemple** : On calcule le sous-terme de $t_0$ à la position $x_3$. Le sous-système de $\xi_0$ associé à $x_3$ est :

$$\xi_{0_{x_3}} = \begin{cases} x_3 = g(x_2, x_3) \\ x_2 = f(x_4, x_3) \\ x_4 = x \end{cases}$$

L'algorithme de reconstruction du terme donne :




Littéralement : $t_{0_{\uparrow x_3}} = x_3 : g(x_2 : f(x_4 : x, x_3), x_3)$. ♢

Avant de donner la définition de remplacement, on introduit une nouvelle notation. On note $\xi[x_1 := x_2]$ le système obtenu en remplaçant toutes les occurrences de l'étiquette $x_1$ par l'étiquette $x_2$ dans le système $\xi$.

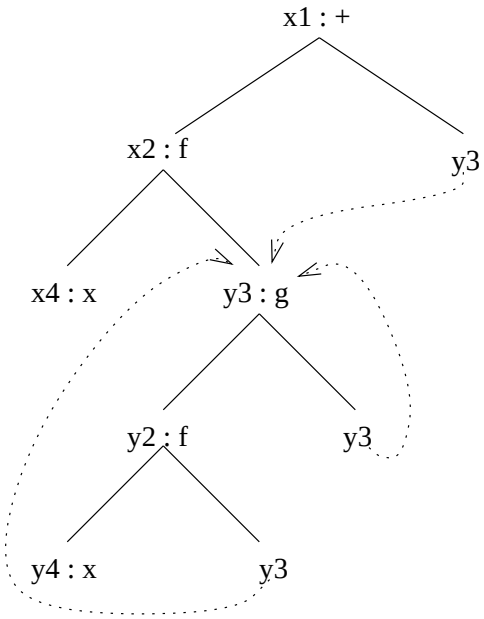
**Définition 24** (GREFFE , REMPLACEMENT) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t_1$ , $t_2$ deux termes, $z \in \mathrm{Pos}(t_1)$ une position de $t_1$, et $M \supseteq \mathrm{Label}(t_1)$ un ensemble d'étiquettes. On appelle **greffe de $t_2$ sur $t_1$ à la position z** ou **remplacement par $t_2$ à la position $z$ dans** $t_1$, noté $t_1[z \leftarrow t_2]$ le terme tel que :

- Si $z = \mathrm{Rac}(t_1)$ alors $t_1[z \leftarrow t_2] = t_2$
- Sinon :
  1. (a) On renomme $t_2$ pour éviter les conflits d'étiquettes : soit $t'_2 =_\alpha t_2$ tel que $\mathrm{Label}(t'_2) \cap M = \emptyset$.
     (b) On note $\xi'_2$ le système associé à $t'_2$ : $\xi'_2 = \mathrm{System}(t'_2)$.
  2. (a) Soit $\xi_1$ le système associé à $t_1$ : $\xi_1 = \mathrm{System}(t_1)$.
     (b) On enlève de $\xi_1$ l'équation associée à l'étiquette $z$ : $\xi'_1 = \xi_1 - \{z = \phi\}$.
     (c) On remplace toutes les occurrences de $z$ par la racine de $t'_2$ dans le système $\xi'_1$ : $\xi''_1 = \xi'_1[z := \mathrm{Rac}(t'_2)]$.
  3. On pose enfin :
$$t_1[z \leftarrow t_2] = \mathrm{Term}(\mathrm{Rac}(t_1), \xi''_1 \cup \xi'_2)$$

**Exemple** : Calculons $t_0[x_3 \leftarrow t_{0\uparrow x_3}]$. Nous obtenons :

$$\xi'_2 = \left\{ \begin{array}{l} y_3 = g(y_2, y_3) \\ y_2 = f(y_4, y_3) \\ y_4 = x \end{array} \right. \quad \text{et } \xi''_1 = \left\{ \begin{array}{l} x_1 = +(x_2, y_3) \\ x_2 = f(x_4, y_3) \\ x_4 = x \end{array} \right.$$

Le terme $t_0[x_3 \leftarrow t_{0\uparrow x_3}]$ vaut donc :




Littéralement : $t_0[x_3 \leftarrow t_{0\uparrow x_3}] = x_1 : +(x_2 : f(x_4 : x, y_3 : g(y_2 : f(y_4 : x, y_3), y_3)), y_3)$. Or le terme $t_0$ avait pour expression : $t_0 = x_1 : +(x_2 : f(x_4 : x, x_3), x_3 : g(x_2, x_3))$ $\diamondsuit$

On constate, là encore, qu'en général, $t$ et $t[z \leftarrow t_{\uparrow z}]$ ne sont pas syntaxiquement égaux. Par contre, ils sont égaux modulo la bisimulation, introduite ci-après.

## 2.4 Bisimulation

Dans cette section, nous définissons une nouvelle relation de congruence sur les termes. Informellement, deux termes sont bisimilaires si on n'arrive pas à les distinguer lorsqu'on parcourt

leurs arcs respectifs. Cette notion existe aussi dans la théorie des automates. Nous proposons une définition formelle en passant par les systèmes.

### 2.4.1 Relations de bisimulation sur les systèmes

**Définition 25** (BISIMULATION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soit $\xi$ un système bien formé. On appelle **simulation** toute relation $\rho \subseteq \text{Label}(\xi) \times \text{Label}(\xi)$ telle que $\forall (x_1, x_2) \in \rho$,

1. $x_1 = x \in \xi \implies x_2 = x \in \xi$
2. $x_1 = c \in \xi \implies x_2 = c \in \xi$
3. $$x_1 = f(u_1, \ldots, u_n) \in \xi \implies \left\{ \begin{array}{l} x_2 = f(v_1, \ldots, v_n) \in \xi \text{ et} \\ \forall i \in 1 \ldots n,\ (u_i, v_i) \in \rho \end{array} \right.$$

On appelle **bisimulation** toute simulation qui soit une relation d'équivalence.

**Remarque** : La fermeture réflexive (resp. symétrique , transitive) d'une simulation reste une simulation. $\diamondsuit$

**Proposition 26** : Tout système bien défini $\xi$ admet une plus grande bisimulation (au sens de $\subseteq$) que l'on note $\dot{\sim}_\xi$ (ou simplement $\dot{\sim}$).

**Preuve** : On note $\mathcal{P}(\xi) = \mathbf{2}^{\text{Label}(\xi) \times \text{Label}(\xi)}$ l'ensemble des relations binaires sur les étiquettes de $\xi$. Soit $\Phi : \mathcal{P}(\xi) \to \mathcal{P}(\xi)$ l'application telle que

$$\Phi(\rho) = \left\{ \begin{array}{l} (x_1, x_2) \in \text{Label}(\xi) \times \text{Label}(\xi) \text{ tel que} \\ x_1 = x \in \xi \implies x_2 = x \in \xi \\ x_1 = c \in \xi \implies x_2 = c \in \xi \\ x_1 = f(u_1, \ldots, u_n) \in \xi \implies \left\{ \begin{array}{l} x_2 = f(v_1, \ldots, v_n) \in \xi \\ \forall i \in 1 \ldots n,\ (u_i, v_i) \in \rho \end{array} \right. \end{array} \right\}$$

$\Phi$ est croissante sur $\mathcal{P}(\xi)$. De plus, $\langle \mathcal{P}(\xi), \subseteq \rangle$ est un treillis complet fini. Donc $\Phi$ est continue. Donc $\Phi$ admet un plus grand point fixe. Compte tenu de la définition de $\Phi$, ce point fixe est une simulation. De plus, elle contient toutes les autres simulations de $\mathcal{P}(\xi)$. Donc elle est fermée par réflexivité, par symétrie, et par transitivité, ce qui en fait une relation d'équivalence. C'est donc une bisimulation. En résumé, $\xi$ admet une plus grande bisimulation. $\diamondsuit$

**Définition 27** (SYSTÈME CANONIQUE) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\xi$ un système bien formé, et $z$ une de ses étiquettes.

- On désigne par $\widehat{z}$ la **classe d'équivalence** de $z$ par rapport à $\dot{\sim}_\xi$.
- On distingue un élément dans chacune des classes d'équivalence. On note $\overline{z}$ l'**étiquette distinguée** de $\widehat{z}$.
- On définit le **système canonique** associé à $\xi$, noté $\overline{\xi}$, en posant :

$$\begin{array}{rcl} \overline{\xi} & = & \{\overline{z} = x \mid z = x \in \xi\} \\ & \cup & \{\overline{z} = c \mid z = c \in \xi\} \\ & \cup & \{\overline{z} = f(\overline{z_1}, \ldots, \overline{z_n}) \mid z = f(z_1, \ldots, z_n) \in \xi\} \end{array}$$

La transformation $\Phi$ et le théorème de Kleene nous donne un algorithme pour calculer la plus grande bisimulation. Notez que [16] ont donné un algorithme en $O(n \log m)$ pour un automate à $n$ états et $m$ transitions.

### 2.4.2 Bisimulation sur les termes

**Définition 28** (TERMES BISIMILAIRES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t_1$, $t_2$ deux termes. Soient $t'_1$ , $t'_2$ tels que

1. $t'_1 =_\alpha t_1$
2. $t'_2 =_\alpha t_2$
3. $\mathrm{Label}(t'_1) \cap \mathrm{Label}(t'_2) = \emptyset$

Soit $\xi = \mathrm{System}(t'_1) \cup \mathrm{System}(t'_2)$.

On dit que $t_1$ et $t_2$ sont **bisimilaires**, et on note $t_1 \doteq t_2$ si leurs racines sont équivalentes dans la plus grande bisimulation de $\xi$ :

$$t_1 \doteq t_2 \iff \mathrm{Rac}(t'_1) \dot{\sim}_\xi \mathrm{Rac}(t'_2)$$

**Proposition 29** : $\doteq$ est une congruence sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$.

**Preuve** : La réflexivité, la symétrie et la transitivité sont assez immédiates. Montrons que $\doteq$ est une congruence. Soient $f \in \Omega_{s_1 \ldots s_n, s}$, $t_i \doteq t'_i \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_{s_i}$ des termes, et $u, v \in \mathcal{L}_s$. Sans perte de généralité, nous supposons que les termes ont des ensembles d'étiquettes disjoints deux à deux. Soient $\xi_i = \mathrm{System}(t_i)$, et $\xi'_i = \mathrm{System}(t'_i)$. Soit

$$\xi = \{u = f(\mathrm{Rac}(t_1), \ldots, \mathrm{Rac}(t_n)), v = f(\mathrm{Rac}(t'_1), \ldots, \mathrm{Rac}(t'_n))\} \cup \bigcup_{i=1}^{n} (\xi_i \cup \xi'_i)$$

Clairement, $\dot{\sim}_{(\xi_i \cup \xi'_i)} \subseteq \dot{\sim}_\xi$. Or, par hypothèse, $\mathrm{Rac}(t_i) \dot{\sim}_{(\xi_i \cup \xi'_i)} \mathrm{Rac}(t'_i)$. Donc $\mathrm{Rac}(t_i) \dot{\sim}_\xi \mathrm{Rac}(t'_i)$. D'où, par définition de $\dot{\sim}_\xi$, $u \dot{\sim}_\xi v$, ce qui implique finalement : $u : f(t_1, \ldots, t_n) \doteq v : f(t'_1, \ldots, t'_n)$.
$\diamondsuit$

- On note $\widehat{t}$ la classe d'équivalence du terme $t$. On note :

$$\mathcal{T}(\widehat{\Sigma, \mathcal{L}}, \mathcal{X}) = \biguplus_{s \in \mathcal{S}} \mathcal{T}_s(\widehat{\Sigma, \mathcal{L}}, \mathcal{X})$$

**l'ensemble des termes quotienté par la relation de bisimulation.**

- On définit le **terme canonique** associé à $t$, unique à l'$\alpha$-conversion près, noté $\overline{t}$, en posant :

$$\overline{t} = \mathrm{Term}(\overline{\mathrm{Rac}(t)}, \overline{\mathrm{System}(t)})$$

On note :

$$\overline{\mathcal{T}}(\Sigma, \mathcal{L}, \mathcal{X}) = \biguplus_{s \in \mathcal{S}} \overline{\mathcal{T}}(\Sigma, \mathcal{L}, \mathcal{X})_s$$

**l'ensemble des termes canoniques.**

**Remarque** :

1. $=_\alpha \ \subset \doteq$
2. $\overline{t} \in \widehat{t}$
3. Si $t_1 \doteq t_2$, alors $t[x \leftarrow t_1] \doteq t[x \leftarrow t_2]$
4. Si $t_1 \doteq t_2$ et $u \in \mathrm{Pos}(t_1)$ alors il existe $v \in \mathrm{Pos}(t_2)$ telle que $t_{1\uparrow u} \doteq t_{2\uparrow v}$
5. $t \doteq \mathrm{Term}(\mathrm{Rac}(t), \mathrm{System}(t))$
6. $t \doteq t[x \leftarrow t_{\uparrow x}]$

$\diamondsuit$

## 2.5 Problème d'unification

Dans cette dernière section, nous nous intéressons au problème d'unification, c'est-à-dire à la résolution d'équations syntaxiques sur nos graphes. Nous commençons par définir les substitutions, puis nous posons le problème d'unification de deux termes, et nous donnons un algorithme d'unification.

### 2.5.1 Substitution

Informellement, une substitution est une application qui associe des termes à des variables libres. Son extension à l'ensemble des termes est unique à la bisimulation près. Le reste des notions est classique.

**Définition 30** (SUBSTITUTION) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes.

- On appelle **substitution** toute application $\sigma : \mathcal{X} \to \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ telle que $\sigma(\mathcal{X}_s) \subseteq \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_s$. On note **SUB** l'ensemble des substitutions.
- L'ensemble $\mathrm{Dom}(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq z : x\}$ est appelé **domaine** de la substitution $\sigma$. On note Id la substitution de domaine vide.
- L'ensemble $\mathrm{Im}(\sigma) = \bigcup_{x \in \mathrm{Dom}(\sigma)} \mathrm{Var}(\sigma(x))$ est appelé **image** de la substitution $\sigma$ ; c'est l'ensemble des variables libres introduites par $\sigma$. On dit que $\sigma$ est **fermée** si $\mathrm{Im}(\sigma) = \emptyset$. On note **SUBF** l'ensemble des substitutions fermées.

**Définition 31** (RELATION $\doteqdot$) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\sigma_1$ et $\sigma_2$ deux substitutions. On étend la relation de bisimulation définie sur les termes en une relation d'équivalence sur les substitutions. On définit ainsi $\doteqdot$ en posant :

$$\sigma_1 \doteqdot \sigma_2 \iff \forall x \in \mathcal{X}, \sigma_1(x) \doteq \sigma_2(x)$$

**Définition 32** (EXTENSION DES SUBSTITUTIONS) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\sigma$ une substitution, $t$ un terme, et $M \subseteq \mathcal{L}$ un ensemble d'étiquettes contenant $L = \mathrm{Label}(t)$. On définit l'**extension de $\sigma$ à $t$ en dehors de $M$**, notée $\sigma^\sharp$, en posant :

$$t \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{\sigma^\sharp(t) = \sigma_M(t)}$$

où :

1. $\sigma_M(z) = z$
2. $\sigma_M(z : x) = t_3$ avec :
   (a) $t_1 = \sigma(x)$
   (b) $t_2 =_\alpha t_1$ avec $\mathrm{Label}(t_2) \cap M = \emptyset$
   (c) $t_3 = t_2[\mathrm{Rac}(t_2) := z]$
3. $\sigma_M(z : c) = z : c$
4. $\sigma_M(z : f(t_1, \ldots, t_n)) = z : f(\sigma_{M_1}(t_1), \ldots, \sigma_{M_i}(t_i), \ldots, \sigma_{M_n}(t_n))$ avec $M_1 = M$ et $M_{j+1} = M_j \cup \mathrm{Label}(\sigma(t_j))$

Par abus de notation, nous écrirons $\sigma$ à la place de $\sigma^\sharp$.

**Exemple** : Soit $\sigma$ telle que $\sigma(x) = \sigma(y) = x_1 : h(x_1, x_2 : z)$. On a $\mathrm{Dom}(\sigma) = \{x, y\}$ et $\mathrm{Im}(\sigma) = \{z\}$. Soit $t = x_1 : f(x_2 : x, x_3 : y)$. On a :

$$\sigma(t) = x_1 : f(x_2 : h(x_2, x_4 : z), x_3 : h(x_3, x_5 : z))$$

$\diamondsuit$

**Définition 33** (Restriction d'une substitution) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\sigma$ une substitution, et $V \subseteq \mathcal{X}$ un ensemble de variables. On note $\sigma_{\uparrow V}$ la **restriction** de $\sigma$ à l'ensemble $V$, définie par $\sigma_{\uparrow V}(x) = \sigma(x)$ si $x \in V$, et $\sigma_{\uparrow V}(x) = z : x$ sinon.

**Définition 34** (Composition , idempotence) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature. Soient $\sigma_1$ et $\sigma_2$ deux substitutions.

- On note $\sigma_1 \circ \sigma_2$ la **composition** des substitutions $\sigma_1$ et $\sigma_2$.
- On dit qu'une substitution $\sigma$ est **idempotente** si $\sigma \circ \sigma \doteqdot \sigma$.

**Proposition 35** : Soit $\sigma$ une substitution. $\sigma \circ \sigma \doteqdot \sigma \iff \mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma) = \emptyset$.

**Preuve** :

- $\Longleftarrow$ : Supposons $\mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma) = \emptyset$. On a : $\sigma \circ \sigma \doteqdot \sigma_{\uparrow \mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma)} \circ \sigma \doteqdot \sigma$.
- $\Longrightarrow$ : Supposons $\sigma \circ \sigma \doteqdot \sigma$. Si $\mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma) \neq \emptyset$, alors il existe $x \in \mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma)$. Comme $x \in \mathrm{Im}(\sigma)$, il existe $y \in \mathrm{Dom}(\sigma)$, $t \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$, et $u \in \mathrm{Pos}(t)$ tels que $\sigma(y) \doteq t[u \leftarrow x]$. Mais $x \in \mathrm{Dom}(\sigma)$, donc par définition, $\sigma(x) \neq z : x$, donc $\sigma(\sigma(y)) \neq \sigma(y)$ : contradiction.

$\diamondsuit$

### 2.5.2 Problème d'unification

Dans ce paragraphe, nous cherchons à résoudre des équations syntaxiques sur les termes, c'est-à-dire à trouver tous les unificateurs qui rendent bisimilaires deux termes. Après une définition plus formelle, nous donnons un algorithme permettant de calculer l'unificateur le plus général à la bisimulation près.

**Définition 36** (Unification) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t_1, t_2$ deux termes.

- On dit que $t_1$ et $t_2$ sont **unifiables** s'il existe une substitution $\sigma$ telle que $\sigma(t_1) \doteq \sigma(t_2)$. On dit que $\sigma$ est un **unificateur**.
- Trouver un unificateur, c'est résoudre le **problème d'unification**.
- On note $\mathbf{U}(t_1, t_2)$ l'ensemble des unificateurs de $t_1$ et $t_2$.

**Définition 37** (Préordre $\dot{\preceq}$) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature. Soient $\sigma_1$ et $\sigma_2$ deux substitutions. On définit le préordre $\dot{\preceq}$ sur les substitutions à partir de la relation d'équivalence $\doteqdot$ en posant :

$$\sigma_1 \dot{\preceq} \sigma_2 \iff \exists \delta \in \mathbf{SUB} \text{ tel que } \delta \circ \sigma_1 \doteqdot \sigma_2$$

**Théorème 38** : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $t_1, t_2$ deux termes.

1. Si $t_1$ et $t_2$ sont unifiables, alors il existe une substitution $\sigma \in \mathbf{U}(t_1, t_2)$ telle que :

$$\forall \sigma' \in \mathbf{U}(t_1, t_2) \ , \sigma \dot{\preceq} \sigma'$$

   $\sigma$ est unique à la relation de bisimulation $\dot{\equiv}$ près. On l'appelle **plus petit unificateur** de $t_1$ et $t_2$, ou **unificateur le plus général**[1] de $t_1$ et $t_2$.

2. De plus, on peut toujours choisir $\sigma$ idempotent ($\mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma) = \emptyset$). Et si $\mathrm{Var}(t_1) \cap \mathrm{Var}(t_2) = \emptyset$, alors on a : $\mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma) = \mathrm{Var}(t_1) \cup \mathrm{Var}(t_2)$.

**Exemple** : Considérons les termes $t_1 = x_1 : +(v_1 : x, x_1)$ et $t_2 : +(x_2, v_2 : x)$. Les substitutions $\sigma : x \mapsto y_1 : +(y_1, y_1)$ et $\sigma' : x \mapsto y_2 : +(y_2, y_3 : +(y_3, y_3))$ sont deux unificateurs les plus généraux, égaux entre eux (au sens de $\dot{\equiv}$). $\diamondsuit$

### 2.5.3 Algorithme d'unification

Nous donnons ici les règles d'inérence de l'algorithme d'unification :

- Problème : Soient $t_1$ , $t_2$ deux termes tels que $\mathrm{Label}(t_1) \cap \mathrm{Label}(t_2) = \emptyset$. Soit $\xi = \mathrm{System}(t_1) \cup \mathrm{System}(t_2)$. Le problème d'unification de $t_1$ et $t_2$ se pose ainsi :

$$(\{\overline{\mathrm{Rac}(t_1)} \diamond \overline{\mathrm{Rac}(t_2)}\} \ ; \ \overline{\xi} \ ; \ \emptyset)$$

- Règles :

1. Élimination :

$$\frac{(\Delta \cup \{x_1 \diamond x_1\} \ ; \ \xi \ ; \ S)}{(\Delta \ ; \ \xi \ ; \ S)}$$

2. Substitution 1 :

$$\frac{(\Delta \cup \{x_1 \diamond x_2\} \ ; \ \xi \cup \{x_1 = x, x_2 = y\} \ ; \ S)}{(\Delta \ ; \ \xi \cup \{x_1 = x\} \ ; \ S \cup \{x_1 = y\})[x_2 := x_1]}$$

3. Substitution 2 :

$$\frac{(\Delta \cup \{x_1 \diamond x_2\} \ ; \ \xi \cup \{x_1 = x, x_2 = \phi\} \ ; \ S)}{(\Delta \ ; \ \xi \cup \{x_2 = \phi\} \ ; \ S \cup \{x_2 = x\})[x_1 := x_2]} \ , \ \phi \notin \mathcal{X}$$

4. Substitution 3 :

$$\frac{(\Delta \cup \{x_1 \diamond x_2\} \ ; \ \xi \cup \{x_1 = \phi, x_2 = x\} \ ; \ S)}{(\Delta \ ; \ \xi \cup \{x_1 = \phi)\} \ ; \ S \cup \{x_1 = x\})[x_2 := x_1]} \ , \ \phi \notin \mathcal{X}$$

5. Décomposition :

$$\frac{(\Delta \cup \{x_1 \diamond x_2\} \ ; \ \xi \cup \{x_1 = f(u_1, \ldots, u_n), x_2 = f(v_1, \ldots, v_n)\} \ ; \ S)}{(\Delta \cup \{u_1 \diamond v_1, \ldots, u_n \diamond v_n\} \ ; \ \xi \cup \{x_1 = f(u_1, \ldots, u_n)\} \ ; \ S)[x_2 := x_1]}$$

[1] most general unifier, en anglais.

6. Échec 1 :
$$\frac{(\Delta \cup \{x_1 \diamond x_2\} \ ; \ \xi \cup \{x_1 = f\vec{M}, x_2 = g\vec{N}\} \ ; \ S)}{\mathcal{FAIL}}$$
7. Échec 2 :
$$\frac{(\Delta \ ; \ \xi \ ; \ S)}{\mathcal{FAIL}}$$
si $\exists s \in \mathcal{S}_{\infty}$ , $x \in \text{Label}_s(\xi)$ tels que $x \rightsquigarrow_\xi{}^+ x$.

- On peut obtenir deux résultats :
  1. $\mathcal{FAIL}$ : $t_1$ et $t_2$ ne sont pas unifiables.
  2. $(\emptyset \ ; \ \xi \ ; \ S)$ : $t_1$ et $t_2$ sont unifiables. Un de leurs unificateurs le plus général est $\sigma$ tel que :
$$\sigma = \{x \mapsto \text{Term}(x_i, \xi) \mid (x_i = x) \in S\}$$

# Chapter 3

# Sémantiques

Dans ce chapitre, nous nous intéressons à la sémantique dénotationnelle et à la sémantique opérationnelle d'une spécification. La première exprime le sens mathématique unique qu'a cette spécification. La seconde explique comment exécuter correctement cette spécification, considérée comme un programme.

## A . Sémantique dénotationnelle

Dans cette partie, nous proposons une sémantique dénotationnelle pour toute spécification. Elle est donnée par son E-modèle standard, qu'il s'agit de définir. D'abord, il faut remarquer que nous avons identifié des termes en définissant la congruence $\doteq$, dite de bisimulation. Celle-ci n'est pas spécifiable dans notre syntaxe. Donc nous travaillons directement avec l'ensemble des termes quotienté par $\doteq$ : $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})$.

Dans un premier temps, nous nous intéressons aux spécifications dont la signature n'est pas égalitaire. Nous définissons d'abord les notions d'interprétation et de satisfaction. Nous définissons ensuite ce qu'est un modèle, puis nous parlons du modèle standard, intersection de tous les modèles.

Lorsqu'on travaille avec une signature égalitaire, ce modèle standard n'est plus satisfaisant : on y manipule le symbole d'égalité comme un prédicat quelconque (à la Prolog). Pour pallier cet inconvénient, nous définissons les E-interprétations, puis et les E-modèles. Le E-modèle standard est l'intersection de tous les E-modèles.

### 3.1 Modèle standard

Dans cette section, nous nous intéressons aux spécifications dont la signature n'est pas égalitaire. Nous ne développons que les notions utiles pour exprimer les calculs, la sémantique opérationnelle d'une spécification. Nous définissons d'abord les notions d'interprétation et de satisfaction. Nous définissons ensuite ce qu'est un modèle, puis nous parlons du modèle standard, intersection de tous les modèles. La sémantique d'une spécification non égalitaire est donnée par ce dernier.

### 3.1.1 Interprétations syntaxiques et satisfaction

**Définition 39** (INTERPRÉTATION SYNTAXIQUE) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On appelle **interprétation syntaxique** tout triplet

$$\mathcal{IH} = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}}) \ , \alpha, \beta \rangle$$

tel que :

- $\mathcal{T}(\widehat{\Sigma, \mathcal{L}}) = \biguplus_{s \in \mathcal{S}} \mathcal{T}_s(\widehat{\Sigma, \mathcal{L}})$ soit l'ensemble des termes fermés , quotienté par la relation de bisimulation $\doteq$ définie précédemment.
- $\alpha$ soit une famille $\mathcal{S}^\star \times \mathcal{S}$-indicée d'applications :

$$\alpha = \{\alpha_{w,s} : \Omega_{w,s} \longrightarrow \ \mathcal{T}_s(\widehat{\Sigma, \mathcal{L}})^{\mathcal{T}_w(\widehat{\Sigma, \mathcal{L}})} \mid (w, s) \in \mathcal{S}^\star \times \mathcal{S}\}$$

avec :

1.
$$\left.\begin{array}{l} c \in \Omega_{\lambda,s} \\ z \in \mathcal{L}_s \end{array}\right\} \Longrightarrow \boxed{\alpha_{\lambda,s}(c) = \widehat{z : c}}$$

2.
$$\left.\begin{array}{l} f \in \Omega_{s_1 \ldots s_n, s} \\ \widehat{t_i} \in \mathcal{T}_{s_i}(\widehat{\Sigma, \mathcal{L}}) \\ z \in \mathcal{L}_s \end{array}\right\} \Longrightarrow \boxed{\alpha_{s_1 \ldots s_n, s}(f)(\widehat{t_1}, \ldots, \widehat{t_n}) = z : \widehat{f(t_1, \ldots, t_n)}}$$

- $\beta$ soit une famille $\mathcal{S}^+$-indicée d'applications :

$$\beta = \{\beta_w : \Pi_w \longrightarrow \ \mathbf{2}^{\mathcal{T}_w(\widehat{\Sigma, \mathcal{L}})} \mid w \in \mathcal{S}^+\}$$

avec :

$$P \in \Pi_{s_1 \ldots s_n} \implies \beta(P) \subseteq \mathcal{T}_{s_1}(\widehat{\Sigma, \mathcal{L}}) \times \ldots \times \mathcal{T}_{s_n}(\widehat{\Sigma, \mathcal{L}})$$

**Remarque** :

- $\mathcal{T}_{s_1 \ldots s_n}(\widehat{\Sigma, \mathcal{L}})$ désigne le produit cartésien $\mathcal{T}_{s_1}(\widehat{\Sigma, \mathcal{L}}) \times \ldots \times \mathcal{T}_{s_n}(\widehat{\Sigma, \mathcal{L}})$.
- Un symbole de fonction est interprété de manière "syntaxique" , comme dans les interprétations de Herbrand.
- A tout prédicat $P \in \Pi_{s_1 \ldots s_n}$ , on associe une partie de $\mathcal{T}_{s_1}(\widehat{\Sigma, \mathcal{L}}) \times \ldots \times \mathcal{T}_{s_n}(\widehat{\Sigma, \mathcal{L}})$.

$\diamondsuit$

**Définition 40** (AFFECTATION SUR $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})$) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On appelle **affectation sur** $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})$ toute application $a : \mathcal{X} \to \mathcal{T}(\widehat{\Sigma, \mathcal{L}})$ telle que $a(\mathcal{X}_s) \subseteq \mathcal{T}_s(\widehat{\Sigma, \mathcal{L}})$. On définit en deux temps l'unique **extension** de $a$ à l'ensemble des termes $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$, notée $a^\sharp$ :

1. Soit $\sigma_a : \mathcal{X} \to \mathcal{T}(\Sigma, \mathcal{L})$ une substitution telle que $\forall x \in \mathcal{X}, a(x) = \widehat{\sigma_a(x)}$

2. On pose :

$$t \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{a^\sharp(t) = \widehat{\sigma_a(t)}}$$

**Définition 41** (SATISFACTION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soit $\mathcal{IH} = \langle \widehat{\mathcal{T}(\Sigma, \mathcal{L})}, \alpha, \beta \rangle$ une interprétation syntaxique.

- Soient $a : \mathcal{X} \to \widehat{\mathcal{T}(\Sigma, \mathcal{L})}$ une affectation et $P(t_1, \ldots, t_n)$ un atome. On dit que $P(a^\sharp(t_1), \ldots, a^\sharp(t_n))$ est **vrai** dans $\mathcal{IH}$ si $(a^\sharp(t_1), \ldots, a^\sharp(t_n)) \in \beta(P)$. Par abus de notation, on pose $a^\sharp(P(t_1, \ldots, t_n)) = P(a^\sharp(t_1), \ldots, a^\sharp(t_n))$.
- Soient $A_0, \ldots, A_n$ des atomes.
  - On dit que **le fait** $A_0$ **est valide** dans $\mathcal{IH}$ si pour toute affectation $a$, $a^\sharp(A_0)$ est vrai dans $\mathcal{IH}$.
  - On dit que **la clause** $A_0 \Leftarrow A_1 \ldots A_n$ **est valide** dans $\mathcal{IH}$ si pour toute affectation $a$, $a^\sharp(A_0)$ est vrai dans $\mathcal{IH}$ chaque fois que $a^\sharp(A_1), \ldots, a^\sharp(A_n)$ sont vrais dans $\mathcal{IH}$.

  On note par $\mathcal{IH} \models c$ le fait que $c$ soit une clause valide dans l'interprétation syntaxique $\mathcal{IH}$.

### 3.1.2 Modèles syntaxiques et modèle standard

**Définition 42** (MODÈLE SYNTAXIQUE) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification, et $\mathcal{IH} = \langle \widehat{\mathcal{T}(\Sigma, \mathcal{L})}, \alpha, \beta \rangle$ une interprétation syntaxique. On dit que $\mathcal{IH}$ est un **modèle syntaxique** de $\mathcal{SP}$ si :

$$\forall c \in \mathcal{HC} \ , \ \mathcal{IH} \models c$$

**Remarque** : La classe des modèles syntaxiques correspond à la classe des modèles librement engendrés de la théorie classique. $\diamondsuit$

**Définition 43** (INTERSECTION DE MODÈLES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $\mathcal{IH}_1 = \langle \widehat{\mathcal{T}(\Sigma, \mathcal{L})}, \alpha, \beta_1 \rangle$ , $\mathcal{IH}_2 = \langle \widehat{\mathcal{T}(\Sigma, \mathcal{L})}, \alpha, \beta_2 \rangle$ deux modèles syntaxiques de $\mathcal{SP}$. On appelle **intersection de** $\mathcal{IH}_1$ **et** $\mathcal{IH}_2$ l'interprétation syntaxique :

$$\mathcal{IH} = \mathcal{IH}_1 \cap \mathcal{IH}_2 = \langle \widehat{\mathcal{T}(\Sigma, \mathcal{L})}, \alpha, \beta \rangle$$

telle que :

$$\forall P \in \Pi_w \ , \ \beta(P) = \beta_1(P) \cap \beta_2(P)$$

**Proposition 44** : L'intersection de deux modèles syntaxiques est encore un modèle syntaxique.

**Preuve** :

- Soit $A_0$ un fait de $\mathcal{HC}$. Soit $a$ une affectation. $\mathcal{IH}_1$ et $\mathcal{IH}_2$ étant deux modèles syntaxiques de $\mathcal{SP}$, on a $a^\sharp(A_0) \in \beta_1(P)$ et $a^\sharp(A_0) \in \beta_2(P)$ et donc $a^\sharp(A_0) \in \beta_1(P) \cap \beta_2(P)$. Comme $a$ est quelconque, on a finalement : $\mathcal{IH}_1 \cap \mathcal{IH}_2 \models A_0$

- Soit $A_0 \Longleftarrow A_1 \ldots A_n$ une clause de $\mathcal{HC}$. Soit $a$ une affectation telle que $a^\sharp(A_0) \in \beta_1(P) \cap \beta_2(P)$. On a donc $a^\sharp(A_0) \in \beta_1(P)$ et $a^\sharp(A_0) \in \beta_2(P)$. Comme $\mathcal{IH}_1$ et $\mathcal{IH}_2$ sont deux modèles syntaxiques de $\mathcal{SP}$, on a : $\forall i \in 1 \ldots n, a^\sharp(A_i) \in \beta_1(P)$ et $a^\sharp(A_i) \in \beta_2(P)$. Donc : $\forall i \in 1 \ldots n, a^\sharp(A_i) \in \beta_1(P) \cap \beta_2(P)$, et donc finalement : $\mathcal{IH}_1 \cap \mathcal{IH}_2 \models A_0 \Longleftarrow A_1 \ldots A_n$.

$\diamondsuit$

**Définition 45** (MODÈLE STANDARD) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. On appelle **plus petit modèle syntaxique de $\mathcal{SP}$** ou **modèle standard de $\mathcal{SP}$** l'intersection de tous les modèles syntaxiques de $\mathcal{SP}$. On le note $\mathcal{MS}_{\mathcal{SP}}$.

## 3.2 E-modèle standard

Nous nous intéressons maintenant aux spécifications dont la signature est égalitaire. Nous définissons dans un premier temps la notion de E-interprétation, ainsi que la relation de satisfaction. Nous définirons ensuite ce qu'est un E-modèle, puis nous parlerons du E-modèle standard, l'intersection de tous les E-modèles. La sémantique d'une spécification égalitaire est donnée par ce dernier.

### 3.2.1 E-interprétations syntaxiques et satisfaction

**Définition 46** (AXIOMES DE L'ÉGALITÉ) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. On définit l'ensemble des **axiomes de l'égalité sur** $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})$ pour la signature $\Sigma$, noté $\mathcal{AXE}(\Sigma)$, l'ensemble des clauses[1] suivantes :

1. $x == x$
2. $x == y \Longleftarrow y == x$
3. $x == z \Longleftarrow x == y \ , \ y == z$
4. $z_1 : f(x_1, \ldots, x_n) == z_2 : f(y_1, \ldots, y_n) \Longleftarrow x_1 == y_1, \ldots, x_n == y_n$ pour tous les symboles d'opérateurs $f \in \Omega_{s_1 \ldots s_n, s}$
5. $P(x_1, \ldots, x_n) \Longleftarrow P(y_1, \ldots, y_n), x_1 == y_1, \ldots, x_n == y_n$ pour tous les symboles de prédicats $P \in \Pi_{s_1 \ldots s_n}$ différents de l'égalité ($\{=_{ss} | \ s \in \mathcal{S}\}$).

**Proposition 47** : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire , $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. On définit la spécification :

$$\mathcal{SP}' = \langle \Sigma, \mathcal{HC} \cup \mathcal{AXE}(\Sigma) \rangle$$

Soit $\mathcal{MS}_{\mathcal{SP}'} = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}}) \ , \alpha, \beta \rangle$ le modèle standard de $\mathcal{SP}'$. On définit

$$\cong \ = \{\cong_s | \ s \in \mathcal{S}\} \ , \text{ où } \ \cong_s \ = \ \beta(=_{ss})$$

$\cong$ est la plus petite congruence engendrée par la spécification $\mathcal{SP}$ sur $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})$. On définit $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong} = \biguplus_{s \in \mathcal{S}} \mathcal{T}_s(\widehat{\Sigma, \mathcal{L}})_{/\cong}$ l'ensemble tel que $\mathcal{T}_s(\widehat{\Sigma, \mathcal{L}})_{/\cong}$ soit le quotient de $\mathcal{T}_s(\widehat{\Sigma, \mathcal{L}})$ par $\cong_s$. Les éléments de $\mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}$ sont notés $[\ \hat{t}\ ]$.

[1] == désigne n'importe quel symbole d'égalité $=_{ss}$

**Preuve** : $\cong$ est une congruence sur les termes de $\mathcal{T}(\widehat{\Sigma,\mathcal{L}})$, compte tenu des axiomes qu'elle satisfait. C'est la plus petite parce qu'on travaille dans le plus petit modèle syntaxique de $\mathcal{SP}'$. $\diamondsuit$

**Définition 48** (E-INTERPRÉTATION SYNTAXIQUE) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On appelle **E-interprétation syntaxique** tout triplet

$$\mathcal{EIH} = \langle \mathcal{T}(\widehat{\Sigma,\mathcal{L}})_{/\cong}, \alpha, \beta \rangle$$

tel que :

- $\alpha$ soit une famille $\mathcal{S}^\star \times \mathcal{S}$-indicée d'applications :

$$\alpha = \{\alpha_{w,s} : \Omega_{w,s} \longrightarrow \mathcal{T}_s(\widehat{\Sigma,\mathcal{L}})_{/\cong}^{\mathcal{T}_w(\widehat{\Sigma,\mathcal{L}})_{/\cong}} \mid (w,s) \in \mathcal{S}^\star \times \mathcal{S}\}$$

avec :

1. $$\left.\begin{array}{l} c \in \Omega_{\lambda,s} \\ z \in \mathcal{L}_s \end{array}\right\} \Longrightarrow \boxed{\alpha_{\lambda,s}(c) = [\ \widehat{z : c}\ ]}$$

2. $$\left.\begin{array}{l} f \in \Omega_{s_1 \ldots s_n, s} \\ [\ \widehat{t_i}\ ] \in \mathcal{T}_{s_i}(\widehat{\Sigma,\mathcal{L}})_{/\cong} \\ z \in \mathcal{L}_s \end{array}\right\} \Longrightarrow \boxed{\alpha_{s_1 \ldots s_n, s}(f)([\ \widehat{t_1}\ ], \ldots, [\ \widehat{t_n}\ ]) = [\ z : \widehat{f(t_1, \ldots, t_n)}\ ]}$$

- $\beta$ soit une famille $\mathcal{S}^+$-indicée d'applications :

$$\beta = \{\beta_w : \Pi_w \longrightarrow \mathbf{2}^{\mathcal{T}_w(\widehat{\Sigma,\mathcal{L}})_{/\cong}} \mid w \in \mathcal{S}^+\}$$

avec :

1. $\boxed{\beta(=_{ss}) = \{(x,x) \mid x \in \mathcal{T}_s(\widehat{\Sigma,\mathcal{L}})_{/\cong}\}}$
2. $P \in \Pi_{s_1 \ldots s_n} \implies \beta(P) \subseteq \mathcal{T}_{s_1}(\widehat{\Sigma,\mathcal{L}})_{/\cong} \times \ldots \times \mathcal{T}_{s_n}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$

**Remarque** :

- On notera que l'interprétation des sortes se fait dans $\mathcal{T}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$.
- $\mathcal{T}_{s_1 \ldots s_n}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$ désigne un produit cartésien $\mathcal{T}_{s_1}(\widehat{\Sigma,\mathcal{L}})_{/\cong} \times \ldots \times \mathcal{T}_{s_n}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$.
- Un symbole de fonction est interprété de manière "syntaxique", comme dans les E-structures de Herbrand.
- A tout prédicat $P \in \Pi_{s_1 \ldots s_n}$ , on associe une partie de $\mathcal{T}_{s_1}(\widehat{\Sigma,\mathcal{L}})_{/\cong} \times \ldots \times \mathcal{T}_{s_n}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$.

$\diamondsuit$

**Définition 49** (AFFECTATION SUR $\mathcal{T}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On appelle **affectation sur** $\mathcal{T}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$ toute application $a : \mathcal{X} \to \mathcal{T}(\widehat{\Sigma,\mathcal{L}})_{/\cong}$ telle que $a(\mathcal{X}_s) \subseteq \mathcal{T}_s(\widehat{\Sigma,\mathcal{L}})_{/\cong}$. On définit en deux temps l'unique **extension** de $a$ à l'ensemble des termes $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$, notée $a^\sharp$ :

1. Soit $\sigma_a : \mathcal{X} \to \mathcal{T}(\Sigma, \mathcal{L})$ une substitution telle que $\forall x \in \mathcal{X}, a(x) = [\ \widehat{\sigma_a(x)}\ ]$
2. On pose :

$$t \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{a^\sharp(t) = [\ \widehat{\sigma_a(t)}\ ]}$$

**Définition 50** (SATISFACTION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soit $\mathcal{EIH} = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}, \alpha, \beta \rangle$ une E-interprétation syntaxique.

- Soient $a : \mathcal{X} \to \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}$ une affectation et $P(t_1, \ldots, t_n)$ un atome. On dit que $P(a^\sharp(t_1), \ldots, a^\sharp(t_n))$ est **vrai** dans $\mathcal{EIH}$ si $(a^\sharp(t_1), \ldots, a^\sharp(t_n)) \in \beta(P)$. par abus de notation, on pose $a^\sharp(P(t_1, \ldots, t_n)) = P(a^\sharp(t_1), \ldots, a^\sharp(t_n))$.
- Soient $A_0, \ldots, A_n$ des atomes.
  - On dit que **le fait** $A_0$ **est valide** dans $\mathcal{EIH}$ si pour toute affectation $a$, $a^\sharp(A_0)$ est vrai dans $\mathcal{EIH}$.
  - On dit que **la clause** $A_0 \Longleftarrow A_1 \ldots A_n$ **est valide** dans $\mathcal{EIH}$ si pour toute affectation $a$, $a^\sharp(A_0)$ est vrai dans $\mathcal{EIH}$ chaque fois que $a^\sharp(A_1), \ldots, a^\sharp(A_n)$ sont vrais dans $\mathcal{EIH}$.

  On note par $\mathcal{EIH} \models c$ le fait que $c$ soit une clause valide dans l'E-interprétation syntaxique $\mathcal{EIH}$.

### 3.2.2 E-modèles syntaxiques et E-modèle standard

**Définition 51** (E-MODÈLES SYNTAXIQUES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. Soient $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification, et $\mathcal{EIH} = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}, \alpha, \beta \rangle$ une E-interprétation syntaxique. On dit que $\mathcal{EIH}$ est un **E-modèle syntaxique** de $\mathcal{SP}$ si :

$$\forall c \in \mathcal{HC}\ ,\ \mathcal{EIH}\ \models c$$

**Définition 52** (INTERSECTION DE E-MODÈLES) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $\mathcal{EIH}_1 = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}, \alpha, \beta_1 \rangle$ , $\mathcal{EIH}_2 = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}, \alpha, \beta_2 \rangle$ deux E-modèles syntaxiques de $\mathcal{SP}$. On appelle **intersection de** $\mathcal{EIH}_1$ **et** $\mathcal{EIH}_2$ la E-interprétation syntaxique :

$$\mathcal{EIH}\ = \mathcal{EIH}_1\ \cap\ \mathcal{EIH}_2 = \langle \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}, \alpha, \beta \rangle$$

telle que :

$$\forall P \in \Pi_w\ ,\ \beta(P)\ =\ \beta_1(P)\ \cap\ \beta_2(P)$$

**Proposition 53** : L'intersection de deux E-modèles syntaxiques est encore un E-modèle syntaxique.

**Preuve** : Aux notations près, la même que précédemment. $\diamondsuit$

**Définition 54** (E-MODÈLE STANDARD) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. On appelle **plus petit E-modèle syntaxique de** $\mathcal{SP}$ ou **E-modèle standard de** $\mathcal{SP}$ l'intersection de tous les E-modèles syntaxiques de $\mathcal{SP}$. On le note $\mathcal{EMS}_{\mathcal{SP}}$.

**Remarque** : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit $A \in \mathcal{AT}(\Sigma, \mathcal{L}, \mathcal{X})$ un atome. On pourrait montrer :

$$\mathcal{EMS}_{\mathcal{SP}} \models A \iff \mathcal{MS}_{\mathcal{SP}'} \models A$$

où $\mathcal{MS}_{\mathcal{SP}'}$ est le modèle standard de la spécification $\mathcal{SP}$ auquelle on a ajouté les axiomes de l'égalité (cf. définition de la congruence $\cong$). $\diamondsuit$

**Remarque** : Nous ne nous sommes pas lancer dans une étude générale, avec des modèles quelconques, parce que celà nécessiterait d'interpréter nos termes comme plus petits points fixes de systèmes d'équations dans des algèbres continues. Il nous faudrait alors travailler directement avec des arbres infinis réguliers (voir [6]), ce qui nécessiterait beaucoup de "matériels". On montrerait alors facilement que la sémantique d'un programme est donnée par un E-modèle initial dans la catégorie des E-modèles (voir [17] pour une bonne idée de la preuve). On pourrait aussi utiliser une transformation continue sur ces algèbres, comme celà a été fait pour Prolog. La sémantique d'un programme serait donnée par le plus petit point fixe de cette transformation (voir [18] pour des détails sur cette transformation). $\diamondsuit$

## 3.3 Congruence $=^{\mathbf{i}}_{\mathcal{SP}}$

Nous allons maintenant travailler dans le E-modèle standard. Nous commençons par définir la théorie équationnelle inductive, $=^{\mathbf{i}}_{\mathcal{SP}}$, c'est-à-dire l'ensemble des couples de termes qui sont égaux dans le E-modèle standard. Autrement dit, c'est l'ensemble des équations qui sont valides dans ce E-modèle. Cette relation est une congruence sur les termes. Nous l'étendons ensuite en une relation d'équivalence sur les substitutions, $\equiv^{\mathbf{i}}_{\mathcal{SP}}$, puis nous nous servons de cette dernière pour définir un préordre sur les substitutions, $\preceq^{\mathbf{i}}_{\mathcal{SP}}$. Les définitions introduites seront utilisées dans la prochaine partie.

**Définition 55** (RELATION $=^{\mathbf{i}}_{\mathcal{SP}}$) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. On définit la relation $=^{\mathbf{i}}_{\mathcal{SP}}$ sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ en posant :

$$\forall t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}), \boxed{t_1 =^{\mathbf{i}}_{\mathcal{SP}} t_2 \iff \mathcal{EMS}_{\mathcal{SP}} \models t_1 == t_2}$$

**Proposition 56** : $=^{\mathbf{i}}_{\mathcal{SP}}$ est une congruence sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$.

**Preuve** : Trivial, car si $\mathcal{EMS}_{\mathcal{SP}} \models t_1 == t_2$, alors pour toute affectation $a : \mathcal{X} \to \mathcal{T}(\widehat{\Sigma, \mathcal{L}})_{/\cong}$, $a^\sharp(t_1)$ est identique à $a^\sharp(t_2)$. $\diamondsuit$

**Définition 57** (RELATION $\equiv^{\mathbf{i}}_{\mathcal{SP}}$) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit $V \subseteq \mathcal{X}$ un ensemble de variables libres. Soient $\sigma_1$ et $\sigma_2$ deux substitutions. On étend la relation de congruence $=^{\mathbf{i}}_{\mathcal{SP}}$ définie sur les termes en une relation d'équivalence sur les substitutions. On définit ainsi $\equiv^{\mathbf{i}}_{\mathcal{SP}}$ en posant :

$$\sigma_1 \equiv^{\mathbf{i}}_{\mathcal{SP}} \sigma_2 \ [\ V\ ] \iff \forall x \in V, \sigma_1(x) =^{\mathbf{i}}_{\mathcal{SP}} \sigma_2(x)$$

**Définition 58** (PRÉORDRE $\preceq^{\mathbf{i}}_{\mathcal{SP}}$) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit $V \subseteq \mathcal{X}$ un ensemble de variables libres. Soient $\sigma_1$ et $\sigma_2$ deux substitutions. On définit le préordre $\preceq^{\mathbf{i}}_{\mathcal{SP}}$ sur les substitutions à partir de la relation d'équivalence $\equiv^{\mathbf{i}}_{\mathcal{SP}}$ en posant :

$$\sigma_1 \preceq^{\mathbf{i}}_{\mathcal{SP}} \sigma_2 \ [\ V\ ] \iff \exists \delta \in \mathbf{SUB} \text{ tel que } \delta \circ \sigma_1 \equiv^{\mathbf{i}}_{\mathcal{SP}} \sigma_2 \ [\ V\ ]$$

# B . Sémantique opérationnelle

Dans cette partie, nous allons décrire la sémantique opérationnelle d'un programme, soit la manière dont il s'exécute. Nous parlons de calcul. Dans la première partie, nous allons définir précisément ce qu'est un calcul, puis nous en proposons un : la SLDEI-résolution.

## 3.1 Notion de calcul

Informellement, un calcul correspond à la résolution de but :

**Définition 59** (BUT) : Soit $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature.

On appelle **but** toute conjonction d'atomes $A_0 \dots A_n$.

**Remarque** : Nous rappelons deux définitions qui ont été donnée par ailleurs :

1. On appelle **clause de but** la négation d'un but (notée $\Longleftarrow A_0 \dots A_n$).
2. On appelle **clause vide**, notée $\Box$, la clause dont l'ensemble des atomes est vide. C'est le symbole de la contradiction logique.

$\Diamond$

**Définition 60** (SOLUTION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit $B = A_0 \dots A_n$ un but. Une substitution $\sigma$ est une **solution** pour le but $B$ si pour tout $i$, $\sigma(A_i)$ est valide dans le E-modèle standard de $\mathcal{SP}$ :

$$\forall i \in 0 \dots n,\ \mathcal{EMS}_{\mathcal{SP}} \models \sigma(A_i)$$

On note $\mathbf{SOL}(\mathcal{SP}, B)$ l'ensemble de toutes les solutions possibles pour le but $B$.

**Définition 61** (ENSEMBLE COMPLET DE SOLUTIONS) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $B = A_0 \dots A_n$ un but, et $W$ un ensemble de variables libres contenant $V = \mathrm{Var}(B)$. Soit $S$ un ensemble de substitutions. On dit que $S$ est **un ensemble complet de solutions pour $B$ en dehors de $W$** si :

1. $S \subseteq \mathbf{SOL}(\mathcal{SP}, B)$ (cohérence)

2. $\forall \theta \in \mathbf{SUBF} \cap \mathbf{SOL}(\mathcal{SP}, B)$ , $\exists \sigma \in S$ tel que $\sigma \preceq^{\mathbf{i}}_{\mathcal{SP}} \theta[V]$ (complétude)
3. $\forall \sigma \in S, \mathrm{Dom}(\sigma) \subseteq V$ et $\mathrm{Im}(\sigma) \cap W = \emptyset$

**Remarque** : Nous nous intéressons à la complétude par rapport aux solutions fermées, la complétude par rapport aux solutions ouvertes étant indécidables dans le cas général. $\diamondsuit$

**Définition 62** (CALCUL) **:** On appelle **calcul** un ensemble de règles de déduction de substitutions pour les buts. On dit qu'un calcul est **cohérent** si pour tout but $B$, toutes les substitutions calculées sont effectivement solutions de $B$. On dit qu'un calcul est **complet** si pour tout but $B$, l'ensemble des substitutions calculées est complet (au sens de la définition précédente).

La règle de SLD-résolution est sans doute l'exemple le plus connu de calcul cohérent et complet. C'est le calcul effectué dans Prolog. Une extension a été faite pour pouvoir utiliser des termes infinis (voir [18] (chapitre 6)). Cette étude débouche sur la programmation logique temps réel, les processus communicants et concurrents ...

## 3.2 SLDEI-résolution

Nous proposons maintenant un premier calcul. Nous allons utiliser une extension de la SLD-résolution, permettant un traitement des équations indépendant ; nous nous inspirons de la SLDEI-résolution, définie R. Echahed dans [19]. Ce calcul ne marche pas dans le cadre le plus général, et nous allons donc poser des conditions quant aux clauses avec lesquelles nous travaillons.

Nous supposons dans tout ce paragraphe que nous disposons d'une signature égalitaire $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$, d'un ensemble $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ de variables, d'un ensemble $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ d'étiquettes, et d'une spécification $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ où l'ensemble des clauses $\mathcal{HC}$ est l'union d'un ensemble de clauses de têtes non équationnelles $\mathcal{CL}$, et d'un ensemble d'équations conditionnelles $\mathcal{E}$ :

$$\mathcal{HC} = \mathcal{CL} \uplus \mathcal{E}$$

avec :

$$\mathcal{E} = \{ g_i == d_i \iff g_i^1 == d_i^1, \ldots, g_i^{j_i} == d_i^{j_i} \mid i \in 1 \ldots k \}$$

Le calcul présenté traite de manière différente les équations et les autres littéraux des buts. Plus précisément, il permet de déléguer la résolution des équations à des algorithmes spécialisés.

Dans un premier temps, nous précisons ce que doit retourner un tel algorithme de résolution d'équations. Nous en présenterons un dans le chapitre suivant. Dans un deuxième temps, nous développons le calcul général.

### 3.2.1 Problème de EI-unification

Résoudre une équation, c'est trouver toutes les substitutions qui rendent deux termes égaux dans le E-modèle standard. Il s'agit plus précisément de trouver les substitutions qui valident une équation dans le E-modèle standard.

**Définition 63** (EI-UNIFICATION) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $t_1$ , $t_2$ deux termes.

- On dit que $t_1$ et $t_2$ sont **EI-unifiables** s'il existe une substitution $\sigma$ telle que $\sigma(t_1) =^{\mathbf{i}}_{\mathcal{SP}} \sigma(t_2)$, c'est-à-dire telle que $\mathcal{EMS}_{\mathcal{SP}} \models \sigma(t_1) == \sigma(t_2)$. On dit que $\sigma$ est un **EI-unificateur** de $t_1$ et $t_2$.
- Trouver un EI-unificateur, c'est résoudre le **problème de EI-unification**.
- On note $\mathbf{EIU}(t_1, t_2, \mathcal{SP})$ l'ensemble des EI-unificateurs de $t_1$ et $t_2$. On note $\mathbf{EIUF}(t_1, t_2, \mathcal{SP})$ l'ensemble des EI-unificateurs fermés de $t_1$ et $t_2$.

Comme pour l'unification syntaxique, il existe une infinité de *EI*-unificateurs. Néanmoins, nous avons vu que lorsque deux termes sont unifiables, il existe un unificateur le plus général (unique à $\doteqdot$ près), et que tous les autres peuvent se déduire de lui. Ce n'est plus vrai pour la EI-unification. Dans le cas général, il existe plusieurs EI-unificateurs incomparables entre eux. Nous cherchons donc à obtenir des ensembles de EI-unificateurs. Il serait agréable de pouvoir "déduire" tous les autres EI-unificateurs à partir de ses ensembles. Or, c'est un problème indécidable dans le cas général. Nous sommes conduits à définir la notion d'ensemble complet d'EI-unificateurs, dont nous exigeons la complétude par rapport aux EI-unificateurs fermés.

**Définition 64** (Complétude d'un ensemble de EI-unificateurs) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $t_1$ , $t_2$ deux termes, et $W \subseteq \mathcal{X}$ un ensemble de variables contenant $V = \mathrm{Var}(t_1) \cup \mathrm{Var}(t_2)$. Un ensemble $ES$ de substitutions est un **ensemble complet de EI-unificateurs de $t_1$ et $t_2$ en dehors de $W$** si :

1. $ES \subseteq \mathbf{EIU}(t_1, t_2, \mathcal{SP})$ (cohérence)
2. $\theta \in \mathbf{EIUF}(t_1, t_2, \mathcal{SP}) \implies \exists \sigma \in ES$ tel que $\sigma \preceq^{\mathbf{i}}_{\mathcal{SP}} \theta$ [ $V$ ] (complétude)
3. $\forall \sigma \in ES, \mathrm{Dom}(\sigma) = V$ et $\mathrm{Im}(\sigma) \cap W = \emptyset$

**Remarque** : Cette définition s'inspire de celle donnée dans [20], un précurseur du domaine. $\diamondsuit$

### 3.2.2 SLDEI-résolution

Nous présentons maintenant le calcul général. Il s'inspire de celui proposé par [19] dans le cadre des termes finis.

**Définition 65** (SLDEI-résolution) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit

$$B = \Longleftarrow A_0 \ldots A_m \ldots A_n$$

une clause de but[2]. On déduit la clause de but $B'$ à partir de $B$ par **SLDEI-résolution** dans les deux cas suivants :

1. $A_m$ est de la forme $P(t_1, \ldots, t_p)$.
   (a) Soit $C'$ une variante d'une clause dans $\mathcal{HC}$ de la forme

$$P(t'_1, \ldots, t'_p) \Longleftarrow D_1 \ldots D_k$$

[2] $A_m$ est appelé l'**atome sélectionné**.

(b) $B' = \Longleftarrow A_1, \ldots, A_{m-1}, t_1 == t'_1, \ldots, t_p == t'_p, D_1, \ldots, D_k, A_{m+1}, \ldots, A_n$

2. $A_m$ est une équation de la forme $t_1 == t_2$
   (a) $B' = \theta(A_1, \ldots, A_{m-1}, A_{m+1}, \ldots, A_n)$
   (b) $\theta \in ES(t_1, t_2, SP)$ où $ES(t_1, t_2, SP)$ est un ensemble complet de $EI$-unificateurs de $t_1$ et $t_2$.

La clause de but $B'$ est appelée **EI-résolvante** de $B$. On note cette dérivation par : $B \overset{\mathcal{SLDEI}}{\longrightarrow}_{[m,C',\mathrm{Id}]} B'$ dans le premier cas, et $B \overset{\mathcal{SLDEI}}{\longrightarrow}_{[m,\emptyset,\theta]} B'$ dans le second.

**Définition 66** (SLDEI-DÉRIVATION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit $B$ une clause de but. Une **SLDEI-dérivation à partir de** $B$ consiste en :

1. une suite de clauses de buts : $B_0 (= B), \ldots, B_n$
2. une suite de variantes de clauses (éventuellement vides) : $C_0, \ldots, C_{n-1}$
3. une suite de EI-unificateurs : $\theta_0, \ldots, \theta_{n-1}$

telles que :

$$B_0 (= B) \overset{\mathcal{SLDEI}}{\longrightarrow}_{[C_0,\theta_0]} \ldots B_{n-1} \overset{\mathcal{SLDEI}}{\longrightarrow}_{[C_{n-1},\theta_{n-1}]} B_n$$

**Définition 67** (SLDEI-RÉFUTATION) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soit $B$ un but. Une **SLDEI-réfutation** de $B$ est une SLDEI-dérivation à partir de $\Longleftarrow B$ qui aboutit à la clause vide $\square$.

**Définition 68** (SOLUTION CALCULÉE) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $B$ un but, et $V = \mathrm{Var}(B)$ l'ensemble des variables libres apparaissant dans $B$. Une **solution calculée par SLDEI-résolution pour le but** $B$ est la restriction aux variables dans $B$ de la composition $\theta_{n-1} \circ \ldots \circ \theta_0$, c'est-à-dire $(\theta_{n-1} \circ \ldots \circ \theta_0)_{\uparrow V}$, où les $\theta_i$ sont des EI-unificateurs utilisés lors d'une SLDEI-réfutation de $B$.

**Théorème 69** : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $B$ un but, et $W$ un ensemble de variables libres contenant $V = \mathrm{Var}(B)$. L'ensemble des solutions calculées par SLDEI-résolution pour le but $B$ est un ensemble complet de solutions pour $B$ en dehors de $W$. Autrement dit, la SLDEI-résolution définit un calcul cohérent et complet.

**Remarque** : Il y a même complétude forte : L'ensemble des solutions pour un but est indépendant de l'ordre dans lequel on sélectionne les atomes d'un but, c'est-à-dire de l'atome qu'on sélectionne à chaque pas d'une dérivation. $\diamondsuit$

Nous ne faisons pas la démonstration de ce théorème par manque de temps, mais elle s'inspirerait de celle concernant la SLD-résolution (voir [21], et [18]), et celle concernant la SLDEI-résolution dans le cadre de termes finis (voir [19]).

# Chapter 4

# Surréduction

Nous avons vu au chapitre précédent qu'un calcul fondé sur la règle de SLDEI-résolution permettait l'utilisation d'algorithmes de résolution d'équations. Informellement, résoudre une équation, c'est trouver un ensemble complet de EI-unificateurs, c'est-à-dire un ensemble de substitutions qui rendent deux termes égaux dans le E-modèle standard d'une spécification, et telle que tout autre EI-unificateur fermé puisse être obtenu (par composition) à partir d'un EI-unificateur de l'ensemble. L'objectif de ce chapitre est de donner un premier calcul cohérent et complet de tels ensembles de EI-unificateurs, dans le cas de systèmes de réécriture de graphes non conditionnels.

Dans une première partie, nous précisons un certain nombre de points généraux. Dans une seconde partie, nous étudions la réécriture de graphes, en définissant plusieurs relations utiles par la suite. Enfin, nous définissons la relation de surréduction, ainsi qu'un calcul d'ensembles complets de EI-unificateurs ; nous montrons sa cohérence et sa complétude.

## 4.1 Généralités

Nous commençons par rappeler la définition de EI-unificateur, et celle d'ensemble complet de EI-unificateurs. Nous définissons ensuite le cadre d'étude de ce chapitre. Puis nous parlons de la théorie équationnelle que nous utilisons : $=_{\mathcal{E}}$. Enfin, nous donnons les définitions de confluence et de terminaison d'une relation (qui diffèrent de celles utilisées dans le cas où on manipule des termes finis).

### 4.1.1 Rappel : EI-unification

**Définition 70** (EI-UNIFICATION) **:** Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature égalitaire, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $t_1$ , $t_2$ deux termes.

- On dit que $t_1$ et $t_2$ sont **EI-unifiables** s'il existe une substitution $\sigma$ telle que $\sigma(t_1) =^{\mathbf{i}}_{\mathcal{SP}} \sigma(t_2)$, c'est-à-dire telle que $\mathcal{EMS}_{\mathcal{SP}} \models \sigma(t_1) == \sigma(t_2)$. On dit que $\sigma$ est un **EI-unificateur** de $t_1$ et $t_2$.
- Trouver un EI-unificateur, c'est résoudre le **problème de EI-unification**.
- On note $\mathbf{EIU}(t_1, t_2, \mathcal{SP})$ l'ensemble des EI-unificateurs de $t_1$ et $t_2$. On note $\mathbf{EIUF}(t_1, t_2, \mathcal{SP})$ l'ensemble des EI-unificateurs fermés de $t_1$ et $t_2$.

**Définition 71** (Complétude d'un ensemble de EI-unificateurs) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification. Soient $t_1$ , $t_2$ deux termes, et $W \subseteq \mathcal{X}$ un ensemble de variables contenant $V = \mathrm{Var}(t_1) \cup \mathrm{Var}(t_2)$. Un ensemble $ES$ de substitutions est un **ensemble complet de EI-unificateurs de $t_1$ et $t_2$ en dehors de $W$** si :

1. $ES \subseteq \mathbf{EIU}(t_1, t_2, \mathcal{SP})$ (cohérence)
2. $\theta \in \mathbf{EIUF}(t_1, t_2, \mathcal{SP}) \implies \exists \sigma \in ES$ tel que $\sigma \preceq^{\mathbf{i}}_{\mathcal{SP}} \theta \ [\ V\ ]$ (complétude)
3. $\forall \sigma \in ES, \mathrm{Dom}(\sigma) = V$ et $\mathrm{Im}(\sigma) \cap W = \emptyset$

### 4.1.2 Cadre d'étude

Nous supposons dans tout ce chapitre que nous disposons d'une signature égalitaire $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$, d'un ensemble $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ de variables, d'un ensemble $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ d'étiquettes, et d'une spécification $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ où l'ensemble des clauses $\mathcal{HC}$ est l'union d'un ensemble d'équations non conditionnelles $\mathcal{E}$ et d'un ensemble de clauses de têtes non équationnelles $\mathcal{CL}$ :

$$\mathcal{HC} = \mathcal{CL} \uplus \mathcal{E}$$

avec :

$$\mathcal{E} = \{g_i == d_i \mid i \in 1 \ldots k\}$$

### 4.1.3 Théorie équationnelle

**Définition 72** (Théorie équationnelle) : Soient $\Sigma = \langle \mathcal{S}, \Omega, \Pi \rangle$ une signature, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{s \in \mathcal{S}} \mathcal{L}_s$ un ensemble d'étiquettes. On définit $=_{\mathcal{E}}$ sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ comme la plus petite relation telle que :

1. $\boxed{g == d \in \mathcal{E} \implies g =_{\mathcal{E}} d}$
2. $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}), \sigma \in \mathbf{SUB} \implies \boxed{t_1 =_{\mathcal{E}} t_2 \implies \sigma(t_1) =_{\mathcal{E}} \sigma(t_2)}$
3. $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{t_1 \doteq t_2 \implies t_1 =_{\mathcal{E}} t_2}$
4. $t \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{t =_{\mathcal{E}} t}$
5. $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{t_1 =_{\mathcal{E}} t_2 \implies t_2 =_{\mathcal{E}} t_1}$
6. $t_1, t_2, t_3 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \implies \boxed{t_1 =_{\mathcal{E}} t_2 \ , \ t_2 =_{\mathcal{E}} t_3 \implies t_1 =_{\mathcal{E}} t_3}$
7. $$\left.\begin{array}{l} f \in \Omega_{s_1 \ldots s_n, s} \\ t_i, t'_i \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})_{s_i}, (i \in 1 \ldots n) \\ z, z' \in \mathcal{L}_s \\ \\ \boxed{t_i =_{\mathcal{E}} t'_i}, (i \in 1 \ldots n) \end{array}\right\} \implies \boxed{z : f(t_1, \ldots, t_n) =_{\mathcal{E}} z' : f(t'_1, \ldots, t'_n)}$$

**Proposition 73** : $=_{\mathcal{E}}$ est une relation de congruence.

**Proposition 74** : $=_{\mathcal{E}}$ est fermée par substitution et par remplacement :

- $t_1 =_\mathcal{E} t_2 \implies \sigma(t_1) =_\mathcal{E} \sigma(t_2)$
- $t_1 =_\mathcal{E} t_2 \implies t[z \leftarrow t_1] =_\mathcal{E} t[z \leftarrow t_2]$

**Proposition 75** :

1. $t_1 =_\mathcal{E} t_2 \implies t_1 =^{\mathbf{i}}_{\mathcal{SP}} t_2$
2. Si $t_1, t_2$ sont fermés , $t_1 =_\mathcal{E} t_2 \iff t_1 =^{\mathbf{i}}_{\mathcal{SP}} t_2$

### 4.1.4 Propriétés générales sur les relations

Soit $\rightharpoonup$ une relation sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$. On note $\overset{+}{\rightharpoonup}$ sa fermeture transitive, et $\overset{\star}{\rightharpoonup}$ sa fermeture réflexive transitive. On note $\rightleftharpoons$ sa fermeture symétrique, $\overset{+}{\rightleftharpoons}$ sa fermeture symétrique transitive, et $\overset{\star}{\rightleftharpoons}$ sa fermeture symétrique, réflexive et transitive.

**Définition 76** (CONFLUENCE, NŒTHÉRIANITÉ, CANONICITÉ) :

1. On dit que $\rightharpoonup$ est **nœthérienne** s'il n'existe aucun terme duquel part une dérivation infinie $(t_1 \rightharpoonup t_2 \rightharpoonup \ldots)$.
2. $\rightharpoonup$ est **confluente** si :
$$\left.\begin{array}{l} t_1 \doteq t_2 \\ t_1 \overset{\star}{\rightharpoonup} t'_1 \\ t_2 \overset{\star}{\rightharpoonup} t'_2 \end{array}\right\} \implies \exists\, t''_1, t''_2 \text{ tels que } \left\{\begin{array}{l} t'_1 \overset{\star}{\rightharpoonup} t''_1 \\ t'_2 \overset{\star}{\rightharpoonup} t''_2 \\ t''_1 \doteq t''_2 \end{array}\right.$$
3. $\rightharpoonup$ est **canonique** si elle est nœthérienne et confluente.

**Remarque** : Cette notion de confluence n'est pas la même que dans le cas classique. C'est en fait la confluence modulo la bisimulation. Notez qu'on retrouve la confluence classique si on travaille dans $\mathcal{T}(\widehat{\Sigma, \mathcal{L}}, \mathcal{X})$, plutôt que dans $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$. $\diamondsuit$

**Définition 77** (FORME NORMALE) :

- On dit qu'un terme $t$ est sous **forme normale** par rapport à $\rightharpoonup$ s'il n'est pas **réductible**, c'est-à-dire : $\neg(\exists t' \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \mid t \rightharpoonup t')$
- Lorsque $t \overset{\star}{\rightharpoonup} t'$ et que $t'$ est sous forme normale, on dit que $t'$ est *une* forme normale de $t$.
- Si $\rightharpoonup$ est nœthérienne, tout terme admet au moins une forme normale.
- Si $\rightharpoonup$ est confluente, tout terme admet au plus une forme normale, unique à la bisimularité près, lorsqu'elle existe.
- Enfin, quand $\rightharpoonup$ est canonique, tout terme admet une forme normale, unique à la bisimularité près.

**Définition 78** (SUBSTITUTION NORMALISÉE) : On dit qu'une substitution $\sigma$ est **normalisée** par rapport à $\rightharpoonup$ si pour tout $x \in \mathrm{Dom}(\sigma)$ , $\sigma(x)$ est sous forme normale.

## 4.2 Réécriture

Dans cette partie, nous nous intéressons à la réécriture de graphes. Après la définition de ce que nous entendons par système de réécriture de graphes, nous étudions plusieurs relations : $\longrightarrow^\alpha$, $\Longrightarrow$, $\rightsquigarrow$ et $\longmapsto$. Le dernier paragraphe rappelle nos résultats.

### 4.2.1 Système de réécriture de graphes

**Définition 79** (Règle et système de réécriture) : Une **règle de réécriture non conditionnelle** est un couple de termes de même sorte, noté $g \longrightarrow d$ tels que $\mathrm{Var}(d) \subseteq \mathrm{Var}(g)$ et $g \notin \mathcal{X}$. Un **système de réécriture de graphe**, ou **SRG**, est un ensemble de règles de réécriture.

L'intérêt des systèmes de réécriture est de fournir un moyen mécanique pour résoudre des équations dans la théorie équationnelle $=_{\mathcal{E}}$.

Nous associons à l'ensemble d'équations $\mathcal{E}$, le système de réécriture $\vec{\mathcal{E}} = \{g \to d \mid g==d \in \mathcal{E}\}$.

### 4.2.2 Relations $\longrightarrow^{\alpha}$ et $\Longrightarrow$

**Définition 80** (Relation de réduction) : On définit la **relation de réduction** sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$, notée $\longrightarrow^{\alpha}$, comme étant la plus petite relation contenant $\vec{\mathcal{E}}$, fermée par substitution et remplacement :

1. $\boxed{\vec{\mathcal{E}} \subseteq \longrightarrow^{\alpha}}$
2. $$\left.\begin{array}{l} t, t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \\ x \in \mathrm{Pos}(t) \end{array}\right\} \Longrightarrow \boxed{t_1 \longrightarrow^{\alpha} t_2 \Longrightarrow t[x \leftarrow t_1] \longrightarrow^{\alpha} t[x \leftarrow t_2]}$$
3. $$\left.\begin{array}{l} t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X}) \\ \sigma \in \mathbf{SUB} \end{array}\right\} \Longrightarrow \boxed{t_1 \longrightarrow^{\alpha} t_2 \Longrightarrow \sigma(t_1) \longrightarrow^{\alpha} \sigma(t_2)}$$

**Définition 81** (Réduction d'un terme) : On dit qu'un terme $t_1$ **se réduit** ou **se réécrit** en un terme $t_2$ par $\longrightarrow^{\alpha}$ si il existe $x \in \mathrm{Pos}^{\star}(t_1)$[1], une règle $g \to d \in \vec{\mathcal{E}}$, et une substitution $\sigma \in \mathbf{SUB}$ tels que :

1. $\sigma(g) =_{\alpha} t_{1\uparrow x}$
2. $t_2 = t_1[x \leftarrow \sigma(d)]$

On le note : $t_1 \longrightarrow^{\alpha}_{[x, g \to d, \sigma]} t_2$ , ou $t_1 \longrightarrow^{\alpha}_{[x, g \to d]} t_2$ , ou simplement $t_1 \longrightarrow^{\alpha} t_2$.

Cette première relation ne peut pas nous permettre de résoudre des équations à elle seule, puisqu'elle ne dit rien de la bisimulation.

**Définition 82** ($\Longrightarrow$) : On définit une nouvelle relation sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ :

$$\Longrightarrow \; = \longrightarrow^{\alpha} \cup \doteq$$

**Proposition 83** : $\overset{\star}{\Longleftrightarrow}$ est une relation de congruence sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$.

**Preuve** : Clairement, $\overset{\star}{\Longleftrightarrow}$ est une relation d'équivalence. Montrons que c'est une congruence. Soient $f : s_1 \ldots s_n \longrightarrow s$, et $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$. Montrons d'abord par induction que si $t_1 \overset{\star}{\Longleftrightarrow} t'_1$ alors $f(t_1, t_2, \ldots, t_n) \overset{\star}{\Longleftrightarrow} f(t'_1, t_2, \ldots, t_n)$. Clairement, si $t_1 \overset{0}{\Longleftrightarrow} t'_1$, le résultat est établi. Supposons le résultat établi lorsque $t_1 \overset{n}{\Longleftrightarrow} t'_1$. Soit $t_1 \overset{n+1}{\Longleftrightarrow} t'_1$. Il y a trois cas :

[1] Une position de $t_1$ qui ne dénote pas une variable.

1. $t_1 \doteq t \overset{n}{\Longleftrightarrow} t'_1$ : comme $\doteq$ est une congruence, et du fait de l'hypothèse de récurrence, on a : $f(t_1, t_2, \ldots, t_n) \doteq f(t, t_2, \ldots, t_n) \overset{n}{\Longleftrightarrow} f(t'_1, t_2, \ldots, t_n)$.
2. $t_1 \longrightarrow^{\alpha} t \overset{n}{\Longleftrightarrow} t'_1$ : comme $\longrightarrow^{\alpha}$ est fermée par remplacement, on a, en utilisant l'hypothèse de récurrence : $f(t_1, t_2, \ldots, t_n) \longrightarrow^{\alpha} f(t, t_2, \ldots, t_n) \overset{n}{\Longleftrightarrow} f(t'_1, t_2, \ldots, t_n)$.
3. $t_1 \longleftarrow^{\alpha} t \overset{n}{\Longleftrightarrow} t'_1$ : même cas que le précédent.

On établit donc le résultat pour $n+1$, et donc pour tout $n$ : si $t_1 \overset{\star}{\Longleftrightarrow} t'_n$ alors

$$f(t_1, t_2, \ldots, t_n) \overset{\star}{\Longleftrightarrow} f(t'_1, t_2, \ldots, t_n)$$

On refait la même preuve pour $t_2$ à partir de $f(t'_1, t_2, \ldots, t_n)$. Et ainsi de suite. Par transitivité de $\overset{\star}{\Longleftrightarrow}$, on montre finalement que si $t_1 \overset{\star}{\Longleftrightarrow} t'_1, \ldots, t_n \overset{\star}{\Longleftrightarrow} t'_n$ alors $f(t_1, \ldots, t_n) \overset{\star}{\Longleftrightarrow} f(t'_1, \ldots, t'_n)$. Et donc, $\overset{\star}{\Longleftrightarrow}$ est une relation de congruence. $\diamondsuit$

**Théorème 84** : $\boxed{\overset{\star}{\Longleftrightarrow} \; = \; =_{\mathcal{E}}}$

**Preuve** :

- $\supseteq$ : $=_{\mathcal{E}}$ est la plus petite congruence sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ contenant $\vec{\mathcal{E}}$ et $\doteq$. Comme $\overset{\star}{\Longleftrightarrow}$ est aussi une congruence contenant $\doteq$ et $\vec{\mathcal{E}}$ (par l'intermédiaire de $\longrightarrow^{\alpha}$), on a : $\overset{\star}{\Longleftrightarrow} \; \supseteq \; =_{\mathcal{E}}$.
- $\subseteq$ :
  - $\doteq \; \subseteq \; =_{\mathcal{E}}$.
  - Montrons que $\longrightarrow^{\alpha} \; \subseteq \; =_{\mathcal{E}}$ : Si $t_1 \longrightarrow^{\alpha}_{[x, g \to d, \sigma]} t_2$ , alors $\sigma(g) =_{\alpha} t_{1\uparrow x}$ et $t_2 = t_1[x \leftarrow \sigma(d)]$. On a $t_1 \doteq t_1[x \leftarrow t_{1\uparrow x}] \doteq t_1[x \leftarrow \sigma(g)]$. Donc $t_1 =_{\mathcal{E}} t_1[x \leftarrow \sigma(g)]$. Or $g \to d \in \vec{\mathcal{E}}$, donc $g =_{\mathcal{E}} d$, et donc $\sigma(g) =_{\mathcal{E}} \sigma(d)$. De plus, si $l =_{\mathcal{E}} r$, alors $t[x \leftarrow l] =_{\mathcal{E}} t[x \leftarrow r]$. Donc $t_1 =_{\mathcal{E}} t_1[x \leftarrow \sigma(d)]$ et donc $t_1 = t_2$. Par conséquent : $\longrightarrow^{\alpha} \; \subseteq \; =_{\mathcal{E}}$.
  - Conclusion : $\overset{\star}{\Longleftrightarrow} \; \subseteq \; =_{\mathcal{E}}$.
- Conclusion : $\overset{\star}{\Longleftrightarrow} \; = \; =_{\mathcal{E}}$.

$\diamondsuit$

Maintenant, dans le cas des termes finis, lorsqu'on montre que $=_{\mathcal{E}} \; = \; \overset{\star}{\rightleftharpoons}$ et que $\rightharpoonup$ est canonique, alors on montre que $t_1 =_{\mathcal{E}} t_2$ si les formes normales de $t_1$ et $t_2$ sont égales. Or ici, comme $\Longrightarrow \; \supseteq \; \doteq$, donc $\Longrightarrow$ ne peut pas être nœthérienne : il existe une infinité de termes bisimilaires à un terme donné. On définit donc une nouvelle relation dont nous allons imposer la nœthérianité.

### 4.2.3 Relation $\rightsquigarrow$

**Définition 85 ($\rightsquigarrow$) :** On définit la relation $\rightsquigarrow$ sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ en posant :

$$\rightsquigarrow \; = \; \doteq \; \cdot \; \longrightarrow^{\alpha} \; \cdot \; \doteq$$

**Proposition 86** : $\boxed{\overset{\star}{\rightsquigarrow} \; \subset \; \overset{\star}{\Longrightarrow}}$

**Preuve** : L'inclusion est évidente. Elle est stricte, parce que si $t_1 \doteq t_2$ alors $t_1 \overset{\star}{\Longrightarrow} t_2$. Mais la définition de $\rightsquigarrow$ nous oblige à faire un pas de $\longrightarrow^{\alpha}$, et donc : $\neg(t_1 \doteq t_2 \implies t_1 \overset{\star}{\rightsquigarrow} t_2)$. $\diamondsuit$

Nous supposerons dans toute la suite que $\rightsquigarrow$ est canonique. Nous noterons $t\!\downarrow$ la forme normale de tout terme $t$.

**Théorème 87** : $\boxed{t_1 \overset{\star}{\Longleftrightarrow} t_2 \text{ ssi } t_1\!\downarrow \doteq t_2\!\downarrow}$

**Preuve** :

- $\Longleftarrow$ : Supposons $t_1\!\downarrow \doteq t_2\!\downarrow$. Comme $\overset{\star}{\rightsquigarrow} \subset \overset{\star}{\Longrightarrow}$, on a : $t_1 \overset{\star}{\Longrightarrow} t_1\!\downarrow$ et $t_2 \overset{\star}{\Longrightarrow} t_2\!\downarrow$. Comme $t_1\!\downarrow \doteq t_2\!\downarrow$, $t_1\!\downarrow \overset{\star}{\Longleftrightarrow} t_2\!\downarrow$. Donc $t_1 \overset{\star}{\Longleftrightarrow} t_2$.
- $\Longrightarrow$ : Nous raisonnons par induction sur la longueur de la dérivation $t_1 \overset{\star}{\Longleftrightarrow} t_2$. Clairement, si $t_1 \overset{0}{\Longleftrightarrow} t_2$, alors $t_1\!\downarrow \doteq t_2\!\downarrow$. Supposons le résultat établi pour tout $k < n$. Soit $t_1 \overset{n}{\Longleftrightarrow} t_2$.
  - Si $t_1 \doteq t_2$, alors par confluence de $\rightsquigarrow$, $t_1\!\downarrow \doteq t_2\!\downarrow$.
  - Nous supposons maintenant que $\neg(t_1 \doteq t_2)$. Il existe alors une dérivation de la forme : $t_1 \doteq u_0 \doteq \ldots \doteq u_i \longrightarrow^{\alpha} v_0 \overset{j}{\Longleftrightarrow} t_2$ ou bien $t_1 \doteq u_0 \doteq \ldots \doteq u_i \longleftarrow^{\alpha} v_0 \overset{j}{\Longleftrightarrow} t_2$.
    1. Dans le premier cas, on a : $t_1 \doteq u_i \longrightarrow^{\alpha} v_0 \doteq v_0 \overset{j}{\Longleftrightarrow} t_2$, donc $t_1 \rightsquigarrow v_0 \overset{j}{\Longleftrightarrow} t_2$. Donc $t_1\!\downarrow \doteq v_0\!\downarrow$, et comme par hypothèse $v_0\!\downarrow \doteq t_2\!\downarrow$, on a $t_1\!\downarrow \doteq t_2\!\downarrow$.
    2. Dans le second cas, on a : $t_1 \doteq u_i \longleftarrow^{\alpha} v_0 \doteq v_0 \overset{j}{\Longleftrightarrow} t_2$, donc $t_2 \overset{j}{\Longleftrightarrow} v_0 \rightsquigarrow t_1$. Et de même, $t_2\!\downarrow \doteq t_1\!\downarrow$.
- Conclusion : $t_1 \overset{\star}{\Longleftrightarrow} t_2$ ssi $t_1\!\downarrow \doteq t_2\!\downarrow$

$\Diamond$

Pour simplifier la preuve de complétude de la surréduction, nous allons encore "raffiner" la réduction des termes, en définissant une dernière relation.

### 4.2.4 Relation $\mapsto$

**Définition 88** ($\longmapsto$) : On définit la relation $\longmapsto$ sur $\mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$. On pose $t_1 \longmapsto_{[x, g \to d, \sigma]} t_2$ si il existe $x \in \text{Pos}^{\star}(t_1)$, une règle $g \to d \in \vec{\mathcal{E}}$, et une substitution $\sigma \in \mathbf{SUB}$ tels que :

1. $\sigma(g) \doteq t_{1\uparrow x}$
2. $t_2 \doteq t_1[x \leftarrow \sigma(d)]$

**Proposition 89** : $t_1 \longmapsto_{[x, g \to d, \sigma]} t_2 \implies t_1 \rightsquigarrow t_2$

**Preuve** : $\longmapsto$ est un cas particulier de $\rightsquigarrow$. En effet, on a : $t_1 \doteq t_1[x \leftarrow t_{1\uparrow x}]$. Par définition de $\longmapsto$, on sait que $\sigma(g) \doteq t_{1\uparrow x}$. Donc, soit $u \in \mathcal{T}(\Sigma, \mathcal{L}, \mathcal{X})$ tel que $u \doteq t_{1\uparrow x}$ et $u =_{\alpha} \sigma(g)$. On a : $t_1[x \leftarrow t_{1\uparrow x}] \doteq t_1[x \leftarrow u]$. Par conséquent : $t_1 \doteq t_1[x \leftarrow u] \longrightarrow^{\alpha}_{[x, g \to d, \sigma]} t_1[x \leftarrow \sigma(d)] \doteq t_2$. Et donc : $t_1 \rightsquigarrow t_2$. $\Diamond$

**Proposition 90** :

- $\overset{\star}{\longmapsto} \subset \overset{\star}{\rightsquigarrow}$
- Si $\rightsquigarrow$ est nœthérienne, alors $\longmapsto$ est nœthérienne.

**Preuve** : L'inclusion découle de la propriété précédente. Elle est stricte parce que pour faire un pas de réduction avec $\rightsquigarrow$ ($t_1 \doteq u_1 \longrightarrow^{\alpha} u_2 \doteq t_2$), on peut choisir n'importe quel premier terme bisimilaire ($u_1$), alors que celui-ci est imposé lors d'une réduction avec $\longmapsto$ ($t_1[x \leftarrow t_{1\uparrow x}]$). La noethérianité de $\longmapsto$ découle de celle de $\rightsquigarrow$. $\Diamond$

**Proposition 91** : $t_1 \rightsquigarrow t_2 \implies \exists t_2'$ tel que $t_1 \longmapsto t_2'$.

**Remarque** : $t_2$ et $t'_2$ ne sont pas bisimilaires dans le cas général. $\diamondsuit$

**Preuve** : Soit $t_1 \rightsquigarrow t_2$. Il existe donc $u_1, u_2$ tels que : $t_1 \doteq u_1 \longrightarrow^{\alpha}_{[x,g\to d,\sigma]} u_2 \doteq t_2$ avec $u_{1\uparrow x} =_\alpha \sigma(g)$. Comme $t_1 \doteq u_1$, il existe $y \in \mathrm{Pos}^\star(t_1)$ tel que $t_{1\uparrow y} \doteq u_{1\uparrow x}$. Par transitivité de $\doteq$, $t_{1\uparrow y} \doteq \sigma(g)$. Soit $t'_2 \doteq t_1[y \leftarrow \sigma(d)]$. On a : $t_1 \longmapsto_{[y,g\to d,\sigma]} t'_2$ $\diamondsuit$

**Proposition 92** :

- Si $\rightsquigarrow$ est confluente alors $\longmapsto$ est confluente.
- $t\downarrow \doteq t\wr$

**Preuve** : On sait que $\longmapsto$ est nœthérienne, et donc que les formes normales existent. Soit $t$ un terme et $t_1$ une de ses formes normales par rapport à $\longmapsto$. Si $t_1$ n'est pas sous forme normale par rapport à $\rightsquigarrow$, alors il existe $t_2$ tel que $t_1 \rightsquigarrow t_2$. Et dans ce cas, il existe $t'_2$ tel que $t_1 \longmapsto t'_2$, ce qui est impossible. Par conséquent, toute forme normale de $t$ par rapport à $\longmapsto$ est une forme normale par rapport à $\rightsquigarrow$. Comme cette dernière est unique à la bisimulation près ($\rightsquigarrow$ est confluente), c'est aussi le cas pour les formes normales par rapport à $\longmapsto$. D'où la propriété. $\diamondsuit$

### 4.2.5 Résumé

**Théorème 93** : Si $\rightsquigarrow$ est canonique, alors pour tout terme $t_1$, $t_2$ :

$$\begin{array}{lllll} t_1 =_{\mathcal{E}} t_2 & \text{ssi} & t_1 & \overset{\star}{\Longleftrightarrow} & t_2 \\ & \text{ssi} & t_1\wr & \doteq & t_2\wr \\ & \text{ssi} & t_1\downarrow & \doteq & t_2\downarrow \end{array}$$

## 4.3 Surréduction

Nous définissons maintenant la relation de surréduction. Nous montrons ensuite qu'elle définit un calcul cohérent. Puis nous montrons qu'elle définit un calcul complet. Nous rèunissons ces deux propriétés dans un théorème final.

### 4.3.1 Définitions

**Définition 94** (SURRÉDUCTION) : On dit qu'un terme $t_1$ **se surréduit** en un terme $t_2$ s'il existe $x \in \mathrm{Pos}^\star(t_1)$, et une règle $g \to d \in \vec{\mathcal{E}}$ tels que :

1. $t_{1\uparrow x}$ et $g$ soient unifiables, d'unificateur minimal $\sigma$ : $\sigma(t_{1\uparrow x}) \doteq \sigma(g)$
2. $t_2 = \sigma(t_1[x \leftarrow d])$

On le note : $t_1 — \mathcal{N} \longrightarrow_{[x,g\to d,\sigma]} t_2$ ou $t_1 — \mathcal{N} \longrightarrow t_2$.

**Définition 95** (DÉRIVATION DE SURRÉDUCTION) : Une **dérivation de surréduction** issue d'un terme $t$ consiste en :

1. une suite de terme $t(= t_0), \ldots, t_n$
2. une suite de variantes de règles $g_0 \to d_0, \ldots, g_{n-1} \to d_{n-1}$
3. une suite d'unificateurs $\sigma_0, \ldots, \sigma_{n-1}$

tels que $\forall j \in 0 \ldots n-1, t_j — \mathcal{N} \longrightarrow_{[x_j,g_i\to d_i,\sigma_j]} t_{j+1}$.

Nous considérons maintenant le symbole d'égalité == comme un symbole fonctionnel particulier, distinct de tous les autres.

### 4.3.2 Théorème de cohérence de la surréduction

**Lemme 96** : Soient $t_1, t_2$ des termes, $x \in \mathrm{Pos}^\star(t_1)$, $g \to d \in \vec{\mathcal{E}}$, et $\sigma \in \mathbf{SUB}$ tels que $t_1 \,—\mathcal{N}\longrightarrow_{[x,g\to d,\sigma]}\, t_2$. Alors pour toute substitution $\theta \in \mathbf{SUB}$ :

$$(\theta \circ \sigma)(t_1) \longmapsto_{[x,g\to d]} \theta(t_2)$$

**Preuve** : On suppose : $t_1 \,—\mathcal{N}\longrightarrow_{[x,g\to d,\sigma]}\, t_2$. Donc $\sigma(t_{1\uparrow x}) \doteq \sigma(g)$ et $t_2 = \sigma(t_1[x \leftarrow d])$. Comme $x$ est une étiquette de $t_1$, on a $(\theta \circ \sigma)(t_{1\uparrow x}) \doteq (\theta \circ \sigma)(t_1)_{\uparrow x} \doteq (\theta \circ \sigma)(g)$. Par conséquent, $(\theta\circ\sigma)(t_1) \longmapsto_{[x,g\to d,\theta\circ\sigma]} t$ avec $t$ quelconque tel que $t \doteq (\theta \circ \sigma)(t_1)[x \leftarrow (\theta \circ \sigma)(d)] \doteq (\theta\circ\sigma)(t_1[x \leftarrow d])$. C'est en particulier vrai si on prend $t = \theta(t_2)$. Dans ce cas, on a bien : $(\theta \circ \sigma)(t_1) \longmapsto_{[x,g\to d]} \theta(t_2)$. $\diamondsuit$

**Théorème 97** : Soit $t_1 == t'_1$ une équation. On suppose qu'il existe une dérivation de surréduction :

$$\begin{array}{lll} (t_1 == t'_1) & —\mathcal{N}\longrightarrow_{[x_1,g_1\to d_1,\sigma_1]} & (t_2 == t'_2) \\ & —\mathcal{N}\longrightarrow \ \ldots & \\ & —\mathcal{N}\longrightarrow_{[x_{n-1},g_{n-1}\to d_{n-1},\sigma_{n-1}]} & (t_n == t'_n) \end{array}$$

telle que $t_n$ et $t'_n$ soient unifiables, d'unificateur minimal $\mu$. Alors, $\mu \circ \sigma_{n-1} \circ \ldots \sigma_1$ est un $EI$-unificateur de $t_1$ et de $t'_1$.

**Preuve** : En utilisant récursivement le lemme précédent, on a :

$$\begin{array}{lll} (\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_2 \circ \sigma_1)(t_1 == t'_1) & \longmapsto_{[x_1,g_1\to d_1]} & (\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_2)(t_2 == t'_2) \\ & \longmapsto_{[x_2,g_2\to d_2]} & (\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_3)(t_3 == t'_3) \\ & \longmapsto \ \ldots & \\ & \longmapsto_{[x_{n-1},g_{n-1}\to d_{n-1}]} & \mu(t_n == t'_n) \end{array}$$

Or $\mu(t_n) \doteq \mu(t'_n)$. Donc $(\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1)(t_1)\downarrow \doteq (\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1)(t'_1)\downarrow$. On a donc : $(\mu \circ \sigma_{n-1}\circ\ldots\circ\sigma_1)(t_1) =_\varepsilon (\mu\circ\sigma_{n-1}\circ\ldots\circ\sigma_1)(t'_1)$. Et donc : $(\mu\circ\sigma_{n-1}\circ\ldots\circ\sigma_1)(t_1) =^{\mathrm{i}}_{\mathcal{SP}} (\mu\circ\sigma_{n-1}\circ\ldots\circ\sigma_1)(t'_1)$. Par conséquent, $\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1$ est un $EI$-unificateur de $t_1$ et de $t'_1$. $\diamondsuit$

### 4.3.3 Théorème de complétude de la surréduction

**Lemme 98** : Soient $s_1, t_1$ des termes, $\theta_1$ une substitution **normalisée** telle que $t_1 = \theta_1(s_1)$, $V_1$ un ensemble de variables tel que $\mathrm{Var}(s_1) \cup \mathrm{Dom}(\theta_1) \subseteq V_1$. On suppose que $t_1$ peut se réécrire avec $\longmapsto$ à la position $x$ en utilisant une variante de la règle $g \to d \in \vec{\mathcal{E}}$, telle que $\mathrm{Var}(g) \cap V_1 = \emptyset$. Alors, il existe $s_2, t_2$ deux termes et $\sigma, \theta_2$ deux substitutions tels que :

1. $t_1 \longmapsto_{[x,g\to d]} t_2$
2. $s_1 \,—\mathcal{N}\longrightarrow_{[x,g\to d,\sigma]}\, s_2$
3. $\mathrm{Var}(s_2) \cup \mathrm{Dom}(\theta_2) \subseteq V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma)$
4. $t_2 = \theta_2(s_2)$
5. $\theta_2 \circ \sigma \stackrel{\cdot}{\equiv} \theta_1 \ [\ V_1\ ]$
6. $\theta_2$ normalisée.

**Preuve** : Elle s'inspire pour beaucoup de celle donnée dans [22], quant à la gestion des variables libres. Soit $t$ un terme *quelconque* tel que $t_1 \longmapsto_{[x,g\to d]} t$. Il existe donc une substitution $\tau$ telle que $t_{1\uparrow x} \doteq \tau(g)$ avec $\mathrm{Dom}(\tau) \subseteq \mathrm{Var}(g)$, et $t \doteq t_1[x \leftarrow \tau(d)]$.

- – Comme $\theta_1$ est normalisée , $t_{1\uparrow x} = \theta_1(s_1)_{\uparrow x} \doteq \theta_1(s_{1\uparrow x})$. Par conséquent, $\theta_1(s_{1\uparrow x}) \doteq \tau(g)$.
  – Par hypothèse, $\mathrm{Dom}(\theta_1) \subseteq V_1$, et comme $\mathrm{Dom}(\tau) \subseteq \mathrm{Var}(g)$, $\mathrm{Dom}(\tau) \cap V_1 = \emptyset$. On peut donc définir la substitution $\underline{\mu = \tau \cup \theta_1}$, et on a : $\mu(s_{1\uparrow x}) \doteq \mu(g)$.
  – Du fait de l'expression précédente, $s_{1\uparrow x}$ et $g$ sont unifiables. Soit $\sigma$ un de leurs unificateurs minimaux : $\underline{\sigma(s_{1\uparrow x}) \doteq \sigma(g)}$.
  – Par définition de la relation de surréduction, on a donc : $\boxed{s_1 \;—\mathcal{N}\!\longrightarrow_{[x,g\to d,\sigma]}\; s_2}$ avec $\underline{s_2 = \sigma(s_1[x \leftarrow d])}$.
  – On sait d'autre part que $\sigma \dot{\preceq} \mu$. Donc il existe une substitution $\rho$ telle que $\underline{\rho \circ \sigma \doteqdot \mu}$.
  – Enfin, on pose $\underline{V_2 = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma)}$ et $\underline{\theta_2 = \rho_{\uparrow V_2}}$.
- – On a $\mathrm{Dom}(\theta_2) \subseteq V_2$.
  – D'autre part, $s_2 = \sigma(s_1[x \leftarrow d])$, et $\mathrm{Var}(d) \subseteq \mathrm{Var}(g)$, donc $\mathrm{Var}(s_2) \subseteq \mathrm{Var}(\sigma(s_1[x \leftarrow g]))$. Or :

$$\begin{array}{rcl} \sigma(s_1[x \leftarrow g]) & \doteq & \sigma(s_1)[x \leftarrow \sigma(g)] \\ & \doteq & \sigma(s_1)[x \leftarrow \sigma(s_{1\uparrow x})] \\ & \doteq & \sigma(s_1) \end{array}$$

  Donc $\mathrm{Var}(s_2) \subseteq \mathrm{Var}(\sigma(s_1)) \subseteq \mathrm{Im}(\sigma) \subseteq V_2$.
  – On a donc $\boxed{\mathrm{Var}(s_2) \cup \mathrm{Dom}(\theta_2) \subseteq V_2}$.

- Maintenant, par définition, $\theta_2 = \rho_{\uparrow V_2}$. Donc :

$$\begin{array}{rcl} \theta_2(s_2) & = & \rho(s_2) \\ & \doteq & (\rho \circ \sigma)(s_1[x \leftarrow d]) \\ & \doteq & \mu(s_1[x \leftarrow d]) \\ & \doteq & \mu(s_1)[x \leftarrow \mu(d)] \end{array}$$

  – Clairement, $\mu_{\uparrow \mathrm{Var}(s_1)} = \theta_{1\uparrow \mathrm{Var}(s_1)}$.
  – D'autre part, $\mathrm{Var}(d) \subseteq \mathrm{Var}(g)$ et comme $\mu_{\uparrow \mathrm{Var}(g)} = \tau_{\uparrow \mathrm{Var}(g)}$, on a $\mu_{\uparrow \mathrm{Var}(d)} = \tau_{\uparrow \mathrm{Var}(d)}$.

  Par conséquent :

$$\begin{array}{rcl} \theta_2(s_2) & \doteq & \mu(s_1)[x \leftarrow \mu(d)] \\ & \doteq & \theta_1(s_1)[x \leftarrow \tau(d)] \\ & \doteq & t_1[x \leftarrow \tau(d)] \end{array}$$

  Nous avons laissé $t$ arbitraire pour faire la démonstration, avec $t \doteq t_1[x \leftarrow \tau(d)]$. Nous fixons maintenant : $t = \theta_2(s_2) = t_2$. Et nous obtenons $\boxed{t_1 \longmapsto_{[x,g\to d]} t_2}$, avec $\boxed{t_2 = \theta_2(s_2)}$.

- Nous montrons maintenant que $\theta_2 \circ \sigma \doteqdot \theta_1\ [\ V_1\ ]$.

$$(\theta_2 \circ \sigma)_{\uparrow V_1} \doteqdot (\theta_{2\uparrow \mathrm{Im}(\sigma)} \circ \sigma)_{\uparrow V_1} \cup \theta_{2\uparrow (V_1 - \mathrm{Dom}(\sigma))}$$

  Or $\theta_2 = \rho_{\uparrow V_2}$ avec $V_2 = (V_1 - \mathrm{Dom}(\sigma)) \cup \mathrm{Im}(\sigma)$. Donc on a :

$$\begin{array}{rcl} (\theta_2 \circ \sigma)_{\uparrow V_1} & \doteqdot & (\rho_{\uparrow \mathrm{Im}(\sigma)} \circ \sigma)_{\uparrow V_1} \cup \rho_{\uparrow (V_1 - \mathrm{Dom}(\sigma))} \\ & \doteqdot & (\rho \circ \sigma)_{\uparrow V_1} \\ & \doteqdot & \mu_{\uparrow V_1} \end{array}$$

  Mais : $\mu_{\uparrow V_1} = \theta_{1\uparrow V_1}$. Et donc $\boxed{\theta_2 \circ \sigma \doteqdot \theta_1\ [\ V_1\ ]}$.

- Il reste à montrer que $\theta_2$ est normalisée. Avant, nous allons préciser l'expression de $V_2$.
  – Montrons que $V_2 = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma) = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma_{\uparrow V_1})$.
    * Par définition, $\sigma$ est un unificateur minimal de $s_{1\uparrow x}$ et $g$. Nous pouvons toujours le supposer idempotent ce qui se traduit par $\mathrm{Dom}(\sigma) \cap \mathrm{Im}(\sigma) = \emptyset$. D'autre part, comme $\mathrm{Var}(s_{1\uparrow x}) \cap \mathrm{Var}(g) = \emptyset$, on peut le choisir tel que $\mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma) = \mathrm{Var}(s_{1\uparrow x}) \cup \mathrm{Var}(g)$. On a donc
$$\mathrm{Im}(\sigma) \subseteq \mathrm{Var}(s_{1\uparrow x}) \cup \mathrm{Var}(g)$$
    * Soit $x \in \mathrm{Im}(\sigma)$. Par idempotence, $x \notin \mathrm{Dom}(\sigma)$. Il y a deux cas :
      1. Si $x \in \mathrm{Var}(s_{1\uparrow x})$, alors $x \in V_1 - \mathrm{Dom}(\sigma)$.

2. Si $x \in \mathrm{Var}(g)$ , alors $x \notin \mathrm{Var}(s_{1\uparrow x})$. Néanmoins, $x \in \mathrm{Var}(\sigma(s_{1\uparrow x}))$. Donc $x \in \mathrm{Im}(\sigma_{\uparrow \mathrm{Var}(s_{1\uparrow x})})$. Comme $\mathrm{Var}(s_{1\uparrow x}) \subseteq V_1$, $x \in \mathrm{Im}(\sigma_{\uparrow V_1})$.

On a donc $\mathrm{Im}(\sigma) \subseteq V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma_{\uparrow V_1})$. D'où finalement

$$V_2 = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma) \subseteq V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma_{\uparrow V_1})$$

* Réciproquement, comme $\mathrm{Im}(\sigma_{\uparrow V_1}) \subseteq \mathrm{Im}(\sigma)$, on a aussi $V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma_{\uparrow V_1}) \subseteq V_2$.
* Et donc, $\boxed{V_2 = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma) = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma_{\uparrow V_1})}$.

– Maintenant, montrons que $\theta_2$ est normalisée.
* Comme $\mathrm{Dom}(\theta_2) \subseteq V_2$, il nous suffit de montrer que $\theta_{2\uparrow V_2}$ est normalisée.
* Rappelons les hypothèses :
  1. $\theta_{1\uparrow V_1}$ est normalisée
  2. $\theta_2 \circ \sigma \doteq \theta_1 \ [\ V_1\ ]$
  3. $V_2 = V_1 - \mathrm{Dom}(\sigma) \cup \mathrm{Im}(\sigma_{\uparrow V_1})$
* Soit $x \in V_2$.
  · Si $x \in V_1 - \mathrm{Dom}(\sigma)$, alors $\theta_2(x) = (\theta_2 \circ \sigma)(x) \doteq \theta_1(x)$ qui est normalisé par hypothèse.
  · Sinon, $x \in \mathrm{Im}(\sigma_{\uparrow V_1})$, donc il existe $y \in V_1$ tel que $x \in \mathrm{Var}(\sigma(y))$. Donc $\theta_2(x)$ est bisimilaire à un sous terme de $(\theta_2 \circ \sigma)(y) \doteq \theta_1(y)$. $\theta_1(y)$ est normalisé, et $\theta_2(x)$ est bisimilaire à un sous terme de $\theta_1(y)$, donc $\theta_2(x)$ est normalisé.
* En conséquence, $\boxed{\theta_2 \text{ est normalisée}}$.

$\diamondsuit$

**Théorème 99** : Soient $t_1, t'_1$ deux termes $EI$-unifiables, et $\rho$ une substitution **fermée** solution de $t_1 == t'_1$ : $\rho(t_1) =^{\mathbf{i}}_{\mathcal{SP}} \rho(t'_1)$. Soit $V \supseteq \mathrm{Var}(t_1) \cup \mathrm{Var}(t'_1)$ un ensemble de variables. Alors il existe une dérivation de suréduction :

$$\begin{array}{lll} (t_1 == t'_1) & —\mathcal{N}\longrightarrow_{[x_1, g_1 \to d_1, \sigma_1]} & (t_2 == t'_2) \\ & —\mathcal{N}\longrightarrow \ \ldots & \\ & —\mathcal{N}\longrightarrow_{[x_{n-1}, g_{n-1} \to d_{n-1}, \sigma_{n-1}]} & (t_n == t'_n) \end{array}$$

telle que $t_n$ et $t'_n$ soient unifiables. Si $\mu$ est un de leurs unificateurs minimaux, alors :

$$\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1 \preceq^{\mathbf{i}}_{\mathcal{SP}} \rho \ [\ V\ ]$$

**Preuve** : Par hypothèse, $\rho(t_1) =^{\mathbf{i}}_{\mathcal{SP}} \rho(t'_1)$ et $\rho$ est fermée. Donc $\rho(t_1) =_{\mathcal{E}} \rho(t'_1)$. Il y a deux cas :

1. Si $\rho(t_1) \doteq \rho(t'_1)$, alors soit $\mu$ un unificateur minimal de $t_1$ et $t'_1$. On a $\mu \dot{\preceq} \rho$, donc $\mu \preceq^{\mathbf{i}}_{\mathcal{SP}} \rho\ [\ V\ ]$.
2. Sinon :
   - Soit $\theta_1$ la substitution normalisée associée à $\rho$ : $\forall x, \theta_1(x) = \ \rho(x)\downarrow$. Comme $\theta_1 \equiv^{\mathbf{i}}_{\mathcal{SP}} \rho$, nous avons toujours $\theta_1(t_1) =_{\mathcal{E}} \theta_1(t'_1)$.
   - Maintenant, en appliquant récursivement le lemme précédent, on construit les deux dérivations suivantes :
     (a)

$$\begin{array}{lll} \theta_1(t_1 == t'_1) & \longmapsto_{[x_1, g_1 \to d_1]} & \theta_2(t_2 == t'_2) \\ & \longmapsto \ \ldots & \\ & \longmapsto_{[x_{n-1}, g_{n-1} \to d_{n-1}]} & \theta_n(t_n == t'_n) \end{array}$$

     (b)

$$\begin{array}{lll} (t_1 == t'_1) & —\mathcal{N}\longrightarrow_{[x_1, g_1 \to d_1, \sigma_1]} & (t_2 == t'_2) \\ & —\mathcal{N}\longrightarrow \ \ldots & \\ & —\mathcal{N}\longrightarrow_{[x_{n-1}, g_{n-1} \to d_{n-1}, \sigma_{n-1}]} & (t_n == t'_n) \end{array}$$

telle que $\theta_n(t_n) \doteq \theta_n(t'_n)$. C'est toujours possible puisque $\longmapsto$ est canonique. Les hypothèses du lemme imposent :

(a) i. $V_1 = V \cup \mathrm{Dom}(\theta_1)$

ii. $\forall i \in 2 \ldots n, V_i = V_{i-1} - \mathrm{Dom}(\sigma_{i-1}) \cup \mathrm{Im}(\sigma_{i-1}) \supseteq \mathrm{Var}(t_i == t'_i) \cup \mathrm{Dom}(\theta_i)$

(b) $\forall i \in 1 \ldots n$ , $\theta_i$ est normalisée

(c) $\forall i \in 1 \ldots n-1, (\theta_{i+1} \circ \sigma_i) \doteqdot \theta_i \ [\ V_i\ ]$

- On commence par montrer : $\forall i \in 1 \ldots n, \theta_i \circ \sigma_{i-1} \circ \ldots \sigma_1 \doteqdot \theta_1 \ [\ V_1\ ]$ Le cas est trivial pour $i = 1$. Supposons que ce soit vrai pour $i$. Nous avons : $V_i = V_{i-1} - \mathrm{Dom}(\sigma_{i-1}) \cup \mathrm{Im}(\sigma_{i-1})$. Donc $\mathrm{Im}(\sigma_{i-1} \circ \ldots \circ \sigma_1) \subseteq V_i$. De plus, $\mathrm{Dom}(\theta_i) \subseteq V_i$. Et enfin, $(\theta_{i+1} \circ \sigma_i) \doteqdot \theta_i \ [\ V_i\ ]$. Donc $\theta_{i+1} \circ \sigma_i \circ \ldots \circ \sigma_1 \doteqdot \theta_1 \ [\ V_1\ ]$.
- Nous avons par conséquent : $\theta_n \circ \sigma_{n-1} \circ \ldots \circ \sigma_1 \doteqdot \theta_1 \ [\ V_1\ ]$. D'autre part, $\theta_n$ est un unificateur de $t_n$ et $t'_n$. Soit $\mu$ un de leurs unificateurs minimal. Nous avons par définition : $\mu \dot{\preceq} \theta_n$, c'est-à-dire qu'il existe $\xi$ telle que $\xi \circ \mu \doteqdot \theta_n$. Par conséquent : $\xi \circ \mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1 \doteqdot \theta_1 \ [\ V_1\ ]$. Or $\theta_1 \equiv^{\mathbf{i}}_{S\mathcal{P}} \rho$ d'une part, et $V_1 \subseteq V$ d'autre part. Donc $\xi \circ \mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1 \equiv^{\mathbf{i}}_{S\mathcal{P}} \rho \ [\ V\ ]$. Finalement : $\mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1 \preceq^{\mathbf{i}}_{S\mathcal{P}} \rho \ [\ V\ ]$.

$\diamondsuit$

### 4.3.4 Calcul des ensembles complets de EI-unificateurs

**Théorème 100** : Soient $t_1, t'_1$ deux termes, et $V \supseteq \mathrm{Var}(t_1) \cup \mathrm{Var}(t'_1)$ un ensemble de variables. Soit $ES$ l'ensemble des substitutions $\sigma$ où $\sigma \in ES$ si et seulement si il existe une dérivation de surréduction :

$$\begin{array}{lll} (t_1 == t'_1) & -\!\!\!-\mathcal{N}\!\!\!\longrightarrow_{[x_1, g_1 \to d_1, \sigma_1]} & (t_2 == t'_2) \\ & -\!\!\!-\mathcal{N}\!\!\!\longrightarrow \ \ldots & \\ & -\!\!\!-\mathcal{N}\!\!\!\longrightarrow_{[x_{n-1}, g_{n-1} \to d_{n-1}, \sigma_{n-1}]} & (t_n == t'_n) \end{array}$$

telle que $t_n$ et $t'_n$ soient unifiables, d'unificateur minimal $\mu$ et $\sigma \doteqdot \mu \circ \sigma_{n-1} \circ \ldots \circ \sigma_1$. Alors $ES$ est un ensemble complet de $EI$-unificateurs de $t_1$ et $t'_1$ en dehors de $V$.

**Preuve** : La cohérence et la complétude découlent des deux lemmes précédents. $\diamondsuit$

# Chapter 5

# Sous-sortes

Dans ce chapitre, nous traitons le problème des sous-sortes. Nous commençons par définir une nouvelle syntaxe, telle que la signature possède un ensemble de sortes partiellement ordonné. Puis nous travaillons sur la traduction d'une spécification ordo-sortée vers une spécification multi-sortée. Enfin, nous utilisons cette transformation pour définir la sémantique dénotationnelle des spécifications ordo-sortées.

## 5.1 Syntaxe

Dans cette section nous définissons la syntaxe des spécifications ordo-sortées. Nous commençons par les signatures ordo-sortées : nous structurons l'ensemble des sortes en définissant des sortes maximales (par exemple, les listes, les entiers ...) puis nous définissons un ordre sur les sortes, tel qu'une sous-sorte ne soit comparable qu'avec une unique sorte maximale (par exemple, les listes triées, les entiers positifs, ...).

Nous imposons à chaque sous-sorte $s$ d'être définie à l'aide d'un prédicat distingué : Is_$s$. Un objet d'une certaine sorte ne peut appartenir à une sous-sorte que s'il vérifie ce prédicat. C'est du typage dynamique. Nous sommes donc amené à définir la notion de terme et d'atome potentiel, c'est-à-dire les entités que l'on peut écrire et qui peuvent être bien typées ou non.

Nous définissons enfin les spécifications ordo-sortées.

### 5.1.1 Signature ordo-sortée

**Définition 101** (Signature ordo-sortée) **:** Soient $\Im$ et $\Re$ deux symboles distingués. Une **signature ordo-sortée** $\Sigma$ est un triplet

$$\Sigma = \langle \mathcal{S}, \leq, \Omega, \Pi \rangle$$

tel que

- $\mathcal{S}$ soit un ensemble fini de **sortes**, union disjointe d'un **ensemble de sortes $\Im$inies** $\mathcal{S}_\Im$ et d'un **ensemble de sortes $\Re$ationnelles** $\mathcal{S}_\Re$ :

$$\mathcal{S} = \mathcal{S}_\Im \uplus \mathcal{S}_\Re$$

- $\mathcal{S}$ soit muni d'un ordre partiel $\leq$ tel que :

1. Toute sous-sorte $s$ admet une unique sorte maximale, notée $\check{s}$ :

$$s_1 \leq s_2 \implies \check{s_1} = \check{s_2}$$

On note $\check{\mathcal{S}}$ l'ensemble des **sortes maximales**. $\mathcal{S}$ est donc l'union disjointe d'ensembles $\check{\mathcal{S}}$-indicés :

$$\mathcal{S} = \biguplus_{\check{s} \in \check{\mathcal{S}}} \mathcal{S}_{\check{s}} \text{ avec } \forall s \in \mathcal{S}_{\check{s}} \ , \ s \leq \check{s}$$

2. Les ensembles $\mathcal{S}_{\check{s}}$ conservent la finitude ou la rationalité des sortes, c'est-à-dire si $s \leq s'$ alors :

$$s \in \mathcal{S}_{\Im} \text{ (resp. } \mathcal{S}_{\Re}) \iff s' \in \mathcal{S}_{\Im} \text{ (resp. } \mathcal{S}_{\Re})$$

- $\Omega$ soit une famille $\check{\mathcal{S}}$-indicée d'ensembles, eux-mêmes union disjointe d'ensembles $\mathcal{S}^\star \times \mathcal{S}$-indicés (de noms) de **fonctions** :

$$\Omega = \{\ddot{\Omega}_{\check{s}} \mid \check{s} \in \check{\mathcal{S}}\}$$

avec :

$$\ddot{\Omega}_m = \biguplus_{(w,s) \in \mathcal{S}^\star \times \mathcal{S}} \Omega_{w,s} \text{ où } \check{s} = m$$

- $\Pi$ soit une famille $\check{\mathcal{S}}^+$-indicée d'ensembles, eux-mêmes union disjointe d'ensembles $\mathcal{S}^+$-indicés (de noms) de **prédicats** :

$$\Pi = \{\ddot{\Pi}_{\check{s_1} \ldots \check{s_n}} \mid \check{s_1} \ldots \check{s_n} \in \check{\mathcal{S}}^+\}$$

avec :

$$\ddot{\Pi}_{u_1 \ldots u_n} = \biguplus_{s_1 \ldots s_n \in \mathcal{S}^+} \Pi_{s_1 \ldots s_n} \text{ où } \check{s_i} = u_i$$

Nous supposons par ailleurs :

- $\forall s \in \mathcal{S}$ , $=_{ss} \in \Pi_{ss}$
- $\forall s \in \mathcal{S}$, $\boxed{s < \check{s} \implies \text{Is\_}s \in \Pi_{\check{s}}}$

Pour chaque sous-sorte $s$ qui n'est pas maximale ($s < \check{s}$), nous exigeons du programmeur qu'il spécifie les éléments de sorte $s$ à l'aide du prédicat Is_$s$.

**Remarque** : Cette définition des ensembles de fonctions et de prédicats permet d'éviter le polymorphisme dit de *sous-sorte*. Ainsi, on ne peut pas définir + à la fois sur les naturels et sur les entiers positifs :

- $pos < nat$
- $+ : nat\ nat \longrightarrow nat$
- $+ : pos\ pos \longrightarrow pos$

Dans cet exemple, le problème du polymorphisme de sous-sorte tient à ce que le code associé au + sur les *nat* peut-être différent de celui associé au + sur les *pos*. Que peut valoir la somme de deux entiers positifs, si elle diffère de la somme de ces mêmes entiers considérés comme des *nat* ? Des propositions ont été faites pour des cas particuliers (par exemple, les signatures régulières ou prérégulières (voir [13])). Elle ne s'adapte pas ici, parce que la définition sémantique que nous développerons plus tard confondrait les deux opérations distinctes +. $\diamondsuit$

Nous donnons maintenant un exemple qui sera repris plus tard. Nous commençons par construire un type fini : celui des entiers naturels `nat`, et deux de ses sous-types : celui des entiers positifs `pos`, et celui des âges `age` :

```
finite sort nat

   0 : -> nat
   s : nat -> nat

   <= : nat nat

subsort

   pos < nat

   Is_pos : nat

subsort

   age < nat

   Is_age : age
```

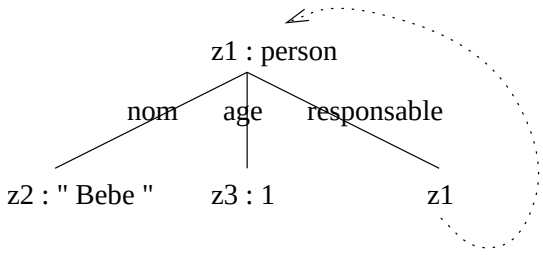
Nous construisons maintenant le type des `humains` et le sous-type des `adultes`. Informellement, un humain a un nom[1], un âge, et un responsable qui doit être un adulte. Un adulte est un humain âgé de plus de 18 ans. Comme nous souhaitons que les adultes soient responsables d'eux-mêmes, le type des humains est rationnel.

```
rational sort human

   person : nom age adult -> human

   nom : human -> nom
   age : human -> age
   responsable : human -> adult

subsort

   adult < human

   Is_adult : human
```

### 5.1.2 Termes potentiels

Nous cherchons maintenant à définir les termes et les atomes. Mais à l'inverse du cas multi-sorté, on ne peut pas décider syntaxiquement si un terme est bien typé ou non.

[1] Ce type est construit par ailleurs.




Ce terme $t_0$ est mal formé, parce qu'un bébé de 1 an ne peut pas être responsable de lui-même. Mais nous ne pouvons le prouver que dynamiquement, en résolvant le but Is_$adult(t_0)$. Bref, nous sommes conduits à définir des **termes potentiels**. Ceux-ci seront des **termes au sens classique** lorsque nous aurons montré (dynamiquement) leurs bons typages.

Nous commençons par structurer les ensembles d'étiquettes et de variables :

- Les étiquettes forment un ensemble $\mathcal{L}$, union disjointe d'ensembles $\check{\mathcal{S}}$-indicés : $\mathcal{L} = \biguplus_{\check{s}\in\check{\mathcal{S}}} \mathcal{L}_{\check{s}}$. Une information supplémentaire sur le typage des étiquettes n'est pas nécessaire.
- Les variables forment un ensemble $\mathcal{X}$, union disjointe d'ensembles $\mathcal{S}$-indicés, $\mathcal{X} = \biguplus_{s\in\mathcal{S}} \mathcal{X}_s$. Nous autorisons le programmeur à utiliser des variables des sous-sortes.

**Définition 102** (TERMES POTENTIELS) : Soit $\Sigma = \langle \mathcal{S}, \leq, \Omega, \Pi\rangle$ une signature ordo-sortée, $\mathcal{X} = \biguplus_{s\in\mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{\check{s}\in\check{\mathcal{S}}} \mathcal{L}_{\check{s}}$ un ensemble d'étiquettes.

- Comme dans le cas multi-sorté, on commence par définir l'ensemble des **pré-termes potentiels** $\breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X}) = \biguplus_{\check{s}\in\check{\mathcal{S}}} \breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}}$ en posant :
  1. $z \in \mathcal{L}_{\check{s}} \Longrightarrow\ z \in \breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}}$
  2. $x \in \mathcal{X}_s, z \in \mathcal{L}_{\check{s}} \Longrightarrow\ z : x \in \breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}}$
  3. $c \in \Omega_{\lambda,s}, z \in \mathcal{L}_{\check{s}} \Longrightarrow\ z : c \in \breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}}$
  4. $$\left.\begin{array}{l} f \in \Omega_{s_1\ldots s_n, s} \\ z \in \mathcal{L}_{\check{s}} \\ t_i \in \breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s_i}}, (i \in 1\ldots n) \end{array}\right\} \Longrightarrow\ z : f(t_1, \ldots, t_n) \in \breve{\mathcal{P}}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}}$$
- Comme dans le cas multi-sorté, il faut ensuite définir pour tout pré-terme potentiel $t$, l'ensemble de ses étiquettes Label($t$), de ses positions Pos($t$) et la relation d'accessibilité $\rightsquigarrow_t$ . On définit ensuite l'ensemble des **termes potentiels (bien formés)** :
$$\mathcal{P}(\Sigma, \mathcal{L}, \mathcal{X}) = \biguplus_{\check{s}\in\check{\mathcal{S}}} \mathcal{P}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}}$$
en posant les mêmes conditions que dans le cas multi-sorté : toutes les étiquettes d'un terme doivent être définies une et une seule fois, et on ne "boucle" pas sur les étiquettes de sortes finies. Pour plus de détail, voir le premier chapitre.

### 5.1.3 Atomes potentiels , clauses et spécification ordo-sortée

**Définition 103** (ATOMES POTENTIELS) : Soit $\Sigma = \langle \mathcal{S}, \leq, \Omega, \Pi \rangle$ une signature ordo-sortée, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, et $\mathcal{L} = \biguplus_{\check{s} \in \check{\mathcal{S}}} \mathcal{L}_{\check{s}}$ un ensemble d'étiquettes. On définit l'ensemble des **atomes potentiels** $\mathcal{AP}(\Sigma, \mathcal{L}, \mathcal{X})$ en posant :

$$\left.\begin{array}{l} P \in \Pi_{s_1 \ldots s_n} \\ t_i \in \mathcal{P}(\Sigma, \mathcal{L}, \mathcal{X})_{\check{s}_i}, (i \in 1 \ldots n) \end{array}\right\} \Longrightarrow P(t_1, \ldots, t_n) \in \mathcal{AP}(\Sigma, \mathcal{L}, \mathcal{X})$$

Rien ne change dans la définition des clauses de Horn définies, par rapport au cas multi-sorté.

**Définition 104** (SPÉCIFICATION ORDO-SORTÉE) : Une **spécification ordo-sortée** dans la logique des clauses de Horn avec égalité est un couple

$$\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$$

où $\Sigma$ est une signature ordo-sortée, et $\mathcal{HC}$, un ensemble de clauses de Horn définies.

**Remarque** : Dans la suite, nous utiliserons le prédicat d'égalité == qui réfère tous les prédicats $=_{\check{s}\check{s}}$ (de sortes maximales). Nous n'autorisons pas le programmeur de spécifier les prédicats d'égalités $=_{ss}$ (de sortes quelconques) pour des problèmes d'incohérence de spécification (polymorphisme de sous-sortes ...) $\diamondsuit$

Nous donnons maintenant les équations et les clauses associées aux signatures précédentes :

- Celles concernant le type `nat` et ses sous-types :

```
x, y : nat .

0 <= x .
s( x ) <= s( y ) <== x <= y .

Is_pos( s( x ) ).

Is_age( x ) <== x <= 130
```

- Celles concernant le type `humain` et son sous-type `adulte` :

```
nom( person( n : nom , a : age , r : human ) ) == n .
age( person( n : nom , a : age , r : human ) ) == a .
responsable( person( n : nom , a : age , r : human ) ) == r .

Is_adult( x : person( n : nom , a : age , x ) ) <==  18 <= a
```

## 5.2 Transformation des spécifications ordo-sortées

Pour définir la sémantique dénotationnelle d'une spécification ordo-sortée, nous allons la traduire en une spécification multi-sortée. Cette transformation n'est pas immédiate.

En effet, considérons la spécification du type `humain`, et le terme `x:person(Bebe,1,x)` , donné précédemment sous forme de graphe. Ce terme est mal typé dans la sorte maximale `humain` : un bébé de 1 an ne peut pas être responsable de lui-même ; seul les adultes de plus de 18 ans peuvent l'être.

Le problème est le suivant : il faut utiliser le prédicat Is_*adult* et ses clauses de définition pour prouver que ce terme est mal formé, et donc qu'il n'appartient pas au type `humain`. Or, on ne peut utiliser un prédicat qu'avec des termes bien typés. Aussi, il faudrait que $t_0$ soit un `humain` pour prouver qu'il ne l'est pas !

Pour résoudre ce paradoxe, nous sommes conduits à compléter les signatures en y ajoutant de nouvelles sortes maximales : les **sur-sortes**. Tous les termes mal typés seront typables dans cette sur-sorte. Tous les termes bien typés seront typables dans la sorte maximale actuelle. C'est cette complétion de signature que nous décrivons dans le paragraphe suivant.

### 5.2.1 Complétion d'une spécification ordo-sortée

Soient $\Sigma = \langle \mathcal{S}, \leq, \Omega, \Pi \rangle$ une signature ordo-sortée, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{\check{s} \in \check{\mathcal{S}}} \mathcal{L}_{\check{s}}$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification ordo-sortée. On transforme la signature $\Sigma$ en une autre signature ordo-sortée $\Sigma'$,

$$\Sigma' = \langle \mathcal{S}', \leq', \Omega', \Pi' \rangle$$

telle que :

- On ajoute à chaque ensemble $\mathcal{S}_{\check{s}}$ une nouvelle sorte, dite **sur-sorte**, notée $\tilde{s}$. On note $\tilde{\mathcal{S}}$ l'ensemble des sur-sortes.
- $\leq'$ est défini par : $\leq' = \leq \cup \{(\check{s}, \tilde{s})\}$.
- On n'ajoute pas de nouveaux symboles de fonctions. On a donc $\Omega' = \{\ddot{\Omega}'_{\tilde{s}} \mid \tilde{s} \in \tilde{\mathcal{S}}\}$, avec $\ddot{\Omega}'_{\tilde{s}} = \ddot{\Omega}_{\check{s}}$ .
- Par contre, l'ensemble des noms de prédicats change. Premièrement, on ajoute le prédicat d'égalité $=_{\tilde{s}\tilde{s}}$ pour chaque sur-sorte $\tilde{s}$. Deuxièmement, on ajoute l'ensemble des prédicats Is_$s'$ : $\tilde{s}$ pour tous les $s' <' \tilde{s}$, et en particulier pour $\check{s}$. Troisièmement, on supprime les prédicats Is_$s'$ : $\check{s}$. On obtient donc $\Pi' = \{\ddot{\Pi}_{\tilde{s_1} \ldots \tilde{s_n}} \mid \tilde{s_1} \ldots \tilde{s_n} \in \tilde{\mathcal{S}}^+\}$ avec :
  1. $\forall \tilde{s} \in \tilde{\mathcal{S}}$ , $\ddot{\Pi}'_{\tilde{s}\tilde{s}} = \{=_{\check{s}\check{s}}\} \cup \ddot{\Pi}_{\check{s}\check{s}}$
  2. $\forall \tilde{s} \in \tilde{\mathcal{S}}$ , $\ddot{\Pi}'_{\tilde{s}} = \{\text{Is\_}s' : \tilde{s} \mid s' <' \tilde{s}\} \cup (\ddot{\Pi}_{\check{s}} - \{\text{Is\_}s' : \check{s} \mid s' <' \check{s}\})$
  3. Dans les autres cas, les noms de prédicats ne changent pas : $\ddot{\Pi}'_{\tilde{s_1} \ldots \tilde{s_n}} = \ddot{\Pi}_{\check{s_1} \ldots \check{s_n}}$

  **Proposition 105** : La signature $\Sigma' = \langle \mathcal{S}', \leq', \Omega', \Pi' \rangle$ est une signature ordo-sortée.

Pour finir, on transforme la spécification $\mathcal{SP}$ en une autre spécification ordo-sortée $\mathcal{SP}' = \langle \Sigma', \mathcal{HC}' \rangle$. Il s'agit d'introduire de nouvelles clauses, celles qui définissent les symboles Is_$\check{s}$ : $\tilde{s}$. On se donne un ensemble d'étiquettes $\mathcal{L}' = \biguplus_{\tilde{s} \in \tilde{\mathcal{S}}} \mathcal{L}'_{\tilde{s}}$, et un ensemble de variables $\mathcal{X}' = \biguplus_{s \in \mathcal{S}'} \mathcal{X}'_s$. On pose $\mathcal{HC}' = \mathcal{HC} \cup \mathcal{CL}'$ avec :

$$\left.\begin{array}{l} s \in \mathcal{S} \\ c \in \Omega'_{\lambda,s} \\ z \in \mathcal{L}'_{\tilde{s}} \end{array}\right\} \Longrightarrow \boxed{\text{Is\_}\check{s}(z : c) \in \mathcal{CL}'} \quad \text{et} \quad \left.\begin{array}{l} s \in \mathcal{S} \\ f \in \Omega'_{s_1 \ldots s_n, s} \\ x_i \in \mathcal{X}'_{s_i} \\ z \in \mathcal{L}'_{\tilde{s}} \end{array}\right\} \Longrightarrow \boxed{\text{Is\_}\check{s}(z : f(x_1, \ldots, x_n)) \in \mathcal{CL}'}$$

En guise d'exemple, voici la définition du type `humain'` obtenue à partir du type `humain`. On constatera que `x : person(Bebe, 1, x)` est un terme potentiel du langage, qu'il est bien typé dans `humain'`, et qu'il est mal typé dans `humain` :

```
rational sort human'

subsort

   human < human'

   person : nom age adult -> human

   nom : human -> nom
   age : human -> age
   responsable : human -> adult

   Is_human : human'

subsort

   adult < human'

   Is_adult : human'

clauses

   nom( person( n : nom , a : age , r : human ) ) == n .
   age( person( n : nom , a : age , r : human ) ) == a .
   responsable( person( n : nom , a : age , r : human ) ) == r .

   Is_adult( x : person( n : nom , a : age , x ) ) <==  18 <= a

   Is_human( person( n : nom , a : age , r : human ) ) .
   Is_human( responsable( h : human ) ) .
```

### 5.2.2 Transformation de la signature ordo-sortée complétée

Dans ce paragraphe, nous définissons une signature multi-sortée à partir d'une signature ordo-sortée complétée.

On suppose construit $\Sigma' = \langle \mathcal{S}', \leq', \Omega', \Pi' \rangle$. On définit $\tilde{\Sigma} = \langle \tilde{\mathcal{S}}, \tilde{\Omega}, \tilde{\Pi} \rangle$ telle que :

- $\tilde{\mathcal{S}}$ soit l'ensemble des sortes maximale de $\mathcal{S}'$ (le $\tilde{\mathcal{S}}$ défini lors de la complétion).
- $\tilde{\Omega}$ soit la famille $\{\tilde{\Omega}_{w,s} \mid (w,s) \in \tilde{\mathcal{S}}^\star \times \tilde{\mathcal{S}}\}$ avec :
  1. $c \in \Omega'_{\lambda,s} \implies \tilde{c} \in \tilde{\Omega}_{\lambda,\tilde{s}}$
  2. $f \in \Omega'_{s_1 \ldots s_n, s} \implies \tilde{f} \in \tilde{\Omega}_{\tilde{s_1} \ldots \tilde{s_n}, \tilde{s}}$
- $\tilde{\Pi}$ soit la famille $\{\tilde{\Pi}_w \mid w \in \tilde{\mathcal{S}}^+\}$ avec : $P \in \Pi'_{s_1 \ldots s_n} \implies \tilde{P} \in \tilde{\Pi}_{\tilde{s_1} \ldots \tilde{s_n}}$

### 5.2.3 Transformation des termes et des atomes

Dans ce paragraphe, nous donnons les définitions nécessaires pour transformer la spécification ordo-sortée complétée en une spécification multi-sortée.

#### Compilation des termes et des atomes

Ce paragraphe vise à traduire les termes et les atomes de $\mathcal{SP}'$ en des termes et des atomes de $\tilde{\mathcal{SP}}$. On se donne un ensemble d'étiquettes $\tilde{\mathcal{L}} = \biguplus_{\tilde{s} \in \tilde{\mathcal{S}}} \tilde{\mathcal{L}}_{\tilde{s}}$, et un ensemble de variables $\tilde{\mathcal{X}} = \biguplus_{\tilde{s} \in \tilde{\mathcal{S}}} \tilde{\mathcal{X}}_{\tilde{s}}$.

On définit l'application $\mathcal{R} : \mathcal{P}(\Sigma', \mathcal{L}', \mathcal{X}') \to \mathcal{T}(\tilde{\Sigma}, \mathcal{L}, \tilde{\mathcal{X}})$ pour tout terme potentiel $t \in \mathcal{P}(\Sigma', \mathcal{L}', \mathcal{X}')$ en posant :

1. $\mathcal{R}(z) = \tilde{z}$
2. $\mathcal{R}(z : x) = \tilde{z} : \tilde{x}$
3. $\mathcal{R}(z : c) = \tilde{z} : \tilde{c}$
4. $\mathcal{R}(z : f(t_1, \ldots, t_n)) = \tilde{z} : \tilde{f}(\mathcal{R}(t_1), \ldots, \mathcal{R}(t_n))$

Nous aurons aussi besoin de sa réciproque $\mathcal{R}^{-1}$ lors de la définition de la sémantique dénotationnelle d'une spécification ordo-sortée. Nous ne la définissons que sur les termes fermés en posant :

1. $\mathcal{R}^{-1}(\tilde{z}) = z$
2. $\mathcal{R}^{-1}(\tilde{z} : \tilde{c}) = z : c$
3. $\mathcal{R}^{-1}(\tilde{z} : \tilde{f}(\tilde{t_1}, \ldots, \tilde{t_n})) = z : f(\mathcal{R}^{-1}(\tilde{t_1}), \ldots, \mathcal{R}^{-1}(\tilde{t_n}))$

#### Contraintes de types

Ce paragraphe définit une application qui à tout terme potentiel $t$ associe une conjonction de buts. Elle correspond aux contraintes de type qu'il faut montrer dans la spécification multi-sortée pour prouver que $t$ est un terme bien typé dans la spécification ordo-sortée.

On définit l'application $\mathcal{C}$ pour tout terme potentiel $t \in \mathcal{P}(\Sigma', \mathcal{L}', \mathcal{X}')$ en posant :

$$\mathcal{C}(t) = \bigwedge_{z \in \mathrm{Pos}(t)} \mathcal{C}'(z, t)$$

où $\mathcal{C}'$ est définie de la manière suivante :

1. Contraintes de type sur les variables :
   (a) Cas des variables de sorte maximale :

$$\left.\begin{array}{l} t_{\uparrow z} = z : x \\ x \in \mathcal{X}'_{\check{s}} \\ z \in \mathcal{L}'_{\check{s}} \end{array}\right\} \Longrightarrow \mathcal{C}'(z, t) = \emptyset$$

(b) Cas des variables de sorte quelconques :

$$\left.\begin{array}{l} t_{\uparrow z} = z : x \\ x \in \mathcal{X'}_s \\ s < \check{s} \\ z \in \mathcal{L'}_{\tilde{s}} \end{array}\right\} \Longrightarrow \mathcal{C}'(z,t) = \mathcal{R}(\text{Is\_}s(z:x))$$

2. Contraintes de type sur les constantes :

(a) Cas des constantes de sorte maximale :

$$\left.\begin{array}{l} t_{\uparrow z} = z : c \\ c \in \Omega'_{\lambda,\check{s}} \\ z \in \mathcal{L'}_{\tilde{s}} \end{array}\right\} \Longrightarrow \mathcal{C}'(z,t) = \emptyset$$

(b) Cas des constantes de sorte quelconque :

$$\left.\begin{array}{l} t_{\uparrow z} = z : c \\ c \in \Omega'_{\lambda,s} \\ s < \check{s} \\ z \in \mathcal{L'}_{\tilde{s}} \end{array}\right\} \Longrightarrow \mathcal{C}'(z,t) = \mathcal{R}(\text{Is\_}s(z:c))$$

3. Contraintes de type sur les termes fonctionnels :

$$\left.\begin{array}{l} t_{\uparrow z} = z : f(\phi_1, \ldots, \phi_n) \\ f \in \Omega'_{s_1 \ldots s_n, s} \\ \phi_i \in \check{\mathcal{P}}(\Sigma', \mathcal{L}', \mathcal{X}')_{\tilde{s_i}} \\ \quad \text{avec } \text{Rac}(\phi_i) = z_i \\ \quad (i \in 1 \ldots n) \\ z \in \mathcal{L}_{\tilde{s}} \end{array}\right\} \Longrightarrow \mathcal{C}'(z,t) = \mathcal{R}(\text{Is\_}s(t_{\uparrow z})) \wedge \bigwedge_{i=1}^{n} \mathcal{R}(\text{Is\_}s_i(t_{\uparrow z_i}))$$

Nous étendons $\mathcal{C}$ aux atomes en posant :

$$\left.\begin{array}{l} P : s_1 \ldots s_n \\ t_i \in \mathcal{P}(\Sigma', \mathcal{L}', \mathcal{X}')_{\tilde{s_i}} \\ (i \in 1 \ldots n) \end{array}\right\} \Longrightarrow \mathcal{C}(P(t_1, \ldots, t_n)) = \bigwedge_{i=1}^{n} (\mathcal{R}(\text{Is\_}s_i(t_i) \wedge \mathcal{C}(t_i))$$

Pour simplifier les écritures, nous introduisons une dernière application sur les atomes potentiels (de $\mathcal{AP}(\Sigma', \mathcal{L}', \mathcal{X}')$) qui est la conjonction des deux précédentes. On la note $\mathcal{B}$, et on pose pour tout atome potentiel $A$ :

$$\mathcal{B}(A) = \mathcal{R}(A) \wedge \mathcal{C}(A)$$

On étend cette même application aux buts en posant :

$$\mathcal{B}(A_0 \wedge \ldots \wedge A_n) = \mathcal{B}(A_0) \wedge \ldots \wedge \mathcal{B}(A_n)$$

### 5.2.4 Transformation de la spécification

Nous sommes maintenant en mesure de transformer une spécification ordo-sortée en une spécification multi-sortée. On rappelle que l'on dispose de la signature ordo-sortée $\Sigma' = \langle \mathcal{S}', \leq' , \Omega', \Pi' \rangle$, de la signature multi-sortée $\tilde{\Sigma} = \langle \tilde{\mathcal{S}}, \tilde{\Omega}, \tilde{\Pi} \rangle$, et de la spécification ordo-sortée $\mathcal{SP}' = \langle \Sigma', \mathcal{HC}' \rangle$.

On définit la spécification multi-sortée $\tilde{\mathcal{SP}} = \langle \tilde{\Sigma}, \tilde{\mathcal{HC}} \rangle$ en posant :

$$\begin{aligned} \tilde{\mathcal{HC}} = \quad & \{\mathcal{R}(A_0) \impliedby \mathcal{C}(A_0) \mid A_0 \in \mathcal{HC}'\} \cup \\ & \{\mathcal{R}(A_0) \impliedby \mathcal{C}(A_0)\mathcal{B}(A_1)\ldots\mathcal{B}(A_n) \mid A_0 \impliedby A_1 \ldots A_n \in \mathcal{HC}'\} \end{aligned}$$

## 5.3 Sémantique dénotationnelle

Nous définissons maintenant la sémantique dénotationnelle des spécifications ordo-sortées. Nous commençons par définir une famille de relations de congruence partielles sur les termes potentiels. Puis nous donnons la sémantique dénotationnelle d'une spécification ordo-sortée.

### 5.3.1 Congruence $\asymp_{\mathcal{SP}}$

**Définition 106** (RELATION $\asymp_{\mathcal{SP}}$) **:** Soient $\Sigma = \langle \mathcal{S}, \leq, \Omega, \Pi \rangle$ une signature ordo-sortée, $\mathcal{X} = \biguplus_{s \in \mathcal{S}} \mathcal{X}_s$ un ensemble de variables, $\mathcal{L} = \biguplus_{\check{s} \in \check{\mathcal{S}}} \mathcal{L}_{\check{s}}$ un ensemble d'étiquettes, et $\mathcal{SP} = \langle \Sigma, \mathcal{HC} \rangle$ une spécification ordo-sortée. Soient $\tilde{\Sigma} = \langle \tilde{\mathcal{S}}, \tilde{\Omega}, \tilde{\Pi} \rangle$ la signature multi-sortée transformée de $\Sigma$, et $\tilde{\mathcal{SP}} = \langle \tilde{\Sigma}, \tilde{\mathcal{HC}} \rangle$ la spécification multi-sortée transformée de $\mathcal{SP}$. Soient $t_1$ et $t_2$ deux termes de $\mathcal{P}(\Sigma, \mathcal{L}, \mathcal{X})_{\tilde{s}}$. On définit la famille $\mathcal{S}$-indicée de relation $\asymp_{\mathcal{SP}} = \{\asymp^s_{\mathcal{SP}} \mid s \in \mathcal{S}\}$ en posant :

$$t_1 \asymp^s_{\mathcal{SP}} t_2 \iff \mathcal{EMS}_{\tilde{\mathcal{SP}}} \models \mathcal{B}(t_1 == t_2) \wedge \mathcal{R}(\text{Is\_}s(t_1)) \wedge \mathcal{R}(\text{Is\_}s(t_2))$$

**Proposition 107** : $\asymp^s_{\mathcal{SP}}$ est une relation de congruence partielle[2] sur $\mathcal{P}(\Sigma, \mathcal{L}, \mathcal{X})$. On note $[\ t\ ]_{\asymp^s_{\mathcal{SP}}}$ la classe d'équivalence de tout terme $t$ par rapport à $\asymp^s_{\mathcal{SP}}$.

**Remarque** : Soient $s_1 \leq s_2$ deux sortes, et $t$ un terme potentiel de sorte $\tilde{s_2}$. Alors :

- soit $[\ t\ ]_{\asymp^{s_1}_{\mathcal{SP}}} = \emptyset$
- soit $[\ t\ ]_{\asymp^{s_1}_{\mathcal{SP}}} = [\ t\ ]_{\asymp^{s_2}_{\mathcal{SP}}}$

$\diamondsuit$

**Exemple** : Nous commençons par définir le type `nat` des entiers naturels à partir de la constante 0 et de la fonction successeur $s$ :

```
0 : -> nat
s : nat -> nat
```

Nous définissons maintenant le sous-type des entiers naturels positifs :

```
subsort pos < nat

Is_pos( s( x ) ).
```

[2] Une congruence est partielle si la réflexivité n'est pas définie sur tout le domaine considéré (voir [15]).

On a :

- $[\ 0\ ]^{pos}_{\asymp_{\mathcal{SP}}} = \emptyset \subset [\ 0\ ]^{nat}_{\asymp_{\mathcal{SP}}}$
- $[\ s(x)\ ]^{pos}_{\asymp_{\mathcal{SP}}} = [\ s(x)\ ]^{nat}_{\asymp_{\mathcal{SP}}}$

$\diamondsuit$

### 5.3.2 Sémantique

On note le E-modèle standard de $\tilde{\mathcal{SP}}$ par :

$$\mathcal{EMS}_{\tilde{\mathcal{SP}}} = \langle \widehat{\mathcal{T}(\tilde{\Sigma}, \tilde{\mathcal{L}})}_{/\cong}, \tilde{\alpha}, \tilde{\beta} \rangle$$

La sémantique de la spécification ordo-sortée $\mathcal{SP}$ est donnée par la structure suivante $\mathcal{ST}$ :

$$\mathcal{ST} = \langle A, \alpha, \beta \rangle$$

où :

- $A$ est une union $\mathcal{S}$-indicée d'ensembles :

$$A = \bigcup_{s \in \mathcal{S}} A_s$$

tel que :

$$A_s = \{[\ t\ ]^s_{\asymp_{\mathcal{SP}}} \mid t \in \mathcal{P}(\Sigma, \mathcal{L})_{\check{s}}\}$$

- $\alpha$ est une famille $\mathcal{S}^\star \times \mathcal{S}$-indicée d'ensembles d'applications :

$$\alpha = \{\alpha_{w,s} : \Omega_{w,s} \longrightarrow {A_s}^{A_w} \mid (w, s) \in \mathcal{S}^\star \times \mathcal{S}\}$$

avec :

1. $c \in \Omega_{\lambda,s} \implies \alpha_{\lambda,s}(c) = [\ z : c\ ]^s_{\asymp_{\mathcal{SP}}}$
2. $$f \in \Omega_{s_1 \ldots s_n, s} \implies \alpha_{s_1 \ldots s_n, s}(f) \left| \begin{array}{l} A_{s_1} \times \ldots \times A_{s_n} \longrightarrow A_s \\ ([\ t_1\ ]^{s_1}_{\asymp_{\mathcal{SP}}}, \ldots, [\ t_n\ ]^{s_n}_{\asymp_{\mathcal{SP}}}) \longmapsto r \end{array} \right.$$

   où :

   (a) $r = [\ \mathcal{R}^{-1}(t)\ ]^s_{\asymp_{\mathcal{SP}}}$

   (b) $t \in \tilde{\alpha}_{\tilde{s_1} \ldots \tilde{s_n}, \tilde{s}}(\tilde{f})([\ \widehat{\mathcal{R}(t_1)}\ ], \ldots, [\ \widehat{\mathcal{R}(t_n)}\ ])$

   (c) $\mathcal{R}^{-1}$ transforme les termes fermés sur la signature multi-sortée en termes (effectifs, non potentiels) sur la signature ordo-sortée (cf. p. 8).

- $\beta$ est une famille $\mathcal{S}^+$-indicée d'ensembles d'applications :

$$\beta = \{\beta_w : \Pi_w \longrightarrow \mathbf{2}^{A_w} \mid w \in \mathcal{S}^+\}$$

avec :

$$P \in \Pi_{s_1 \ldots s_n} \implies \beta_{s_1 \ldots s_n}(P) = \{([\ t_1\ ]^{s_1}_{\asymp_{\mathcal{SP}}}, \ldots, [\ t_n\ ]^{s_n}_{\asymp_{\mathcal{SP}}}) \mid \mathcal{EMS}_{\tilde{\mathcal{SP}}} \models \mathcal{B}(P(t_1, \ldots, t_n))\}$$

### 5.3.3 Spécification cohérente

Nous terminons ce chapitre en définissant la notion de spécification cohérente. Informellement, une spécification ordo-sortée $\mathcal{SP}$ est dite cohérente si il y a adéquation entre l'ordre partiel sur les sortes et les extensions des prédicats Is_$s$ d'une part, et si aucune des fonctions définies n'est vide d'autre part.

**Définition 108** (SPÉCIFICATION COHÉRENTE) **:** Une spécification est **cohérente** si :

- $s_1 \leq s_2 \implies A_{s_1} \subseteq A_{s_2}$
- $\forall f : s_1 \ldots s_n \rightarrow s, \alpha(f) \neq \emptyset$

# Conclusion

## Bilan du projet

Dans notre projet, nous avons défini les rudiments d'un nouveau langage logico-fonctionnel où le programmeur peut utiliser des graphes et des sous-types. Pour celà, nous avons proposé une syntaxe qui permet à la fois :

- de définir des structures de données traditionnelles (sortes de premier ordre) telles que les *entiers naturels*, les *arbres binaires*, ...
- de définir des structures de données à base de graphes. Rappelons ici leur importance en base de données, où elles permettent de modéliser simplement les relations sémantiques qui existent entre différents objets (par exemple : les *humains*, les *villes*,...).
- de définir des sous-types. Plus précisément, elle permet de caractériser des ensembles de données satisfaisant un même invariant. Les invariants sont définis par des clauses de Horn (avec égalité).
- de définir des fonctions à l'aide de systèmes de réécriture de graphe. Les domaines de définition peuvent être décrits de manière précise en utilisant les sous-types.
- de définir des prédicats à l'aide de clauses de Horn avec égalité.

Nous avons ensuite défini une sémantique dénotationnelle pour les programmes que nous considérons. Dans le cas où ils n'utilisent pas de sous-type, elle est donnée par le E-modèle standard d'un programme, c'est-à-dire l'intersection de tous ses E-modèles syntaxiques (de Herbrand). Dans le cas où un programme $P$ utilise des sous-types, nous avons défini sa sémantique en fonction de celle d'un programme $Q$ sans sous-types. Le programme $Q$ est le résultat de l'application d'une transformation, que nous avons décrite, sur le programme $P$.

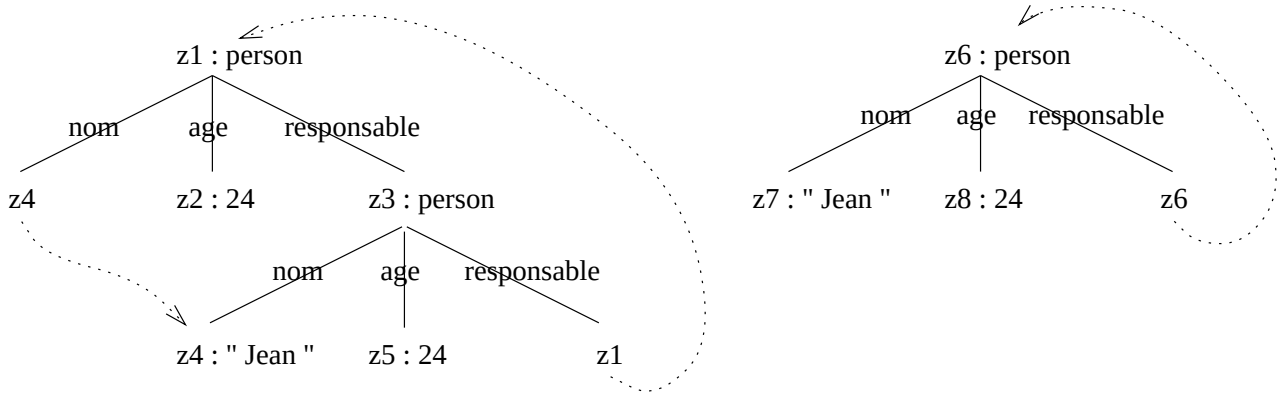
Pour finir, nous avons proposé une sémantique opérationnelle pour la classe de programmes considérée. Cette sémantique à pour objectif la résolution des buts. Pour celà, nous avons proposé l'utilisation d'une unique règle d'inférence : la règle de SLDEI-résolution. Cette règle est paramétrée par un algorithme général de résolution d'équations. Nous en avons proposé un dans le cadre des systèmes de réécriture de graphe non conditionnels. Nous en avons montré la cohérence et la complétude. Nous soulignons de nouveau qu'aucun langage logico-fonctionnel n'est doté d'une telle sémantique opérationnelle actuellement.

## Perspectives

Beaucoup de travail reste à faire sur le sujet.

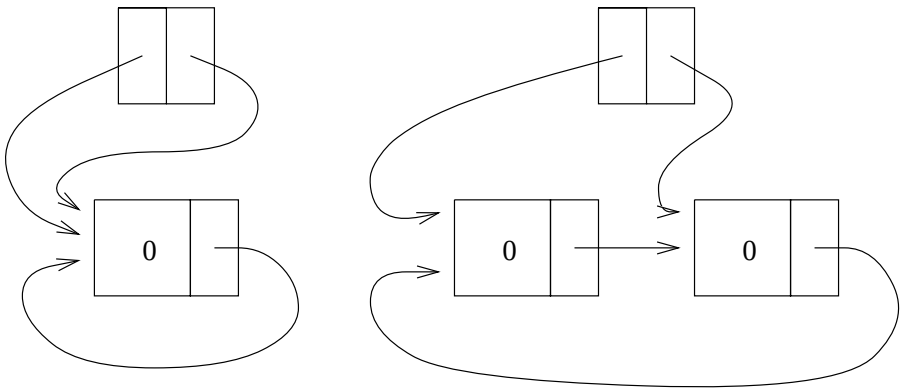
### Structures de données

Les graphes que nous manipulons sont bien adaptés pour traiter des exemples de bases de données. En effet, nous avons défini une congruence, appelée bisimulation ($\doteq$), qui identifie des graphes représentant la même information :

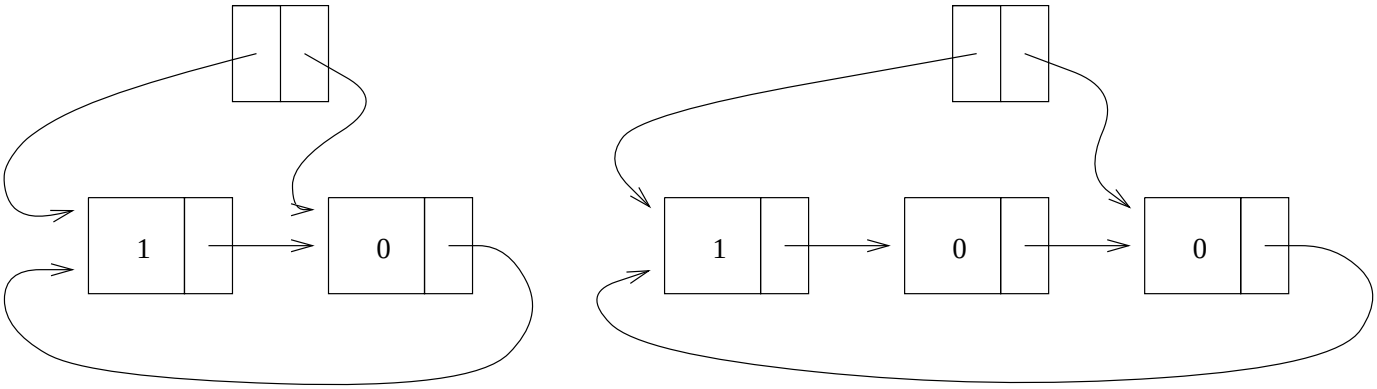



Ces deux graphes "signifient" que Jean, 24 ans, est responsable de lui-même. Ils sont bisimilaires au sens où nous l'avons définis. De plus, le second graphe est dit canonique ; c'est le plus petit graphe qui représente l'information.

Maintenant, la relation de bisimulation n'est pas toujours souhaitable lorsqu'on manipule des structures de données quelconque. Prenons les deux graphes suivants, qui représentent des listes circulaires de 0.




Ces deux graphes sont bisimilaires. Maintenant, si on insère l'élément 1 en tête des listes, les résultats obtenus sont les suivants :




Ces deux graphes ne sont plus bisimilaires. Autrement dit, l'insertion n'est pas compatible avec la bisimulation. Or les listes circulaires sont des éléments qu'un programmeur doit pouvoir manipuler.

Nous constatons donc que la relation de bisimulation $\doteq$ est bien adaptée pour certaines structures de données et qu'elle est mal adaptée pour d'autres. Il est donc nécessaire d'entreprendre une étude approfondie sur les différents prédicats d'égalité que l'on souhaite avoir dans un langage de programmation (par exemple la bisimulation $\doteq$, l'$\alpha$-conversion $=_\alpha$, le prédicat d'égalité logique $==$, l'égalité syntaxique sur les étiquettes ...).

### Optimisation du calcul

Dans ce projet, nous avons donné une première sémantique opérationnelle aux programmes. Elle se fonde sur la SLDEI-résolution, qui nécessite un algorithme de résolution d'équations. Nous avons effectivement proposé un algorithme dans le cadre où nous avons affaire à un système de réécriture de graphes non conditionnel.

Il est fondé sur une extension de la relation de surréduction, aux graphes. Toutefois, il développe un espace de recherche qui contient des informations redondantes dans le cas général. Nous envisageons donc de l'améliorer en proposant des stratégies de calcul. Il nous faudra ensuite considérer les systèmes de réécriture de graphe conditionnels. Dans le cadre des langages à termes finis, ce sujet n'est pas encore très bien maîtrisé. Ils posent des problèmes fondamentaux : terminaison, confluence... Ce sont des problèmes ouverts dans le cadre des graphes.

### Sous-types

Pour tenir compte des sous-types, nous avons proposé un mécanisme qui traduit un programme avec sous-sortes en un programme sans sous-sortes. Cela nous permet de donner la sémantique dénotationnelle et opérationnelle de tout programme. Cependant, la traduction que nous avons proposée ne permet pas d'envisager un calcul optimal. Nous allons voir pourquoi sur un exemple.

Considérons l'exemple des listes triées. Nous commençons par définir le type des *listes finies d'entiers naturels*, construit à partir de `nil` (la liste vide), et de `cons` :

```
nil : -> list
cons : (nat list) -> list
```

Nous définissons maintenant le type des *listes triées d'entiers*, un sous-type de celui des *listes*. Nous donnons donc la définition du prédicat `Is_sorted_list` : d'abord, la liste vide et la liste

à un élément sont des listes triées ; ensuite, une liste quelconque est triée si le premier élément est plus petit que le second, et que la queue de la liste est triée. Ceci s'écrit :

```
sorted_list < list

Is_sorted_list( nil ).
Is_sorted_list( cons(x, nil) ).
Is_sorted_list( cons(x, cons(y , l)) )
          <== (x <= y), Is_sorted_list( cons(y, l) ).
```

Considérons maintenant une opération qui insère un entier naturel dans une liste. Nous en donnons le profil, puis l'ensemble des clauses équationnelles qui la définissent :

```
insert : nat sorted_list -> sorted_list

insert( x, nil ) == cons(x, nil).

insert( x, cons(y, l) ) == cons(x, cons(y, l))
          <== ( x <= y ).

insert( x, cons(y, l) ) == cons(y, insert( x, l ))
          <== ( x > y ).
```

La fonction `insert` insère un *entier naturel* dans une *liste triée* et "renvoie" une *liste triée*. Pour exécuter ce programme, il faut commencer par le traduire vers un programme sans sous-sorte. La transformation que nous avons proposée donne :

```
insert( x, nil ) == cons(x, nil)
          <== Is_sorted_list( nil ),
              Is_sorted_list( cons(x, nil) ).

insert( x, cons(y, l) ) == cons(x, cons(y, l))
          <== ( x <= y ),
              Is_sorted_list( cons(y, l) ),
              Is_sorted_list( cons(x, cons(y, l)) ).

insert( x, cons(y, l) ) == cons(y, insert( x, l ))
          <== ( x > y ),
              Is_sorted_list( cons(y, l) ),
              Is_sorted_list( cons(y, insert( x, l )) ).
```

Le calcul à partir du programme issu de la transformation exige des vérifications de typage redondantes. En effet :

1. La première clause nécessite de montrer que la liste vide et la liste à un seul élément sont triés. C'est trivial dans la mesure où ce sont des faits que nous avons donné dans la définition de `Is_sorted_list`. Nous pouvons prouver statiquement que ce sont des buts toujours vrais.

2. Ensuite, les mêmes buts sont prouvés plusieurs fois. Par exemple, pour montrer qu'une liste est triée, il est nécessaire de la parcourir entièrement. Or l'opération `insert` est récursive (s'appelle elle-même), et la transformation exige que l'on prouve à chaque fois le bon typage des arguments. Donc le calcul doit vérifier plusieurs fois que les sous-listes sont triées.

Clairement, une optimisation de la transformation est nécessaire pour faire une implantation. Plusieurs solutions sont envisageables. D'abord, il est nécessaire de supprimer tous les tests inutiles. Nous pouvons y arriver en construisant un générateur d'obligations de preuve, qui seraient démontrées par ailleurs (à l'aide d'un démonstrateur). Ensuite, il est nécessaire de mémoriser ce que l'on a déjà prouvé au niveau des termes que l'on manipule. Une solution ([23]) consiste par exemple à décorer les termes avec les noms des types dont nous avons déjà montré l'appartenance.

# Bibliography


[1] D. DeGroot and G. Lindstrom, editors. *Logic Programming, Functions, Relations, and Equations*. Prentice Hall, 1986.

[2] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[3] J. A. Goguen, C. Kirchner, H. Kirchner, A. Mégrelis, J. Meseguer, and T. Winkler. An introduction to obj 3. In *LNCS 308*, pages 258–263. Springer Verlag, 1987.

[4] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. In *Proc. 21st ACM Symposium on Principles of Programming Languages*, pages 268–279, Portland, 1994.

[5] H. Dörr, editor. *Efficient Graph Rewriting and Its Implementation*. Springer LNCS 922, 1995.

[6] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[7] A. Colmerauer, H. Kanoui, and M. Van Caneghem. Prolog, bases théoriques et développement actuels. *TSI*, 2.4:271–312, 1983.

[8] H. Aït-Kaci and A. Podelski. Towards a meaning of life. Technical report PRL-11, DEC Paris Research Laboratory, 1992. Disponible sur le Web : http://www.isg.sfu.ca/life/.

[9] H. Aït-Kaci and al. Life : a natural language for natural languages. Mcc technical report ACA-ST-074-88, DEC Paris Research Laboratory, 1988.

[10] H. Aït-Kaci and al. A database interface for complex objects. Technical report PRL-27, DEC Paris Research Laboratory, 1993. Disponible sur le Web : http://www.isg.sfu.ca/life/.

[11] J. A. Goguen and J. Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming, Functions, Relations, and Equations*, pages 295–363. Prentice Hall, 1986.

[12] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *JCSS*, 25:239–266, 1982.

[13] J. A. Goguen and J. Meseguer. Order-sorted algebra 1 : Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Technical report SRI-CSL-89-10, SRI International, 1989.

[14] B. Courcelle. Arbres infinis et systèmes d'équations. *RAIRO Theoretical Informatics*, 13(1):31–48, 1979.

[15] A. Poigné. Identity and existence, and types in algebra - a survey of sorts. pages 53–78. Springer Verlag LNCS 785, 1992.

[16] R. Paije and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computation*, 16(7):973–989, 1987.

[17] J. A. Goguen, J. W. Thatcher end E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1):68–95, 1977.

[18] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, Berlin, 1987. first edition : 1984.

[19] R. Echahed. Sur l'intégration des langages algébriques et logiques. Thèse de l'Institut National Polytechnique de Grenoble, 1990.

[20] J.-M. Hullot. Compilation de formes canoniques dans les théorie équationnelles. Thèse de l'Université de Paris-Sud Centre d'Orsay, 1980.

[21] R. Kowalski and D. Kuehner. Linear resolution with selection function. In *Artificial Intelligence*, pages 227–260, 1971.

[22] A. Middeldörp and E. Hamoen. Counterexamples to completeness results for basic narrowing (extended abstract). In *Proceedings of the Third International Conference on Algebraic and Logic Programming*, pages 244–258, Volterra, Italy, September 1992. Disponible sur le Web : http://SunSITE.Informatik.RWTH-Aachen.de/dblp/db/indices/a-tree/m/Middeldorp:Aart.html.

[23] C. Hintermeier. Déduction avec sortes ordonnées et égalités. Thèse de l'Université Henri Poincaré - Nancy I, 1995.