Combining second order matching and first order E-matching

Régis Curien

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

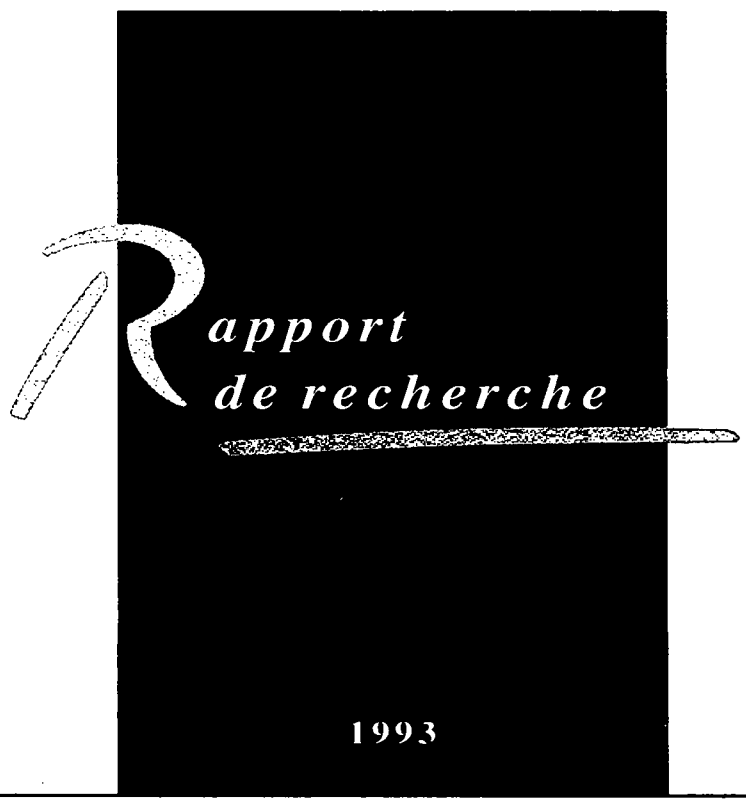
Combining Second Order Matching and First Order E-Matching

Régis CURIEN

N° 2012
Août 1993

PROGRAMME 2

Calcul symbolique,
programmation
et génie logiciel



Combining Second Order Matching and First Order E-Matching

Combinaison du Filtrage du Second Ordre et du E-Filtrage du Premier Ordre

Régis Curien*

Abstract : We propose an algorithm for combining second order matching and first order matching in an algebraic first order theory E. This algorithm has the flavor of the higher order E-unification algorithm of Nipkow and Qian [NQ91], but relies on the classical second order matching algorithm of Huet and Lang [HL78] instead of higher order unification. Since matching is simpler than unification, we are able to prove the termination of our algorithm when the algebraic theory E respects some conditions. We show that it is possible to preserve the termination when we relax some of these conditions by adapting the previous algorithm. It allows us to use AC1, ACI and ACI1 for example. These algebraic theories are the more useful for our purpose (recognizing logical or functional schemata).We have implemented our algorithm for the AC and AC1 theories, and we show examples of possible applications.

Keywords : matching. second-order. equational theories, AC-matching.

Résumé : Nous proposons un algorithme de combinaison du filtrage d'ordre deux et du filtrage modulo une théorie algèbrique du premier ordre. Si cet algorithme rappelle la E-unification d'ordre supérieur de Nipkow et Qian [NQ91], il repose en fait sur le filtrage du second ordre de Huet et Lang [HL78]. L'avantage du filtrage sur l'unification se manifeste ici par le fait que si la théorie E respecte certaines conditions, nous pouvons prouver la terminaison de cet algorithme. Certaines de ces conditions peuvent d'ailleurs être levées en adaptant l'algorithme de base. Il est alors possible de traiter les cas AC, AC1 ou ACI par exemple. Ces théories étant parmi celles qui nous intéressent pour reconnaître des motifs logiques ou fonctionnels dans des formules. Cet algorithme a été implanté pour les théories AC et AC1, et des exemples d'applications sont présentés.

Mots clés : filtrage, second ordre, théories équationnelles, filtrage AC.

*CRIN - INRIA Lorraine, BP239, 54506 Vandoeuvre-les-Nancy Cedex, France. E-mail: curien@loria.fr
Partially supported by the Esprit basic research working group 6028, CCL.

Combining Second Order Matching and First Order E-Matching

Régis Curien
CRIN and INRIA-Lorraine
BP239, 54506 Vandoeuvre-les-Nancy Cedex, France

Abstract

We propose an algorithm for combining second order matching and first order matching in an algebraic first order theory E. This algorithm has the flavor of the higher order E-unification algorithm of Nipkow and Qian [NQ91], but relies on the classical second order matching algorithm of Huet and Lang [HL78] instead of higher order unification. Since matching is simpler than unification, we are able to prove the termination of our algorithm when the algebraic theory E respects some conditions. We show that it is possible to preserve the termination when we relax some of these conditions by adapting the previous algorithm. It allows us to use AC1, ACI and ACI1 for example. These algebraic theories are the more useful for our purpose (recognizing logical or functional schemata).We have implemented our algorithm for the AC and AC1 theories, and we show examples of possible applications.

Keywords : matching, second-order, equational theories, AC-matching.

1 Introduction

Pattern matching is one of the most basic tools in A.I. and computer science and is used in many applications. The expressiveness of pattern matching has been widely enhanced by performing this process modulo a set of first order axioms (called an equational theory). The most common ones are Associativity Commutativity axioms (for short AC). $AC = \{f(x, f(y, z)) \sim f(f(x, y), z); f(x, y) \sim f(y, x)\}$ (for instance $\wedge$ and $\vee$ connectives are AC). Another useful pattern matching process deals with higher order pattern matching and is required when we are dealing with schemes instead of first order terms. There has been a growing interest in higher order topics since researchers have realized that they can be handled much more easily than it seemed at first glance.

Huet was the pioneer of higher order unification, pre-unification [Hue76] and matching [HL78]. We have used this matching algorithm as a base. Snyder and Gallier have revisited Huet's work in [SG90]. Nipkow and Qian have combined regular theories with this higher order unification [NQ91] and later on, Qian and Wang [QW92] generalized to arbitrary theories.
In many applications, second order matching is all that we need. In this paper, we consider second order pattern matching modulo a first order equational theory as a tool needed in a higher order toolbox which can be used in applications like higher order theorem proving, first order theorem proving or higher order rewriting.

Indeed, formula simplification is a useful process to carry out, before performing the automated deduction, in order to simplify the search for a proof. Therefore, the main difference between our second order E-matching and current work on E-unification is that we must obtain a termination property. We shall provide examples which show how second order E-matching can be used to find recurrence patterns in a logic program and make simplifications in a first order logic formula in order to prove this formula.

Section 2 contains general definitions used in this paper. We first give in section 3 a basic algorithm wich combines second order matching and first order E-matching for size and root preserving theories (see below for formal definitions). The required properties of this algorithm (soundness, completeness and termination) are proved in the section 4. We then show in section 5 how the root-preserving condition can be relaxed. Furthermore, collapsing axioms can be added, as described in section 6. We give runing examples and discuss about the implementation in section 7 and conclude in section 8.

2 Preliminaries

2.1 Terms

The reader is assumed to be familiar with typed λ-calculus [HS86] and first order matching. Then, we shall give the definitions and notions that we actually need.

Types : the set of the types T is inductively defined by :

- $\tau \in T_0 \Rightarrow \tau \in T$ where T_0 is the set of base types,
- $\alpha, \beta \in T \Rightarrow \alpha \rightarrow \beta \in T$.

Example 1

$$T_0 = \{Bool, Int\} \Rightarrow (Int \rightarrow Int) \rightarrow Bool \in T$$

Note that $\rightarrow$ associates to the right and in the sequel, we shall write $\alpha_1 \times \alpha_2 \times \cdots \times \alpha_n \rightarrow \beta$ for $(\alpha_1 \rightarrow \cdots (\alpha_n \rightarrow \beta) \cdots)$ or $\overline{\alpha_n} \rightarrow \beta$ which is a more convenient notation.

Order of a type : the order O of a type τ is defined by :

- $O(\tau) = 1$ if $\tau \in T_0$
- $O(\overline{\alpha_n} \rightarrow \beta) = max\{O(\alpha_i) \mid 1 \leq i \leq n\} + 1$.

Example 2 *The order of the previous type* $(Int \rightarrow Int) \rightarrow Bool$ *is three.*

Signature :

- for each type $\tau \in T$, there is a denumerable set $\mathcal{V}_\tau$ of variable symbols of type τ. The set of all variable symbols is $\mathcal{V}$,
- for each type $\tau \in T$, there is a denumerable set $\mathcal{C}_\tau$ of constant symbols of type τ. The set of all constant symbols is $\mathcal{C}$

Convention : In the following, we shall use (unless stated otherwise) α, β, *Bool* and *Int* for base types and τ for any type: $a, b, \ldots$ for constants of base type (i.e. of order one); $f, g, \ldots$ for constants of higher order; F, G, H and P for variables of second order; x, y, z for variables of any order and ϕ for any function symbol.

Atoms : the set of atoms is $\mathcal{A} = \mathcal{V} \cup \mathcal{C}$

Terms : the set $\mathcal{T}$ of terms is defined by $\mathcal{T} = \bigcup_{\alpha \in T} \mathcal{T}_\alpha$ with

Application : $(\phi\ t) \in \mathcal{T}_\beta$ if $\phi \in \mathcal{T}_{\alpha \to \beta}$ and $t \in \mathcal{T}_\alpha$

Abstraction : $\lambda x.t \in \mathcal{T}_{\tau_1 \to \beta}$ if $t \in \mathcal{T}_\beta$ and $x \in \mathcal{V}_{\tau_1}$

Remark: The order of a term is the order of its type. We shall note $(\cdots(\phi\ t_1)\cdots t_n)$ as $\phi(t_1, t_2, \ldots, t_n)$ or $\phi(\overline{t_n})$. t and s will denote terms. A language of order n contains constants of order at most $n+1$, and variables of order at most n. We study here a language of order two.

Example 3 *Let $x, y \in \mathcal{V}$ with $\tau(x) = \tau(y) = Bool$, $And \in \mathcal{C}$ with $\tau(And) = Bool \times Bool \to Bool$. Then, the logical function And will be written as the term $\lambda y.\lambda x.((And\ x)\ y)$ or, in a more practical notation $\lambda xy.And(x, y)$ or $\lambda xy.x \wedge y$.*

As usually defined, in the term $\lambda\overline{x_n} \cdot t$, $\overline{x_n}$ is called the *binder*, t is the *matrix* and the occurrences of the variable x_i in t are called *bound* ones. $BV(t)$ is the set of all the variables which are bound in t and $FV(t)$ is the set of all the variables which occur free in t. For each term $t = \lambda\overline{x_m} \cdot \phi(\overline{t_n})$, ϕ is the *head* of the term. We denote it by $\mathcal{H}(t) = \phi$. The term t is said *rigid* if its head is a constant or a bound variable and *flexible* otherwise. In the sequel, $[x \leftarrow t_2]t_1$ will be the result of replacing each free occurrence of x in t_1 by t_2. We consider terms modulo α-conversion, i.e. two terms which are equivalent modulo bound variables renaming are considered as syntactically equivalent.

λcalculus rules :

β-reduction : $((\lambda x.s)t) \to_\beta [x \leftarrow t]s$

η-reduction : if $x \notin FV(t)$ then $(\lambda x.(t\ x)) \to_\eta t$.

$\to_\beta$ denotes a single application of the β-reduction. Analogously for $\to_\eta$.
$\to_{\beta\eta}$ is defined as $\to_\beta \cup \to_\eta$. The reflexive, symmetric and transitive closures of $\to_\beta$, $\to_\eta$ and $\to_{\beta\eta}$ are respectively denoted by $=_\beta$, $=_\eta$ and $=_{\beta\eta}$.

Normal form : when no β-reduction can apply to a term t, t is said to be in β-normal form.

A **long-$\beta\eta$-normal form** is a β-normal form $\lambda\overline{x_m} \cdot \phi(\overline{t_n})$ where $\tau(\phi(\overline{t_n})) \in T_0$ and each t_i is in long-$\beta\eta$-normal form ($l\beta\eta$-nf). $l\beta\eta$-nf is unique.

Example 4 *The term $t = And$ is in β-nf but not in $l\beta\eta$-nf. We have to apply the η-expansion and t becomes $\lambda xy.And(x, y)$ which is its $l\beta\eta$-nf.*

Substitutions : a substitution (represented by σ or θ) is a function from $\mathcal{V}$ to $\mathcal{T}$ such that $\tau(\sigma(x)) = \tau(x)\ \forall x \in \mathcal{V}$.

Domain : $Dom(\sigma)$ is the set $\{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. We shall always assume that $BV(t) \cap Dom(\sigma) = \emptyset$. Hence, we can say that $\sigma(\lambda\overline{x_n}.t_1) = \lambda\overline{x_n}.\sigma(t_1)$.

The **range** of a substitution is $Ran(\sigma) = \bigcup_{x \in Dom(\sigma)} FV(\sigma(x))$.

Composition : let σ and θ be two substitutions, their composition $(\theta \circ \sigma)x = \theta(\sigma(x))$ is defined by $(\theta \circ \sigma) = \{< x, \theta t > | < x, t > \in \sigma\} \cup \{< x, t > \in \theta \mid x \notin Dom(\sigma)\}$.

Union : if $Dom(\sigma) \cap Dom(\theta) = \emptyset$, then $\sigma \cup \theta = \{< x, t > \in \sigma\} \cup \{< x, t > \in \theta\}$ and is not defined otherwise.

Normalization : σ is **normalized** iff $\forall x \in Dom(\sigma)$, $\sigma(x)$ is in $I\beta\eta$-nf.

Order on substitutions : let W be a set of variables. If $\forall x \in W \sigma(x) = \theta(x)$, σ and θ are equal over W and we write $\sigma = \theta[W]$. *β-equality* is defined by $\sigma =_\beta \theta[W]$ iff $\forall x \in W \sigma(x) =_\beta \theta(x)$. So, σ is more general than θ over W (denoted by $\sigma \leq_\beta \theta[W]$ iff there exists a substitution γ such that $\theta =_\beta \sigma \circ \gamma[W]$.

Idempotent substitution : σ is said *idempotent* when $\sigma \circ \sigma = \sigma$. Snyder [Sny88] has shown that for any substitution σ and a set of variables W containing $Dom(\sigma)$, there exists σ' idempotent such that $Dom(\sigma) = Dom(\sigma')$, $\sigma' \leq_{\beta\eta} \sigma[W]$. We then shall use now only idempotent substitutions without loss of generality.

2.2 Theory

We now define algebraic terms [NQ91].

Algebraic terms : let A_0 be $\bigcup_{\alpha \in T_0} \mathcal{A}_\alpha$. The set of the algebraic terms $\mathcal{T}^E$ is the smallest one such that :

- $A_0 \in \mathcal{T}^E$
- if $f \in \mathcal{C}_{\overline{\alpha_n} \to \beta}$ with $\alpha_i \in T_0$ $1 \leq i \leq n$ and $s_i \in \mathcal{T}^E \cap \mathcal{T}_{\alpha_i}$ $1 \leq i \leq n$ then $f(\overline{s_n}) \in \mathcal{T}^E$.

Equation : an equation is an unordered pair $t \sim s$ with $\tau(t) = \tau(s)$ and t and s are algebraic terms.

Theory : A set E of equations is an algebraic *theory*. For a given theory E, $\mathcal{T}^E$ denotes its terms and $\mathcal{C}^E$ the constants appearing in it.

Equivalence : the *equivalence modulo* E denoted by $=_E$ is the smallest equivalence relation on the terms such that $t[\sigma(l)] =_E t[\sigma(r)]$ $\forall t, \sigma$ and $l \sim r \in E$. So, we define the $\beta\eta E$-equivalence ($=_{\beta\eta E}$) by the reflexive, transitive and symmetric closure of $=_E \cup \to_{\beta\eta}$. A useful result [BT88] is that for any pair of terms $< t_1, t_2 >$, $t_1 =_{\beta\eta E} t_2$ iff $t_1 \downarrow_{I\beta\eta} =_E t_2 \downarrow_{I\beta\eta}$ ($t \downarrow_{I\beta\eta}$ is the $\beta\eta$-nf of t).

Example 5 *We define a theory which illustrates the associativity-commutativity of the And function.*

$$E = \{And(And(x,y),z) \sim And(x,And(y,z));$$
$$And(x,y) \sim And(y,x)\}$$

Size of a term :

we now define the size of terms [HL78] :

$$\text{the size } |\ t\ | \text{ is } \begin{cases} |\ x\ | = 1 \\ |\ \lambda\overline{x_m} \cdot \phi(\overline{t_n})\ | = 1 + \sum_{i=1}^{n} |\ t_i\ | \end{cases}.$$

Size preserving theories : we call so, theories which have the following properties :

- $\{\{\mathcal{V}(l)\}\} = \{\{\mathcal{V}(r)\}\}$ for each equation $l \sim r \in E$ (where $\{\{\mathcal{V}(l)\}\}$ is the multiset of the variables in l),
- $|\ l\ | = |\ r\ |$ for each equation $l \sim r \in E$.

Lemma 1 *If a theory has the previous properties, then this is regular and if t_1 and t_2 are two terms with $t_1 =_E t_2$ then $|\ t_1\ | = |\ t_2\ |$.*

Proof: The regularity is trivial because $\{\{\mathcal{V}(l)\}\} = \{\{\mathcal{V}(r)\}\}$ implies $\{\mathcal{V}(l)\} = \{\mathcal{V}(r)\}$ which is the definition of regularity.
$=_E$ is the smallest equivalence relation on the terms such that $t[\sigma(l)] =_E t[\sigma(r)]\ \forall t, \sigma$ and $l \sim r \in E$. The number of constant symbols is the same for l and r for each equation of the theory, (because the total number of symbols is constant and the also number of variables). The only way to make a term grow is to apply a substitution σ. But σ is applied to the both sides and if $x \in Dom(\sigma)$, then $x \in \mathcal{V}(t_1) \iff x \in \mathcal{V}(t_2)$ and it appears the same number of times in both sides. That means that we preserve the property $|\ \sigma(t_1)\ | = |\ \sigma(t_2)\ |$.

□

Root-preserving theories : a theory will be called root-preserving if each axiom of this theory is such that $\mathcal{H}(l) = \mathcal{H}(r)$ for each $l \sim r \in E$. AC is such a theory for example.

3 Higher-Order E-Matching

Higher Order E-Matching will be studied here with terms and theory as defined in the previous section.

3.1 Definition of E-matching

Matching pair : $t_1 \stackrel{?}{=} t_2$ is a matching pair if t_1 and t_2 are two terms in $\lambda\beta\eta$-nf and t_2 is rigid, and $\tau(t_1) = \tau(t_2)$. We assume that t_2 doesn't contain any free variable. If it is the case, they can be frozen during the matching. The theory we shall use cannot introduce free variables in this term.

E-matcher : σ is an E-matcher of a set of matching pairs iff for each pair of this set, we have $\sigma t_1 =_{\beta\eta E} t_2$. $\mathcal{M}_E$ denotes the set of all such E-matchers.

Solved form : $x \stackrel{?}{=} t$ is said to be solved in a set $\mathcal{S}$ of matching pairs if x is a variable and $x \notin FV(\mathcal{S} - \{x \stackrel{?}{=} t\})$. $\mathcal{S}$ is solved if all its matching pairs are solved or if $\mathcal{S}$ is empty. Note that if x is of order two, x is not in λ-$\beta\eta$-nf in $x \stackrel{?}{=} t$. In fact, a solved form $x \stackrel{?}{=} t$ means $x \leftarrow t$.

A complete set of E-matchers of a set of matching pairs $\mathcal{S}$ denoted by $CSM_E(\mathcal{S})$ is a set of substitutions requiring the following properties :

Soundness : $\sigma \in CSM_E(\mathcal{S})$ implies $\sigma \in M_E(\mathcal{S})$
(i.e. $CSM_E(\mathcal{S}) \subseteq M_E(\mathcal{S})$)

Completeness : $\forall \sigma \in M_E(\mathcal{S})$, $\exists \gamma \in CSM_E(\mathcal{S})$ such that $\gamma \leq_{\beta\eta E} \sigma[FV(\mathcal{S})]$ (i.e. for each solution in M_E there exists a smaller one in $CSM_E(\mathcal{S})$).

Protectiveness : $\forall \sigma \in CSM_E(\mathcal{S})$ such that $Dom(\sigma) \subseteq FV(\mathcal{S}), Ran(\sigma) \cap (Dom(\sigma) \cup W) = \emptyset$, we have σ normalized. This property is assumed without loss of generality [NQ91] because each substitution σ has a normalized equivalent θ away from $Dom(\sigma) \cup W$.

3.2 The algorithm : rules and strategy

We shall describe the algorithm in three parts. In the first one, we shall see what *abstraction* is and the rule which use the matching in E. The second one contains the rest of the rules and the third one, the strategy.

3.2.1 Abstraction and matching in E

Let $t_1 \stackrel{?}{=} t_2$ be a matching pair. Assume $\mathcal{H}(t_1)$ and $\mathcal{H}(t_2)$ are constants of the theory E. We then have to do first order matching in E. But before, we have to abstract t_1 to make it pure in E.

Example 6 *Let us consider the matching pair :* $\lambda xy.And(P(And(x,y)),y) \stackrel{?}{=} \lambda xy.And(y,f(a))$. *To find a solution, we have to consider the fact that And is commutative. To use the matching algorithm of the theory,* t_1 *must be pure in it. So, abstraction will transform the pair into the system :*

$$\{\lambda xy.And(X_1,X_2) \stackrel{?}{=} \lambda xy.And(y,f(a)); X_1 \stackrel{?}{=} P(And(x,y)); X_2 \stackrel{?}{=} y\}$$

and we shall use AC-matching for the first pair.

We shall describe it more formally. First, we introduce the notion of *maximal alien subterm* [NQ91] :

MAS : let $\lambda\overline{x_m} \cdot \phi(\overline{t_n})$ be a term with ϕ a constant of the theory. We have :

$$MAS(\lambda\overline{x_m} \cdot \phi(\overline{t_n})) = \begin{cases} \{t_i\} \cup FV(\phi(\overline{t_n})) & \text{if } \mathcal{H}(t_i) \notin \mathcal{C}^E \\ MAS(t_i) & \text{otherwise} \end{cases}$$

For example, $MAS(\lambda xy.And(P(And(x,y)),y))$ is $\{P(And(x,y)); y\}$

Then, the abstraction of the first term of the matching pair can be written as a rule by : (ϕ and ϕ' are constants of the theory)

$$\frac{\lambda\overline{x_m} \cdot \phi(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x'_m} \cdot \phi'(\overline{s'_{n'}})}{\lambda\overline{x_m}.[t_i \leftarrow X_i]\phi(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x'_m} \cdot \phi'(\overline{s'_{n'}})}\textbf{Abs}$$

where

- $\{t_i\} = MAS(\lambda\overline{x_m} \cdot \phi(\overline{s_n}))$,
- X_i are new distinct variables of the appropriate type.

Note that the pairs $X_i \stackrel{?}{=} t_i$ are in solved form. As seen in the previous example, we want to use the matching algorithm on the first pair obtained by abstraction. The first term is built with constants of E and new variables X_i, therefore, the result of E-matching are pairs $X_i \leftarrow t'_i$. Hence, we shall have to make the correspondence between these terms t'_i and the alien subterms t_i. By compacting all this work (abstraction and forming the pairs), we obtain the following rule :

$$\frac{\lambda\overline{x_m} \cdot \phi(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x'_m} \cdot \phi'(\overline{s'_{n'}})}{\lambda\overline{x_m}.t_i \stackrel{?}{=} \lambda\overline{x'_m}.t'_i}\textbf{E-matching}$$

where

- $\{t_i\} = MAS(\lambda\overline{x_m} \cdot \phi(\overline{s_n}))$,
- X_i are new distinct variables of the appropriate type,
- t'_i are such that $X_i \stackrel{?}{=} t'_i$ are solution of $\{\lambda\overline{x_m}.[t_i \leftarrow X_i]\overline{s_n} \stackrel{?}{=} \lambda\overline{x'_m} \cdot \phi'(\overline{s'_{n'}})\}$ in E.

Let us apply it to our example.

Example 7 *We shall detail the two steps of the rule :*

$$\frac{\lambda xy.And(P(And(x,y)),y) \stackrel{?}{=} \lambda xy.And(y,f(a))}{\lambda xy.And(X_1,X_2) \stackrel{?}{=} \lambda xy.And(y,f(a))}\textbf{Abs}$$

Then, the AC-matching of the first pair will give us for first solution $\sigma_1 = \{X_1 \leftarrow f(a); X_2 \leftarrow y\}$ *(*$f(a)$ *and* y *are the* t'_i *defined in the E-matching rule). Hence, we get :*

$$\frac{\lambda xy.And(P(And(x,y)),y) \stackrel{?}{=} \lambda xy.And(y,f(a))}{\lambda xy.P(And(x,y)) \stackrel{?}{=} \lambda xy.f(a);\ \lambda xy.y \stackrel{?}{=} \lambda xy.y}\textbf{E-matching}$$

The first order E-matching algorithm need to deal with free constants in order to consider alien subterms in the rigid term as free constants.

3.2.2 The other rules

We shall introduce the rest of the rules by showing with an example in which case they are needed.

Example 8

$$\lambda Fx.F(a,x) \stackrel{?}{=} \lambda Gy.G(a,y)$$

Since we are working modulo α*-equivalence, and since* F *and* G *are variables of the same type, to match this pair is to match the subterms. i.e.* $\{\lambda Fx.a \stackrel{?}{=} \lambda Gy.a;\ \lambda Fx.x \stackrel{?}{=} \lambda Gy.y\}$.

This is the purpose of the following rule :

$$\frac{\lambda B_1.F(\overline{t_m}) \stackrel{?}{=} \lambda B_2.G(\overline{t'_m})}{\lambda B_1.t_1 \stackrel{?}{=} \lambda B_2.t'_1;\ \ldots;\ \lambda B_1.t_m \stackrel{?}{=} \lambda B_2.t'_m}\textbf{Decomp1}$$

where

- $B_1 = x_1 \ldots x_i\, F\, x_{i+1} \ldots x_n$.
- $B_2 = x'_1 \ldots x'_i\, G\, x'_{i+1} \ldots x'_n$.

Example 9 *The simplest case is when $\mathcal{H}(t_1) = \mathcal{H}(t_1)$ and are constants. For example :*

$$\lambda xy.And(x,y) \stackrel{?}{=} \lambda x'y'.And(a,b)$$

We then have, as in the previous case, to examine the subterms $\{\lambda xy.x \stackrel{?}{=} \lambda x'y'.a;\ \lambda xy.y \stackrel{?}{=} \lambda x'y'.b\}$

So, we write this second decomposition :

$$\frac{\lambda\overline{x_m} \cdot f(\overline{t_n}) \stackrel{?}{=} \lambda\overline{x'_m} \cdot f(\overline{t'_n})}{\lambda\overline{x_m}.t_1 \stackrel{?}{=} \lambda\overline{x'_m}.t'_1;\ \cdots;\ \lambda\overline{x_m}.t_n \stackrel{?}{=} \lambda\overline{x'_m}.t'_n}\textbf{Decomp2}$$

In the sequel *Decomp* means either *Decomp*1 or *Decomp*2.

Example 10 *If we have the matching pair :*

$$\lambda xy.F(And(x,y),a) \stackrel{?}{=} \lambda x'y'.And(x',y')$$

we need to transform F into a projection, i.e. to form a new pair $F \stackrel{?}{=} \lambda x_1 x_2.x1$ (which is a solved one). So, we shall obtain the system $\{\lambda xy.And(x,y) \stackrel{?}{=} \lambda x'y'.And(x',y');\ F \stackrel{?}{=} \lambda x_1 x_2.x1\}$.

This is achieved by the projection rule :

$$\frac{\lambda\overline{x_m} \cdot F(\overline{t_p}) \stackrel{?}{=} \lambda\overline{x'_m}.s}{\lambda\overline{x_m}.t_i \stackrel{?}{=} \lambda\overline{x'_m}.s;\ F \stackrel{?}{=} \lambda\overline{x_p}.x_i}\textbf{Projection}$$

Note that the head of s is either a variable or a constant.

Example 11 *Assume we have to match the following pair :*

$$\lambda xy.G(x,y) \stackrel{?}{=} \lambda x'y'.f(x',y',x')$$

We shall try then to imitate the function f with the variable G. For our example, we imitate f by : $G \stackrel{?}{=} \lambda x_1 x_2.f(H_1(x_1,x_2),H_2(x_1,x_2),H_3(x_1,x_2))$, where H_i are new function variables of the appropriate type. Applying the substitution $\{[G \leftarrow \lambda x_1 x_2.f(H_1(x_1,x_2),H_2(x_1,x_2),H_3(x_1,x_2))]\}$ to the first term and after β-reduction, we obtain the pair $\lambda xy.f(H_1(x,y),H_2(x,y),H_3(x,y)) \stackrel{?}{=} \lambda x'y'.f(x',y',x')$.

So, we give the imitation rule :

$$\frac{\lambda\overline{x_n} \cdot F(\overline{t_m}) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})}{[F \leftarrow \lambda\overline{x_m}.f(\overline{u_{m'}})]t_F \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}});\ F \stackrel{?}{=} \lambda\overline{x_m}.f(\overline{u_{m'}})}\textbf{Imitation}$$

where

- t_F is $\lambda\overline{x_n} \cdot F(\overline{t_m})$, and $\overline{x_m}$ are new variables of the appropriate type
- with the following definition of u_i (due to Huet and Lang [HL78]) :
  if $\tau(t'_i) \in T_0$ then $u_i = H_i(\overline{x_m})$
  if $\tau(t'_i) = (\alpha_1 \times \cdots \times \alpha_n \rightarrow \beta)$ then $u_i = \lambda w_1, w_2, \ldots, w_s \cdot H_i(\overline{x_m}, \overline{w_s})$
  and $\tau(w_j) = \alpha_j\ 1 \leq j \leq s$
  Where w_i and H_i are new variables of the appropriate type.

Remarks:

Decomposition denotes either Decomp1 or Decomp2. We are working with second order language combined with first order theory. Therefore, in the imitation rule, if f is in E, then the order of f is at most two (because E is of order 1). The consequence is that the case $\tau(t'_i) = (\alpha_1 \times \alpha_2 \times \cdots \times \alpha_n \rightarrow \beta)$ only appears for an imitation of a symbol f out of the theory.

The set of rules $\{E\text{–}matching; Decomp; Projection; Imitation\}$ will be called ρ. ρ_i will be any of them.

3.2.3 The strategy

We have to define now how to use these rules. First, let us define what success and failure cases are.

Success case : $\mathcal{S}$ is a success case if it's empty or solved. We can then build a substitution $\sigma_{\mathcal{S}}$ with all the pairs in solved form of $\mathcal{S}$, and we have $\sigma_{\mathcal{S}} t_1 =_{\beta\eta E} t_2$ for all $t_1 \stackrel{?}{=} t_2$ in the initial set $\mathcal{S}_0$.

Failure case : $\mathcal{S}$ is a failure case if it contains a pair such that :
$\lambda\overline{x_m} \cdot \phi(\overline{t_n}) \stackrel{?}{=} \lambda\overline{x'_m} \cdot \phi'(\overline{t'_{n'}})$ with ϕ and ϕ' two different constants not in E.

To start the E-matching, we need a set $\mathcal{S}_0$ of matching pairs such that all the terms in $\mathcal{S}_0$ are in $l\beta\eta$-nf, and such that $\tau(t_1) = \tau(t_2)$ for each pair $t_1 \stackrel{?}{=} t_2$ in $\mathcal{S}_0$.

The strategy : if $\mathcal{S}$ is neither in solved form nor empty nor in a failure case, we select arbitrarily a pair.

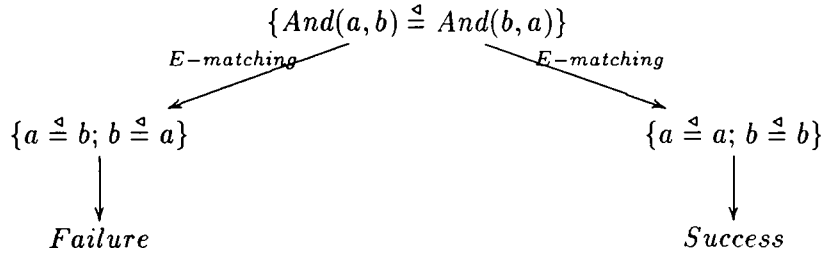
- **a) If we have a pair rigid-rigid :**
  - if $\mathcal{H}(t_1) \in E$ and $\mathcal{H}(t_2) \in E$, then E-matching will be applied (cf. following example),
  - if $\mathcal{H}(t_1) \notin E$ and $\mathcal{H}(t_2) \notin E$ and $\mathcal{H}(t_1) = \mathcal{H}(t_2)$, then use Decomp,
  - if $\mathcal{H}(t_1)$ and $\mathcal{H}(t_2)$ are bound variables, use Decomp.
  - it's a failure otherwise.

- **b) If we have a pair flexible-rigid :**
  - if $O(\mathcal{H}(t_1)) = 1$, then either the pair is in solved form, and we do not select it, or it's a failure case,
  - otherwise $O(\mathcal{H}(t_1)) = 2$ and then
    if $\mathcal{H}(t_2) \in \mathcal{V}$, we apply the Projection rule as many times (at most the number of arguments of $\mathcal{H}(t_1)$) as the type constraints permit it, and each time, we shall try to find a different solution.
    i.e. when we have $t_1 = \lambda\overline{x_m} \cdot F(\overline{t_p})$, we shall try a new matching with each $\lambda\overline{x_m}.t_i$ such that $\tau(t_i) = \tau(F(\overline{t_p}))$.
    if $\mathcal{H}(t_2) \in \mathcal{C}$, we shall then apply the projections as in the previous case, and we shall add another solution which will be given by the application of the imitation.

Remark:

The case of a pair *flexible – flexible* doesn't appear here because in a matching pair, t_2 is rigid. i.e. its head cannot be a free variable.

Example 12 *This illustrates the case in which the E-matching generates more than one solution.*



Example 13 *:*

Imitation of a symbol of the theory :

$$\lambda xy.F(x,y) \stackrel{?}{=} \lambda xy.And(p(x),y) \longrightarrow^{projections} \ldots$$

$$\downarrow imitation$$

$$\lambda xy.And(H_1(x,y),H_2(x,y)) \stackrel{?}{=} \lambda xy.And(p(x),y); F \stackrel{?}{=} \lambda x_1x_2.And(H_1(x_1,x_2),H_2(x_1,x_2))$$

$$\downarrow E-matching$$

$$\{\lambda xy.(H_1(x,y)) \stackrel{?}{=} \lambda xy.p(x); \lambda xy.(H_2(x,y)) \stackrel{?}{=} \lambda xy.y; F \stackrel{?}{=} \lambda x_1x_2.And(H_1(x_1,x_2),H_2(x_1,x_2))\}$$

$$\{\lambda xy.(H_1(x,y)) \stackrel{?}{=} \lambda xy.y; \lambda xy.(H_2(x,y)) \stackrel{?}{=} \lambda xy.p(x); F \stackrel{?}{=} \lambda x_1x_2.And(H_1(x_1,x_2),H_2(x_1,x_2))\}$$

The E-matching rule gives two matching problems.

Imitation of a symbol out of the theory :

$$\lambda xy.F(x,y) \stackrel{?}{=} \lambda xy.f(p(x),y) \longrightarrow^{projections} \ldots$$
$$\Big\downarrow imitation$$
$$\lambda xy.f(H_1(x,y),H_2(x,y)) \stackrel{?}{=} \lambda xy.f(p(x),y); F \stackrel{?}{=} \lambda x_1x_2.f(H_1(x_1,x_2),H_2(x_1,x_2))$$
$$\Big\downarrow Decomp$$
$$\lambda xy.(H_1(x,y)) \stackrel{?}{=} \lambda xy.p(x); \lambda xy.(H_2(x,y)) \stackrel{?}{=} \lambda xy.y; F \stackrel{?}{=} \lambda x_1x_2.f(H_1(x_1,x_2),H_2(x_1,x_2))$$

4 Termination, soundness and completeness

We shall prove the essential properties of this algorithm for size and root preserving theories. The next section improves the algorithm for dealing with all size preserving theories and then, we show how to allow collapsing axioms in the following section.

4.1 Soundness

Lemma 2 *Let $\mathcal{S}$ and $\mathcal{S}'$ be sets of matching pairs, $\frac{\mathcal{S}}{\mathcal{S}'}\rho_i$ implies $CSM_E(\mathcal{S}') \subseteq CSM_E(\mathcal{S})$.*

Proof: E-matching : Let $[t_1 \leftarrow t_2]$ mean that we replace all the occurrences of t_1 by t_2, then we summarize the result of *E−matching* by :

$$[t_i \leftarrow t'_i]\lambda\overline{x_m} \cdot \phi(\overline{s_n}) =_E \lambda\overline{x'_m} \cdot \phi'(\overline{s'_{n'}}) \qquad (1)$$

because $[t_i \leftarrow X_i][X_i \leftarrow t'_i] \equiv [t_i \leftarrow t'_i]$. Then, it is easy to see that if there is a σ such that $\sigma(\lambda\overline{x_m}.t_i) =_{\beta\eta E} \lambda\overline{x'_m}.t'_i$ for each i, from (1) we can conclude that $\sigma(\lambda\overline{x_m} \cdot \phi(\overline{s_n})) =_{\beta\eta E} \lambda\overline{x'_m} \cdot \phi'(\overline{s'_{n'}})$ (t'_i cannot contain variables).
Decomp : if we have a σ such that $\sigma(\lambda\overline{x_m}.t_i) =_{\beta\eta E} \lambda\overline{x'_m}.t'_i$, it is straightforward to say that $\sigma(\lambda\overline{x_m} \cdot \phi(\overline{t_n})) =_{\beta\eta E} \lambda\overline{x'_m} \cdot \phi'(\overline{t'_n})$ if ϕ and ϕ' are either the same constant or bound variables because we are working modulo α-conversion.
Projection : Let σ be such that $\sigma(\lambda\overline{x_m}.t_i) =_{\beta\eta E} \lambda\overline{x'_m}.s$. $[F \leftarrow \lambda\overline{x_p}.x_i]\lambda\overline{x_m}.F(\overline{t_p}) =_{\beta\eta} \lambda\overline{x_m}.t_i$ where $[F \leftarrow \lambda\overline{x_p}.x_i]$ is a possible projection. Then,
$(\sigma \cup [F \leftarrow \lambda\overline{x_p}.x_i])\lambda\overline{x_m}.F(\overline{t_p}) =_{\beta\eta E} \lambda\overline{x'_m}.s$ (with F and $x_1, \ldots, x_p$ of the appropriate type).
Imitation : We have σ such that

$$\sigma(\lambda\overline{x_n}.f(\lambda\overline{w_{k_1}}.H_1(\overline{t_m},\overline{w_{k_1}}),\ldots,\lambda\overline{w_{k_{m'}}}.H_{m'}(\overline{t_m},\overline{w_{k_{m'}}}))) =_{\beta\eta E} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$$

Then, if we note θ the imitation, $\theta = [F \leftarrow \lambda x^1 \ldots x^m.f(\lambda\overline{w_{k_1}}.H_1(x^1,\ldots,x^m,\overline{w_{k_1}}),\ldots$
$,\lambda\overline{w_{k_{m'}}}.H_{m'}(x^1,\ldots,x^m,\overline{w_{k_{m'}}}))]$,
$\theta(\lambda\overline{x_n} \cdot F(\overline{t_m})) =_{\beta\eta} \lambda\overline{x_n}.f(\lambda\overline{w_{k_1}}.H_1(\overline{t_m},\overline{w_{k_1}}),\ldots,\lambda\overline{w_{k_{m'}}}.H_{m'}(\overline{t_m},\overline{w_{k_{m'}}}))$ Hence, $(\sigma \cup \theta)(\lambda\overline{x_n} \cdot F(\overline{t_m})) =_{\beta\eta E} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$

□

4.2 Termination

We just need some tools and two lemmata :

- the *Alternation* is defined for a term by :

$$IAST(\lambda\overline{x_m} \cdot \phi(\overline{t_n})) = \begin{cases} IAST(t_i) \cup \{t_j\} \mid \mathcal{H}(t_i) \notin E \text{ and } \mathcal{H}(t_j) \in E \text{ if } \phi \notin E \\ IAST(t_i) \cup \{t_j\} \mid \mathcal{H}(t_i) \in E \text{ and } \mathcal{H}(t_j) \notin E \text{ if } \phi \in E \end{cases}$$

We denote the alternation of a term by $Alt(t)$ such that :

$$Alt(t) = 1 + max_{s \in IAST(t)} Alt(s)$$

For a set $\mathcal{S}$ of matching pairs, we define $Alt_1(\mathcal{S})$ by :

$$Alt_1(\mathcal{S}) = \{\{Alt(t_1)\} \mid t_1 \stackrel{?}{=} t_2 \in \mathcal{S} \text{ and not solved}\}$$

- $\{\{\}\}$ stands for multisets.
- multisets of integers are compared using the multiset extension of $<$.
- we now define the depth of a system of matching pairs [HL78] :
  let $t_1 \stackrel{?}{=} t_2$ be a matching pair and $\mathcal{S}$ a set of matching pairs,

$$\xi_1(\mathcal{S}) = \sum \mid t_1 \mid \text{ for all } t_1 \text{ such that } t_1 \stackrel{?}{=} t_2 \text{ not solved in } \mathcal{S}$$
$$\xi_2(\mathcal{S}) = \sum \mid t_2 \mid \text{ for all } t_2 \text{ such that } t_1 \stackrel{?}{=} t_2 \text{ not solved in } \mathcal{S}$$

- notion of subterm :

$$\text{subterms of } \lambda\overline{x_m} \cdot \phi(\overline{t_n}) \text{ are } \begin{cases} \lambda\overline{x_m} \cdot \phi(\overline{t_n}) \\ t_i \\ \text{subterms of } t_i \end{cases}$$

- t' is a strict subterm of t if t' is a subterm of t and $t' \neq t$.

So, to prove the termination of the algorithm, we shall prove for each rule that the complexity triple $CT(\mathcal{S}) =< Alt_1(\mathcal{S}), \xi_2(\mathcal{S}), \xi_1(\mathcal{S}) >$ is decreasing. We shall compare $CT(\mathcal{S})$ and $CT(\mathcal{S}')$ lexicographically. Before, this lemma :

Lemma 3 *if t' is a subterm of the term t, then we have :*

$$\mid t' \mid \leq \mid t \mid$$

$$Alt(t') \leq Alt(t)$$

if t' is a strict subterm of the term t, then we have :

$$\mid t' \mid < \mid t \mid$$

$$Alt(t') \leq Alt(t).$$

Proof: If $t = t'$, we trivially have $|\ t'\ | = |\ t\ |$. If t' is a strict subterm of t, then we have $|\ t'\ | < |\ t\ |$ because t' is at most one of the t_i in $t = \lambda\overline{x_m} \cdot \phi(\overline{t_n})$. So, by definition, $|\ t\ | = 1 + \sum_{i=1}^{n} |\ t_i\ |$ and trivially $|\ t\ | < |\ t_i\ |\ \forall i$.
If we take off the head of a term, we cannot increase its alternation. We can just decrease it, or keep it to the same value. Then we have $Alt(t') \leq Alt(t)$. □

We just need the following lemma that justifies the use of root and size preserving theories.

Lemma 4 *Let $f(x_1, x_2, \ldots, x_n) \stackrel{?}{=} f(t_1, t_2, \ldots, t_n)$ be a matching problem where $f \in E$ and x_i are all distinct variables, then, if E is a size and root preserving theory, each solution $\sigma_i = \cup_{k \in 1 \ldots n} x_k \leftarrow s_k$ will be such that $|\ s_k\ | < |\ f(t_1, t_2, \ldots, t_n)\ |$.*

Proof: By definition, $|\ \sigma_i f(x_1, x_2, \ldots, x_n)\ | = |\ f(t_1, t_2, \ldots, t_n)\ |$. Then, we have for each $x_k \leftarrow s_k \in \sigma_i$, at least $|\ s_k\ | \leq |\ f(x_1, x_2, \ldots, x_n)\ | - 1$. Therefore, $|\ s_k\ | < |\ f(t_1, t_2, \ldots, t_n)\ |$.
□

We can then formulate the termination lemma for such theories.

Lemma 5 *(Termination (for root and size preserving theories)) : for any derivation with the given set of rules, $\frac{\mathcal{S}_1}{\mathcal{S}_2}\rho_i$, either $\mathcal{S}_2$ is a terminal case, or $CT(\mathcal{S}_2) <_{lex} CT(\mathcal{S}_1)$.*

Proof: Note that the strategy cannot generate an infinite set of solutions because the E-matching is assumed to give a finite set of more general solutions. We have to examine all the rules.

E-matching : in this rule, t_i is a maximal alien subterm of $\lambda\overline{x_m} \cdot \phi(\overline{s_n})$. It means that ϕ is a constant of the theory, and the head of t_i is not in E. Therefore, the alternation has decreased of 1 between $\lambda\overline{x_m} \cdot \phi(\overline{s_n})$ and each t_i. We then have $Alt_1(\mathcal{S}_2) < Alt_1(\mathcal{S}_1)$ which implies $CT(\mathcal{S}_2) <_{lex} CT(\mathcal{S}_1)$. It is clear here that regularity is required to avoid free variables in terms t'_i.

Decomp : if $\mathcal{H}(t_1)$ and $\mathcal{H}(t_2)$ are constants, they are not in E, otherwise, we would apply the E-matching rule. If they are variables, they are bound ones (otherwise, it's a Flexible-Flexible case). For these two cases, the alternation cannot increase. $\xi_1(\mathcal{S})$ and $\xi_2(\mathcal{S})$ are decreasing because t_i and t'_i are strict subterms of respectively $F(\overline{t_m})$ or $f(\overline{t_n})$ and $G(\overline{t'_m})$ or $f(\overline{t'_n})$. Therefore $CT(\mathcal{S}_2) <_{lex} CT(\mathcal{S}_1)$.

Projection : Again, t_i is a strict subterm of $\lambda\overline{x_m} \cdot F(\overline{t_p})$, then Alt_1 cannot increase (by lemma 2). $F \stackrel{?}{=} \lambda\overline{x_p}.x_i$ is in solved form. $\lambda\overline{x_m}.s$ doesn't change. So, ξ_2 doesn't increase. And ξ_1 is decreasing because t_i is a strict subterm of $\lambda\overline{x_m} \cdot F(\overline{t_p})$. Then, $CT(\mathcal{S}_2) <_{lex} CT(\mathcal{S}_1)$.

Imitation : We have to distinguish two cases :

- 1- imitation of a symbol f out of the theory.

$$\lambda\overline{x_n} \cdot F(\overline{t_m}) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$$

Imitation : $F \leftarrow \lambda x_1 \ldots x_m.f(\lambda w_1 \ldots w_{k_1} \cdot H_1(x_1, \ldots, x_m, w_1, \ldots, w_{k_1}), \ldots,$
$\lambda w_1 \ldots w_{k_{m'}} \cdot H_{m'}(x_1, \ldots, x_m, w_1, \ldots, w_{k_{m'}}))$
After application of imitation and β-reduction, we have the pair :

$$\lambda\overline{x_n}.f(\lambda w_1 \ldots w_{k_1}.H_1(\overline{t_m}, \overline{w_{k_1}}), \ldots, \lambda w_1 \ldots w_{k_{m'}}.H_{m'}(\overline{t_m}, \overline{w_{k_{m'}}})) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$$

In this case, the only rule we can apply is the decomposition. The result is the following set of matching pairs.

$$\{(\lambda\overline{x_n w_{k_1}}.H_1(\overline{t_m}, \overline{w_{k_1}}) \stackrel{?}{=} \lambda\overline{x'_n}.t'_1), \ldots, (\lambda\overline{x_n w_{k_{m'}}}.H_{m'}(\overline{t_m}, \overline{w_{k_{m'}}})) \stackrel{?}{=} \lambda\overline{x'_n}.t'_{m'})\}$$

We then have $Alt_1(\mathcal{S}_1) = Alt(\lambda\overline{x_n} \cdot F(\overline{t_m}))$ with $\phi \notin E$ and $Alt_1(\mathcal{S}_2) = Alt(\lambda\overline{x_n w_{k_i}}.H_i(\overline{t_m}, \overline{w_{k_i}})) = Alt_1(\mathcal{S}_1)$ because $H_i \notin E$. But we have $\xi_2(\mathcal{S}_2) = \sum_i \mid t'_i \mid = \xi_2(\mathcal{S}_1) - 1$. therefore, $\xi_2(\mathcal{S}_2) < \xi_2(\mathcal{S}_1)$, and then $CT(\mathcal{S}_2) <_{lex} CT(\mathcal{S}_1)$.

- 2- imitation of a symbol f of the theory. After application of the imitation and β-reduction, we have the same pair :

$$\lambda\overline{x_n}.f(H_1(\overline{t_m}), \ldots, H_{m'}(\overline{t_m})) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$$

But $f \in E$ and $H_i \notin E$. Then, trivially $MAS(\lambda\overline{x_n}.f(H_1(\overline{t_m}), \ldots, H_{m'}(\overline{t_m}))) = \{H_1(\overline{t_m}); \ldots; H_{m'}(\overline{t_m})\}$. The E-matching rule is then applied on the pair $f(x_1, x_2, \ldots, x_{m'}) \stackrel{?}{=} f(\overline{t'_{m'}})$ where the x_i's are the abstractions of the $(H_i(\overline{t_m}))$'s. The k (assumed finite) more general solutions will be :

		$sol.1$					$sol.k$	
$H_1(\overline{t_m})$	x_1	$\leftarrow$	s_1^1	$\cdots$	$\cdots$	x_1	$\leftarrow$	s_1^k
$\vdots$		$\vdots$					$\vdots$	
$H_{m'}(\overline{t_m})$	$x_{m'}$	$\leftarrow$	$s_{m'}^1$	$\cdots$	$\cdots$	$x_{m'}$	$\leftarrow$	$s_{m'}^k$

We shall have then the k matching problems :

$\mathcal{S}_1 = \cup_{i \in 1 \ldots m'}(H_i(\overline{t_m}) \stackrel{?}{=} s_i^1)$

$\vdots$

$\mathcal{S}_k = \cup_{i \in 1 \ldots m'}(H_i(\overline{t_m}) \stackrel{?}{=} s_i^k)$

We then have $Alt(H_i(\overline{t_m})) = Alt(F(\overline{t_m}))$, and $\xi(H_i(\overline{t_m})) = \xi(F(\overline{t_m}))$. Therefore, the only way to prove termination with this method is to assume $\xi(s_i^j) < \xi(f(\overline{t'_{m'}}))$, $\forall i \forall j$. i.e. for a matching problem $f(x_1, x_2, \ldots, x_{m'}) \stackrel{?}{=} f(t'_{m'})$, solutions are of the form $\cup_i[x_i \leftarrow s_i]$, where each s_i is such that $\mid s_i \mid < \mid f(\overline{t'_{m'}}) \mid$. Lemma 3 shows it's true for root and size preserving theories.

□

4.3 Completeness

Completeness means that for each solution of a given matching problem, our strategy generates a sequence of rules which gives an equivalent solution. To prove this, we could adapt the proof given by Snyder and Gallier in [SG90]. But our context is simpler, then we give a simpler one. We first prove the following lemma.

Lemma 6 *Let $t_1 \stackrel{?}{=} t_2$ be a matching pair not solved. If $\sigma \in CSM_E(\{t_1 \stackrel{?}{=} t_2\})$, then there exists a rule ρ_i in ρ such that $\frac{t_1 \stackrel{?}{=} t_2}{\mathcal{S}}\rho_i$ and σ is solution of $\mathcal{S}$.*

Proof: Let us consider the different cases for the form of this pair and let $\sigma \in CSM_E(\{t_1 \stackrel{?}{=} t_2\})$.

Rigid-rigid :

$\mathcal{H}(t_1) \in \mathcal{C}^E$ **and** $\mathcal{H}(t_2) \in \mathcal{C}^E$: if we consider the *abstracted* t_1 i.e. $t_1[t_i \leftarrow X_i]$ where t_i are the maximal alien subterms of t_1, then $Dom(\sigma) \cap FV(\textit{abstracted}\ t_1) = \emptyset$ because *abstracted* t_1 is just made of constants of the theory and new variables.

$$\text{Hence, } \sigma t_1 = [t_i \leftarrow \sigma t_i]t_1 \quad (1)$$

Let $\theta \in CSM_E(\{\textit{abstracted}\, t_1 \stackrel{?}{=} t_2\})$ i.e. $\theta([t_i \leftarrow X_i]t_1) =_E t_2$. Then, θ will be of the form $\{[X_i \leftarrow t'_i]\}$.

$$\text{Therefore, } t_2 =_E [t_i \leftarrow X_i][X_i \leftarrow t'_i]t_1 \quad (2)$$

On the other hand, we assumed that $\sigma t_1 =_{\beta\eta E} t_2$. Then, by (1) and (2), we have (assuming that we have all the substitutions $\theta \mid \theta \in CSM_E(\{\textit{abstracted}\, t_1 \stackrel{?}{=} t_2\})$),

$$\sigma t_1 = [t_i \leftarrow \sigma t_i]t_1 =_{\beta\eta E} t_2 =_E [t_i \leftarrow t'_i]t_1$$

Therefore, $[t_i \leftarrow \sigma t_i]t_1 =_{\beta\eta E} [t_i \leftarrow t'_i]t_1$. Hence, $\sigma t_i =_{\beta\eta E} t'_i$ for each i, and this is the result of the E-matching rule. Remember that the E-matching algorithm considers the maximum alien subterms of the rigid term as constants not in the theory.

$\mathcal{H}(t_1) = \mathcal{H}(t_2) \in \mathcal{C}\backslash\mathcal{C}^E$: then $\mathcal{H}(t_1) \in \mathcal{C}$ implies $\mathcal{H}(t_1) \notin Dom(\sigma)$, therefore, $\sigma(\lambda\overline{x_m} \cdot f(\overline{t_n})) =_{\beta\eta E} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$ implies $\lambda\overline{x_m} \cdot f(\overline{\sigma t_n}) =_{\beta\eta E} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$.

Then, trivially, $\sigma t_i =_{\beta\eta E} t'_i$ which is the result of Decomp1.

$\mathcal{H}(t_1) \in BV(t_1)$ **and** $\mathcal{H}(t_2) \in BV(t_2)$: $\sigma(\lambda B_1.F(\overline{t_m})) =_{\beta\eta E} \lambda B_2.G(\overline{t'_m})$.

With $\sigma(\lambda B_1.F(\overline{t_m})) = \lambda B_1.F(\overline{\sigma t_m})$ as before. Then, $\sigma t_i = t'_i$ which is Decomp2.

Flexible-rigid : then, for the pair $\lambda\overline{x_m} \cdot F(\overline{t_p}) \stackrel{?}{=} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$, we have $\sigma \mid \sigma(\lambda\overline{x_m} \cdot F(\overline{t_p})) =_{\beta\eta E} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$. Hence, $[F \leftarrow s] \in \sigma$ because $F \in FV(\lambda\overline{x_m} \cdot F(\overline{t_p}))$. s is in $l\beta\eta$-nf, and $\tau(s) = \tau(F) = \tau(t_1) \rightarrow (\tau(t_2) \rightarrow (\cdots \rightarrow (\tau(t_p) \rightarrow \tau_0)\cdots))$. For a term of this type in $l\beta\eta$-nf, there are only two cases :

$s = \lambda\overline{x_p}.x_i$: with $\tau(x_i) = \tau_0$ which is the case of application of the projection. Then, $[F \leftarrow s](\lambda\overline{x_m} \cdot F(\overline{t_p})) =_{\beta\eta} \lambda\overline{x_m} \cdot t_i$. And we know that σ is a solution of the matching pair $\lambda\overline{x_m} \cdot t_i \stackrel{?}{=} \lambda\overline{x'_m} \cdot f(\overline{t'_m})$ which is the result of one of the projections.

$s = \lambda\overline{x_p}.s'$ with $s' \neq x_i$ We still have two subcases.

$f \notin E$ we then have $\mathcal{H}(s') = f$ and $s = \lambda\overline{x_p} \cdot f(\lambda\overline{w_{k_1}}.s_1(\overline{x_p}, \overline{w_{k_1}}), \ldots, \lambda\overline{w_{k_n}}.s_n(\overline{x_p}, \overline{w_{k_n}}))$
hence, $[F \leftarrow s]\lambda\overline{x_m} \cdot F(\overline{t_p}) =_{\beta\eta} \lambda\overline{x_p} \cdot f(\lambda\overline{w_{k_1}}.s_1(\overline{t_p}, \overline{w_{k_1}}), \ldots, \lambda\overline{w_{k_n}}.s_n(\overline{t_p}, \overline{w_{k_n}}))$ We then have $\forall i \in 1 \ldots n$, $\sigma(\lambda\overline{x_m w_{k_i}}.s_i(\overline{t_p}, \overline{w_{k_i}})) =_{\beta\eta E} \lambda\overline{x'_m} \cdot t'_i$ which is the result of the imitation and decomposition. Note that decomposition is the only rule we can apply to the pair
$\lambda\overline{x_m} \cdot f(\lambda\overline{w_{k_1}}.s_1(\overline{t_p}, \overline{w_{k_1}}), \ldots, \lambda\overline{w_{k_n}}.s_n(\overline{t_p}, \overline{w_{k_n}})) \stackrel{?}{=} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$.

$f \in E$ We then have $s = \lambda\overline{x_p} \cdot f(s_1(\overline{x_p}), \ldots, s_n(\overline{x_p}))$
hence $[F \leftarrow s]\lambda\overline{x_m} \cdot F(\overline{t_p}) =_{\beta\eta} \lambda\overline{x_m} \cdot f(s_1(\overline{t_p}), \ldots, s_n(\overline{t_p}))$ We then know that σ is solution of the matching problem
$\lambda\overline{x_m} \cdot f(s_1(\overline{t_p}), \ldots, s_n(\overline{t_p})) \stackrel{?}{=} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$ with $f \in E$ (1).
Hence, $\exists\theta \in CSM_E(f(x_1, \ldots, x_n) \stackrel{?}{=} f(\overline{t'_n}))$ i.e. $\theta f(x_1, \ldots, x_n) =_E f(\overline{t'_n})$ s.t. $(s_i(\overline{t_p}) = \theta x_i)$. This solution will be generated by imitation followed by E-matching (assuming that the E-matching algorithm gives the complete set of more general solutions).

□

Remark : We work here with root and size preserving theories, but the next section shows how to relax the root preserving condition.

The E-matching algorithm has to give the set of all solutions in a finite time (to preserve termination). Then, this set has to be finite. It is the case for size preserving theories, as shown by the following lemma.

Lemma 7 *Let E be a theory such that for each matching problem of the form $f(x_1, x_2, \ldots, x_n) \stackrel{?}{=} f(t_1, t_2, \ldots, t_n)$ where $f \in E$ and x_i are all distinct variables, we have for each E-solution $\sigma_i = \cup_{k \in 1 \ldots n} x_k \leftarrow s_k$, $\mid s_k \mid < c$ where c is a constant which depends on the size of $f(t_1, t_2, \ldots, t_n)$, then the number of solutions is finite.*

Proof: Each s_k is closed, and the number of constant symbols used in the theory is finite. Then, the number of ways to build closed terms which size is bound by c with a finite number of constants is finite. A solution of the previously defined matching problem is a set of n such terms (corresponding one to one to the x_k). Hence, we cannot built an infinite set of such substitutions. □

Let $\mathcal{S}$ be a set of matching pairs in solved form. Each element of $\mathcal{S}$ will be of the form $x \stackrel{?}{=} s$. We shall denote by $\sigma_{\mathcal{S}}$ the substitution $\{[x \leftarrow s]\}$.

Theorem 1 *(Completeness for root-preserving theories) : Let $\mathcal{S}$ be a set of matching pairs not solved. If $\sigma \in CS\mathcal{M}_E(\mathcal{S})$, then there exists a sequence of rules of ρ generated by the strategy which gives a set of solved matching pairs $\mathcal{S}_m$ such that $\sigma_{\mathcal{S}_m} =_{\beta\eta E} \sigma[FV(\mathcal{S})]$.*

Proof: By previous lemma, we know that for each $\mathcal{S}$, if we have $\sigma \in CS\mathcal{M}_E(\mathcal{S})$, then $\exists \rho_i \in \rho$ s. t. $\frac{\mathcal{S}}{\mathcal{S}'}\rho_i$ with $\sigma \in CS\mathcal{M}_E(\mathcal{S}')$. And such a sequence of rules application is finite (termination) and sound (soundness). Then $\mathcal{S}_0, \ldots, \mathcal{S}_m$ exists, $\sigma_{\mathcal{S}_m} \in CS\mathcal{M}_E(\mathcal{S}_0)$ and $\sigma \in CS\mathcal{M}_E(\mathcal{S}_m)$. Then $\sigma_{\mathcal{S}_m} =_{\beta\eta E} \sigma[FV(\mathcal{S})]$.

□

This result is valid for root and size preserving theories. This class is a little wider than permutative theories, and we shall extend it in the next section to all the size preserving theories. But theories like AC1, ACI or ACI1 are neither size preserving nor root preserving just because of their collapsing axioms. We then show in the section 6 how to deal with such theories.

5 Relaxing the root-preserving condition

After the simplest case, we shall relax the root-preserving condition on theories. In order to do that, we shall generalize the imitation rule, and modify the strategy. In fact, the only difference appears when we have to imitate a function f of the theory. If each axiom of this theory is of the form $l \sim r$ with $\mathcal{H}(l) = \mathcal{H}(r)$, then we just have to imitate f by f. But here, we have to consider all the function symbols g such that $f(\overline{x_n}) =_E g(\overline{x_n})$ has solutions. Note that the variables $\overline{x_n}$ are the same on both side by definition of size preserving theories, but the arity of g can be different from f's. That's why we have

- to adapt the definition of the imitation,
- to make the strategy generate all the appropriate imitations,
- and to verify that proofs are still correct.

First of all, if we define a **compatibility pair** as an unordered pair (f, g) where f and g are such that $f(\overline{x_n}) =_E g(\overline{x_n})$ has solutions, then we can provide, as suggested in [NQ91], the set of such pairs with the first order E-matching algorithm. Furthermore, we can say in our case that this set is finite, at least because there are a finite number of axioms in the theory, and then a finite number of $\mathcal{H}(l)$ s. t. $l \sim r \in E$. Then we shall call $CPS(E)$ this compatibility pairs set for a given theory E.

5.1 New imitation rule

The modification comes from the arity of the imitated function. The arity of f is m' and g's one is l.

$$\frac{\lambda\overline{x_n} \cdot F(\overline{t_m}) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})}{[F \leftarrow \lambda\overline{x_m}.g(\overline{u_l})]t_F \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}});\ F \stackrel{?}{=} \lambda\overline{x_m}.g(\overline{u_l})}\textbf{Imitation}$$

where

- t_F is $\lambda\overline{x_n} \cdot F(\overline{t_m})$, and $\overline{x_p}$ are new variables of the appropriate type
- with the definition of u_i :
  if $\tau(t'_i) \in T_0$ then $u_i = H_i(\overline{x_m})$ with $i \in [1 \cdots m']$ for f and $i \in [1 \cdots l]$ for any $g \neq f$
  if $\tau(t'_i) = (\alpha_1 \times \cdots \times \alpha_n \rightarrow \beta)$ then $u_i = \lambda w_1, w_2, \ldots, w_s \cdot H_i(\overline{x_m}, \overline{w_s})$ and $\tau(w_j) = \alpha_j\ 1 \leq j \leq s$
  Where w_i and H_i are new variables of the appropriate type.

Remark:

For an imitation of a symbol in the theory, the case where $\tau(t'_i) = (\alpha_1 \times \cdots \times \alpha_n \rightarrow \beta)$ is avoided because the theory is assumed of order one. Hence, a constant symbol cannot be of oder three.

5.2 Modification of the strategy

In the case of an imitation of a function f out of the theory, of course, nothing changes. We are in the case of the previous section , $f = g$, and there is only one imitation to perform. But to imitate a function of the theory, all the compatible functions will be imitated in order to keep completeness. Then the strategy for the flexible-rigid case becomes :

- if $O(\mathcal{H}(t_1)) = 1$, then either the pair is in solved form, and we did not select it, or it's a failure case,
- otherwise $O(\mathcal{H}(t_1)) = 2$ and then if $\mathcal{H}(t_2) \in \mathcal{V}$, we apply the rule Projection as many times (at most the number of arguments of $\mathcal{H}(t_1)$) as the type constraints permit it, and each

time, we shall try to find a different solution.
i.e. when we have $t_1 = \lambda\overline{x_n} \cdot F(\overline{t_m})$, we shall try a new matching with each $\lambda\overline{x_n}.t_i$ such that $\tau(t_i) = \tau(F(\overline{t_m}))$.
if $\mathcal{H}(t_2) \in \mathcal{C}$ but $\mathcal{H}(t_2) \notin E$ we shall apply the projections and we shall add the solution which will be given by the application of the imitation with $g = f$.

if $\mathcal{H}(t_2) \in \mathcal{C}$ and $\mathcal{H}(t_2) \in E$ we shall apply the projections and we shall generate all the imitations of f by f $(g = f)$ and all the compatible functions (i.e. f is imitated by each functions g such that (f, g) or (g, f) is in $CPS(E)$).

Let us give an example to illustrate such a case.

Example 14 *let $E = \{g(k(x, y)) \sim f(h(x), y)\}$. E is size preserving but is not root-preserving. Obviously $CPS(E) = \{(f, g)\}$. Then, if we have the matching problem :*

$$\lambda yz \cdot F(y, z) \stackrel{?}{=} \lambda y'z' \cdot f(h(a), b),$$

we shall generate projections and :

$$\lambda yz \cdot F(y,z) \stackrel{?}{=} \lambda y'z' \cdot f(h(a),b)$$

imitation by f ↓ — *imitation by g* →

$$\lambda yz \cdot f(H_1(y,z), H_2(y,z)) \stackrel{?}{=} \lambda y'z' \cdot f(h(a),b) \qquad \lambda yz \cdot g(H_1(y,z)) \stackrel{?}{=} \lambda y'z' \cdot f(h(a),b)$$

5.3 Modifications in the proofs

We begin with the soundness of the imitation rule. We have substituted f by g and m' by l. We have σ such that

$$\sigma(\lambda\overline{x_n}.g(H_1(\overline{t_m}), \ldots, H_l(\overline{t_m}))) =_{\beta\eta E} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$$

Then, if we note θ the imitation, $\theta = [F \leftarrow \lambda x_1 \ldots x_m.g(H_1(x_1, \ldots, x_m), \ldots, H_l(x_1, \ldots, x_m))]$. $\theta(\lambda\overline{x_n} \cdot F(\overline{t_m})) =_{\beta\eta} \lambda\overline{x_n}.g(H_1(\overline{t_m}), \ldots, H_l(\overline{t_m}))$ Hence, $(\sigma \cup \theta)(\lambda\overline{x_n} \cdot F(\overline{t_m})) =_{\beta\eta E} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$

and all is still correct.

For the termination in the case of an imitation of a function of the theory, we have to verify two things. First the number of generated imitations is finite because $CPS(E)$ is finite. Second one, the part of the termination proof concerning this operation, becomes :

After application of the imitation and β-reduction, we have the same pair :

$$\lambda\overline{x_n}.g(H_1(\overline{t_m}), \ldots, H_l(\overline{t_m})) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})$$

But f and $g \in E$ and $H_i \notin E$. Then, trivially $MAS(\lambda\overline{x_n}.g(\overline{H_l(\overline{t_m})})) = \cup_{i=1..l}\{H_i(\overline{t_m})\}$. The E-matching rule is then applied on the pair $g(x_1, x_2, \ldots, x_l) \stackrel{?}{=} f(\overline{t'_{m'}})$ where the x_i's are the

abstractions of the $(H_i(\overline{t_m}))$'s. The k (assumed finite) solutions will be :

	$sol.1$					$sol.k$		
$H_1(\overline{t_m})$	x_1	$\leftarrow$	s_1^1	$\cdots$	$\cdots$	x_1	$\leftarrow$	s_1^k
$\vdots$		$\vdots$					$\vdots$	
$H_l(\overline{t_m})$	x_l	$\leftarrow$	s_l^1	$\cdots$	$\cdots$	x_l	$\leftarrow$	s_l^k

We shall have then the k matching problems :
$\mathcal{S}_1 = \cup_{i\in 1\ldots l}(H_i(\overline{t_m}) \stackrel{?}{=} s_i^1)$
$\vdots$
$\mathcal{S}_k = \cup_{i\in 1\ldots l}(H_i(\overline{t_m}) \stackrel{?}{=} s_i^k)$
We then have $Alt(H_i(\overline{t_m})) = Alt(F(\overline{t_m}))$, and $\xi(H_i(\overline{t_m})) = \xi(F(\overline{t_m}))$. Therefore, the only way to prove termination with this method is to assume $\xi(s_i^j) < \xi(f(\overline{t'_{m'}}))$, $\forall i \forall j$. i.e. for a matching problem $g(x_1, x_2, \ldots, x_l) \stackrel{?}{=} f(t'_{m'})$, solutions are of the form $\cup_i[x_i \leftarrow s_i]$, where each s_i is such that $|\ s_i\ | < |\ f(\overline{t'_{m'}})\ |$.

And this is true for theories which preserve size, equally not root-preserving.

For completeness, we are only interesting in the flexible-rigid subcase where $f \in E$. We have then

$s = \lambda\overline{x_p} \cdot g(s_1(\overline{x_p}), \ldots, s_l(\overline{x_p}))$ hence $[F \leftarrow s]\lambda\overline{x_m} \cdot F(\overline{t_p}) =_{\beta\eta} \lambda\overline{x_m} \cdot g(s_1(\overline{t_p}), \ldots, s_l(\overline{t_p}))$. We then know that σ is solution of the matching problem
$\lambda\overline{x_m} \cdot g(s_1(\overline{t_p}), \ldots, s_l(\overline{t_p})) \stackrel{?}{=} \lambda\overline{x'_m} \cdot f(\overline{t'_n})$ with f and $g \in E$ (1).
Hence, $\exists\theta \in CS\mathcal{M}_E(g(x_1, \ldots, x_l) \stackrel{?}{=} f(\overline{t'_n}))$ i.e. $\theta g(x_1, \ldots, x_l) =_E f(\overline{t'_n})$ Then, a solution of (1) which corresponds to a solution generated by the appropriate imitation (here, the imitation by g). The strategy says that we try all the possible ones.

We then preserve the three properties of soundness, termination and completeness. We can conclude that what Nipkow and Qian [NQ91] suggested in order to reduce nondeterminism can be used here to increase the class of theories we can deal with, without losing neither completeness nor termination.

6 Collapsing axioms

We make now the assumption that the theory E we use, is the union of E_1 and E_2 such that E_1 preserves root and size, and E_2 is just made of regular collapsing axioms.

Regular collapsing axiom : a regular axiom is an axiom $l \sim r$ such that $\mathcal{V}(l) = \mathcal{V}(r)$. And it is a collapsing one if $r = x$ where x is a variable. Then, a regular collapsing axiom has the form $l \sim x$ with $\mathcal{V}(l) = \{x\}$, but $l \neq x$. Note that it implies $|\ l\ | > 1$.

Collapsed form : we say that a term is in collapsed form if no collapsing axiom can be applied to it in the direction $l \rightarrow x$ at any occurence.

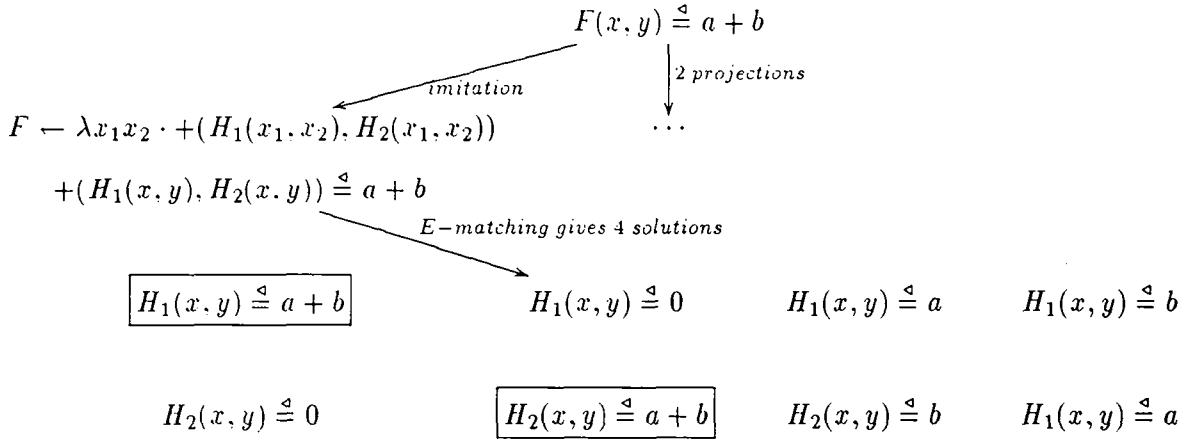
By application of a collapsing rule, a term can only get smaller. Then the collapsed form t of a term is smaller than any others in an infinite equivalence class $\|t\|_E$ such that $t' \in \|t\|_E$ iff

$t' \rightarrow^{+}_{l \rightarrow x} t$ (+ means at least one time) because $t' \rightarrow^{+}_{l \rightarrow x} t \Rightarrow |t'| > |t|$ is obvious.

Because each term has a collapsed form, we shall assume we only have terms in collapsed form. Of course, we have to assume now, that the first order E-matching algorithm gives a finite complete set of more general E-matchers because we have an infinity of solutions as one term has an infinity of E-equivlent terms.

Then, what will change? Nothing for first order (rigid-rigid pairs). The E-matching rule will make the alternation of the complexity triple decrease strictly, and the number of more general solutions is assumed finite. Then, no termination problem. The real problem is again the imitation rule. We shall illutrate it with the following example:

Example 15 *The theory we consider is AC1. AC1 for + is* $AC \cup \{x + 0 \sim x\}$. *Assume we have the following matching pair :* $F(x, y) \stackrel{?}{=} a + b$ *we then have the following computation :*



We see that the two framed new matching pairs are equivalent to $F(x, y) \stackrel{?}{=} a+b$. Then, with the simple idea that two same problems have the same solutions, and it's no need to compute them twice, we transform the strategy for flexible-rigid terms into :

- if $O(\mathcal{H}(t_1)) = 1$, then either the pair is in solved form, and we do not select it, or it is a failure case,
- (this part is the usual one) if $\mathcal{H}(t_2) \notin E$, then if $\mathcal{H}(t_2) \in \mathcal{V}$, we apply the rule Projection as many times (at most the number of arguments of $\mathcal{H}(t_1)$) as the type constraints permit it, and each time, we shall try to find a different solution.
  i.e. when we have $t_1 = \lambda \overline{x_m} \cdot F(\overline{t_p})$, we shall try a new matching with each $\lambda \overline{x_m}.t_i$ such that $\tau(t_i) = \tau(F(\overline{t_p}))$.
  if $\mathcal{H}(t_2) \in \mathcal{C}$, we shall then apply the projections as in the previous case, and we shall add another solution which will be given by the application of the imitation followed by the decomposition.
- otherwise we have a matching pair $P = \{\lambda \overline{x_n} \cdot F(\overline{t_m}) \stackrel{?}{=} \lambda \overline{x'_n} \cdot f(\overline{t'_{m'}})\}$ with $f \in E$. We apply the projection rule as in the previous cases, and we apply the imitation followed by E-matching. We continue with each matching pair of each solution of the E-matching except the ones which are of the form $\{\lambda \overline{x_n} \cdot H_i(\overline{t_m}) \stackrel{?}{=} \lambda \overline{x'_n} \cdot f(\overline{t'_{m'}})\}$. The solutions of these pruned matching pairs are the solutions that we find for P.

There is no soundness problem because we don't change the rules. It is the same for completeness since we remember that we are working with a first order E-matching algorithm which gives all the more general solutions.

For termination, we have to look a little more precisely. For rigid-rigid pairs, there is no change because E-matching gives a finite number of solutions. In fact, the strategy just cut a looping branch of the search tree. Then, if each matching pair (except $\{\lambda\overline{x_n} \cdot H_i(\overline{t_m}) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})\}$) terminates, all the matching problem terminates. Indeed, the solutions of P are the solutions of the pairs $\{\lambda\overline{x_n} \cdot H_i(\overline{t_m}) \stackrel{?}{=} \lambda\overline{x'_n} \cdot f(\overline{t'_{m'}})\}$. Therefore, if we have the solutions of P, we have their solutions.

We just have to verify that the complexity triple is decreasing with the other matching pairs to valid the termination result.

As previously, we have $Alt(H_i(\overline{t_m})) = Alt(F(\overline{t_m}))$ and $\xi(H_i(\overline{t_m})) = \xi(F(\overline{t_m}))$. Then, we have to assume again that for each $s_i \neq f(\overline{t'_{m'}})$, we have $|\ s_i\ | < |\ f(\overline{t'_{m'}})\ |$. This is done by the following lemma :

Lemma 8 *For a theory E such that $E = E_1 \cup E_2$ with E_1 a root and size preserving theory, and E_2 a set of regular collapsing axioms, then the more general solutions $\{x_i \leftarrow s_i\}$ of a matching problem $f(\overline{x_m}) \stackrel{?}{=} f(\overline{t_m})$ are such that $s_i =_E f(\overline{t_m})$, or $|\ s_i\ | < |\ f(\overline{t_m})\ |$.*

Proof: Assume s_i in collapsed form such that $\{x_i \leftarrow s_i\}$ is in the solution.
Then, $f(x_1, \ldots, s_i, \ldots, x_n) =_E f(\overline{t_m})$. There are two cases :

- $f(x_1, \ldots, x_i, \ldots, x_n) =_{E_2} x_i$ then, either $s_i =_{E_1} f(\overline{t_m})$ and $|\ s_i\ | = |\ f(\overline{t_m})\ |$, or s_i is not in collapsed form,
- or $f(x_1, \ldots, s_i, \ldots, x_n) =_{E_1} f(\overline{t_m})$ which implies by definition of size preserving theories that $|\ s_i\ | < |\ f(\overline{t_m})\ |$.

□

Then, we preserve termination for such theories. It permits in particular to deal with AC1, ACI and ACI1. Note that the root preserving condition can be removed here as in the previous section.

Remark about complexity : We can see the search of solutions as a tree which has $\mathcal{S}_0$ as root and inference rules as nodes. A leaf is then a success or a failure. Then, if we can find a binding of the height of this tree and a binding of the complexity of the rules, we shall be able to bind the complexity of the algorithm for finding a solution.
Let $T(n, a)$ be the height of the search tree for the matching pair $t_1 \stackrel{?}{=} t_2$ where n is the number of symbols in the matrix of t_1 and a is an upper bound of the function symbol arities. One can see that for each inference rule, we have the tree height which is O(n).

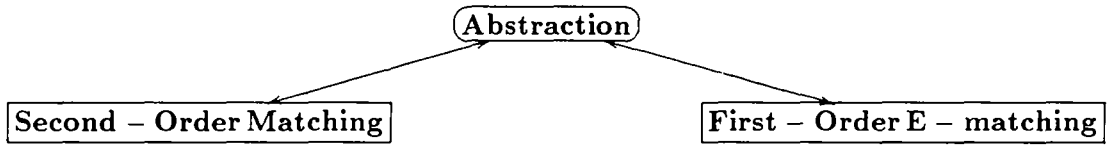
For instance, by decomposition, we obtain a new problems if a is the arity of the decomposed function. Then, $T(n, a) = T(n_1, a) + T(n_2, a) + \cdots + T(n_p, a)$ with $p \leq a$ and $\Sigma n_i = n - 1$. Then,

if we assume that $T(n,a) \leq C.n$ for a constant C, by reccurence, we have $T(n,a) \leq C.\Sigma n_i = C(n-1)$. Similar reasoning can be held for the other rules. We then have a height which is linear for the number of symbols in t_1.
In the other hand, the most complex rule is obviously the *E-matching* rule. For instance, if we take the AC theory, we know that the first order AC-matching is NP. Hence, the complexity for finding a solution to a second order AC-matching problem via our algorithm is NP.

7 Implementation

Our main purpose was to implement this algorithm, and a main aspect of this work is modularity. This concept appears in different ways.

- **Reusability :** first of all, we use two known algorithms. The second order matching algorithm described by Huet and Lang in [HL78], and an E-matching algorithm for first order terms. The E-matching rule permits us to make the link using the abstraction which is like a pipe between them.



  We can say it in two ways :

  - To first order E-matching, we add second order to obtain second order E-matching,
  - or to second order matching, we add a theory to obtain second order E-matching.

- **Adaptability, modifiability :** for our practical application, E has been instantiated with the AC-theory for the collapse-free theories case, and with AC1 for the collapsing case. But it can be replaced by other E-matching algorithms if E is any collapsing size preserving theory.

- **Hidding :** We do not need to know how E-matching is done. We just need to be sure that it is sound, complete and can deal with free constants.

We shall show now examples which point out the many possible applications of such algorithms.

Indeed, there are many applications of second order E-matching. Each domain where one tries to recognize higher order patterns will be described more precisely if first order algebraic properties are taken into account. This enhances the expressiveness of the syntactical approach in many A.I. applications. We show in the next example how we can recognize a recurrent pattern in a logic program. But it can be applied also to automated reasoning, program transformation, higher order rewriting systems, ...

7.1 Recursive functions

We give first the same example as in [HL78]. The purpose was to give automatically an iterative version of a given recursive program. This example doesn't need any algebraic theory. But on the same scheme, it is easy to imagine matching problems where function algebraic properties must be taken into account in order to find any solution. *fixpt* represents the fixpoint operator and *cond* is the conditional one. The first term represents the recursive functions of the form :

$$f(x) = \text{If } a(x) \text{ then } b(x) \text{ else } h(d(x), f(e(x))).$$

The recursive function we try to match is the reverse function :

$$rev(x) = \text{If } null(x) \text{ then } nil \text{ else } append(rev(cdr(x)), cons((car(x)), nil))$$

Then, the matching problem

$$\lambda y.\mathtt{fixpt}(\lambda f\ x.\mathtt{cond}(a(x), b(x), h(d(x), (f(e(x))))), y)$$

$$\stackrel{\lhd}{=}$$

$$\lambda y.\mathtt{fixpt}(\lambda rev\ x.\mathtt{cond}(\mathtt{null}(x), \mathtt{nil}, \mathtt{append}(\mathtt{rev}(\mathtt{cdr}(x)), \mathtt{cons}((\mathtt{car}\ x), \mathtt{nil}))), y)$$

has for solutions:

$a \leftarrow \lambda x_6.\mathtt{null}(x_6)$
$b \leftarrow \lambda x_5.\mathtt{nil}$
$e \leftarrow \lambda x_4.\mathtt{cdr}(x_4)$
$d \leftarrow \lambda x_3.x_3$
$h \leftarrow \lambda x_1\ x_2.\mathtt{append}(x_1, \mathtt{cons}(\mathtt{car}(x_2), \mathtt{nil}))$

$a \leftarrow \lambda x_6.\mathtt{null}(x_6)$
$b \leftarrow \lambda x_5.\mathtt{nil}$
$e \leftarrow \lambda x_4.\mathtt{cdr}(x_4)$
$d \leftarrow \lambda x_3.\mathtt{car}(x_3)$
$h \leftarrow \lambda x_1\ x_2.\mathtt{append}(x_1, \mathtt{cons}(x_2, \mathtt{nil}))$

$a \leftarrow \lambda x_6.\mathtt{null}(x_6)$
$b \leftarrow \lambda x_5.\mathtt{nil}$
$e \leftarrow \lambda x_4.\mathtt{cdr}(x_4)$
$d \leftarrow \lambda x_3.\mathtt{cons}(\mathtt{car}(x_3), \mathtt{nil})$
$h \leftarrow \lambda x_1\ x_2\mathtt{append}.(x_1, x_2)$

This example illustrates very well the second order case. We could match functions in which the role of the *append* function is performed by a function which is in a theory like AC1.

7.2 Finding a recurrence pattern in a logic program

Assuming we are working with Horn clauses, we know that a recurrence schema on the integers for any predicate P has the form : 'P(0) $\wedge$ P(suc(x)) : $-$P(x)'. Then, if we try to match a logic program with this pattern, the algorithm will return all the relations which can instantiate P. i.e. all the relations on which a recurrence is applied. For instance :

$$\mathtt{F}(\mathtt{P}(0) \wedge (\mathtt{P}(\mathtt{suc}(x)) :- \mathtt{P}(x))) \stackrel{\lhd}{=} \mathtt{b} :- \mathtt{a} \wedge \mathtt{p}(\mathtt{suc}(x_1)) :- \mathtt{p}(x_1) \wedge \mathtt{d} :- \mathtt{c} \wedge \mathtt{p}(0) \wedge \mathtt{g} :- \mathtt{e}$$

will give the solutions[1] :

$$\begin{array}{lll}
x \leftarrow x_1 & F \leftarrow \lambda y.(g : -e \wedge & x \leftarrow x_1 \\
P \leftarrow \lambda y \cdot p(y) & \quad d : -c \wedge & P \leftarrow \lambda y \cdot p(y) \\
F \leftarrow \lambda y.\ (g : -e \wedge & \quad p(0) \wedge & F \leftarrow \lambda y.\ (g : -e \wedge \quad [\cdots] \\
\quad d : -c \wedge & \quad b : -a \wedge & \quad d : -c \wedge \\
\quad b : -a \wedge & \quad p(suc(x_1)) : -p(x_1)) & \quad y \wedge \\
\quad y) & & \quad b : -a)
\end{array}$$

Without considering the AC properties of '$\wedge$', we would not be able to produce this result. First order E-matching should try to match at each possible occurrence, but furthermore with each relation P occurring in the program. This pattern recognition is also useful in fold-unfold program transformation where the folding rule rapidly becomes unmanageable by hand and requires some mechanical tool.

7.3 Simplification of a logic formula

As previously stated, looking for tautologies can be applied in a pre-processing step in order to simplify formulae before invoking a theorem prover. We shall illustrate here a new way for simplifying formulae using this method. Consider the following formula : $(p(x) \Rightarrow q(x)) \wedge (q(y) \Rightarrow r(y))$ where x and y are universally quantified. Trivially, by transitivity of the implication, we can simplify into $p(x) \Rightarrow r(x)$. In order to make such simplifications in the formulae, we perform some abstraction on this scheme to obtain the more general scheme :
$\psi = F((P(x) \Rightarrow Q(x)) \wedge (Q(y) \Rightarrow R(y)))$ where P Q and R are second order variables. We can then match the pair $\psi \stackrel{?}{=} \varphi$ with $\varphi = ((q(x) \Rightarrow r(x)) \wedge (p(a) \vee r(b)) \wedge (p(x) \Rightarrow q(x))) \Rightarrow (r(a) \vee r(b))$. The matching problem

$$F((P(x) \Rightarrow Q(x)) \wedge (Q(y) \Rightarrow R(y))) \stackrel{?}{=} ((q(x_1) \Rightarrow r(x_1)) \wedge (p(a) \vee r(b)) \wedge (p(x_1) \Rightarrow q(x_1))) \Rightarrow (r(a) \vee r(b))$$

has a solution :

$$\begin{array}{l}
F \leftarrow \lambda z \cdot (z \wedge (p(a) \vee r(b))) \Rightarrow (r(a) \vee r(b)) \\
P \leftarrow \lambda z \cdot p(z) \\
Q \leftarrow \lambda z \cdot q(z) \\
R \leftarrow \lambda z \cdot r(z) \\
x \leftarrow x_1 \\
y \leftarrow x_1
\end{array}$$

Hence, we obtain : $\varphi' = ((p(x) \Rightarrow r(x)) \wedge (p(a) \vee r(b))) \Rightarrow (r(a) \vee r(b))$ after simplification. Again, this result could not be obtained without taking account of the AC properties of '$\wedge$'. To use first order, we should have tried all the combinations to instantiate P Q and R, and to try to match at each position in φ.

[1] Note that x_1 is a frozen variable (other solutions do exist).

8 Conclusions

The main problem related to the implementation, is the number of solutions and this has two essential reasons. First, we deal with second order. This produces a lot of useless solutions, in particular, solutions which assign closed functions (terms like $\lambda x.t$ with $x \notin t$) to second order variables. One can be easily convinced by looking at the presented examples. Secondly, we cannot avoid the number of solutions given by the first order E-matching. Furthermore, these two reasons can be combined. For example, if one solution is a closed function $\lambda\overline{x_n} \cdot s$ we shall have as other solutions, all the closed functions $\lambda\overline{x_n} \cdot s'$ such that s' is equivalent to s modulo the considered theory. These solutions are obviously useless. One solution could be to keep only one element of this class which would be the normal form. But we are not interested at all by these solutions and computing only one good solution is a much better way of improving the pattern recognition step in Automated Theorem Proving. We are convinced that for our purpose, the first solution which have no closed function in its range associated with a second order variable symbol is the one we need.
Note that experiments have shown that to use a term representation à la De Bruijn or a similar encoding will not provide a substantial improvement since the most important wasting of time is due to irrelevant solutions.

What we have presented is a second order matching algorithm which is able to take into account algebraic properties. The termination property is obtained for collapsing size preserving theories and an appropriate strategy. This algorithm is a first step toward a toolbox using higher order mechanisms in Automated Theorem Proving which is currently under development.

Acknowledgments : We are very grateful to Denis Lugiez for his help during this work and to Eric Domenjoud for helpful discussions about theories.

References

[BT88] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 82–90, 1988.

[HL78] G. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[HS86] J. Roger Hindley and Johnathan P. Seldin. *Introduction to Combinators and Lambda-calculus.* Cambridge University, 1986.

[Hue76] G. Huet. *Résolution d'equations dans les langages d'ordre 1,2, ...,ω.* Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[NQ91] T. Nipkow and Z. Qian. Modular higher-order E-unification. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 1991.

[QW92] Z. Qian and K. Wang. Higher-order E-unification for arbitrary theories. In K. Apt, editor, *LOGIC PROGRAMMING : Proceedings of 1992 joint international conference and symposium on logic programming*, 1992.

[SG90] W. Snyder and J. Gallier. Higher-order unification revisited: Complete sets of tranformations. In Claude Kirchner, editor, *Unification*, pages 565–604. Academic Press, London, 1990.

[Sny88] W.S. Snyder. *Complete Sets of Transformations for General Unification.* PhD thesis, University of Pennsylvania, 1988.

Unité de Recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)

Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399