Compiling Pattern Matching by Term Decomposition

LAURENCE PUEL§† AND ASCANDER SUAREZ‡‡

§LRI URA 410 du CNRS, bat 490, Université Paris Sud, 91405 Orsay, FRANCE
‡‡DIGITAL PRL, 85 Av Victor Hugo, 92568 Rueil Malmaison, FRANCE

(Received 17 July 1990)

We present a method for compiling pattern matching on lazy languages based on previous work by Laville, Huet and Lévy. It consists of coding ambiguous linear sets of patterns using "Term Decomposition," and producing non-ambiguous sets over terms with structural constraints on variables. The method can also be applied to strict languages giving a match algorithm that include only unavoidable tests when such an algorithm exists.

1. Introduction

We are interested in compiling pattern matching in case of partially evaluated terms in order to do only necessary computations for the match. This is a kind of lazy computation over partially defined terms. Huet and Lévy (1979, 1991) defined a method for constructing match trees for non-ambiguous linear term rewriting systems (this method is also explained by Klop and Middeldorp (1991)). However, the application of their results to the problem of compiling pattern matching as in the ML language was not clear until 1988 when Laville (1988a, 1988b) showed that it is possible to use their method for ambiguous term rewriting systems with a given priority on rules. This priority is necessary to decide which rule has to be used in case of conflict. Laville designed a new match predicate that takes into account the priority when building the match trees. When this construction is successful, the leaves of the match tree form a Minimal Extended Set of Patterns equivalent (from the match point of view) to the original system in the case of finite signatures.

Our method is to code ambiguous ordered term rewriting systems into non-ambiguous ones over constrained terms. We replace the priority rule between left parts of the rewriting system by constraints over terms. Therefore the match predicate is that of Huet and Lévy but over constrained terms. Their results are then extended to these terms. Furthermore, as a result of the computation of the non-ambiguous set of terms of the system, we also obtain a characterization of the set of partially evaluated terms for which every matching algorithm will loop. We call it the strict set of the system. Although some
algorithms may loop on other terms, an optimal algorithm, if it exists, will only loop on the strict set.

1.1. CONSTRAINED TERMS

A term with variables is a representation of all ground terms obtained by replacing its variables by terms with no variables. A subset of a given set can be defined either by a description of its elements or as the complement of another subset. For example, a variable $x$ represents the set of all the ground terms and $F(A, y)$ a subset of $x$. We can split the set $x$ into three subsets. First, the set of instances of $F(A, y)$. Then, the set of terms for which we can decide that they are not instances of $F(A, y)$, denoted $\{x \mid x \notin \{F(A, y)\}\}$. Finally, the set that contains partially evaluated terms of the form $F(\ldots)$ whose first argument cannot be evaluated (its computation loops) denoted $F(\bullet, y)$ as well as the non-evaluated term that we denote $\bullet$. With the set notation, this partition of the set of terms is written:

$$x = F(A, y) \cup \{x \mid x \notin \{F(A, y)\}\} \cup \bullet \cup F(\bullet, y)$$

Following this idea, we define the concept of constrained terms and give some of their algebraic properties.

With this formalism an ordered ambiguous set of terms can be transformed into a non-ambiguous set of constrained terms. For instance, the set of terms $F(A, y), F(x, y)$ is ambiguous as the term $F(A, B)$ is an instance of both of them. The set of constrained terms $F(A, y), \{F(x, y) \mid x \notin \{A\}\}, \{x \mid x \notin \{F(\ldots)\}\}, \bullet, F(\bullet, y)$ is not ambiguous since $F(\bullet, y)$ is not an instance of $\{F(x, y) \mid x \notin \{A\}\}$. Now a given term is an instance of exactly one of these constrained terms.

1.2. PATTERN MATCHING

Call by pattern matching is one of the main features of the ML language (Harper, Milner and Tofte, 1988), (Weis, Aponte, Laville, Mauny and Suárez 1989) and was inherited from HOPE (Burstall, MacQueen and Sannella, 1980). It may be viewed as a generalization of the “case” statement of imperative languages. In ML, one can define one’s own structural types and very easily write operations over them. We will introduce call by pattern matching by extending the Pascal definitions of “enumerated types”

In Pascal, it is possible to use the case statement to select among different cases by the value of an expression of an “enumerated type”:

```pascal
  type T = (C_1, \ldots, C_n);  
  var x:T;  
  ...  
  case x of  
    C_1 : <exp_1>  
    | ...  
    | C_i : <exp_i>  
    | otherwise : <exp>  
```

The natural extension of this construct is to allow matching not only constant values but more general data structures as in the following example in the language ML where there are two cases in the definition of the type of trees: Leaf to represent the leaves of trees and the constructor Tree for the other nodes.
type Tree =
  Leaf of number
| Tree of number * tree * tree;

val tree: Tree = Tree(3, Leaf(2), Tree(4, Leaf(7), Leaf(9)));

In this example the value of the variable "tree" will match the second and the fourth cases but taking the first one as the priority holder, the expression <<exp2>> will be executed.

1.3. Compilation

If patterns are non-ambiguous, there is a decision procedure due to Huet and Lévy (1979, 1991) that determines whether an optimal match exists for a set of patterns and, in the case where such a match exists, produces a search tree that allows to compile the match problem. This method can be illustrated with the following example.

Suppose that we want to match pairs of terms (x, y) of Booleans by the set of patterns (true, true), (_, false) and (false, true). We choose to look first at a column that only contains constants (in our example the second one), and divide the patterns by the constants appearing in that column. The result is a transformed program in which there is always one column in the pattern to look at.

\[
\begin{align*}
\text{case } (x, y) \text{ of} & \quad \text{case } y \text{ of} \\
\text{true , true ) } \rightarrow 1 & \quad \text{true } \rightarrow (\text{case } x \text{ of } \\
\text{false, true } \rightarrow 3 & \quad \text{false } \rightarrow 3)
\end{align*}
\]

There are some sets of patterns, namely non-sequential patterns, for which the method of Huet and Lévy (1979) fails. The typical example is the Gustave's function, due to Berry (1978):

\[(A,B,\_), \ (B,\_,A), \ (_,A,B)\]

In this example the patterns are non-ambiguous but there is no column in which we can make the decision. So if we want to avoid looping in the evaluation of this match, a parallel mechanism that inspects simultaneously all three columns is necessary.

But the restriction that patterns must be non-ambiguous is a burden to the programmer especially when the program contains data structures with many different constructors. This is one of the reasons why most programming languages that feature call by pattern matching accept ambiguities and impose a priority rule between different patterns. In this paper we do not discuss assignment of priorities. In ML and other programming languages, for instance, the order of patterns in the text is used and the programmer
has to write the more specific cases before the more general ones. Another possibility to automatically assign higher priority to specific cases and still use textual ordering for those that are compatible. Both priority rules have the same expressive power as any set of patterns can be ordered to work exactly in the same way with any of them.

When ambiguities are allowed and a priority rule is imposed, the method of Huet and Lévy (1979) does not apply directly, as shown below:

\[
\text{case } (x,y) \text{ of } \\
| (\_, \text{true}) \rightarrow 1 \\
| (\text{false}, \_ ) \rightarrow 2 \\
| (\_, \_ ) \rightarrow 3 \\
\]

Now take any pair \((x,y)\), if \(y=\text{true}\) then the pair \((x,y)\) matches the first case. Otherwise, if \(x=\text{false}\), it matches the second one. Finally in every other case, it matches the third one. Remark that it is slightly subtle to find the set of pairs which match this three cases. The first case corresponds to any pair \((x,\text{true})\), the second one to \(\text{false},\text{false}\) and the third one to \((\text{true},\text{false})\). In this example, where both the first and the second column only have one constant, the method of Huet and Lévy (1979) does not apply directly. It can be adapted (as done by Laville (1988a)) by imposing priorities to the patterns to make them non-ambiguous.

Our approach in this work is to use the data structure of constrained terms to represent sets of patterns ordered by priorities such that the disambiguating rule becomes part of the representation. In the previous example, the set of constrained terms that represents the match problem is: \{((\_ , \text{true}), (\text{false}, \neq \text{true}), (\neq \text{false}, \neq \text{true}))\} in which \(\neq C\) represents any value different from \(C\) and the strict set is: \{((\_ , \_ ),(\_ , \_ ))\}. Notice that an algorithm that evaluates from left to right will also loop on the term \((\_ , \_ )\) while an algorithm that evaluates this pair from right to left will not. With the non-ambiguous set of terms given above it becomes possible to apply the method of Huet and Lévy (1979) and choose the second column as the one to look at first (where \(\neq C\) is considered as a constant).

A program for the compilation of patterns with priorities was written by Laville (1988a), in CAML (Weis, Aponte, Laville, Mauny and Suárez 1989), a dialect of the ML family. The construction of the new set of non-ambiguous patterns is embedded in the control of the program. In our work, the transformation from ambiguous to non-ambiguous patterns will be achieved at the level of source programs. This makes the program transformation explicit and independent of the pattern matching process. Furthermore, the algorithm presented here produces very compact representations, especially in the matching of terms with arbitrarily large signature.

2. Terms and Constraints

2.1. Linear Terms

Let \(X\) be a denumerable set of variables and \(\Sigma\) a set containing function symbols and an additional symbol \(\cdot\). To each function symbol is associated its arity. For our purpose the language of terms \(T(\Sigma, X)\) is defined by:

\[
t := F(t_1, \ldots, t_n) \mid x \mid \cdot
\]
where the function symbol $F$ is a symbol of $\Sigma$ of arity $n$, the variable $x$ is in $X$, $t_1, \ldots, t_n$ are terms and, for $i \neq j$, the sets of variables of $t_i$ and $t_j$ are disjoint. The set of terms without variable is the set $T(\Sigma)$ of ground terms. The set of partially evaluated terms is the set $T(\Sigma \setminus \{\cdot\}, X)$.

The set $O(t)$ of occurrences of a term $t$ is recursively defined by:

\[ \epsilon \in O(t) \]
\[ i.u \in O(F(t_1, \ldots, t_n)) \text{ if } u \in O(t_i) \text{ (} 1 \leq i \leq n \text{)} \]

If $u \in O(t)$ the subterm $t/u$ of $t$ is defined by:

\[ t/\epsilon = t \]
\[ F(t_1, \ldots, t_n)/i.u = t_i/u \]

and $t(u)$ denotes the root label of $t/u$.

**Definition 2.1.**

1. A (ground) substitution $\sigma$, is a mapping over terms defined by replacing a finite set of variables by (ground) terms which transforms any term $t$ into $\sigma(t)$. The term $\sigma(t)$ is called an instance of $t$.

2. The quasi-ordering $\preceq$ over terms is defined by $t \preceq t'$ if there exists a substitution $\sigma$ such that $\sigma(t) = t'$ and $t$ is said to be a prefix of $t'$. Its extension to substitutions is defined by $\sigma \preceq \sigma'$ if and only if there exists a substitution $\eta$ such that $\sigma' = \eta \circ \sigma$. Thus $\sigma$ is said to be more general than $\sigma'$.

3. Two terms $t$ and $t'$ are comparable if either $t \preceq t'$ or $t' \preceq t$ and compatible or unifiable, denoted $t \uparrow t'$, if there exists a substitution $\sigma$, such that $\sigma(t) = \sigma(t')$, in which case $\sigma$ is a unifier of $t$ and $t'$.

4. The least upper bound of two terms $t$ and $t'$, denoted $t \sqcup t'$, is the smallest term that has both $t$ and $t'$ as prefixes. This term is unique up to variable renaming, which is the equivalence induced by $\preceq$. A unifier that produces this bound if it exists is called a most general unifier (m.g.u.). The greatest lower bound of two terms $t$ and $t'$, denoted $t \sqcap t'$ is the greatest prefix of $t$ and $t'$ (with respect to $\preceq$).

In the following it will be convenient to identify a term with the set of its ground instances. For example, let $\Sigma = \{F, A, B, \cdot\}$. The term $t = F(x, y)$ represents the set \{F(A, A), F(B, A), F(\cdot, A), F(F(A, A), A), \ldots\}. Any ground term $t$ represents the set \{t\}.

The relation $\preceq$ is the opposite of the set inclusion of the ground instances. A substitution can be seen as an operation that allows to build terms from the root to the leaves. The special term $\cdot$ will denote terms that cannot be built as, for instance, those whose construction does not terminate; a substitution $\sigma$ such that $\sigma(x) = \cdot$ can be assimilated to a construction that never ends and consequently, it is meaningless to compare $\cdot$ with a given term. This leads to the following notion of a computable incompatibility between two terms defined by $t_1 \not\preceq t_2$ if and only if there exists an occurrence $u \in O(t_1) \cap O(t_2)$ such that $t_1(u) \neq t_2(u)$ and $t_i(u) \in \Sigma \setminus \{\cdot\}$, ($i = 1, 2$).

Now we want to represent more precisely sets of terms; for instance the subset of all terms which are not instances of $F(x, y)$. Thus we classify all the terms in three parts: those that are instances of $F(x, y)$, the term $\cdot$ which represents terms that cannot be
built and those that are instances of any $G(\ldots)$ with $G \neq F$ and $G \neq \bullet$. The last part represents terms for which we know that they are not instances of $F(x, y)$ while the second part represents terms for which we cannot say anything. Considering only substitutions that produce terms without $\bullet$ as a subterm, the subset of $F(x, y)$ of all terms different from $F(A, B)$ is the set of ground terms $\{F(\sigma(x), \sigma(y))\mid \sigma(x) \notin \{A\} \text{ or } \sigma(y) \notin \{B\}\}$. With the finite signature $\Sigma = \{F, A, B, \bullet\}$, this set is represented as the union of $F(x, A)$, $F(x, F(y, z))$, $F(B, x)$ and $F(F(x, y), z)$. This representation depends on the number of elements of the signature, for instance using $\Sigma' = \{F, A, B, C, \bullet\}$ the representation as union of terms has two extra components: $F(x, C)$ and $F(C, y)$. With an infinite signature it is not possible to represent this set as a finite union of instances of terms. Notice that the two terms $F(\bullet, y)$ and $F(x, \bullet)$, which are instances of $F(x, y)$, do not belong to $\{F(\sigma(x), \sigma(y))\mid \sigma(x) \notin \{A\} \notin \{B\}\}$. It is more concise to represent those sets by terms with variables with constraints. In the examples below, we introduce the notation $x \Diamond \{A\}$ as a constraint, which means that $x$ can be instantiated in any term different from $A$ excluding the term $\bullet$. This can be illustrated as follows:

1. $\{t = F(x, y) \text{ such that } t \neq F(A, B)\} = \{F(x, y)\} \cup \{F(x, y)\} \cup \{F(x, y)\} \cup \{F(x, y)\}$

2. $F(B, A) \cup F(B, F(x, y)) \cup F(F(x, y), A) \cup F(F(x, y), F(z, t)) = \{F(x, y)\} \cup \{F(x, y)\} \cup \{F(x, y)\}$

3. $F(F(x, y), A) \cup F(F(x, y), F(z, t)) = \{F(x, y)\} \cup \{F(x, y)\}$

We will now formally introduce the notions of constraint and of constrained term in order to represent such sets of ground terms. Roughly speaking, a constrained term is composed of a term and a constraint which is a predicate over the variables of the term. This predicate restricts the possible instances of the variables in subsequent substitutions as we will see below.

### 2.2. Constraints

**Definition 2.2.** Let $t$ and $t'$ be two terms. The quasi-ordering $\sqsubseteq$ between two terms is defined by: $t \sqsubseteq t'$ if and only if there exists a term $t''$ such that $t \sqsubseteq_0 t'' \sqsubseteq t'$ where $\sqsubseteq_0$ is characterized by the following rules:

- Let $x$ and $y$ be two variables, $F$ a symbol in $\Sigma$ and $t$ a term:
  
  $x \sqsubseteq_0 y$
  
  $F(t_1, \ldots, t_n) \sqsubseteq_0 F(t'_1, \ldots, t'_n)$ if and only if for every $i$ $(1 \leq i \leq n)$, $t_i \sqsubseteq_0 t'_i$
  
  $t \sqsubseteq_0 \bullet$

As we frequently use the negation of the relation $\sqsubseteq$, we give here its characterization:

$t \not\sqsubseteq t'$ if and only if there exists $u \in O(t) \cap O(t')$ such that $t(u) \neq t'(u) \land t(u) \notin X \land t'(u) \neq \bullet$

When the name of variables in a term is not relevant (that is the case for terms in what follows) we will use the symbol $\Omega$ instead of the names of variables.

The least upper bound of two terms with respect to $\sqsubseteq$ can be characterized by the following rules: let $l$ be a term, $x$ a variable and $F$ and $G$ two different symbols in $\Sigma$.

\[
\begin{align*}
x \downarrow t &= l \\
l \downarrow x &= l \\
F(\ldots) \downarrow G(\ldots) &= \bullet
\end{align*}
\]
\[ F(l_1, \ldots, l_n) \cup F(l_1', \ldots, l_n') = F(l_1 \cup l_1', \ldots, l_n \cup l_n') \]

The relation \( \sqsubseteq \) is used to define predicates over terms, that we call constraints. Given a term \( t \) and a set \( L \) of terms, let us define a predicate, denoted \( t \bowtie L \), over substitutions by:

\[
\begin{align*}
    t \bowtie L : & \text{ subst } \longrightarrow \text{ bool} \\
    \sigma & \longmapsto \forall l \in L \ l \not\sqsubseteq \sigma(t)
\end{align*}
\]

Let \( t \) be a term, \( x \) be a variable, \( L \) be a list of terms, \( A \) and \( B \) be constants and \( \sigma \) be a substitution such that \( \sigma(x) = B \). The constraint \( t \bowtie (\{ \Omega \} \cup L) \) is not satisfiable and \( t \bowtie \{ \} \) is a tautology. In what follows \( F \) and \( T \) will denote respectively \( t \bowtie \{ \Omega \} \) and \( t \bowtie \{ \} \). The substitution \( \sigma \) satisfies the predicate \( x \bowtie \{ A \} \) because \( B \not\sqsubseteq A \). The predicate \( B \bowtie \{ A \} \) is equivalent to \( T \) and \( A \bowtie \{ A \} \) is equivalent to \( F \). The constraint \( F(A, B) \bowtie \{ F(A, \bullet) \} \) is satisfiable but not \( F(A, B) \bowtie \{ F(A, \Omega) \} \). Notice that a constraint is satisfiable if and only if it is satisfied by the identity substitution.

These constraints are said to be structural as they are specific to the term structure only as opposed to arbitrary predicates.

**Definition 2.3. (Constraint)** A constraint is recursively defined as either an atomic predicate \( t \bowtie L \) or the disjunction of two constraints or the conjunction of two constraints. In the notation \( t \bowtie L \), we say that \( L \) is a constraint over \( t \).

\[
\text{Constraint : } P ::= \text{ term} \bowtie \text{Set of terms} \\
| \ P \lor P \\
| \ P \land P
\]

The truth value of compound constraints is obtained by standard interpretation of the logical connectives. By using the usual equivalences on connectives or \((\lor)\) and and \((\land)\), a constraint can always be written in disjunctive normal form (as a disjunction of conjunctions of atomic constraints).

**Definition 2.4. (Substitution over a Constraint)** Let \( \sigma \) be a substitution, \( t \bowtie L \) an atomic constraint, \( P_1, P_2 \) two constraints. By definition,

\[
\sigma(t \bowtie L) = \sigma(t) \bowtie L, \quad \sigma(P_1 \lor P_2) = \sigma(P_1) \lor \sigma(P_2) \quad \text{and} \quad \sigma(P_1 \land P_2) = \sigma(P_1) \land \sigma(P_2).
\]

Two constraints \( P \) and \( P' \) are said to be equivalent, denoted \( P \equiv P' \), if and only if, the sets of substitutions satisfying \( P \) and \( P' \) are the same. A constraint \( P \) implies a constraint \( Q \), denoted \( P \Rightarrow Q \), if and only if every substitution satisfying \( P \) also satisfies \( Q \).

**2.3. Constraint simplification**

We can prove using the characterization of \( \sqsubseteq \), that \( F(t_1, \ldots, t_n) \sqsubseteq F(t_1', \ldots, t_n') \) if and only if there exists \( i \) such that \( t_i \not\sqsubseteq t_i' \) and that if \( l \sqsubseteq l' \) then \( t \bowtie \{ l \} \land t \bowtie \{ l' \} \equiv t \bowtie \{ l \} \). From those properties we deduce the following equivalences that associate to each constraint an equivalent normal form where the term in each atomic constraint is a variable:
Let \( t, t_1, \ldots, t_n, t'_1, \ldots, t'_n, l \) be terms, \( L \) be a set of terms, \( x \) be a variable and \( \sigma \) be a substitution.

\[
\begin{align*}
\text{I} \quad & \bullet \{l\} \cup L \equiv \mathcal{F} \\
& F(t_1, \ldots, t_n) \circ \{F(t_1', \ldots, t_n')\} \cup L \equiv \left( \bigvee_{1 \leq i \leq n} t_i \circ \{t'_i\} \right) \land F(t_1, \ldots, t_n) \circ L \\
& \circ \{\} \equiv \mathcal{F} \\
& F(t_1, \ldots, t_n) \circ \{G(\ldots)\} \cup L \equiv F(t_1, \ldots, t_n) \circ L \\
& \circ \{\} \equiv T
\end{align*}
\]

\[
\begin{align*}
\text{II} \quad & x \circ \{\Omega\} \cup L \equiv \mathcal{F} \\
& x \circ \{l, \sigma(i)\} \cup L \equiv x \circ \{l\} \cup L \\
& x \circ \{l\} \cup L \land x \circ \{l'\} \cup L' \equiv x \circ \{l, l'\} \cup L \cup L'
\end{align*}
\]

Notice that rules from group II do not generate any left hand side of an equivalence of group I rules and have no ambiguity with them in the sense of a non trivial superposition of left hand sides. They also transform a formula which is in disjunctive normal form into a new one which is also in disjunctive normal form. Thus the following strategy is valid: first, use rules from group I until they do not apply any more, then put the result in disjunctive normal form and then use rules from group II. The termination of part I is proved using the multiset extension of the subterm ordering over the multiset of the right part of the constraints. The termination of part II is proved using the notherian ordering that compares, for both sides of each equivalence, lexicographically, the multiset of the size of the terms appearing at the right side of the \( \circ \) symbol and the number of \( \land \) symbols. These equivalences, when used as a rewriting system (with rules oriented from left to right), transform a constraint \( P \) into an equivalent constraint in which the atomic components are either \( \mathcal{F} \) or \( T \), or a constraint over variables. Notice that these rules are different from those of disequations of Comon (1988) because we deal explicitly with the symbol \( \bullet \) that represents terms whose evaluation does not terminate. For example \( x \circ \{A\} \lor x \circ \{B\} \neq x \circ \{A\} \lor A \circ \{B\} \) because \( \bullet \) does not satisfy the left part while the right part is equivalent to \( T \).

The restriction of a simplified constraint \( P \) to a given set of variables \( V \) is a constraint \( P' \) obtained by replacing by \( T \) all of the satisfiable atomic constraints of the form \( x \circ L \) such that \( x \not\in V \). For instance the constraint \( (x \circ \{F(\Omega, \Omega)\} \land y \circ \{A\}) \) restricted to \( \{x\} \) is \( (x \circ \{F(\Omega, \Omega)\} \lor T) \) which is equivalent to \( x \circ \{F(\Omega, \Omega)\} \). The restriction of \( (x \circ \{F(\Omega, \Omega)\} \lor y \circ \{A\}) \) to \( \{x\} \) is \( T \). Notice that \( P \Rightarrow P' \) for any restriction \( P' \) of \( P \).

The following lemma shows the relation between the constraints and the prefix ordering.

**Lemma 2.1.** Let \( t, l \) be terms and \( L \) a set of terms.

1. If \( t \subseteq l \) and \( t \not\subseteq l \) then \( t \circ \{l\} \equiv T \).
2. Let \( \sigma, \rho \) two substitutions. \( t \circ \{\sigma(t)\} \Rightarrow t \circ \{\rho(t)\} \) if and only if \( \sigma(t) \subseteq \rho(t) \). More generally, \( \bigwedge_i t \circ \{\sigma_i(t)\} \Rightarrow t \circ \{\rho(t)\} \) if and only if there exists \( i \) such that \( \sigma_i(t) \subseteq \rho(t) \).
3. Either \( t \circ \{l\} \equiv T \) or there exists a substitution \( \sigma \) such that \( t \circ \{l\} \equiv t \circ \{\sigma(t)\} \).
   More generally for any predicate \( t \circ L \) there exists a possibly empty set of terms \( L' = \{l_1, \ldots, l_n\} \) such that \( t \subseteq l_i \) and \( t \circ L \equiv t \circ L' \).

**Proof.**
1. If \( t \subseteq \mathcal{L} \) and \( t \not\subseteq \mathcal{L} \), then there exists an occurrence of \( u \) of both terms such that \( l/u = \ast \) and \( t/u \) is not a variable and is different from \( \ast \). Thus for every substitution \( \eta \) the subterm \( \eta(t)/u \) is also different from \( \ast \) and is not a variable which implies \( l/\eta(t) \).

2. If \( t \otimes \{ \sigma(t) \} \Rightarrow t \otimes \{ \rho(t) \} \), \( \rho \) does not satisfy \( t \otimes \{ \sigma(t) \} \) because it does not satisfy \( t \otimes \{ \rho(t) \} \) and thus \( \sigma(t) \not\subseteq \rho(t) \). Conversely, for every substitution \( \eta \) satisfying \( t \otimes \{ \sigma(t) \} \), \( \sigma(t) \subseteq \rho(t) \). As \( \sigma(t) \not\subseteq \rho(t) \), \( \rho(t) \subseteq \eta(t) \) and we conclude that \( \eta \) satisfies \( t \otimes \{ \rho(t) \} \). The generalization is made by simple manipulation of logical connectives.

3. We remark that \( t \setminus \{ \} \equiv t \setminus \{ \} \cup t \setminus \{ \} \equiv t \setminus \{ \} \) (by induction on the structure of \( l \)). As \( t \subseteq \mathcal{L} \), by part (1) either \( t \setminus \{ \} \equiv T \) or \( t \not\subseteq \mathcal{L} \) that proves the property.

The generalization is made by simple manipulation of logical connectives.

### 2.4. Constrained Terms

**Definition 2.5. (Constrained Term)** Let \( t \) be a term and \( P \) a simplified constraint restricted to the variables of \( t \). A constrained term is defined by:

\[
T ::= \{ t | P \}
\]

Two constrained terms \( T_1 = \{ t_1 | P_1 \} \) and \( T_2 = \{ t_2 | P_2 \} \) are the same, denoted \( T_1 \simeq T_2 \), if and only if \( t_1 = t_2 \) and \( P_1 = P_2 \).

Given any term \( t \) and any constraint \( P \), we write \( \{ t | P \} \) instead of \( \{ t | P' \} \) where \( P' \) is the restriction of a simplified constraint equivalent to \( P \) to the variables of \( t \).

**Example:**

Let \( P = F(x, y) \otimes \{ F(A, B) \} \land y \otimes \{ C \} \land z \otimes \{ A \} \) be a constraint. The restriction of \( P \) to the variables of the term \( t = F(x, y) \) is \( F(x, y) \otimes \{ F(A, B) \} \land y \otimes \{ C \} \) and \( \{ t | P \} \) denotes the term \( \{ F(x, y) | F(x, y) \otimes \{ F(A, B) \} \land y \otimes \{ C \} \} \).

Notice that \( \simeq \) represents the syntactical equality of constrained terms and the symbol \( = \) the equality of sets of terms. To each constrained term \( \{ t | P \} \) is associated the set of ground instances \( \sigma(t) \) of \( t \) such that \( \sigma \) satisfies the constraint \( P \). For instance, \( \{ F(x, y) | F(x, y) \otimes \{ F(A, B) \} \land y \otimes \{ C \} \} = T_1 \cup T_2 \) where \( T_1 = \{ F(x, y) | x \otimes \{ A \} \land y \otimes \{ C \} \} \) and \( T_2 = \{ F(x, y) | y \otimes \{ B, C \} \} \). In general the terms \( T_i \) are not disjoint as in this example where the term \( F(B, A) \) belongs to both \( T_1 \) and \( T_2 \).

The following properties on the sets represented by constrained terms are easy to check:

1. \( \{ t | P \lor Q \} = \{ t | P \} \cup \{ t | Q \} \)
2. \( \{ t | P \land Q \} = \{ t | P \} \cap \{ t | Q \} \)
3. \( \{ t | P \} = \{ \sigma(t) | \sigma(P) \} \) where \( \sigma \) only changes the name of variables.
4. Any constraint \( P \) is equivalent to its disjunctive normal form \( \bigvee_i P_i \) where each \( P_i \) is a conjunction of atomic constraints over variables and thus \( \{ t | P \} = \bigcup_i \{ t | P_i \} \).

A constrained term is a concise representation of a set of ground terms for which its description as an union of instances of pure terms can be large (or infinite if the signature is not finite). For instance \( \{ x | x \otimes \{ A \} \} \) is the union of instances of all the terms of the form \( F(x_1, \ldots, x_n) \) \((n \geq 0)\) such that \( F \neq A \).

In what follows we will call \( t \) the pure part of \( T \) and substitutions over pure terms will be called pure substitutions. The set of occurrences of \( T \) is that of its pure part. The subterms of \( \{ t | P \} \) are of the form \( \{ t' | P \} \) where \( t' \) is a (pure) subterm of \( t \). A linear constrained term is a constrained term whose pure part is linear. When \( P \) is not satisfiable the term \( \{ t | P \} \) represents the empty set of terms that we note \( \emptyset \).
2.5. CONSTRAINED SUBSTITUTIONS

Definition 2.6. We give below, for constrained terms, the definitions of substitution, unifiers, prefix quasi-ordering and equivalence.

Let $\sigma$ be a substitution and $Q$ a constraint. A constrained substitution $\overline{\sigma} = (\sigma, Q)$ is the mapping over constrained terms defined by $\overline{\sigma}([t|P]) = \{\sigma(t)|\sigma(P) \land Q\}$. A substitution $\sigma$ is also a mapping over constrained terms defined by $\sigma([t|P]) = \{\sigma(t)|\sigma(P)\}$.

A constrained substitution $\overline{\sigma} = (\sigma, Q)$ is said to be admissible for $[t|P]$ if and only if $\sigma(P) \land Q$ is satisfiable.

Two constrained terms $T_1$ and $T_2$ are unifiable or compatible, if and only if there exists a constrained substitution $\overline{\sigma}$, such that $\overline{\sigma}(T_1) \simeq \overline{\sigma}(T_2)$. Such a substitution $\overline{\sigma}$ is called a unifier of $T_1$ and $T_2$.

Let $T_1$ and $T_2$ be two constrained terms. The quasi-ordering ($\succeq$) is defined by:

$T_1 \succeq T_2$ if and only if there exists a constrained substitution $\overline{\sigma}$ such that $\overline{\sigma}(T_1) \simeq T_2$.

In this case $T_2$ is an instance of $T_1$.

Notice that when $Q$ is not satisfiable, the substitution $\overline{\sigma} = (\sigma, Q)$ maps every term to the term $\theta$. We compose two constrained substitutions $\overline{\sigma}_1 = (\sigma_1, Q_1)$ and $\overline{\sigma}_2 = (\sigma_2, Q_2)$ as usual and check easily that $\overline{\sigma}_2 \circ \overline{\sigma}_1 = (\sigma_2 \circ \sigma_1, Q_1 \land Q_2(Q_1))$.

We give now a characterization of these concepts in order to compute separately the pure part and the constraint.

Lemma 2.2. Let $T_1 = \{t_1|P_1\}$ and $T_2 = \{t_2|P_2\}$ be two constrained terms.

1. $T_1 \succeq T_2$ if and only if there exists a substitution $\sigma$ such that $\sigma(t_1) = t_2$ and $P_2 \Rightarrow \sigma(P_1)$.

2. The least upper bound of $T_1$ and $T_2$ with respect to $\succeq$ can be computed as: $T_1 \cup T_2 = \{t_1 \cup t_2|\sigma(P_1 \land P_2)\}$ where $\sigma$ is a most general unifier of $t_1$ and $t_2$ such that $\sigma(t_1) = \sigma(t_2) = t_1 \cup t_2$. $\cup$ is not defined when $t_1$ and $t_2$ are not unifiable. The substitution $\overline{\sigma} = (\sigma, \sigma(P_1 \land P_2))$ is a most general unifier for $T_1$ and $T_2$.

3. Let $L$ be a set of partially evaluated terms, $T = \{t|t \triangleright L\}$ and $M = \{m|T\}$ be two constrained terms (in fact the last one is a pure term). $T \succeq M$ if and only if $t \preceq m$ and for each $l \in L$, $m \triangleright l$.

Proof.

1. $T_1 \succeq T_2$ if and only if there exists $\overline{\sigma} = (\sigma, Q)$ such that $\sigma(t_1) = t_2$ and $P_2 \equiv \sigma(P_1) \land Q$. This implies $P_2 \Rightarrow \sigma(P_1)$. Conversely, if there exists $\sigma$ such that $\sigma(t_1) = t_2$ and $P_2 \Rightarrow \sigma(P_1)$, as $P_2 \equiv \sigma(P_1) \land P_2$, $\overline{\sigma} = (\sigma, P_2)$ satisfies $\overline{\sigma}(T_1) \simeq T_2$.

2. Obviously $\{t_1 \cup t_2|\sigma(P_1 \land P_2)\}$ is an upper bound of $T_1$ and $T_2$. Now, let $T = \{t|P\}$ be an upper bound of $T_1$ and $T_2$. By definition of the least upper bound of pure terms, there exists a substitution $\rho$ such that $\rho(t_1 \cup t_2) = t$; as $T$ is an upper bound of $T_1$ and $T_2$, there exist $\sigma_1$ and $\sigma_2$ such that $\sigma_1(t_1) = \sigma_2(t_2) = t$, $P \Rightarrow \sigma_1(P_1)$ and $P \Rightarrow \sigma_2(P_2)$. Consequently $\rho(\sigma(t_1)) = \sigma_1(t_1)$ and $\rho(\sigma(t_2)) = \sigma_2(t_2)$. These equalities hold on the variables of $t_1$ and of $t_2$ that, when applied to the constraints, give $\sigma_1(P_1) = \rho(\sigma(P_1))$ and $\sigma_2(P_2) = \rho(\sigma(P_2))$. In conclusion $P \Rightarrow \rho(\sigma(P_1 \land P_2))$. 


3 By definition, \( T \subseteq M \) if and only if \( t \subseteq m \) and for every \( l \in L \) and for every substitution \( \eta \), \( l \subseteq m(\eta) \). Let \( l \) be an element of \( L \). By hypothesis, \( l(u) \neq \bullet \) for every occurrence \( u \in \mathcal{O}(l) \). In that case, \( l \subseteq m(\eta) \) implies \( m(\eta) \subseteq l \). Conversely, \( m(\eta) \subseteq l \) implies that for every substitution \( \eta \), \( \eta(m) \not\subseteq l \) and \( l \subseteq m(\eta) \).

Example:

A most general unifier \( \overline{\sigma} \) of the terms:

\[
T = \{ G(x, y, z)|x \Diamond \{ H(u) \} \land y \Diamond \{ C \} \land z \Diamond \{ H(H(B)) \} \}
\]

\[
T' = \{ G(P(A), y', H(z'))|y' \Diamond \{ B \} \land z' \Diamond \{ C \} \}
\]

is defined by:

\[
\sigma(x) = P(A)
\]

\[
\sigma(y) = y'
\]

\[
\sigma(z) = H(z')
\]

\[
\overline{\sigma} = (\sigma, y' \Diamond \{ B, C \} \land z' \Diamond \{ H(B), C \})
\]

To each substitution \( \sigma \) we associate the set \( \hat{\mathcal{S}} \) of substitutions corresponding to non-evaluated parts of \( \sigma \).

**Definition 2.7.** Let \( \sigma \) be a substitution. \( \hat{\mathcal{S}} \) is the set of substitutions \( \hat{\sigma} \) defined by \( \hat{\sigma}(x) \) is \( \sigma(x) \) in which at least one non-variable subtree different from \( \bullet \) have been replaced by \( \bullet \).

A substitution \( \sigma \) is considered as the semantics of an operation that defines more precisely a partial term \( T \) and the terms \( \hat{\sigma}(T) \) represent those instances that failed to be evaluated. This is why we call the set \( \{ \hat{\sigma}(T) | \hat{\sigma} \in \hat{\mathcal{S}} \} \), denoted \( \hat{\mathcal{S}}(T) \), the strict part of \( T \) with respect to \( \sigma \).

**Definition 2.8. (Restriction by a Substitution)** Let \( T = \{ t | P \} \) be a constrained term and \( \sigma \) a pure substitution. The restriction of \( T \) by \( \sigma \) is the constrained term:

\[
T|_{\sigma} = \{ t | P \land t \Diamond \{ \sigma(t) \} \}
\]

Notice that restriction is defined only for a pure substitution because there is no constrained term in a constraint.

Example:

For the term \( T = \{ F(x, y)|x \Diamond \{ H(A) \} \} \) and the substitution \( \sigma \) defined by \( \sigma(x) = H(z) \), \( \sigma(y) = A \), we obtain:

\[
\sigma(T) = \{ F(H(z), A)|z \Diamond \{ A \} \}
\]

\[
T|_{\sigma} = \{ F(x, y)|F(x, y) \Diamond \{ F(H(z), A) \} \land x \Diamond \{ H(A) \} \}
\]

\[
\hat{\mathcal{S}}(T) = \{ F(H(z), \bullet)|z \Diamond \{ A \} \} \cup \{ F(\bullet, A)|\mathcal{F} \} \cup \{ F(\bullet, \bullet)|\mathcal{F} \}
\]

Notice that in general the terms of \( T|_{\sigma} \) may have common instances like the term \( F(A, B) \) in this example.
Lemma 2.3. Let $T = \{ t|P \}$ be a constrained term, $\sigma$ be a pure substitution and $\text{Id}$ the identity substitution. $T|_{\text{Id}} = \emptyset$ and the sets $\sigma(T)$, $T|_{\sigma}$ and $\dot{\sigma}(T)$ are pairwise disjoint. The set of instances of $T$ is the union of instances of $\sigma(T)$, $T|_{\sigma}$ and $\dot{\sigma}(T)$.

Proof. By definition if $\eta(t)$ is an element of $\sigma(T)$ then $\sigma(t) \subseteq \eta(t)$ and thus $\eta(t) \circ \{\sigma(t)\}$ is not satisfiable. Furthermore, let $\dot{\sigma} \in \dot{\sigma}$, by definition of $\dot{\sigma}(t)$, $\sigma(t)$ and $\dot{\sigma}(t)$ have no common instances and $\sigma(t) \subseteq \dot{\sigma}(t)$; as a consequence, the sets $\sigma(T)$, $T|_{\sigma}$ and $\dot{\sigma}(T)$ are pairwise disjoint. The last property is proved by cases on the definition of $\subseteq$.

The set $\sigma(T) \cup T|_{\sigma}$ is the computable part of $T$ with respect to $\sigma$ and the complement is the strict set $\{ \dot{\sigma}(T) \mid \dot{\sigma} \in \dot{\sigma} \}$. The proposition below will be useful in the definition of the decomposition of a term.

Proposition 2.1. For every term $T = \{ t|P \}$ and pure substitution $\sigma$, $T$ is the greatest lower bound of $\sigma(T)$, $T|_{\sigma}$ and all the $\dot{\sigma}(T)$ with $\dot{\sigma} \in \dot{\sigma}$.

Proof. Let $T = \{ t|L \}$. It is easy to prove that $T$ is a prefix of $\sigma(T)$, $T|_{\sigma}$ and all the $\dot{\sigma}(T)$ with $\dot{\sigma} \in \dot{\sigma}$. Furthermore, if there exists a common prefix $T_0$ of these terms that is not a prefix of $T$, $T \cup T_0$ is also a common prefix. Now let $T' = \{ t'|P' \}$ be a lower bound of $\sigma(T)$, $T|_{\sigma}$ and all the $\dot{\sigma}(T)$ such that $T \subseteq T'$. By hypothesis on the pure terms, $t \leq t' \leq t$ and thus $t = t'$. The hypothesis on the constraints are:

$$t \circ L \wedge t \circ \{\sigma(t)\} \Rightarrow P' \Rightarrow t \circ L$$
$$\sigma(t) \circ L \Rightarrow \sigma(P')$$
$$\dot{\sigma}(t) \circ L \Rightarrow \dot{\sigma}(P')$$

The first property implies the existence of a constraint $P''$ such that $P' \equiv t \circ L \wedge P''$ and $t \circ \{\sigma(t)\} \Rightarrow P''$. By lemma 2.1, either $P'' \equiv T$ which implies $P' \equiv P$ or $P'' \equiv \wedge_i t \circ \{\rho_i(t)\}$. Consequently the following implications are satisfied:

$$t \circ L \wedge t \circ \{\sigma(t)\} \Rightarrow t \circ L \wedge \bigwedge_i t \circ \{\rho_i(t)\} \tag{2.1}$$

$$\sigma(t) \circ L \Rightarrow \sigma(t) \circ L \wedge \bigwedge_i \sigma(t) \circ \{\rho_i(t)\} \tag{2.2}$$

$$\dot{\sigma}(t) \circ L \Rightarrow \dot{\sigma}(t) \circ L \wedge \bigwedge_i \dot{\sigma}(t) \circ \{\rho_i(t)\} \tag{2.3}$$

Let $\eta$ be a substitution satisfying $t \circ L$. If $\eta$ satisfies $t \circ \{\sigma(t)\}$, by (1), it also satisfies $P''$. Let us suppose now that $\eta$ does not satisfy $t \circ \{\sigma(t)\}$; either $\sigma(t) \leq \eta(t)$ and, by (2), $\eta$ satisfies $P''$, or there exists $t'' = \dot{\sigma}(t)$ with $\dot{\sigma} \in \dot{\sigma}$, such that $\sigma(t) \subseteq t'' \leq \eta(t)$ and, by (3), $\eta$ satisfies $P''$. As a conclusion of the three cases, $P \Rightarrow P'$ and thus, $T = T'$.

In general, the greatest lower bound of a set of terms does not exist. For instance, the common prefixes of $\{ A|T \}$ and $\{ B|T \}$ (with $A \neq B$) are of the form $\{ x|x \circ L \}$, but for each prefix there is a constant $C$ not belonging to $L$ and different from $A$ and $B$ such that $\{ x|x \circ L \cup \{ C \} \}$ is a prefix of both terms less general than $\{ x|x \circ L \}$. In what follows, we associate to any pair constrained terms a principal prefix of these two terms,
which will be used in section 4.1, for the analysis of the sequentiality properties of sets of constrained terms.

**Definition 2.9.** Let \( T_1 = \{ t_1 | P_1 \} \) and \( T_2 = \{ t_2 | P_2 \} \) be two constrained terms and \( t = t_1 \cap t_2 \). Let \( \sigma_i(t) = t_i \), and \( \sigma_i' \) be the substitutions defined by \( \sigma_i'(x') = x \) if \( \sigma_i(x) = x' \) and \( \sigma_i'(x') = x \) otherwise. The principal prefix of \( T_1 \) and \( T_2 \), that we denote \( T_1 \cap T_2 \) is, by definition, the term \( \{ t | \sigma_1'(P_1) \lor \sigma_2'(P_2) \} \). The extension of the principal prefix to finite sets of terms if made in the natural way.

**Example:**

As we have already seen, the glb of \( \{ A | T \} \) and \( \{ B | T \} \) does not exist, but \( \{ A | T \} \cap \{ B | T \} = \{ x | T \} \). Furthermore, even if the glb exists, it might not be equal to the principal prefix; for example, the glb of \( \{ A | T \} \) and \( \{ x|x^\otimes \{ A \} \} \) is \( \{ x|x^\otimes \{ \} \} \) and \( \{ A | T \} \cap \{ x|x^\otimes \{ A \} \} = \{ x | T \} \).

### 3. Term Decomposition

We want to split, following an ordered list \( S \) of partially evaluated terms, named patterns, the set of all terms represented by \( T \) into a set of disjoint ones. The decomposition of \( T \) with respect to \( S \) consists in splitting the set associated to \( T \) into subsets such that each subset contains instances of at most one element of \( S \). For example, let \( S = \{ F(A,H(\Omega)), F(A,\Omega) \} \) and \( T = \{ F(x,y)|T \} \). The part of \( T \) that can be evaluated is the disjoint union of \( T_1, T_2 \) and \( T_3 \) where \( T_1 = \{ F(A,H(x))|T \} \), \( T_2 = \{ F(A,y)|y^\otimes \{ H(\Omega) \} \} \) and \( T_3 = \{ F(x,y)|x^\otimes \{ A \} \} \). The study of these constrained terms and decomposition originates in Puel’s thesis (1987).

#### 3.1. Decomposition

**Definition 3.1. (Decomposition w.r.t. a Pattern)** Let \( T = \{ t | P \} \) be a constrained term, and \( s \) a pattern. If \( \sigma \) is a most general unifier of \( t \) and \( s \), then \( \sigma(T) \) is called a decomposition of \( T \) w.r.t. \( s \). \( \sigma(T) \) is also written \( \text{compat}(T, s) \).

With this definition, \( \text{compat}(T, \Omega) \) and \( T \) represent the same set of terms.

**Definition 3.2. (Decomposition w.r.t. an Ordered Set of Patterns)** The decomposition of a constrained term \( T = \{ t | P \} \) w.r.t. an ordered set of patterns \( S = \{ s_1, \ldots, s_n \} \), Decom \( (T, S) \), is recursively defined as:

\[
\begin{align*}
\text{Decomp}(\emptyset, S) &= \emptyset \\
\text{Decomp}(T, \emptyset) &= \emptyset \\
\text{Decomp}(T, \{ s_1, \ldots, s_n \}) &= \begin{cases} \\
\text{Decomp}(T, \{ s_2, \ldots, s_n \}) \text{ if } t \text{ and } s_1 \text{ are incompatible} \\
\{ \text{compat}(T, s_1) \} \cup \text{Decomp}(T|\sigma_1, \{ s_2, \ldots, s_n \}) \text{ otherwise.} \\
\end{cases}
\end{align*}
\]

Notice that \( \text{Decomp} \) stops when a previous pattern matches \( T \) because the restriction of a term by the identical substitution is the empty set. The instances of \( T \) that are not instances of an element of \( \text{Decomp}(T, S \cup \{ \Omega \}) \) are those for which there is no way to decide if they are instances of one of the elements of \( S \).
DEFINITION 3.3. (Strict Set w.r.t. an Ordered Set of Patterns) Let \( T = \{ t | P \} \) be a term, \( S \) a set of patterns, \( \{ M_1, \ldots, M_k \} = \text{Decomp}(T, S) \) with \( M_i = \{ m_i | P_i \} \) and \( \sigma_i \) a mgu of \( t \) and \( m_i \). The strict set of \( T \) with respect to \( S \) is the set \( \bigcup_{1 \leq i \leq k} \{ \sigma_i(T) \mid \sigma_i \in \mathcal{O}_i \} \). The strict set of \( S \) is the strict set of a variable with respect to \( S \).

PROPOSITION 3.1. Let \( T \) be a constrained term and \( S = \{ s_1, \ldots, s_n \} \) a set of patterns. Then \( T \) is the greatest lower bound of \( \text{Decomp}(T, \{ s_1, \ldots, s_n, \Omega \}) \) and the strict set of \( T \) with respect to \( S \).

PROOF. This property is a consequence of the definition of \( \text{Decomp} \) and of Proposition 2.1. Notice that, in the decomposition of a term, \( S \) is used as an ordered list and thus \( \Omega \) is the last element of this list. This decomposition is a partition of the part of \( T \) that can be evaluated.

3.2. DECOMPOSITION PROCEDURE

Let \( T = \{ t | P \} \) be a constrained term and \( S = \{ s_1, \ldots, s_n \} \) a set of patterns.

Initialization step
\[ \theta_1 \leftarrow T \]

Current step
\[ (\tau_i, \theta_{i+1}) \leftarrow (\sigma_i(\theta_i), \theta_i, |\sigma_i|) \text{ if } \theta_i \text{ and } s_i \text{ are unifiable with m.g.u. } \sigma_i \]
\[ \leftarrow (\emptyset, \theta_i) \text{ if not.} \]

Then \( \text{Decomp}(T, S) = \{ \tau_1, \ldots, \tau_n \} \). Notice that some \( \tau_i \) can be an empty set.

The following lemmas will be used for the pattern matching:

LEMMA 3.1. Let \( S = \{ s_1, \ldots, s_n \} \) be a set of patterns and \( \{ \tau_1, \ldots, \tau_n \} \) the decomposition of a variable \( x \) by the set \( S \). For every \( i \), \( \tau_i = \{ s_i | \bigwedge_{j < i} s_i \bigodot \{ s_j \} \} \) and for every \( i \) and \( j \neq i \), \( \tau_i \cap \tau_j = \emptyset \).

PROOF. The first part is proved by induction over \( i \): As \( \theta_1 = x \), \( \tau_1 = \{ s_1 | T \} \) and \( \theta_2 = \{ x | x \bigodot \{ s_1 \} \} \). Now suppose that \( \tau_i = \{ s_i | \bigwedge_{j < i} s_i \bigodot \{ s_j \} \} \) and \( \theta_{i+1} = \{ x | \bigwedge_{j < i} x \bigodot \{ s_j \} \} \). Then \( \tau_{i+1} = \{ s_{i+1} | \bigwedge_{j < i} s_{i+1} \bigodot \{ s_j \} \} \) and \( \theta_{i+2} = \{ x | \bigwedge_{j < i} x \bigodot \{ s_j \} \bigwedge x \bigodot \{ s_{i+1} \} \} \), that achieves the proof. The second part is a direct consequence of Lemma 2.3.

Example:

When we take the set of patterns:
\[ F(x, B), F(P(z), y), F(x, y), F(H(u), y) \]
The decomposition of the term \( T = \{ F(x, y) | T \} \) gives the following set of constrained terms (in the examples we write \( x \bigodot \{ F \} \) instead of \( x \bigodot \{ F(\Omega, \ldots, \Omega) \} \)):
\[ F(x, B), \{ F(P(z), y) | y \bigodot \{ B \} \}, \{ F(x, y) | x \bigodot \{ P \} \bigwedge y \bigodot \{ B \} \} \]
If we decompose the term \( T = \{ v | T \} \) where \( v \) is a variable, the result is:
\[ F(x, B), \{ F(P(z), y) | y \bigodot \{ B \} \}, \{ F(x, y) | x \bigodot \{ P \} \bigwedge y \bigodot \{ B \} \}, \{ v | v \bigodot \{ F \} \} \]
The strict set of $T$ with respect to the patterns is:

$$ \bullet, F(x, \bullet), \{F(\bullet, y) | y \in \{B\} \} $$

Notice that in this example redundant patterns disappear. As we decompose a term which is a variable without constraints, the union of the new set of constrained patterns and of the strict set of $T$ represent the set of all the terms.

The following lemma allows us to use the decomposition of a variable to compute the decomposition of any term. Afterwards, the decomposition of a term is a unification with these new patterns as was illustrated with the previous example.

**Lemma 3.2.** Let $S = \{s_1, \ldots, s_n\}$ be a set of patterns, $\{\tau_1, \ldots, \tau_n\} = \text{Decomp}(x, S)$ and $T = \{t|P\}$ a constrained term. Then $\text{Decomp}(T, S) = \{\overline{\sigma}(\tau_i) | 1 \leq i \leq n\}$ where, for every $i$, $\overline{\sigma}_i$ is a most general unifier of $\tau_i$ and $T$.

**Proof.** Let $\sigma_i$ be a most general unifier of $t$ and $s_i$ for every $i$ ($1 \leq i \leq n$). By definition $\text{Decomp}(T, S) = \bigcup_{1 \leq i \leq n} \{\sigma_i(t)\sigma_i(P) \wedge (\land_{1 \leq j \leq i-1} \sigma_i(t) \circ \sigma_j(t))\}$. As for each $i$, ($1 \leq i \leq n$) $\overline{\sigma}_i(\tau_i) = \{\sigma_i(t)\sigma_i(P) \wedge (\land_{1 \leq j \leq i-1} \sigma_i(s_j) \circ \sigma_j(s_j))\}$, in order to prove the lemma, it is sufficient to prove the equivalence of the constraints $\sigma_i(t) \circ \sigma_j(t)$ and $\sigma_i(s_j) \circ \sigma_j(s_j)$ for every integer $i, j$ such that $j < i$. $\sigma_i(t) \circ \sigma_j(t) \equiv \sigma_i(s_j) \circ \sigma_j(s_j)$ if and only if for all substitution $\eta$, $\sigma_j(t) \not\subseteq \eta \circ \sigma_i(t)$ if and only if $s_j \not\subseteq \eta \circ \sigma_i(s_i)$. The if part is clear because $\sigma_j(t) = \sigma_j(s_j)$. Let us suppose that there exists $\eta$ such that $\sigma_j(t) \not\subseteq \eta \circ \sigma_i(t)$ and $s_j \not\subseteq \eta \circ \sigma_i(s_i)$ since $s_j$ and $t$ are unifiable with the m.g.u. $\sigma_i$, if $s_j \not\subseteq \eta \circ \sigma_i(t)$ there exists a substitution $\rho_j$ such that $\rho_j \circ \sigma_j(t) = \eta \circ \sigma_i(t)$. Thus $\sigma_j(t) \not\subseteq \eta \circ \sigma_i(t)$ that is a contradiction. Otherwise, let $U = \{u \in O(s_j) | s_j/u \neq \bullet$ and $\eta \circ \sigma_i(t)/u = \bullet\}$. Notice that $U \cap O(t) = \emptyset$ because $t$ and $s_j$ are unifiable. Thus $s_j \not\subseteq \eta \circ \sigma_i(t)[u \leftarrow s_j/u | u \in U]$ which is an instance of $t$. Then, we conclude as in the previous case.

### 3.3. DECOMPOSITION NORMALIZATION

As empty sets are meaningless in a decomposition, from now on, we suppose that all the elements of a decomposition are non empty. It is useful to transform constraints into an easily readable shape and we propose now a normalization algorithm. Its first step is to split these terms into terms with constraints with only one function symbol. Its third step is to normalize the constraint associated to a variable appearing at the same occurrence in two trees in the decomposition, in order to get the same constraint at common occurrences.

**Definition 3.4.** A decomposition $\{M_1, \ldots, M_n\}$ is in normal form if and only if:

1. In each atomic constraint $t \circ \{l_1, \ldots, l_n\}$, each $l_i$ has only one function symbol and $t$ is a variable.
2. The constraint of each term is a conjunction of atomic predicates.
3. If there exist $i \neq j$ and an occurrence $u$ of $M_i$ and $M_j$ such that, for every $u' < \text{prefix } u$, $M_i$ and $M_j$ have the same symbol at $u'$, $M_i/u = \{x|x \circ L_x\}$, $M_j/u = \{y|y \circ L_y\}$ and $L_x$ and $L_y \neq \emptyset$ then, $L_x = L_y$.

**Normalization Algorithm** Let $\{M_1, \ldots, M_n\}$ be a set of simplified constrained terms.
If \( M_i = \{ m | (x \diamond \{ C(m_1^i, \ldots, m_n^i) \} \cup L) \wedge P \} \), \( L \) does not contain a term with \( C \) as head symbol and there exists \( m_i^j \) non-variable:

\[
M_i \Rightarrow \{ m | (x \diamond \{ C(\Omega, \ldots, \Omega) \} \cup L) \wedge P \} \cup \{ m | x \leftarrow C(x_1, \ldots, x_n) \} | (\Lambda j \in J \bigvee 1 \leq i \leq n \bigwedge x_i \diamond \{ m_i^j \}) \wedge P \}
\]

If \( M_i = \{ m | P_1 \lor P_2 \} \):

\[
M_i \Rightarrow \{ m | P_1 \} \cup \{ m | P_2 \}
\]

If there exist \( M_i = \{ m | x \diamond \{ C(\Omega, \ldots, \Omega) \} \cup L_x \} \wedge P \) and \( M_j \) satisfying properties 1. and 2. and hypothesis of 3. above with \( C(\Omega, \ldots, \Omega) \in L_y - L_x \):

\[
M_i \Rightarrow \{ m | (x \diamond \{ C(\Omega, \ldots, \Omega) \} \cup L_x) \wedge P \} \cup \{ m | x \leftarrow C(x_1, \ldots, x_n) \} | P \}
\]

It is easy to check that a decomposition has the same set of ground instances as its corresponding normalized decomposition. By definition of a constrained term, after each step of normalization, new constraints are simplified. Notice that simplification does not generate new redexes (left hand side of a rule) for normalization. As rule 3. does not generate any redex of rules 1. and 2. and has no ambiguity with them in the sense of a non trivial superposition of left hand sides, the strategy in which rules 1. and 2. are used first until they do not apply any more and then rule 3. is used is valid. The normalization algorithm terminates: the ordering used to prove termination of 1. and 2. is the multiset extension of the multiset extension of the subterm ordering where the measure of \( \bigvee i \in I \{ m_i | P_i \} \) is the multiset of the multiset of the right parts of \( P_i \) (restricted to the variables of \( m_i \)). Let \( \{ M_1, \ldots, M_n \} \) be a set of normalized with respect to rules 1. and 2. constrained terms. Let \( U \) be the set of occurrences where the hypothesis of property 3. is satisfied in one of the constrained terms \( M_i \) and

\[
L_u = \bigcup_{M_i | u = x \diamond L_x^i}
\]

Notice that for each \( u \in U \), \( L_u \) remains unchanged after an application of rule 3. The measure associated to a constrained term \( M_i \) is the multiset \( \{ L_u - L_x^i | u \in U \text{ and } M_i | u = x \diamond L_x^i \} \) and the measure associated to a set \( \{ M_1, \ldots, M_n \} \) is the multiset of the measure of each element. Thus, the ordering used to prove the termination of rule 3. is the multiset extension of the multiset extension of the set inclusion.

Example:

The decomposition of the following set of patterns \( F(G(A), B), F(y, B), F(C, z) \), \( x \) gives as result:

\[
F(G(A), B) \quad \{ F(y, B) | y \diamond \{ G(A) \} \} \quad \{ F(C, z) | z \diamond \{ B \} \} \quad \{ x | x \diamond \{ F(\Omega, B), F(C, \Omega) \} \}
\]

We notice that the constraints over \( x \) and \( y \) have to be normalized. The first step of normalization transforms the patterns \( \{ F(y, B) | y \diamond \{ G(A) \} \} \) and \( \{ x | x \diamond \{ F(\Omega, B), F(C, \Omega) \} \} \) into these new patterns:

\[
\{ F(y, B) | y \diamond \{ G \} \} \quad \{ F(G(t), B) | t \diamond \{ A \} \} \\
\{ x | x \diamond \{ F \} \} \quad \{ F(y, z) | y \diamond \{ C \} \wedge z \diamond \{ B \} \}
\]
The third step gives the resulting set:

\[
\begin{align*}
F(G(A), B) & \quad \{F(G(t), B)|t \odot \{A\}\} \quad F(C, B) \\
\{F(y, B)|y \odot \{G, C\}\} & \quad \{F(C, z)|z \odot \{B\}\} \quad \{F(G(t), z)|z \odot \{B\}\} \\
\{F(y, z)||y \odot \{G, C\} \land z \odot \{B\}\} & \quad \{z|x \odot \{F\}\}
\end{align*}
\]

4. Pattern Matching

In this section we use constrained terms to reason about pattern matching over pure terms.

**Definition 4.1.** A set of patterns \( \Pi \) is complete if and only if every ground term which is not an instance of an element of \( \Pi \) is an instance of an element of the strict set of \( \Pi \).

Let \( \Pi = \{m_1, \ldots, m_n\} \) be a set of patterns. The simplest matching predicate over \( \Pi \) is defined by \( \text{match}_\Pi(t) = \text{True} \) if and only if there exists \( m_i \in \Pi \) such that \( m_i \preceq t \) where \( t \) is a pure term. This predicate does not take account of any priority over \( \Pi \) and is not suitable for pattern matching over partially evaluated terms. Laville (1988a) defines a matching predicate over pure terms which takes care of the ordering.

**Definition 4.2.** Let \( \Pi = \{m_1, \ldots, m_n\} \) be a set of patterns ordered by priority, and \( t \) be a pure term. \( \text{Match}_\Pi(t) = \text{True} \) if and only if there exists \( i(1 \leq i \leq n) \) such that \( m_i \preceq t \) and for every \( j < i \), \( t \not\preceq m_j \).

The priority on patterns is necessary to force the matching with a chosen pattern when several patterns are compatible.

With the concept of constrained terms, we replace priority by constraints. We transform the ordered set of patterns into an unordered set of constrained ones using the decomposition algorithm and without losing generality, we work on the instances of terms that are instances of the patterns.

Let \( \Pi = \{m_1, \ldots, m_n, \Omega\} \) be an ordered set of patterns and \( x \) be a variable. The decomposition algorithm computes \( \Pi' = \text{Decomp}(x, \Pi) \) which is the set of constrained patterns \( M'_i = \{m_i|\land_{j<i} m_j \odot \{m_j\}\} \). Remember that \( M'_i \cap M'_j = \emptyset \) for \( i \neq j \) and that the redundant patterns, represented by empty constrained terms, are eliminated.

**Definition 4.3.** (Pattern Matching) Let \( \Pi' = \{M_1, \ldots, M_n\} \) be a set of disjoint constrained patterns, and \( T \) be a constrained term. \( \text{RMatch}_\Pi(T) = \text{True} \) if and only if there exists \( i \) \((1 \leq i \leq n)\) such that \( M_i \preceq T \).

Notice that the relation \( \preceq \) over constrained terms is transitive and the predicate \( \text{RMatch}_\Pi \) is monotonic with the ordering False < True.

In the following theorem, we prove that the predicate \( \text{RMatch}_\Pi \), which only uses the prefix ordering, is as powerful as \( \text{Match}_\Pi \) which uses the prefix ordering and incompatibility tests.

**Theorem 4.1.** Let \( \Pi = \{m_1, \ldots, m_n\} \) be an ordered set of patterns and \( \Pi' = \{M'_1, \ldots, M'_n\} \) be the decomposition of \( x \) by \( \Pi \) where each \( M'_i = \{m_i|\land_{j<i} m_j \odot \{m_j\}\} \). \( \Pi' \) is the set of terms \( M' \) such that for every term \( T, T \preceq M' \) implies \( \text{RMatch}_\Pi(T) = \text{False} \) and such
that $\text{RMatch}_{\Pi'}(T) = \text{True}$ implies that there exists $M' \in \Pi'$ such that $M' \preceq T$ ($\Pi'$ is the set of minimal generators of the term satisfying the predicate $\text{RMatch}_{\Pi'}$). Furthermore for every pure term $t$:

$$\text{Match}_{\Pi}(t) \equiv \text{RMatch}_{\Pi'}(\{t|T\})$$

**Proof.** By definition $\text{RMatch}_{\Pi'}(T) = \text{True}$ if and only if there exists $M' \in \Pi'$ such that $M' \preceq T$. Conversely each non-empty $M' \in \Pi'$ matches its ground instances. Furthermore, as the elements of $\Pi'$ are pairwise incompatible $T \preceq M'$ implies $\text{RMatch}_{\Pi'}(T) = \text{True}$. By definition and lemma 2.2, $\text{RMatch}_{\Pi'}(\{t|T\}) = \text{True}$ if and only if there exists $M'_j \in \Pi'$ such that $m_i \preceq t$ and for every $j < i$, if $m_j$ that is the definition of the predicate $\text{Match}_{\Pi}$ over pure terms.

Notice that $\Pi'$ generates the set of pure terms satisfying $\text{Match}_{\Pi}$ and gives a set of minimal generators more compact than the minimal set of generators described in Lavill'le's thesis (1988a, page 44). For instance, the decomposition of the patterns $F(A, B, z)$, $F(A, A, z)$, $F(x, y, C)$ and $F(x, y, z)$ is the set:

$$F(A, B, z), \{F(x, y, C)|F(x, y, C)\otimes\{F(A, B, \Omega), F(A, A, \Omega)\}\},$$

$$F(A, A, z), \{F(x, y, z)|F(x, y, z)\otimes\{F(A, B, \Omega), F(A, A, \Omega)\}\}$$

Normalization gives the following set:

$$F(A, B, z), \{F(x, y, C)|x\otimes\{A\}\}, \{F(x, y, z)|x\otimes\{A\}, z\otimes\{C\}\},$$

$$F(A, A, z), \{F(x, y, C)|y\otimes\{A, B\}\}, \{F(x, y, z)|y\otimes\{A, B\}, z\otimes\{C\}\}$$

There are several algorithms to check the match of a term by a given set of patterns, named pattern matching algorithms. We will use Search Trees to represent these algorithms. The internal nodes (i.e. different from a leave) of these trees have as labels, pairs of a constrained term and an occurrence of a variable in it. The label of the root is a variable and on each branch the labels are terms more and more instance. The sons of a node with label $T, u$ have as term, either $T[u \leftarrow T']$ where $T'$ contains at most one function symbol and is a prefix of a pattern compatible with $T$ or $T[u \leftarrow \{x|z\otimes L\}]$ where $x$ is the pure part of $T/u$ and $L$ contains the constraint over $x$ in $T$. The leaves of the tree are compatible with exactly one pattern (the occurrence is of no use). The only freedom in the construction is the choice of the occurrence used to develop the subtrees.

For instance, if the choice of the occurrence is always the leftmost variable that leads to the pattern having priority, as it is the choice of many compilers for functional languages, the search tree associated to the patterns $F(A, B)$, $F(y, B)$ and $x$ is:

```
   z[x]
    /\     /
   F(\)  F(y, z)[y]
    \  /\    \ {x|z\otimes\{F(\Omega, \Omega)\})
    /  /  /
   F(A, x)[z]  F(y, z)[y\otimes\{A\}][z]  \{F(y, z)[y\otimes\{A\} z\otimes\{B\}\}
    \  \ /  /  /
    \  /  /  /  /
  A B \{F(A, x)[z\otimes\{B\}\} F(y, B)[y\otimes\{A\}]\{F(y, x)[y\otimes\{A\} z\otimes\{B\}\}
```

The strict set of the match is $\bullet, F(y, \bullet)$ and $F(\bullet, B)$. This algorithm will not give a result for the term $F(\bullet, A)$, which does not belong to the strict set of the match.
The set of instances of the patterns and the set of instances of the elements of the strict set are disjoint, and thus, every algorithm fails to recognize the elements of the strict set. It is not easy to express an optimality criterion for pattern matching algorithms associated to search trees that corresponds to the intuition. Clearly, an optimal algorithm should associate, to each ground term $T$ a corresponding pattern $M$ such that $M \preceq T$ in a minimal number of steps. Nevertheless, not every ground term has a corresponding pattern and not all of the algorithms recognize all of the instances of the patterns. We will prove now, that an algorithm that minimizes for every ground term the number of steps necessary to recognize it, also maximizes the set of ground instances recognized.

**Definition 4.4.** A pattern matching algorithm is optimal if and only if for every ground term $T$, the set of occurrences of the term evaluated by the algorithm is minimal for inclusion ordering.

**Proposition 4.1.** Let $\Pi$ be complete set of patterns. A pattern matching algorithm with respect to $\Pi$ is optimal if and only if it recognizes every ground instance of the patterns (i.e. if and only if it fails to recognize exactly the elements of the strict set).

**Proof.** The ground terms not recognized by an optimal algorithm $A$ are exactly the elements of the strict set, as every instance of a pattern can be recognized by an algorithm which only evaluates the non variable occurrences of the pattern. Conversely, if an algorithm $A$ that only fails to recognize the elements of the strict set were not optimal, there would exist a term $T$ such that $A$ would evaluate an occurrence $u$ while another algorithm would not. This new algorithm would recognize the term $T[u \leftarrow \cdot]$ which consequently would not belong to the strict set. As the algorithm $A$ would fail to recognize this term, there would be a contradiction.

### 4.1. Sequentiality

We say that a pattern matching problem is sequential when it can be computed without looking ahead on a sequential machine. In this section we describe how to decide if a match problem is sequential and in such case, how to build the search tree associated to it. This section adapts the definitions and proofs of Huet and Lévy (1979) to the case of constrained terms.

**Definition 4.5.** (Index, Sequential)

Let $\mathcal{P}$ be a predicate on constrained terms monotonic with respect to $\preceq$, (with the truth values domain ordered as $\text{False} < \text{True}$).

An occurrence $u$ of $T$ is said to be an index of $\mathcal{P}$ in $T$ if and only if

1. $T/u = \{x \mid \not\exists L\}$
2. For every $M \succeq T$, $\mathcal{P}(M) = \text{True}$ implies $(M/u) \not\in (T/u)$.

Then $\mathcal{P}$ is sequential at $T$ if and only if whenever $\mathcal{P}(T) = \text{False}$ and there exists $M \succeq T$ such that $\mathcal{P}(M) = \text{True}$, it follows that there exists an index of $\mathcal{P}$ in $T$.

Finally $\mathcal{P}$ is said to be sequential if and only if it is sequential at every constrained term.
As the predicate $\text{RMatch}_\Pi$ is monotonic, we look for its sequentiality at every term, called the sequentiality of $\Pi$. The set $\text{Dir}_\Pi(T)$ of the indexes of $\text{RMatch}_\Pi$ in $T$ is called the set of directions from $T$ to $\Pi$.

**Lemma 4.1.** Let $T$ be a constrained term and $\Pi$ a set of disjoint constrained patterns. $u \in \text{Dir}_\Pi(T)$ if and only if $T/u = \{x|x \circ L\}$ and, for all $M \in \Pi$ such that $M \uparrow T$, one has $u \in \mathcal{O}(M)$ and $M/u \not\leq T/u$.

**Proof.** Let $u \in \text{Dir}_\Pi(T)$ and $M \in \Pi$ such that $M \uparrow T$. Suppose that $u \notin \mathcal{O}(M)$. There exists a proper prefix $u'$ of $u$ such that $u = u'w$ with $w \neq \varepsilon$, $M/u' = \{y|y \circ L\}$ and $M/u' \not\leq T/u'$. Let $\bar{\sigma}$ be the substitution such that $\bar{\sigma}(y) = T/u'$. The terms $\bar{\sigma}(M)$ and $T$ are compatible and thus, their least upper bound $T''$ is such that $T''/u' = T/u'$ which contradicts the hypothesis. Knowing that $u \in \mathcal{O}(M)$, obviously $M/u \not\leq T/u$. Conversely, if there is a term $T' \geq T$ such that $\text{RMatch}_\Pi(T') = \text{True}$, there is a pattern $M \in \Pi$ compatible with $T$. Thus $M/u \not\leq T/u$ that implies $T''/u \not\leq T/u$ and the equivalence is clear.

**Remark:**

This lemma gives a simple characterization of directions. By normalization, a pattern $M \in \Pi$ is split in several terms $M_1, \ldots, M_n$ which may be compatible. As a consequence of the simplification rules, $M/u \not\leq \{x|x \circ L\}$ if and only if each $M_i/u \not\leq \{x|x \circ L\}$ and thus the set of directions $\text{Dir}_\Pi(T)$ is the set of directions from $T$ to the normalization of $\Pi$.

**Lemma 4.2.** Let $\Pi = \{\{m_i \mid \bigwedge_{j<i} m_i \circ \{m_j\}\}|1 \leq i \leq n\}$ be the decomposition of a variable w.r.t. a complete set of patterns, $T = \{t|P\}$ be a term and $\Pi' \subseteq \Pi$ be the set of patterns compatible with $T$. Either $\Pi'$ is an empty set or $\cap \Pi' \not\leq T$.

**Proof.** Let $\Pi' = \{\{m_i \mid \bigwedge_{j<i} m_i \circ \{m_j\}\}|1 \leq i_1 < i_2 < \ldots < i_p \leq n\}$. By definition, $\Pi' = \{\{m_i \mid \bigwedge_{j<i} m_i \circ \{m_j\}\}|1 \leq i_1 < i_2 < \ldots < i_p \leq n\}$. As a property of the restriction, $\cap \Pi' \not\leq \{\{m_i \mid \bigwedge_{j<i} m_i \circ \{m_j\}\} = \{\{m_i \mid \bigwedge_{j<i} m_i \circ \{m_j\}\}$. Thanks to lemma 2.2, as $t_j m_j$ for $j < i_1$, we only need to prove that $\cap m_{i_1} \preceq t$. In order to achieve this, we prove that there exists a compatible pattern such that its pure part is a prefix of $t$. If the last compatible pattern is of the form $\{x|P\}$, the property holds. Otherwise, the restriction of $T$ by the unifier $\sigma_i t$ of $t$ and $m_{i_1}$ is an empty term that implies $m_{i_1} \preceq t$. In conclusion, $\cap \Pi' \not\leq T$.

**Remark:**

Let $u$ be an occurrence of $T$ such that for all $M \in \Pi'$, $u \in \mathcal{O}(M)$, then $u \in \mathcal{O}(\cap \Pi')$. This property comes from the fact that for all $u \preceq \text{prefix } u$, $M(v) = T(v)$.

**Lemma 4.3.** Let $\Pi$ be the decomposition of a variable w.r.t. a complete set of patterns and $T = \{t|P\}$ be a term. Then:

$$\text{Dir}_\Pi(T) = \text{Dir}_\Pi' (\cap \Pi') \text{ where } \Pi' = \{M \in \Pi | T \uparrow M\}$$

**Proof.** Let $\Pi''$ be the normalization of $\Pi'$. We prove the following inclusions:

$$\text{Dir}_\Pi'(\cap \Pi'') \subseteq \text{Dir}_\Pi(T) \subseteq \text{Dir}_\Pi' (\cap \Pi') \subseteq \text{Dir}_\Pi'(\cap \Pi')$$
Dir'\(\cap Pi''\) ⊂ Dir\(\cap Pi\)(T):

Let \(u \in Dir_{\cap Pi}(\cap Pi'')\). There exists \(M \in Pi''\) such that \(M(u) = C\) with \(C \in \Sigma\), otherwise for every \(M \in Pi''\), \(M/u = \{x|\cap L\}\) because the constraints are normalized and thus \(M/u = \cap Pi''/u\) that contradicts the hypothesis. Furthermore, there exists \(M' \in Pi''\) such that \(M'/u = \{x|\cap L'\}\) and \(C(\Omega, \ldots, \Omega) \in L'\) or \(M'(u) = D\) with \(D \neq C\) otherwise \(\cap Pi''(u) = C\) that would contradict the hypothesis. In both cases \(\cap Pi''/u = \{x|T\}\) and, as the decomposition is complete, \(u \in O(T)\). If \(T(u) = C\), there would exist terms \(M, M' \in Pi''\) such that \(M(u) = C\) and either \(M'(u) = D\) (\(\neq C\)) or \(M'/u = \{x|\cap L'\}\) with \(C \in L'\) and both \(M\) and \(M'\) would have to be compatible with \(T\) which is impossible. Thus \(T/u = \{x|\cap L\}\). Also, if there were \(M \in Pi''\) such that \(M/u \leq T/u\), then \(M/u = \{x|\cap L'\}\) with \(x\cap L \Rightarrow x\cap L'\) and there would exist \(M'(u) = C\) with \(C(\Omega, \ldots, \Omega) \in L'\) that is in contradiction with the compatibility of \(M'\) and \(T\) and with \(x\cap L \Rightarrow x\cap L'\). In conclusion \(u \in Dir_{\cap Pi}(T)\).

Dir\(\cap Pi\)(T) ⊂ Dir'\(\cap Pi'\):

Suppose \(u \in Dir_{\cap Pi}(T)\). Then, by lemma 4.1, \(T/u = \{x|\cap L\}\) and for every \(M \in Pi'\), \(u \in O(M)\), and thus \(u \in O(\cap Pi')\), and \(M/u \not\leq T/u\). By lemma 4.2, \(\cap Pi'/u \leq T/u\) which implies that \(M/u \not\leq \cap Pi'/u\) and thus, \(u \in Dir_{\cap Pi'}(\cap Pi')\).

Dir'\(\cap Pi'\) ⊂ Dir'\(\cap Pi''\):

Let \(u \in Dir_{\cap Pi'}(\cap Pi')\). It is easy to check that \(u \in O(\cap Pi'')\) and \(\cap Pi''(u) = x\) and that if \(t_i \leq t_i'\), \(1 \leq i \leq n\), then \(\cap t_i \leq \cap t_i'\). As the normalization rules transform a term \(M\) into two terms \(M_1, M_2\) such that if \(M_1 \cap M_2 \leq M\), we deduce \(\cap Pi'' \leq \cap Pi'\), that implies \(M/u \not\leq \cap Pi''/u\) and thus \(u \in Dir_{\cap Pi'}(\cap Pi'')\).

This property allows to look for directions only in a finite set of prefixes of patterns.

**Theorem 4.2.** Let \(Pi\) be the decomposition of a variable w.r.t. a complete set of patterns. If \(Pi\) is finite, one can decide if \(Pi\) is sequential, one just checks that \(RMatch_{\cap Pi}\) is sequential at the principal lower bound \(\cap Pi'\) of every subset \(Pi'\) of \(Pi\).

**Proof.** If \(Pi\) is sequential, then \(RMatch_{\cap Pi}\) is sequential at every term \(T\) and in particular at every \(\cap Pi'\) where \(Pi'\) is a subset of \(Pi\). Conversely, let us suppose that \(RMatch_{\cap Pi}\) is sequential at the principal lower bound \(\cap Pi'\) of every subset \(Pi'\) of \(Pi\). Let \(T\) be a term such that \(RMatch_{\cap Pi}(T) = False\) and let \(Pi'\) be the subset of patterns compatible with \(T\). By lemma 4.3, \(Dir_{\cap Pi}(T) \supset Dir_{\cap Pi'}(\cap Pi')\). In order to prove that \(Pi\) is sequential, we just need to prove that \(Dir_{\cap Pi'}(\cap Pi') \neq \emptyset\) which is a consequence of \(RMatch_{\cap Pi'}(\cap Pi') = False\) by sequentiality hypothesis. If \(RMatch_{\cap Pi'}(\cap Pi')\) were \(True\), there would exist \(M' \in Pi\) more general than \(\cap Pi'\) that would contradict the fact that \(RMatch_{\cap Pi}(T) = False\). Thus \(RMatch_{\cap Pi'}(\cap Pi') = False\) and \(Dir_{\cap Pi}(\cap Pi') \neq \emptyset\) which implies \(Dir_{\cap Pi}(T) \neq \emptyset\) and thus \(Pi\) is sequential.

These theorems state that in order to verify the sequentiality of a match problem it is sufficient to verify it on a set of prefixes of the patterns. We propose now an algorithm to build the search tree of a pattern matching problem and we prove that the sequentiality is equivalent to the termination of this algorithm. We also prove the optimality of the
SearchTree\( (N, \Pi) = \)
\( T \) where

If there is no direction from \( N \) to \( \Pi \),
if \( N \in \Pi \), \( \epsilon \) is the only occurrence in \( T \) and \( \text{Root}(T) = N \)
otherwise the algorithm fails
otherwise
let \( u \) be one direction for \( N \) to \( \Pi \) such that \( N/u = \{ x | x \circ L_u \} \)
and \( L = \{ l_1, \ldots, l_p \} \) be the set
\( \{ F(\cdots) \exists M \in \text{Decomp}(N, \Pi) \text{ such that } F(\cdots) \uparrow M/u \} \).
\( \text{Root}(T) = (N, u) \) and
For each \( i \ (1 \leq i \leq p+1) \), \( i \) is an occurrence of \( T \),
\( T/i = \text{SearchTree}(N[u \leftarrow l_i], \Pi) \) for \( 1 \leq i \leq p \)
and \( T/p + 1 = \text{SearchTree}(N[u \leftarrow \{ x | x \circ L_u \cup L \}], \Pi) \)

**Figure 1.** Algorithm SearchTree

algorithm both in the number of tests in each path of the tree and in the number of terms for which the algorithm terminates. Furthermore we prove the equivalence between the sequentiality of a set of patterns and the existence of an optimal pattern matching algorithm.

Remark:

Let \( T, u \) be the label of an internal node of the search tree and \( u \) be the chosen direction for this node. This occurrence \( u \) cannot be any more a direction in an instance \( T' \) of \( T \) because there exists, by construction of the search tree, a pattern \( M \) compatible with \( T' \) such that \( M/u \preceq T'/u \). From this remark, we deduce that if the root is labeled by \( \{ x | T \} \), then for every label \( T, u \) of the search tree \( T/u = \{ x | T \} \) and the condition, \( u \) is a direction if for every compatible pattern \( M, M/u \not\preceq T/u \) becomes \( u \) is a direction if for every compatible pattern \( M, M/u \neq \{ x | T \} \).

**Theorem 4.3.** Let \( \Pi \) be the decomposition of a variable w.r.t. a complete set of patterns. \( \Pi \) is sequential if and only if the algorithm of figure 1 succeeds for \( \Pi \).

**Proof.** If \( \Pi \) is sequential it is possible to build a search tree and the procedure terminates because at each recursive call to SearchTree\( (N, \Pi) \) the number of patterns in \( \Pi \) compatible with \( N \) strictly decreases. Conversely, let us suppose that there exists a search tree SearchTree\( (x, \Pi) \) for a variable \( x \). Let us prove that for any term \( M \), principal lower bound of a subset of \( \Pi \), Dir\( \Pi(M) \neq \emptyset \) and thus that \( \Pi \) is sequential at \( M \). The nodes of this tree are labeled by pairs of terms and occurrences. The subset of the set of labels whose term part is a prefix of \( M \) is not an empty set because it contains at least the label of the root of the tree which is a variable. Furthermore by construction of the search tree, two sons of the same node are incompatible and this subset contains a maximal element with respect to \( \preceq \), that we call \( (T_0, u_0) \). Let us prove that \( u_0 \) which is a direction from \( T_0 \) to \( \Pi \) is also a direction from \( M \) to \( \Pi \). Notice that the pure part of \( M/u_0 \) is a variable otherwise the term part of a son of \( (T_0, u_0) \) in the search tree would be a prefix of \( M \), because the different sons represent all the possible instantiations at the occurrence \( u_0 \) of \( T_0 \) and \( T_0 \) would not be maximal. Also because of the same argument, there cannot
exist a pattern $N$ in $\Pi$ such that $N/u_0 \preceq M/u_0$. In conclusion $u_0$ is a direction from $M$ to $\Pi$.

**Theorem 4.4.** Let $\Pi$ be the decomposition of a variable w.r.t. a complete set of patterns. $\Pi$ is sequential if and only if there exists an optimal pattern matching algorithm for $\Pi$.

**Proof.** When $\Pi$ is sequential, the algorithm SearchTree of figure/restFig build a search tree. The set of terms for which the pattern matching fails, using this search tree, is generated by the terms $T[u \leftarrow \bullet]$ where $(T, u)$ is a label of the search tree. This terms are incompatible with all the patterns because $\{x|T\}$ is the only term compatible with $\{\bullet|T\}$ and, for every pattern $M$ compatible with $T$, $M/u \neq \{x|T\}$. In conclusion the set of terms for which the pattern matching fails is the strict set of a variable with respect to $\Pi$. Conversely, if there exists an optimal pattern matching algorithm while $\Pi$ is not sequential, the algorithm evaluates for a term $T$ an occurrence $u$ which is not a direction from $T$ to $\Pi$. As $u$ is not a direction, there exist instances of some pattern with a $\bullet$ as subterm at the occurrence $u$. These instances are not in the strict set and cannot be recognized by the optimal algorithm, that leads to a contradiction. This property states an equivalence between optimality and sequentiality.

We have extended the sequentiality to constrained terms which allows to compute optimal algorithms for call by pattern matching. If we complete the initial set of patterns by $\Omega$ in order to cover all the cases, we optimize both the success and the failure of the matching. The sequentiality of the set of patterns can be modified by the inclusion of the new element $\Omega$, but, as the search tree covers anyway all the cases, this restriction of the sequentiality has a positive effect on the result.

In case of non-sequential sets of patterns, it is possible to build a non optimal search tree, by ignoring some of the patterns. Two possibilities appear: to ignore, during the direction search, either the pattern with lower priority or those patterns that prevent the existence of directions.

5. Examples

We wrote a prototype of this method in CAML (Weis, Aponte, Laville, Mauny and Suárez 1989) which is used to generate mechanically all the examples in the paper. In this prototype we only represent constraints of depth 1, other constraints are normalized during the application of substitutions. In all the examples we add the term $z$ at the end of the list of patterns to complete the set.

1 With the set of patterns $F(A,B)$, $F(A,z)$, $F(y,B)$ the decomposition produces the following constrained terms:

$$F(A,B) \quad \{F(A,z)|z\Omega\{B\}\} \quad \{F(y,B)|y\Omega\{A\}\}$$

$$\{F(y,z)|y\Omega\{A\} \quad z\Omega\{B\}\} \quad \{z|x\Omega\{F(\Omega,\Omega)\}\}$$

And the strict set is:

$$\bullet, \quad F(z,\bullet) \quad F(\bullet, y)$$

The nodes of search trees are pairs formed by a term and a variable which is a direction in the term. The arcs are labeled by the possible values the direction can take and leaves are represented by the matched patterns.
2 For Gustave’s function: \( G(A, B, x) \), \( G(B, y, A) \), \( G(z, A, B) \) the decomposition of \( G(z, y, x) \) produces:

\[
\begin{align*}
G(A, B, x) &= \{G(z, y, x)|z\circ\{A, B\} y\circ\{A\}\} \\
G(B, y, A) &= \{G(z, y, x)|y\circ\{A, B\} z\circ\{A\}\} \\
G(z, A, B) &= \{G(z, y, x)|z\circ\{A, B\} x\circ\{B\}\}
\end{align*}
\]

As the original patterns have no common instance, they all belong to the decomposition, and there is no direction to start the match.

3 In this example extracted from a CAML program, we try to match lists of Booleans (\( \text{Nil} \) represents the empty list, \( x :: y \) is a list containing the element \( x \) followed by the list \( y \)).

\[
(y :: \text{True} :: u) \quad (\text{False} :: \text{Nil}) \quad \text{Nil}
\]

The decomposition of this example is:

\[
\begin{align*}
(y :: \text{True} :: u) &= \text{Nil} \\
\{x|x\circ\{\text{Nil},::\}\} &= (\text{False} :: \text{Nil}) \\
\{(y :: z)|y\circ\{\text{False}\} z\circ\{::\}\} &= \text{Nil,::}
\end{align*}
\]

the strict set is:

\[
* \quad y :: * \quad :: \text{Nil} \quad y :: * :: u
\]

and the search tree is:

In the decomposition of this example, some of the patterns have the constraint \( x\circ\{\text{Nil},::\} \). In a typed language, if \( \text{Nil} \) and :: are the only list constructors of lists, these patterns represent an empty set. Eliminating them (and assuming that \( \neq \text{True} \) implies \( \text{False} \) and that \( \neq \text{False} \) implies \( \text{True} \)) the decomposition becomes:

\[
(y :: \text{True} :: u) \quad (\text{False} :: \text{Nil}) \quad \text{Nil} \quad (y :: \text{False} :: u) \quad (\text{True} :: \text{Nil})
\]
which is the set of minimal extended patterns as defined by Laville (1988a). The search tree now becomes:

4 The sequentiality of a problem might depend on the signature of terms, for instance the decomposition of $F(x, y)$ by the patterns $F(A, A)$, $F(B, B)$ produces:

$$
F(A, A) \{ F(x, y) | y \diamond \{A, B\} \} \\
F(B, B) \{ F(x, y) | x \diamond \{B\} y \diamond \{A\} \} \\
F(x, y) | x \diamond \{A, B\}
$$

With the following strict set:

$$
F(A, \bullet) , \ F(\bullet, A) , \ F(\bullet, \bullet) , \ F(B, \bullet) , \ F(\bullet, B)
$$

This problem is not sequential because of the patterns $\{ F(x, y) | y \diamond \{A, B\} \}$ and $\{ F(x, y) | x \diamond \{A, B\} \}$. However, if the same match problem where given for a type that is defined with only two constants like the Booleans, those two patterns would represent empty sets and thus could be eliminated. In that case, the decomposition of $F(x, y)$ by the patterns $F(\text{True}, \text{True})$, $F(\text{False}, \text{False})$ produces:

$$
F(\text{True}, \text{True}) , \ F(\text{False}, \text{False}) , \ F(\text{True}, \text{False}) , \ F(\text{False}, \text{True})
$$

And the problem becomes sequential with the search tree:

6. Conclusion

Constrained terms are used to extend the sequentiality to ambiguous sets of patterns. The introduction of an explicit symbol $\bullet$ to represent non-terminating evaluations allows to use constraints for the partially evaluated terms.

The actual compilers for pattern matching use different techniques to improve the code generated for call by pattern matching, like the introduction of heuristics for finding directions, or the analysis of execution tests to improve most frequent cases. Both heuristics
and execution tests analysis become unnecessary as our algorithm computes directions and produces an optimal search tree that includes only unavoidable tests.

The elements of the decomposition are exactly the leaves of the optimal search tree which depends inherently on the match problem. The order of complexity of the substitution and of the restriction is in $O(l)$. For the decomposition it is $O(m \times l)$ and for the search of directions during the construction of a search tree it is $O(m \times l)$ where $m$ is the number of patterns of the match and $l$ is their average size.

The technique presented in this paper allows the implementation of optimal compilers for call by pattern matching in all the languages that support this feature, and encourages language designers to introduce it into new programming languages.

7. Acknowledgements

We are grateful to Jean-Jacques Lévy and Hubert Comon who made numerous suggestions on the presentation of this work. We would like also to acknowledge helpful comments made by Gérard Huet and Hassan Aït Kaci.

References


