Optimal Solutions to Pattern Matching Problems

Laurence Puel[1] and Ascánder Suárez[2]

[1] LRI. URA 410 du CNRS
bat 490, Université Paris Sud
91405 Orsay, FRANCE

[2] LIENS. URA 1327 du CNRS
Ecole Normale Supérieure
45, Rue d'Ulm
75005 Paris, FRANCE

Abstract. We define a *General Pattern Matching problem* and we show that several compiling problems in programming languages, like pattern Matching in ML and the calling mechanism of Prolog can be formalized as instances of the General Pattern Matching problem. As a consequence of this, the solutions of the general problem which are compiling algorithms can be instantiated into compiling algorithms for the instances. In particular, the proof of the decidability of the existence of optimal solutions is a proof of the decidability of the instances. This approach can be used for the meta-compilation of pattern-matching problems, for the implementation of languages or systems that contain different instances of the general problem and for the implementation of systems using *Call by Name* style of evaluation for which completeness and optimality are equivalent.

Introduction

In 1979 Huet and Lévy [1, 2] proposed a method for the compilation of pattern matching problem with linear non ambiguous patterns. In 1988 Laville[4, 5] adapted this result for the matching of linear pattern with priority rule for disambiguation. In 1990 Puel and Suárez [7] extended Huet and Lévy results for a family of terms called constrained terms in order to solve the call by pattern matching problem of the compilation of the language ML.

In this work, we propose a generalization of the pattern matching defined in [1] and a compilation algorithm. We define and prove properties of sequentiality and optimality of this algorithm. We check the correspondence between the notion of sequentiality in [1] and here.

After this theoretical results, we show how the problem addressed in [1] can be seen as a particular case of our general pattern matching problem. Finally, we show how to use this results in practice in order to build compilers of matching primitives of programming languages. Two kinds of instances of the matching problem are defined. Those based on the matching of terms and those based on

unification. We check the inheritance of the general result of sequentiality and optimality for these instances.

1 Terms and Constraints

Definition 1 *Let Σ and X be two disjoint sets of symbols representing respectively variables and term constructors. The set T of terms is composed by:*

$$\begin{array}{l} \textit{terms} \\ \quad T ::= x \qquad\qquad x \in X \\ \qquad\quad | \; F(t_1, \ldots, t_n) \; F \in \Sigma, t_1, \ldots, t_n \in T \; n \geq 0 \end{array}$$

Notation *In what follows, we abbreviate the sequence of terms $t_1, \ldots, t_n$ by $\overline{t_n}$. A* linear term *t is a term in which variable names are pairwise distinct. A* substitution *is a morphism over terms. A term t is* more general *than another term t' denoted $t \preceq t'$ if there exists a substitution s such that $s(t) = t'$; in this case we also say that t' is an* instance *of t. Two terms t and t' are* compatible *denoted $t \uparrow t'$ if there exists a substitution s such that $s(t) = s(t')$. A* closed term *is a term without variables.*

The occurrences *O are sequences of pairs formed by a term constructor $F \in \Sigma$ and an integer i denoted F^i (notice that the sequence of integers represents the usual notion of occurrence). The empty sequence is denoted ϵ. Let t be a term, the set of occurrences of t, denoted $O(t)$, is the subset of O defined by*

$$\begin{array}{rl} O(F) = \{\epsilon\} & F \in \Sigma \cup X \\ O(F(\overline{t_n})) = \{\epsilon\} \cup \{F^i.u | u \in O(t_i), 1 \leq i \leq n\}. & \end{array}$$

The subterm of a term *t* at occurrence *u, denoted t/u is defined by*

$$\begin{array}{c} t/\epsilon = t \\ F(\overline{t_n})/F^i.u = t_i/u \; 1 \leq i \leq n. \end{array}$$

The label *of a term t at occurrence u, denoted $t(u)$ is defined by*

$$\begin{array}{ll} x(\epsilon) = x & \text{if } x \in X \\ t(\epsilon) = F & \text{if } t = F(\overline{t_n}), n \geq 0 \\ t(u) = (t/u)(\epsilon) & \text{if } u \neq \epsilon \end{array}$$

An occurrence v is a prefix of another occurrence u, denoted $v \leq_{\text{prefix}} u$ if and only if there exists an occurrence w such that $u = vw$. An occurrence u is incompatible with a term t if there exist two occurrences v and w such that $u = vw$, $v \in O(t)$, $t/v = F(\overline{t_n})$, $w = G^i.w'$ and $F \neq G$ or $i > n$. An occurrence u overpass a term t if there exists $v <_{\text{prefix}} u$ such that $v \in O(t)$ and $t(v) \in X$ (in other words, neither $u \in O(t)$ nor u is incompatible with t). The set Var(t) *of variables of a term t is $\{x \in X | \exists u \in O(t)$ such that $t(u) = x\}$.*

Example 1 Let $\Sigma = \{f, g, a\}$, $x \in X$ and $t = f(g(a), x)$. $O(t) = \{\epsilon, f^1, f^2, f^1 g^1\}$, $t/\epsilon = t$, $t/f^1 = g(a)$, $t/f^1 g^1 = a$, $t/f^2 = x$. $f^2 g^1$ overpass t and $f^1 f^1$ is incompatible with t.

Definition 2 *Let $\mathcal{A}$ be a set. The* constraints *are predicate logic expressions with constants* true, false *and atoms $A \in \mathcal{A}$.*

$$
\begin{array}{ll}
\textit{constraints} & \\
C ::= A & A \in \mathcal{A} \quad \textit{(Atomic constraints)} \\
\quad \mid \text{true} \mid \text{false} & \\
\quad \mid c_1 \vee c_2 & c_1, c_2 \in C \\
\quad \mid c_1 \wedge c_2 & c_1, c_2 \in C \\
\quad \mid \neg c & c \in C
\end{array}
$$

To each atomic constraint A is associated a subset of O denoted $U(A)$ and a valuation *function $\mathcal{V}(A) : T \to C$. The* valuation *function of atoms is extended to any constraint as follows:*

$$
\begin{array}{ll}
\mathcal{V}(\text{true})(t) & = \text{true} \\
\mathcal{V}(\text{false})(t) & = \text{false}
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{V}(c_1 \vee c_2)(t) = \text{true} & \textit{if } \mathcal{V}(c_1)(t) = \text{true} \\
 & \quad \textit{or } \mathcal{V}(c_2)(t) = \text{true} \\
\mathcal{V}(c_1 \vee c_2)(t) = \text{false} & \textit{if } \mathcal{V}(c_1)(t) = \text{false} \\
 & \quad \textit{and } \mathcal{V}(c_2)(t) = \text{false} \\
\mathcal{V}(c_1 \vee c_2)(t) = \mathcal{V}(c_1)(t) \vee \mathcal{V}(c_2)(t) & \\
 & \quad \textit{otherwise}
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{V}(c_1 \wedge c_2)(t) = \text{true} & \textit{if } \mathcal{V}(c_1)(t) = \text{true} \\
 & \quad \textit{and } \mathcal{V}(c_2)(t) = \text{true} \\
\mathcal{V}(c_1 \wedge c_2)(t) = \text{false} & \textit{if } \mathcal{V}(c_1)(t) = \text{false} \\
 & \quad \textit{or } \mathcal{V}(c_2)(t) = \text{false} \\
\mathcal{V}(c_1 \wedge c_2)(t) = \mathcal{V}(c_1)(t) \wedge \mathcal{V}(c_2)(t) & \\
 & \quad \textit{otherwise}
\end{array}
$$

$$
\begin{array}{lll}
\mathcal{V}(\neg c)(t) & = \text{true} & \textit{if } \mathcal{V}(c)(t) = \text{false} \\
\mathcal{V}(\neg c)(t) & = \text{false} & \textit{if } \mathcal{V}(c)(t) = \text{true} \\
\mathcal{V}(\neg c)(t) & = \neg(\mathcal{V}(c)(t)) & \\
 & & \quad \textit{otherwise}
\end{array}
$$

The instances of a constraint C are the set $\text{Inst}(C) = \{t \in T | \mathcal{V}(C)(t) = \text{true}\}$. *A constraint C implies a constraint D, denoted $C \Rightarrow D$, if* $\text{Inst}(C) \subset \text{Inst}(D)$. *Two constraints C and D are equivalent, denoted $C \equiv D$, if* $\text{Inst}(C) = \text{Inst}(D)$. *Two constraints C and D are compatible, denoted $C \uparrow D$, if* $\text{Inst}(C) \cap \text{Inst}(D) \neq \emptyset$. *Otherwise they are incompatible, denoted $C \not\uparrow D$. A constraint C is compatible with a set of constraints Π if there exists a constraint $D \in \Pi$ compatible with C.*

We define here a set of atoms frequently used below: the predicate $\text{Occ}(u)$ tests if u belongs to the set of occurrences of a term.

Definition 3 (Constraint $\mathrm{Occ}(u)$**)** *To each* $u \in O$ *is associated the predicate* $\mathrm{Occ}(u)$ *over terms such that* $U(\mathrm{Occ}(u)) = \{u\}$ *and its valuation is defined by:*

$$\begin{aligned}\mathcal{V}(\mathrm{Occ}(u))(t) &= \mathrm{true} \quad \textit{if } u \in O(t)\\ &= \mathrm{Occ}(u) \textit{ if } u \textit{ overpass } t\\ &= \mathrm{false} \quad \textit{if } u \textit{ incompatible with } t\end{aligned}$$

2 General Pattern Matching Problem

We define pattern matching over constraints and check its correspondence with pattern matching over terms.

2.1 Match over constraints

Definition 4 *Let* $\Pi = \{M_1, \ldots, M_n\}$ *be a set of pairwise incompatible constraints named patterns. The pattern matching partial function* Match_Π *over constraints is defined by* $\mathrm{Match}_\Pi(C) = i$ *if and only if* $C \Rightarrow M_i$.

Let us show how a matching problem can be seen as a Matching over constraints.

- Atoms for matching and valuation:

  Definition 5 *Let* A *be the set of atoms of the form* $u \doteq F$ *(*$u \in O$, $F \in \Sigma$ *and* $U(u \doteq F) = \{u\}$*) whose valuation is defined by*

  $$\begin{aligned}&\mathcal{V}(u \doteq F)(t) = \mathrm{true} \quad \textit{if } u \textit{ incompatible with } t\\ &\mathcal{V}(u \doteq F)(t) = u \doteq l \quad \textit{if } u \textit{ overpass } t\\ &\mathcal{V}(u \doteq F)(t) = \mathrm{true} && \textit{if } t(u) = F\\ &\mathcal{V}(u \doteq F)(t) = \mathrm{false} \textit{ if } t(u) = G\\ &\qquad\qquad \textit{and } F \neq G\\ &\mathcal{V}(u \doteq F)(t) = u \doteq F && \textit{if } t(u) \in X.\end{aligned}$$

  These atoms correspond to primitive comparison operations between the label of a term at a given occurrence and a constant.

- To each term t is associated the constraint

  $$\mathcal{C}(t) = \bigwedge_{\substack{u \in O(t),\\ t(u) = F}} u \doteq F$$

- **Lemma 1** *Let* t *and* t' *be two terms.* $t \preceq t'$ *if and only if* $\mathcal{C}(t') \Rightarrow \mathcal{C}(t)$.
  Proof: Clearly, if $t \preceq t'$, there exists a constraint D such that $\mathcal{C}(t') \equiv \mathcal{C}(t) \wedge D$ and thus, $\mathcal{C}(t') \Rightarrow \mathcal{C}(t)$. Conversely if $\mathcal{C}(t') \Rightarrow \mathcal{C}(t)$, as $\mathcal{V}(\mathcal{C}(t'))(t') = \mathrm{true}$, for every $u \in O(t)$, such that $t(u) = F$, $\mathcal{V}(u \doteq F)(t') = \mathrm{true}$, $u \in O(t')$ and $t'(u) = t(u)$, otherwise, there would exist $v \in O(t') \cap O(t)$ such that $t'(v) \neq t(v)$ $(= G)$ and thus, $\mathcal{V}(v \doteq G)(t') \neq \mathrm{true}$. In conclusion, $t \preceq t'$.

There are several algorithms to check the match of a constraint by a given set of patterns, named pattern matching algorithms. We will use *Search trees* to represent these algorithms. The nodes of these trees that are not leaves, have constraints as labels. The label of the root is true. The sons of a node with label C are $C \wedge D$ where D is either an atom or the negation of an atom and $C \wedge D$ is compatible with a pattern. The labels of the leaves are pairs (C, i) where C is a constraint and i is the integer such that $C \Rightarrow M_i$. The only freedom in the construction is the choice of the atom used to develop the subtrees. A pattern matching algorithm either associates to a given constraint a pattern or *fails*. The algorithm is said to recognize the constraint C if there exists a leave (D, i) such that $C \Rightarrow D$.

Definition 6 *Let $\Pi = \{\overline{M_n}\}$ a set of pairwise disjoint constraints and s be a search tree corresponding to Π. A term t is recognized by s if it belongs to the instances of one of the leaves of s. The partial function* Search *that follows associates to a pair (s, t) an integer i such that $t \in \mathrm{Inst}(M_i)$.*

$$\begin{aligned} \mathrm{Search}((C,i),t) &= i \quad \textit{if } t \in \mathrm{Inst}(C) \\ \mathrm{Search}(C(\overline{\tau_m}),t) &= \mathrm{Search}(\tau_j,t) \\ &\qquad \textit{if } t \in \mathrm{Inst}(\mathrm{label}(\tau_j)) \\ \mathrm{Search}((C,i),t) &= \mathrm{fail} \ \textit{otherwise.} \end{aligned}$$

2.2 Optimality and sequentiality

We are interested in optimal pattern matching algorithm in the following sense:

Definition 7 (Optimal Pattern Matching) *A pattern matching algorithm is optimal if it recognizes every term that is recognized by any other pattern matching algorithm.*

In the following we propose an algorithm that produces a search tree and prove the optimality of the corresponding pattern matching algorithm. In order to do that we introduce, following Huet and Lévy [1], the sequentiality of a pattern matching problem Π and we prove that the algorithm producing the search tree does not fail if and only if the problem is sequential and that the sequentiality of the problem is decidable and implies the optimality of the pattern matching algorithm.

Definition 8 (Set of Directions of a constraint C with respect to Π) *Let Π be a set of pairwise incompatible constraints. Let C be a constraint and $\Pi' = \{M \in \Pi | M \uparrow C\}$ the set of patterns compatible with C.*

$$\mathrm{Dir}_\Pi(C) = \left\{ D \left| \begin{array}{l} D \in \mathrm{Atom}(\Pi), C \not\Rightarrow D,\ C \not\Rightarrow \neg D, \\ \forall u \in U(D), C \Rightarrow \mathrm{Occ}(u) \textit{ and} \\ \forall M \in \Pi', \forall u \in U(D), M \wedge C \Rightarrow D \vee \neg D \end{array} \right. \right\}$$

Definition 9 (sequentiality) *Let Π be a finite set of pairwise incompatible constraints. The problem Π is sequential, if for every constraint C compatible with Π and such that every pattern $M \in \Pi$, $C \not\Rightarrow M$, $\mathrm{Dir}_\Pi(C) \neq \emptyset$.*

The following algorithm built trees $\mathrm{ST}_{\Pi}(\mathrm{true})$ that we interpret as pattern matching algorithms for a given set of patterns Π.

Definition 10 (Search tree) *Let $\Pi = \{M_1, \ldots, M_n\}$ be a set of patterns and C be a constraint.*

$$
\begin{array}{l}
ST_{\Pi}(C) \\
= \textbf{if } \mathrm{Dir}_{\Pi}(C) \textit{ non empty } \textbf{then let} \\
\quad D \in \mathrm{Dir}_{\Pi}(C), \\
\quad \Pi_1 = \{M \in \Pi | M \wedge C \wedge D \not\equiv \mathrm{false}\} \textbf{ and} \\
\quad \Pi_2 = \{M \in \Pi | M \wedge C \wedge \neg D \not\equiv \mathrm{false}\} \textbf{ in} \\
\quad C(ST_{\Pi}(C \wedge D), ST_{\Pi}(C \wedge \neg D)) \\
\qquad\qquad \textbf{if } \Pi_1 \neq \emptyset \textit{ and } \Pi_2 \neq \emptyset \\
\quad C(ST_{\Pi}(C \wedge D)) \quad \textbf{if } \Pi_1 \neq \emptyset \textit{ and } \Pi_2 = \emptyset \\
\quad C(ST_{\Pi}(C \wedge \neg D)) \textbf{ if } \Pi_1 = \emptyset \textit{ and } \Pi_2 \neq \emptyset \\
= (C, i) \textit{ if } C \Rightarrow M_i \\
= \textit{fail } \textbf{otherwise}
\end{array}
$$

We prove in lemma 3 that all the trees provided by this algorithm for a given set of patterns recognize the same set of patterns. Let us first show how this formalism can describe the usual pattern matching problem for terms.

2.3 Huet and Levy's original problem

To each term t is associated the constraint $C(t)$ defined above. The following notions correspond:

$$
\begin{array}{rcl}
\text{term } t & \text{and} & \text{constraint } \mathcal{C}(t) \\
\{\tau | \exists \sigma \ \ \sigma(t) = \tau\} & \text{and} & \{\tau | \mathcal{V}(\mathcal{C}(t))(\tau) = \mathrm{true}\} \\
\mathrm{match}_{\Pi} & \text{and} & \mathrm{Match}_{\mathcal{C}(\Pi)} \\
Dir_{\Pi}(t) & \text{and} & Dir_{\mathcal{C}(\Pi)}(\mathcal{C}(t))
\end{array}
$$

The equality between $\{\tau | \exists \sigma \ \ \sigma(t) = \tau\}$ and $\{\tau | \mathcal{V}(\mathcal{C}(t))(\tau) = \mathrm{true}\}$ is a consequence of the fact that for every term t , $\mathcal{V}(\mathcal{C}(t))(t) = \mathrm{true}$.

Let $\Pi = \{p_1, \ldots, p_n\}$ be a set of pairwise incompatible linear terms. Let $\mathcal{C}(\Pi) = \{\mathcal{C}(p_1), \ldots, \mathcal{C}(p_n)\}$ be the set of corresponding constraints. Let match_{Π} be the predicate defined by $\mathrm{match}_{\Pi}(t) = \mathrm{true}$ if there exists $p_i \in \Pi$ such that $p_i \preceq t$. A term t satisfies the predicate match_{Π} if and only if the function $\mathrm{Match}_{\mathcal{C}(\Pi)}$ is defined for the corresponding constraint $\mathcal{C}(t)$, as a consequence of lemma 1.

Let $\Pi = \{p_1, \ldots, p_n\}$ be a set of pairwise incompatible linear terms. Let $\mathcal{C}(\Pi) = \{\mathcal{C}(p_1), \ldots, \mathcal{C}(p_n)\}$ be the set of corresponding constraints. Let t be a term compatible with a pattern $p \in \Pi$ such that $\mathrm{match}_{\Pi}(t) = \mathrm{false}$. Let us recall that $u \in Dir_{\Pi}(t)$ if and only if $u \in O(t)$, t/u is a variable and for every pattern $p \in \Pi$ compatible with t, $p/u \not\preceq t/u$. The following property is satisfied.

$$
\begin{array}{l}
u \in Dir_{\Pi}(t) \Leftrightarrow \exists D \in Dir_{\mathcal{C}(\Pi)}(\mathcal{C}(t)) \\
\qquad\qquad \text{such that } u \in U(D).
\end{array}
$$

Let $u \in Dir_\Pi(t)$, there exists a pattern p compatible with t such that $p/u = F(\ldots)$. Let $D = u \doteq F$ and $U(D) = u$, $D \in Atom(\mathcal{C}(\Pi))$ and by definition $\mathcal{C}(t) \Rightarrow \mathrm{Occ}(u)$. As t satisfies $\mathcal{C}(t)$ and t/u is a variable, $\mathcal{C}(t) \not\Rightarrow D$. As p is compatible with t there exists an instance τ of t such that $\tau/u = F(\ldots)$ and thus, $\mathcal{C}(t) \not\Rightarrow \neg D$. Finally let $M \in \Pi$ be a pattern compatible with t. $M/u \not\preceq t/u$ implies $\mathcal{C}(M) \Rightarrow D \vee \neg D$. Conversely let $D = u \doteq F \in Dir_{\mathcal{C}(\Pi)}(\mathcal{C}(t))$. As $\mathcal{C}(t) \Rightarrow \mathrm{Occ}(u)$, t/u is defined. t/u is a variable otherwise either $t(u) = F$ and $\mathcal{C}(t) \Rightarrow D$ or $t(u) = G$ with $G \neq F$ and $\mathcal{C}(t) \Rightarrow \neg D$. For every compatible pattern $p \in \Pi$, p/u is not a variable otherwise $\mathcal{C}(p) \not\Rightarrow D \vee \neg D$ because $\mathcal{V}(D \vee \neg D)(p) \neq \text{true}$ and $\mathcal{V}(\mathcal{C}(p))(p) = \text{true}$.

3 Decidability of sequentiality. Optimality

As in the case of terms we prove that the pattern matching algorithm computed for a sequential set of constraints Π is optimal. First we prove that a finite set of patterns Π is sequential if and only if the set of constraints appearing in the labels of the search tree $ST_\Pi(\text{true})$ is sequential. This property implies the decidability of the sequentiality of a finite set of patterns .

Lemma 2 *Let Π be a finite set of constraints such that $ST_\Pi(\text{true})$ does not fail. For every constraint C compatible with a pattern $M \in \Pi$ there exists a label P in $ST_\Pi(\text{true})$ such that $C \Rightarrow P$ and for every label P', if $C \Rightarrow P'$ then $P \Rightarrow P'$. P is named the maximum prefix of C in $ST_\Pi(\text{true})$. Furthermore, if P is not a leaf,* $\mathrm{Dir}_\Pi(P) \cap \mathrm{Dir}_\Pi(C) \neq \emptyset$.

Proof: Let C be a constraint such that there exists a pattern $M \in \Pi$ compatible with C. Let $\mathcal{P}$ be the set of labels M in $ST_\Pi(\text{true})$ such that $C \Rightarrow M$. $\mathcal{P}$ is not empty because $C \Rightarrow \text{true}$. All the elements of $\mathcal{P}$ belongs to the same branch of $ST_\Pi(\text{true})$ otherwise there exists a constraint D such that $C \Rightarrow D$ and $C \Rightarrow \neg D$ and thus, $C \equiv \text{false}$ that contradicts the fact $D \uparrow M$. Clearly there exists $P \in \mathcal{P}$ such that for every $P' \in \mathcal{P}$, $P \Rightarrow P'$.
Let $D \in \mathrm{Dir}_\Pi(P)$ such that $P \wedge D$ or $P \wedge \neg D$ or both are labels of $ST_\Pi(\text{true})$. Knowing that $D \in \mathrm{Dir}_\Pi(P)$ we prove that $D \in \mathrm{Dir}_\Pi(C)$. $C \not\Rightarrow D$ otherwise $C \Rightarrow P \wedge D$ that contradicts the maximality of P if $P \wedge D$ is a label of $ST_\Pi(\text{true})$ and the compatibility of C with a pattern if not. The same argument proves $C \not\Rightarrow \neg D$. By definition of $\mathrm{ST}_\Pi(\text{true})$, D is a direction of P and thus, $\forall u \in U(D)$, $P \Rightarrow \mathrm{Occ}(u)$. As $C \Rightarrow P$, then $C \Rightarrow \mathrm{Occ}(u)$. The last property is a consequence of the fact that $C \Rightarrow P$ implies $C \wedge M \Rightarrow P \wedge M$.

Lemma 3 (Equivalence of optimal Search Trees) *Let Π be a set of patterns. All the search trees built using the algorithm* $\mathrm{ST}_\Pi(\text{true})$ *have the same set of leaves.*

Proof: Let T and T' be two search trees $\mathrm{ST}_\Pi(\text{true})$ and F be a leaf of T. Notice that $\mathrm{Dir}_\Pi(F) = \emptyset$, $\mathrm{Inst}(F) \neq \emptyset$ and if $F' \neq F$ is a leaf of T, $\mathrm{Inst}(F) \cap \mathrm{Inst}(F') = \emptyset$. There exists by lemma 2 a leaf P' of T' such that $F \Rightarrow P'$. The same reasoning leads to the existence of a leaf F' of T such that $P' \Rightarrow F'$. Thus, $F = F'$.

We deduce from this lemma that all the different trees $ST_\Pi(\text{true})$ recognize the same set of constraints which allows us to use $ST_\Pi(\text{true})$ in order to denote one of them.

Theorem 1 *Let Π be a finite set of constraints. Π is sequential if and only if the algorithm $ST_\Pi(\text{true})$ terminates without any fail. Furthermore the sequentiality of a finite set of patterns is decidable.*
Proof: If Π is sequential each step in $ST_\Pi(\text{true})$ is such that either there exists a direction or a pattern matches the constraint. Conversely, let us suppose that $ST_\Pi(\text{true})$ does not fail. By lemma 2, for every constraint C compatible with Π such that for every $M \in \Pi$, $C \not\Rightarrow M$, there exists a maximal prefix P in $ST_\Pi(\text{true})$ such that $Dir_\Pi(P) \cap Dir_\Pi(C) \neq \emptyset$. Therefore $Dir_\Pi(C) \neq \emptyset$ and Π is sequential. The algorithm $ST\Pi(\text{true})$ terminates because the depth of the tree is bounded by the number of atoms in Π, and thus the sequentiality is decidable.

Lemma 4 *Let $\Pi = \{M_1, \ldots, M_n\}$ be a set of pairwise incompatible constraints. For every integer i and every constraint C such that $C \Rightarrow M_i$, there exist $F_1, \ldots, F_k$ leaves of $ST_\Pi(\text{true})$ such that $C \Rightarrow \bigvee_{1 \leq j \leq k} F_j \Rightarrow M_i$.*
Proof: Let C be a constraint such that $C \Rightarrow M_1$ for instance and P its maximum prefix in $ST_\Pi(\text{true})$. The proof is by induction over the size of the subtree of $ST_\Pi(\text{true})$ with root P. If P is a leave there exists a pattern M_i such that $C \Rightarrow P \Rightarrow M_i$ that implies $M_1 = M_i$ because patterns are incompatible. Otherwise, let $D \in \text{Dir}_\Pi(P)$ such that $P \wedge D$ or $P \wedge \neg D$ or both are labels of $ST_\Pi(\text{true})$. First notice that $C \equiv (C \wedge D) \vee (C \wedge \neg D)$ because $C \Rightarrow M_1 \Rightarrow D \vee \neg D$. As $C \wedge D$ (resp. $C \wedge \neg D$) satisfies the inductive hypothesis, $C \wedge D \Rightarrow \bigvee_{j \in J} F_j \Rightarrow M_1$ (resp. $C \wedge \neg D \Rightarrow \bigvee_{j \in J'} F_j \Rightarrow M_1$) where J (resp. J') is a subset of $[1, k]$. Thus, $C \Rightarrow \bigvee_{j \in J \cup J'} F_j \Rightarrow M_1$. The last part follows straightforwardly.

Theorem 2 *Let Π be a finite set of pairwise incompatible constraints. If Π is sequential then the search tree $ST_\Pi(\text{true})$ is optimal.*
Proof: The optimality is a consequence of lemma 4, by definition.

Definition 11 *Let $\Pi = \{\overline{M_n}\}$ a set of pairwise disjoint constraints and s be a search tree corresponding to Π. A term t is recognized by s if it belongs to the instances of one of the leaves of s. The partial function* Search *that follows associates to a pair (s, t) an integer i such that $t \in \text{Inst}(M_i)$.*

$$\begin{aligned}
\text{Search}((C, i), t) &= i \quad \text{if } t \in \text{Inst}(C) \\
\text{Search}(C(\overline{\tau_m}), t) &= \text{Search}(\tau_j, t) \\
&\qquad \text{if } t \in \text{Inst}(\text{label}(\tau_j)) \\
\text{Search}((C, i), t) &= \text{fail } \textit{otherwise.}
\end{aligned}$$

We present now several instances of the general match problem which can be classed in two groups: Those based on the match operation and those based on the unification operation. The former can be used in term rewriting systems and in functional programming; the later group is oriented to the implementation of logic programming languages.

4 Compilation of Pattern Matching in ML

For the first group, let us consider different variants of the match construct of the ML language whose semantics can be defined by the following conditional rewriting rule in the style of [6].

Definition 12 (Semantics of match in ML) *Let E be a set of expressions of the ML language containing functions defined by cases "*$(\mathbf{fun}\ p_1 \rightarrow e_1 | \ldots | p_n \rightarrow e_n)$*" and the match construct below that corresponds to the application of a function to an expression "*$\mathbf{match}\ e\ \mathbf{with}\ p_1 \rightarrow e_1 | \ldots | p_n \rightarrow e_n \equiv (\mathbf{fun}\ p_1 \rightarrow e_1 | \ldots | p_n \rightarrow e_n)\ e$*" . The set P of language patterns and the relation $i = \mathrm{match}_{\{\overline{p_n}\}}(v)$ where $\{\overline{p_n}\} \subset P$ will be defined for each of the proposed variants of the match construct. The substitution operation $e_i[p_i \leftarrow v]$ produces a partially evaluated expression in which the variables in pattern p_i are replaced by the corresponding parts of the value v. Finally, $e \Rightarrow v$ is the relation that holds if the value of expression e is v, which is defined on the structure of expressions and contains the rule*

$$\frac{e \Rightarrow v \quad i = \mathrm{match}_{\{\overline{p_n}\}}(v) \quad e_i[p_i \leftarrow v] \Rightarrow v'}{\mathbf{match}\ e\ \mathbf{with}\ p_1 \rightarrow e_1 | \ldots | p_n \rightarrow e_n \Rightarrow v'} \quad \text{(match)}$$

The goal of each of these presentations is to define, for each of the instances of the problem, a representation of patterns $\Pi_0 = \{\overline{p_n}\}$ as constraints $\Pi = \mathcal{C}(\Pi_0)$ which, by lemma 2, will lead to the definition of a search tree $s = \mathrm{ST}_{\Pi}(\mathrm{true})$ such that for any value v, $\mathrm{match}_{\Pi_0}(v) = \mathrm{Search}(s, v)$. Under these conditions, the rule above can be replaced by its equivalent compiled version.

$$\frac{\begin{array}{c} s = \mathrm{ST}_{\mathcal{C}(\overline{p_n})}(\mathrm{true}) \\ e \Rightarrow v \qquad i = \mathrm{Search}(s, v) \qquad e_i[p_i \leftarrow v] \Rightarrow v' \end{array}}{\mathbf{match}\ e\ \mathbf{with}\ p_1 \rightarrow e_1 | \ldots | p_n \rightarrow e_n \Rightarrow v'} \quad \text{(match')}$$

4.1 Match of linear non ambiguous patterns

This is a restriction to the match construct of the ML language in which patterns must not have common instances. It correspond to the original work of Huet and Lévy [1]. For this instance of the problem the set P of language patterns is the set T of terms (definition 1); the evaluation of the match construct

$$\mathbf{match}\ e\ \mathbf{with}\ p_1 \rightarrow e_1 | \ldots | p_n \rightarrow e_n \Rightarrow v'$$

is only defined for sets of patterns $\Pi = \{\overline{p_n}\}$ such that for any term $t \in T$, $p_i \preceq t$ and $p_j \preceq t$ if and only if $i = j$; finally, the match predicate is defined by $\mathrm{match}_{\Pi}(v) = i$ if $p_i \preceq v$.

To each pattern $p \in P$ is associated the constraint

$$\mathcal{C}(p) = \bigwedge_{\substack{u \in O(p), \\ p/u = F(\overline{\tau_m})}} u \doteq F$$

The translation of a set Π_0 of patterns is defined by

$$\mathcal{C}(\Pi_0) = \{\mathcal{C}(p) | p \in \Pi_0\}$$

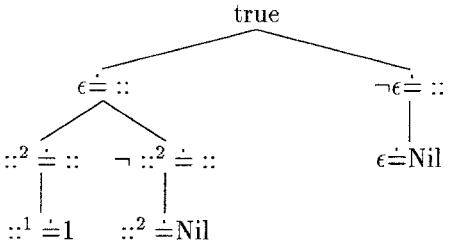
Example 2 Let e_0, e_1, e_2, e_3 be ML expressions :: be a binary constructor in infix notation, Nil, 1 be constants, x, y, z be variables and e be the expression

$$\textbf{match } e_0 \textbf{ with } (1 :: y :: z) \rightarrow e_1 \mid (x :: \text{Nil}) \rightarrow e_2 \mid \text{Nil} \rightarrow e_3.$$

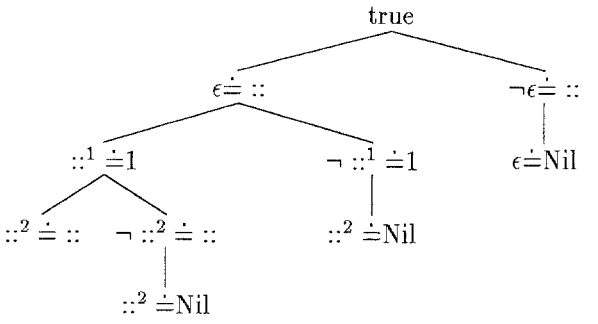
The set of patterns of e is $\Pi_0 = \{1 :: y :: z, x :: \text{Nil}, \text{Nil}\}$. The translation of these patterns produces the constraints

$$\begin{aligned} C_1 &= \epsilon \doteq :: \wedge ::^1 \doteq 1 \wedge ::^2 \doteq ::, \\ C_2 &= \epsilon \doteq :: \wedge ::^2 \doteq \text{Nil}, \\ C_3 &= \epsilon \doteq \text{Nil} \text{ and} \\ \Pi &= \{C_1, C_2, C_3\}. \end{aligned}$$

The optimal search tree associated to this problem is



Another search tree for the same set of patterns could be



Compared to the optimal search tree below, this search tree has an additional step for instances of the second pattern, which means that there are instances of terms that can be recognized by the former and not by the later. That is the case of the term $x :: \text{Nil}$.

4.2 Match of linear patterns ordered by priority

This instance has been studied by its own by Laville[4] and by Puel and Suárez [7]. The presentation proposed here does not cover completely those works; in particular, the strictness characterization of a match problem is not treated here. This instance of the general match problem correspond to the original match construct of the ML language.

We keep in this instance the same set P of language patterns, and change the match predicate as follows:

$$\text{match}_{\Pi}(v) = i \quad \text{if } p_i \preceq v \text{ and for any } j < i, p_i \not\preceq v$$

To each pattern $p \in P$ is associated the constraint

$$\mathcal{C}(p) = \bigwedge_{\substack{u \in O(p), \\ p/u = F(\overline{\tau_m})}} u \doteq F$$

The translation of a set Π_0 of patterns is defined by

$$\mathcal{C}(\Pi_0) = \{\mathcal{C}(p_i) \wedge \bigwedge_{1 \leq j < i} \neg\mathcal{C}(p_j) | 1 \leq i \leq n\}$$

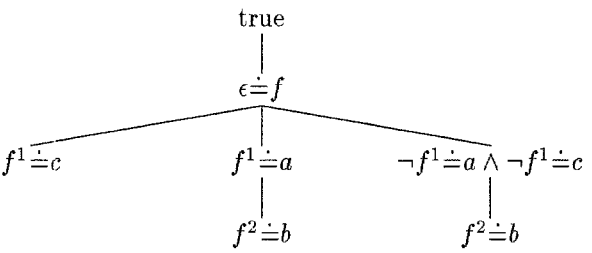
Example 3 Let e_0, e_1, e_2, e_3 be ML expressions, f be a constructor, a, b be constants and x, y be variables and e be the expression

$$\textbf{match } e_0 \textbf{ with } f(a,b) \rightarrow e_1 \mid f(c,x) \rightarrow e_2 \mid f(x,b) \rightarrow e_3.$$

The translation of the patterns of this problem produces the constraints

$$\begin{aligned} C_1 &= \epsilon \doteq f \wedge f^1 \doteq a \wedge f^2 \doteq b, \\ C_2 &= \epsilon \doteq f \wedge f^2 \doteq b, \\ C_3 &= \epsilon \doteq f \wedge f^2 \doteq b \wedge \neg f^1 \doteq a \wedge \neg f^1 \doteq c \text{ and} \\ \Pi &= \{C_1, C_2, C_3\}. \end{aligned}$$

The optimal search tree associated to this problem is



4.3 Match of non linear non ambiguous patterns

This is a variant of the match construct of ML in which variables in patterns might appear several times. The set P of language patterns in this instance is the set of non linear terms that can be represented with the notation (p with $x_1 = y_1, \ldots, x_m = y_m$) which indicates that in the linear pattern p variables x_i and y_i should correspond to the same value. The match predicate of this instance, defined only for non ambiguous sets of patterns is

$$\begin{array}{l}\text{match}_{\Pi}(v) = i \\ \text{if } p_i = (p \text{ with } s),\ p \preceq v \\ \quad \text{and for any pair } x = y \in s, x[p \leftarrow v] = y[p \leftarrow v]\end{array}$$

We introduce now, new atoms that will represent the non linear part of patterns.

Definition 13 *Let A be the set of atoms of the form $u \doteq F$ as in definition 5 completed with* equality constraints *which are expressions of the form $u \dot{\uparrow} v$ where u and v are occurrences and $U(u \dot{\uparrow} v) = \{u, v\}$. The valuation function defined for terms as follows:*

$$\begin{array}{ll}\mathcal{V}(u \dot{\uparrow} v)(t) = \text{true} & \textit{if } u \textit{ incompatible with } t \\ & \quad \textit{or } v \textit{ incompatible with } t \\ \mathcal{V}(u \dot{\uparrow} v)(t) = u \dot{\uparrow} v & \textit{if } u \textit{ overpass } t \textit{ or } v \textit{ overpass } t \\ \mathcal{V}(u \dot{\uparrow} v)(t) = \text{false} & \textit{if } t/v = F(\ldots) \textit{ and } t/u \in \text{Var}(t/v) \\ \mathcal{V}(u \dot{\uparrow} v)(t) = \text{false} & \textit{if } t/u = F(\ldots), t/v = G(\ldots) \textit{ and } F \neq G \\ \mathcal{V}(u \dot{\uparrow} v)(t) = \mathcal{V}(\wedge_{1 \leq i \leq n} u.F^i \dot{\uparrow} v.F^i)(t) & \\ \qquad \textit{if } t/u = F(\overline{t_n}) \textit{ and } t/v = F(\overline{t'_n}) & \\ \mathcal{V}(u \dot{\uparrow} v)(t) = u \dot{\uparrow} v & \textit{otherwise.}\end{array}$$

Equality constraints can be translated into a call to a general equality function often available in functional languages.

To each pattern "τ with $s \in P$" is associated the constraint

$$\mathcal{C}(\tau \text{ with } s) = \bigwedge_{\substack{u \in O(\tau), \\ \tau/u = F(\overline{\tau_m})}} u \doteq F \wedge \bigwedge_{\substack{x = y \in s, \\ \tau/u = x, \\ \tau/v = y}} u \dot{\uparrow} v$$

The translation of a set Π_0 of patterns is defined by

$$\mathcal{C}(\Pi_0) = \{\mathcal{C}(p) | p \in \Pi_0\}$$

Example 4 Let e_0, e_1, e_2, e_3 be ML expressions, f, g, h be constructors, x be a variable and e be the expression

$$\begin{array}{l}\textbf{match } e_0 \textbf{ with } f(g(x), h(x)) \rightarrow e_1 \\ \qquad\qquad\quad \mid f(g(x), x) \rightarrow e_2 \\ \qquad\qquad\quad \mid f(x, x) \rightarrow e_3.\end{array}$$

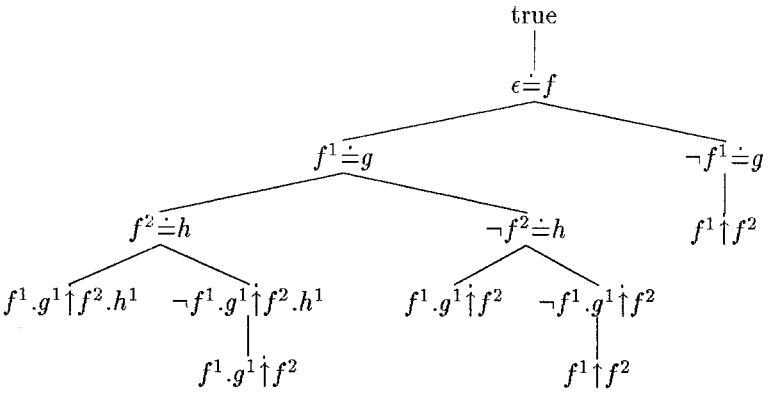
The original patterns associated to this example are

$$\Pi_0 = \left\{ \begin{array}{c} (f(g(x), h(y)) \text{ with } x = y), \\ (f(g(x), y) \text{ with } x = y), \\ (f(x, y) \text{ with } x = y) \end{array} \right\}.$$

The translation of these patterns produces the constraints

$$\begin{array}{l} C_1 = \epsilon \doteq f \wedge f^1 \doteq g \wedge f^2 \doteq h \wedge f^1.g^1 \dot{\uparrow} f^2.h^1, \\ C_2 = \epsilon \doteq f \wedge f^1 \doteq g \wedge f^1.g^1 \dot{\uparrow} f^2, \\ C_3 = \epsilon \doteq f \wedge f^1 \dot{\uparrow} f^2 \text{ and} \\ \Pi = \{C_1, C_2, C_3\}. \end{array}$$

The optimal search tree associated to this problem is



4.4 Match of non linear patterns ordered by priority

This is an extension of the match construct of ML in which both, priorities and non linear patterns are used. The constraints are defined on the set of atoms of definition 13.

To each pattern τ with $s \in P$ is associated the constraint

$$\mathcal{C}(\tau \text{ with } s) = \bigwedge_{\substack{u \in O(\tau), \\ \tau/u = F(\overline{\tau_m})}} u \doteq F \wedge \bigwedge_{\substack{x = y \in s, \\ \tau/u = x, \\ \tau/v = y}} u \dot{\uparrow} v$$

The translation of a set Π_0 of patterns is defined by

$$\mathcal{C}(\Pi_0) = \{\mathcal{C}(p_i) \wedge \bigwedge_{1 \leq j < i} \neg \mathcal{C}(p_j) | 1 \leq i \leq n\}$$

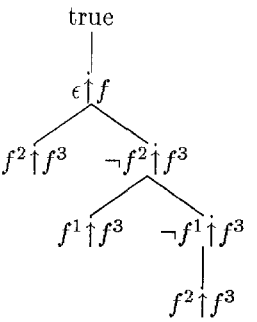
Example 5 Let e_0, e_1, e_2, e_3 be ML expressions, f be a constructor, x, y be variables and e be the expression

$$\begin{aligned}\textbf{match } e_0 \textbf{ with } & f(x,y,y) \rightarrow e_1 \\ | \ & f(x,y,x) \rightarrow e_2 \\ | \ & f(x,x,y) \rightarrow e_3.\end{aligned}$$

The translation of the patterns of this problem produces the constraints

$$\begin{aligned}C_1 &= \epsilon \doteq f \wedge f^2 \dot{\uparrow} f^3, \\ C_2 &= \epsilon \doteq f \wedge f^1 \dot{\uparrow} f^3 \wedge \neg f^2 \dot{\uparrow} f^3, \\ C_3 &= \epsilon \doteq f \wedge f^1 \dot{\uparrow} f^2 \wedge \neg f^2 \dot{\uparrow} f^3 \wedge \neg f^1 \dot{\uparrow} f^3 \text{ and} \\ \Pi &= \{C_1, C_2, C_3\}.\end{aligned}$$

The optimal search tree associated to this problem is



Another possible instance of the general match problem leading to the compilation of an extension of ML is the use of patterns of the form $(t \text{ with } s)$ in which s is any arbitrary predicate. The result will be a matching algorithm in which those predicates and the other matching operations are embedded and launched in the best possible order.

5 Compilation of Resolution in Prolog

The following examples represent different variants of the calling mechanism of logical programming languages. They are based on the unification as a pattern recognition operation.

Definition 14 (Semantics of the calling mechanism of Prolog) *Let $\Pi = \{\overline{p_n}\}$ be a set of patterns and $i = \text{match}_\Pi(t)$ be a relation representing the pattern matching operation that will be defined for each of the instances of the problem. A goal is a term (an element of T). The set Cl of Prolog clauses is defined by*

clauses

$$Cl ::= p :- q_1 \ldots q_n.\ p \in P, \overline{q_n} \in T, n \geq 0$$

The relation $\sigma = mgu(t, t')$ holds if the substitution σ is a most general unifier of terms t and t'. We note "$\Gamma \vdash g \Rightarrow g'$", a resolution step of a set of goals g given the set of clauses Γ producing a set of goals g'. This operation is defined by the conditional rewriting rule below in which $g, g_1 \ldots, g_n$ denote sets of terms representing goals

$$\frac{t(\epsilon) = F \quad \Gamma \Rightarrow F =_{\text{def}} (p_1 :- g_1 \ldots p_n :- g_n) \qquad i = \text{match}_{\{\overline{p_n}\}}(t) \qquad \sigma = mgu(t, p_i)}{\Gamma \vdash \{t\} \cup g \Rightarrow \sigma(g_i) \cup \sigma(g)} \quad \text{(Resolv)}$$

Given an appropriate compilation function $\mathcal{C}$ for patterns, the predicate rule above can be replaced by a compilation rule for definitions of relations and a new rule for the resolution.

$$\frac{s = \text{ST}_{\mathcal{C}(\overline{p_n})}(\text{true})}{\Gamma, (F, p_1 :- g_1 \ldots p_n :- g_n) \Rightarrow \Gamma \cup \{F =_{\text{cmp}} (s, p_1, g_1, \ldots, p_n, g_n)\}} \quad \text{(Cmpl)}$$

$$\frac{t(\epsilon) = F \quad \Gamma \Rightarrow F =_{\text{cmp}} (s, p_1, g_1, \ldots, p_n, g_n) \qquad i = \text{Search}(s, t) \qquad \sigma = \text{m}gu(t, p_i)}{\Gamma \vdash \{t\} \cup g \Rightarrow \sigma(g_i) \cup \sigma(g)} \quad \text{(Resolv}')$$

5.1 Resolution with linear patterns ordered by priority

The first instance is variant of Prolog in which only linear patterns are allowed. A disambiguating rule, priority in this case, is necessary because unification as a pattern matching operation is intrinsicly ambiguous for terms that are prefixes (with respect to $\preceq$) of patterns.

In this instance of the general match problem, the set P of language patterns is the set T of terms and the match function is defined as follows:

$$\begin{aligned} &t \text{ matches } p \text{ if and only if } p \text{ and } t \text{ are unifiable} \\ \text{match}_{\Pi}(t) = i \text{ if } &p_i, \text{ matches } t \\ &\text{and for any } j < i, t \text{ does not match } p_j \end{aligned}$$

The following atomic constraints will be used for the representation of patterns.

Definition 15 (Var Constraint) *To each $u \in O$ is associated a constraint* $\text{Var}(u)$ *defined by:*

$$\begin{aligned} \mathcal{V}(\text{Var}(u))(t) &= true && \text{if } u \in O(t) \text{ and } t(u) \in X \\ &= \text{Var}(u) && \text{if } u \text{ overpass } t \\ &= false && \text{if } u \in O(t) \text{ and } t(u) \in \Sigma \end{aligned}$$

To each pattern $p \in P$ is associated the constraint

$$\mathcal{C}(p) = \bigwedge_{\substack{u \in O(p), \\ p/u = F(\overline{\tau_m})}} (\mathrm{Var}(u) \vee u \doteq F)$$

The translation of a set Π_0 of patterns is defined by

$$\mathcal{C}(\Pi_0) = \{\mathcal{C}(p_i) \wedge \bigwedge_{1 \leq j < i} \neg\mathcal{C}(p_j) | 1 \leq i \leq n\}$$

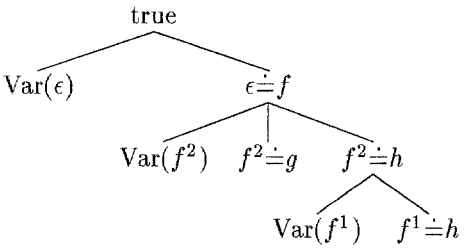
Example 6 Let g_1, g_2 be sets of terms, g, h be constructors, x, y be variables and f be the Prolog predicate defined by

$$f(x, g(y)) :- g_1. \quad f(g(x), h(y)) :- g_2.$$

The constraints associated to this definition are

$$\begin{aligned} C_1 &= (\mathrm{Var}(\epsilon) \vee \epsilon \doteq f) \wedge (\mathrm{Var}(f^2) \vee f^2 \doteq g), \\ C_2 &= (\mathrm{Var}(\epsilon) \vee \epsilon \doteq f) \wedge (\mathrm{Var}(f^1) \vee f^1 \doteq g) \wedge \\ &\quad (\mathrm{Var}(f_2) \vee f^2 \doteq h) \text{ and} \\ \Pi &= \{C_1, C_2\}. \end{aligned}$$

In order to save space in the representation of search trees, we will only represent nodes that are not implied by all of their sons. This convention lead to the representation of trees that are not binary. The optimal search tree associated to this problem is



5.2 Resolution with non linear patterns ordered by priority

This instance of the general match problem correspond to the usual pure Prolog calling mechanism. As for pattern matching, we use the intermediate notation "τ with s" where $s = \{x_1 = y_1, \ldots, x_n = y_n\}, n \geq 0$ to note non linear patterns. The match function is defined for this instance as:

$$\begin{aligned} t \text{ matches } (\tau \text{ with } s) &\text{ if and only if } \tau \text{ and } t \text{ are unifiable} \\ &\quad \text{and for any pair } x = y \in s, \\ &\quad x \text{ and } y \text{ are unifiable} \\ \mathrm{match}_\Pi(t) = i &\text{ if } p_i, \text{ matches } t \text{ and for any } j < i, \\ &\quad t \text{ does not match } p_j \end{aligned}$$

Definition 16 (Unifiability Constraint) *An* unifiability constraint *is an expression of the form* $u\dot{\uparrow}v$ *where* u *and* v *are occurrences and* $U(u\dot{\uparrow}v) = \{u, v\}$. *The valuation function for unifiability constraints is defined as follows:*

$$\begin{aligned}
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{true} \ \textit{if } u \textit{ incompatible with } t \\
&\qquad\qquad \textit{or } v \textit{ incompatible with } t \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{true} \ \textit{if } u \textit{ overpass } t \textit{ or } v \textit{ overpass } t \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{true} \ \textit{if } t/u \in X \textit{ and } t/v \in X \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{true} \ \textit{if } t/u \in X, t/v = F(\overline{t_n}) \\
&\qquad\qquad \textit{and } t/u \notin \mathrm{Var}(t/v) \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{true} \ \textit{if } t/v \in X, t/u = F(\overline{t_n}) \\
&\qquad\qquad \textit{and } t/v \notin \mathrm{Var}(t/u) \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{false} \ \textit{if } t/u = F(\overline{t_n}), t/v = G(\overline{t_n}) \\
&\qquad\qquad \textit{and } F \neq G \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{false} \ \textit{if } t/v = F(\overline{t_n}) \textit{ and } t/u \in \mathrm{Var}(t/v) \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \text{false} \ \textit{if } t/u = F(\overline{t_n}) \textit{ and } t/v \in \mathrm{Var}(t/u) \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= \mathcal{V}(\wedge_{1\leq i\leq n} u.F_i \dot{\uparrow} v.F_i)(t) \\
&\qquad\qquad \textit{if } t/u = F(\overline{t_n}) \\
&\qquad\qquad \textit{and } t/v = F(\overline{t'_n}) \\
\mathcal{V}(u\dot{\uparrow}v)(t) &= u\dot{\uparrow}v \ \textit{otherwise.}
\end{aligned}$$

Unifiability constraints can be implemented by a built-in unification procedure for terms called as an atomic operation.

To each language pattern is associated the constraint

$$\mathcal{C}(\tau \text{ with } s) = \bigwedge_{\substack{u \in O(p), \\ p/u = F(\overline{\tau_m})}} (\mathrm{Var}(u) \vee u \doteq F) \wedge \bigwedge_{\substack{x = y \in s, \\ \tau/u = x, \\ \tau/v = y}} u\dot{\uparrow}v$$

The translation of a set Π_0 of patterns is defined by

$$\mathcal{C}(\Pi_0) = \{\mathcal{C}(p_i) \wedge \bigwedge_{1\leq j<i} \neg\mathcal{C}(p_j) | 1 \leq i \leq n\}$$

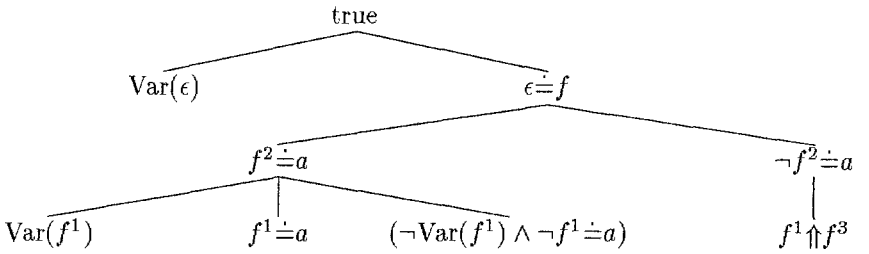
Example 7 Let g_1, g_2, g_3 be sets of terms, a, b be constants, x, y be variables and f be the Prolog predicate defined by

$$f(x, b, x) :- g_1. \quad f(a, a, x) :- g_2. \quad f(x, a, y) :- g_3.$$

The constraints associated to this definition are

$$\begin{aligned}
C_1 &= \mathrm{Var}(\epsilon) \vee (\epsilon \doteq f \wedge (\mathrm{Var}(f^2) \vee f^2 \doteq b) \wedge f^1 \dot{\uparrow} f^3), \\
C_2 &= (\mathrm{Var}(\epsilon) \vee (\epsilon \doteq f \wedge (\mathrm{Var}(f^1) \vee f^1 \doteq a) \wedge (\mathrm{Var}(f^2) \vee f^2 \doteq a))) \wedge \neg C_1, \\
C_3 &= (\mathrm{Var}(\epsilon) \vee (\epsilon \doteq f \wedge (\mathrm{Var}(f^2) \vee f^2 \doteq a))) \wedge \neg C_1 \wedge \neg C_2 \text{ and} \\
\Pi &= \{C_1, C_2, C_3\}.
\end{aligned}$$

Finally, the optimal search tree associated to this problem is



Conclusion

We have given in this work the tools needed for the application of the sequentiality methodology for the resolution of different matching problems. We showed that this is a practical approach for the meta-compilation of pattern matching constructs in programming languages by the development of different variants of the matching constructs of two different programming languages: ML and Prolog. As part of this work, we also developed two interesting and practical instances: the match of terms with non linear patterns and the use of unification as a pattern matching primitive.

References

1. G. Huet and J-J. Lévy. Call by need computations in non ambiguous linear term rewriting systems. Rapport IRIA Laboria 359, INRIA, Domaine de Voluceau, Rocquencourt BP105, 78153 Le Chesnay Cedex. FRANCE, 1979.
2. G. Huet and J-J. Lévy. Call by need computations in orthogonal term rewriting systems. In J.L. Lassez and G. Plotkin, editors, *Computational Logic*. MIT Press, 1991.
3. J.W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *J. Symbolic Computation*, 12(2), 1991.
4. A. Laville. *Evaluation paresseuse des filtrages avec priorité. Application au Langage ML*. PhD thesis, Université Paris 7, 1988. Thèse.
5. A. Laville. Implementation of lazy pattern matching algorithms. In H. Ganzinger, editor, *ESOP'88*, pages 298–316. Lecture Notes in Computer Science 300, March 1988.
6. G. Plotkin. Call-by-name, call-by-value and the λ-calculus. In *TCS*, pages 125–159, 1975.
7. L. Puel and A. Suárez. Compiling pattern matching by term decomposition. In *acm conf. on Lisp and Functional Programming*, pages 273–281. acm Press, June 1990.