Left-to-Right Tree Pattern Matching

Albert Gräf
FB Mathematik, Arbeitsgruppe Informatik
Johannes Gutenberg-Universität Mainz
Email: Graef@dmzrzu71.bitnet

Abstract

We propose a new technique to construct left-to-right matching automata for trees. Our method is based on the novel concept of *prefix unification* which is used to compute a certain *closure* of the pattern set. From the closure a kind of deterministic matching automaton can be derived immediately. We also point out how to perform the construction *incrementally* which makes our approach suitable for applications in which pattern sets change dynamically, such as in the Knuth-Bendix completion algorithm.

Our method, like most others, is restricted to *linear* patterns (the case of non-linear matching can be handled as usual by checking the consistency of variable bindings in a separate pass following the matching phase).

1 Introduction

Terms are usually interpreted as trees labelled with symbols from a ranked alphabet $\Sigma = \bigcup_{n \in \mathbf{N}_0} \Sigma_n$ s.t. each vertex labelled with a symbol α of arity n (i.e. $\alpha \in \Sigma_n$) has exactly n sons. With certain symbols from Σ_0 being distinguished as *variable symbols*, an *instance* of a Σ-tree t can be obtained by replacing leaves labelled with variable symbols by corresponding subtrees.

In this paper we focus on a restricted form of the *tree pattern matching problem* which can be stated as follows:

(*) Given a finite set $P = \{p_1, \ldots, p_n\}$ of Σ-trees (the *patterns*) and another Σ-tree q (the *subject*), determine whether q is an instance of any of the $p_i \in P$.

We generally assume that patterns are *linear* (i.e., double occurrences of variables are not allowed). Non-linear patterns can be handled as usual (do linear matching first, then check for the consistency of variable bindings).

The problem (*) arises in a variety of applications, such as the implementation of equational and functional programming languages. For most applications, the straightforward algorithm to solve (*) (successively check q against all p_i, $i = 1, \ldots, n$) is unacceptable since its running time depends on the number of patterns n.

The usual approach to solve problems like (*) efficiently is to preprocess patterns to produce some sort of *matching automaton*. The tree matching algorithms of Hoffmann and O'Donnell (see [7], and also [4,6] for a discussion of regular tree languages and bottom-up tree acceptors) and the discrimination net approach (see e.g. [3]) are of this kind. Similar work has been done in the field of functional programming language implementation [1,9,10,11]. All these techniques are based on the idea of somehow *factorizing* patterns to speed up the matching process.

We will propose a new method to construct left-to-right matching automata for trees which is easy to implement and is comparable in efficiency with the technique discussed in [11]. That is, matching time is independent of the number of patterns, and the sizes of matching automata are quite reasonable (usually much better than the exponential bounds obtained by other techniques).

A left-to-right tree matching algorithm must be able to handle ambiguities that arise when prefixes of patterns overlap. For instance, consider the patterns $p_1 = f(x, a)$ and $p_2 = f(g(x), b)$ (where x is a variable). When scanning the subject term $q = f(g(y), a)$ from left to right, we are not able to determine that q matches p_1, and does not match p_2, until we have seen the last symbol a of q. Hence the matching algorithm, after having seen the symbols f and g of q, must remember the fact that q at this point could still match the instance $p_1' = f(g(x), a)$ of p_1.

To solve this problem, we introduce the novel concept of *prefix unification* which generalizes ordinary term unification to arbitrary term prefixes. Prefix unification is used to determine overlaps between pattern prefixes, like the overlap between p_1 and p_2 at the position of x in p_1 in the example above. By simultaneously unifying all possible combinations of pattern prefixes we may then compute a (finite) *closure* of the pattern set from which a kind of deterministic matching automaton can be derived immediately.

The formal properties of prefix unification and the closure operation make it easy to systematically derive efficient algorithms for the construction of pattern set closures. In particular, we will also point out how closures can be computed *incrementally*, which is interesting in applications in which pattern sets change dynamically, such as in the Knuth-Bendix completion algorithm [8,12].

The paper is organized as follows. Section 2 introduces basic terminology. In Section 3 we develop a formal model for left-to-right tree matching automata. Section 4 covers prefix unification and the computation of closures. Section 5 states our main result concerning the construction of left-to-right matching automata. In Section 6 we briefly discuss some results related to the complexity of our matching technique. Section 7 summarizes results and discusses open problems.

2 Preliminaries

To make precise the notions of Σ-trees, terms, instances of terms and left-to-right tree matching automata mentioned in the previous section, let us first introduce some terminology.

Given a finite, nonempty alphabet Σ, by Σ^* we denote the set of all strings over Σ. $|\nu|$ is the length of a string $\nu \in \Sigma^*$, and ε is the empty string (the string of length 0). By *Pref*(ν) we denote the set of all prefixes of $\nu \in \Sigma^*$, including the empty string and ν itself. We also write *Pref*(M) for the union of all *Pref*(ν), $\nu \in M \subseteq \Sigma^*$.

We consider a finite *ranked* alphabet Σ which is the disjoint union $\Sigma = \biguplus_{n \in I\!N_0} \Sigma_n$ of alphabets Σ_n, $n \in I\!N_0$. If $\alpha \in \Sigma_n$ we say that α has *arity*, or *rank*, n, and write $\#\alpha = n$. We generally assume that there is a distinguished nullary symbol $\omega \in \Sigma_0$ which plays the role of an *anonymous variable symbol*.

The set T_Σ of Σ-*terms* is inductively defined as the least subset of Σ^* containing all $\alpha t_1 \cdots t_n$ for which $\alpha \in \Sigma_n$, $n \in I\!N_0$, and $t_1, \ldots, t_n \in T_\Sigma$. Observe that, in particular, $\alpha \in T_\Sigma$ for each $\alpha \in \Sigma_0$, and each term $\alpha t_1 \cdots t_n$ naturally represents the labelled tree with root label α and subtrees $t_1, \ldots, t_n$ (if $n = 0$, then $\alpha t_1 \cdots t_n = \alpha$ is a leave). Throughout the rest of this paper, we will use the terms "Σ-term" and "Σ-tree" synonymously.

The length $|t|$ of $t \in T_\Sigma$ as a string is also called the *size* of t, and by $h(t)$ we denote the *height* of t as a tree.

We say that $q \in T_\Sigma$ *matches*, or is an *instance* of, $p \in T_\Sigma$, written $p \leq q$, iff q can be obtained by replacing the ω's in p by Σ-terms. $\leq$ is called the *subsumption ordering* on terms. Note that $p \leq q$ implies that $|p| \leq |q|$ and $h(p) \leq h(q)$, and $p \leq q, q \leq p$ implies that $p = q$. In a slight abuse of the usual terminology we also extend the notion of subsumption to arbitrary strings over Σ, i.e. for $\nu, \mu \in \Sigma^*$ we write $\nu \leq \mu$ iff μ can be obtained by replacing ω's in ν with terms accordingly.

3 Left-to-Right Tree Matching Automata

To specify the notion of a left-to-right matching automaton on Σ-trees, we use finite state systems which in each move may either read one symbol or an entire subterm from the input tape, and change states accordingly.

More precisely, we call a *term acceptor*, or TA, for short, a finite automaton A with *state set* S, a *start state* $s_0 \in S$ and a set of *final states* $F \subseteq S$. The possible moves of A are specified by a *state transition relation* δ which consists of pairs $s\alpha \to s'$, $s, s' \in S$, $\alpha \in \Sigma$. *Configurations* of a TA A are denoted by strings $s\mu$, $s \in S$, $\mu \in \Sigma^*$, and A *accepts* a term $q \in T_\Sigma$ iff $s_0 q \xrightarrow{*} s$ s.t. $s \in F$, where $\xrightarrow{*}$ is the reflexive/transitive

$$\begin{tikzcd}
 & & s_2 \arrow[r, "\omega"] & s_3 & \\
s_0 \arrow[r, "f"'] & s_1 \arrow[ur, "a"] \arrow[r, "\omega"'] \arrow[dr, "g"'] & s_4 \arrow[r, "b"'] & s_5 & \\
 & & s_6 \arrow[r, "\omega"'] & s_7 \arrow[r, "a"'] & s_8
\end{tikzcd}$$

Figure 1: Transition diagram.

closure of the *configuration transition relation* defined by:

$$\begin{array}{lll} s\alpha\mu \rightarrow s'\mu & \text{iff} & s\alpha \rightarrow s' \in \delta,\ \alpha \in \Sigma \setminus \{\omega\} \\ su\mu \rightarrow s'\mu & \text{iff} & s\omega \rightarrow s' \in \delta,\ u \in T_\Sigma. \end{array}$$

That is, a TA A essentially is a *finite* (string) *automaton* (FA, for short) with the additional capability to skip over arbitrary subterms in the input. For a TA A, we will also consider the corresponding FA which performs the same moves as A except that in a transition on ω, the FA may only read the literal symbol ω, and not an arbitrary subterm.

3.1 Lemma. A, as a TA, accepts an input term q if and only if A, as an FA, accepts some term p s.t. $p \leq q$. □

If the transition diagram of A is *acyclic*[1], then A, as an FA, accepts only finitely many terms. Hence:

3.2 Theorem. The following are equivalent:

a) $L \subseteq T_\Sigma$ is the set of all instances of some finite pattern set $P \subseteq T_\Sigma$.

b) $L \subseteq T_\Sigma$ is accepted by an acyclic TA A. □

Theorem 3.2 justifies our choice of acyclic TA's as devices to solve the restricted tree matching problem (*) stated at the beginning of this paper. In the following, we generally assume that TA's are acyclic.

3.3 Example. A TA A accepting the term language L of all instances of some finite pattern set $P \subseteq T_\Sigma$, which even is *deterministic* as an FA, can easily be found by carrying out the following "prefix construction" (also known as the construction of a "trie" or a "discrimination net" for P): $S = \mathit{Pref}(P)$, $s_0 = \varepsilon$, $F = P$ and $\delta = \{s\alpha \rightarrow s' \mid s' = s\alpha\}$. A TA obtained from $P = \{fa\omega, f\omega b, fg\omega a\}$ in this manner ($\#f = 2$, $\#g = 1$, $\#a = \#b = 0$) is depicted in fig. 1. □

The TA of fig. 1, although deterministic as an FA, is *not* deterministic as a TA, i.e. there are inputs μ s.t. $s' \overset{*}{\leftarrow} s\mu \overset{*}{\rightarrow} s''$, but $s' \neq s''$. E.g., for $\mu = fg\omega$ we have that $s_0\mu \overset{*}{\rightarrow} s_4$, but also $s_0\mu \overset{*}{\rightarrow} s_7$. This is the kind of ambiguity in left-to-right matching of trees pointed out in Section 1.

One solution to this problem is to employ some *backtracking strategy* to find an accepting transition chain if there is one. Such a scheme has been used, e.g., in [1] and [3], but it may be quite inefficient because the subject term may have to be scanned as often as the number of patterns (asymptotically).

The second solution is to construct *deterministic TA's* for matching. Unfortunately, it turns out that for many term languages accepted by TA's there are no deterministic TA's accepting the same language, see [5] for details.

Instead, we will now propose another type of TA's, *canonical TA's*, which accept all languages acceptable by (acyclic) TA's, while still admitting a *deterministic matching strategy*.

[1]The *transition diagram* of a TA A, as usual, is obtained by connecting each pair of states s, s' with those edges labelled α for which $s\alpha \rightarrow s' \in \delta$. See fig. 1 for an example.

The idea behind canonical TA matching is as follows. Let us call a TA *weakly deterministic* (*w.d.*) iff for any pair $s \in S$, $\alpha \in \Sigma$ there is at most one transition $s\alpha \to s' \in \delta$. That is, a TA is w.d. iff its underlying string automaton is deterministic. With w.d. TA's, indeterminacy is restricted to situations in which $s\alpha\nu \to s_1\nu$ (by some $s\alpha \to s_1 \in \delta$) and $s\alpha\nu \to s_2$ (by $s\omega \to s_2 \in \delta$) for some $\alpha \in \Sigma \setminus \{\omega\}$, $\alpha\nu \in T_\Sigma$. Let us define the following *canonical* restriction of the configuration transition relation:

$$s\alpha\nu\mu \xrightarrow{c} \begin{cases} s_1\nu\mu & \text{if } s\alpha \to s_1 \in \delta,\ \alpha \neq \omega \\ s_2\mu & \text{otherwise, if } s\omega \to s_2 \in \delta,\ \alpha\nu \in T_\Sigma \end{cases}$$

That is, an ω-transition is taken only when no applicable transition on some $\alpha \neq \omega$ exists. If A is w.d., then $\xrightarrow{c}$ is a partial function by definition, and hence for any input μ and state s there is at most one canonical configuration transition chain $s\mu \xrightarrow{c*} s'$ ($s' \in S$). However, some q accepted by A may not always also be accepted through a canonical configuration transition chain. We say that a w.d. TA A is *canonical* if it meets this condition, i.e. if for any $q \in T_\Sigma$ accepted by A we also have that $s_0q \xrightarrow{c*} s$ for some $s \in F$.

Observe that the TA in fig. 1 is w.d., but not canonical, since $q = fgab \geq f\omega b$, but $s_0q \xrightarrow{c*} s_7b$ which does not lead to a final state. However, this could be fixed by introducing a new "closure pattern" $fg\omega b$ which is determined by "overlapping" the pattern $f\omega b$ and the prefix fg of the pattern $fg\omega a$. In fact, the w.d. TA constructed from the new pattern set P' obtained by adding the closure pattern $fg\omega b$ to $P = \{fa\omega, f\omega b, fg\omega a\}$ yields a canonical TA accepting the same language (note that $fg\omega b \geq f\omega b \in P$ and hence the set of instances of P' is the same as that of P).

In the following section we will develop the tools to construct pattern set closures systematically, by introducing the concept of prefix unification. Section 5 then proposes our canonical TA construction method and proves it correct.

4 Prefix Unification

While ordinary term unification is based on determining least upper bounds w.r.t. the subsumption ordering on terms, we analogously define prefix unification in terms of the following *prefix subsumption ordering*:[2]

4.1 Definition. Let $\nu, \mu \in \Sigma^*$. We write $\nu \trianglelefteq \mu$ iff some prefix of μ is an instance of ν, i.e. there is some $\mu' \in \mathit{Pref}(\mu)$ s.t. $\nu \leq \mu'$. $\trianglelefteq$ is called the *prefix subsumption ordering* on Σ^*. Note that $\trianglelefteq$ is a well-founded ordering on Σ^* with $\varepsilon \trianglelefteq \nu\ \forall\ \nu \in \Sigma^*$, $\nu \trianglelefteq \mu \Rightarrow |\nu| \leq |\mu|$, and $\nu \trianglelefteq \mu, \mu \trianglelefteq \nu \Rightarrow \nu = \mu$ for all $\nu, \mu \in \Sigma^*$. We also remark that if $p, q \in T_\Sigma$, then $p \trianglelefteq q \iff p \leq q$. □

For instance, we have that $\nu = f\omega \trianglelefteq fg\omega a = \mu$ (f, g and a as in Example 3.3), since the prefix $fg\omega$ of μ is an instance of ν.

It is easy to see that for any pair $\nu, \mu \in \Sigma^*$ there is a unique *greatest lower bound* $\nu \wedge \mu \in \Sigma^*$ w.r.t. $\trianglelefteq$ (i.e. $\nu \wedge \mu \trianglelefteq \nu, \mu$, and for any $\lambda \in \Sigma^*$ with $\lambda \trianglelefteq \nu, \mu$ we have that $\lambda \trianglelefteq \nu \wedge \mu$), which can be obtained as follows:

- $\nu \wedge \mu = \alpha(\nu' \wedge \mu')$ if $\nu = \alpha\nu'$, $\mu = \alpha\mu'$, $\alpha \in \Sigma$.
- $\nu \wedge \mu = \omega(\nu' \wedge \mu')$ if ν and μ have different head symbols and $\nu = u\nu'$, $\mu = v\mu'$, $u, v \in T_\Sigma$.
- $\nu \wedge \mu = \varepsilon$ in all other cases.

It then follows from lattice theory [2] that for any pair $\nu, \mu \in \Sigma^*$ which have a common upper bound w.r.t. $\trianglelefteq$, there is a unique least upper bound for ν and μ, denoted $\nu \vee \mu$. As usual, we may complete Σ^* with a new "top" element $\top \notin \Sigma^*$ assumed to be maximal w.r.t. $\trianglelefteq$, and set $\nu \vee \mu = \top$ for all $\nu, \mu \in \Sigma^*$ which do not have a common upper bound w.r.t. $\trianglelefteq$.

4.2 Proposition. $(\Sigma^*_\top, \trianglelefteq) := (\Sigma^* \cup \{\top\}, \trianglelefteq)$ is a complete lattice. □

[2] In fact, we will define prefix unification on arbitrary Σ-strings, but what we are actually interested in is the unification of pattern prefixes, hence the name "prefix unification."

The binary operations $\wedge$ and $\vee$ on $\Sigma^*_\top$ are commonly referred to as *meet* and *join*, respectively. Note that $\wedge$ and $\vee$ are associative, commutative, idempotent and distribute over each other. Also observe that $\vee$ applied to *terms* is really nothing but the ordinary unification of (linear) terms.

The following lemma summarizes some useful rules for the computation of joins in $\Sigma^*_\top$. We assume that $\nu\top = \top\nu = \top \; \forall \, \nu \in \Sigma^*_\top$ (extending the concatenation operation on Σ^*).

4.3 Lemma. Let $\nu, \mu \in \Sigma^*_\top$.

a) $\nu \vee \mu = \top$ if either $\nu = \top$, $\mu = \top$, or ν and μ have different head symbols $\neq \omega$.

b) $\nu \vee \varepsilon = \varepsilon \vee \nu = \nu$.

c) $\alpha\nu \vee \alpha\mu = \alpha(\nu \vee \mu) \; \forall \, \alpha \in \Sigma$.

d) $\omega\nu \vee \alpha\mu = \alpha(\omega^n\nu \vee \mu) \; \forall \, \alpha \in \Sigma_n \setminus \{\omega\}$. □

For instance, we have that $f\omega b \vee fg = f(\omega b \vee g) = fg(\omega b \vee \varepsilon) = fg\omega b$.

We now turn to the definition of the closure operation. It is useful to define the closure operation not only on pattern sets $P \subseteq T_\Sigma$, but also on any set $M \subseteq \Sigma^*$ of term suffixes for the same prefix $\lambda \in \Sigma^*$. As we shall see, this enables us to compute closures by a simple recursive algorithm (Lemma 4.6).

4.4 Definition. $M \subseteq \Sigma^*$ is called a *suffix set* if there is some $\lambda \in \Sigma^*$ s.t. $\lambda M \subseteq T_\Sigma$ (i.e. $\lambda\mu \in T_\Sigma \; \forall \, \mu \in M$). Note that, in particular, $\emptyset$ and $\{\varepsilon\}$ are suffix sets, and if M is a suffix set containing ε, then $M = \{\varepsilon\}$. Furthermore, any subset of a suffix set is again a suffix set.

We say that a suffix set M is *closed* iff for each $\mu \in M$, $\nu \in \mathit{Pref}(M)$ s.t. $\mu \vee \nu \neq \top$ we have that $\mu \vee \nu \in M$. □

In the following we generally assume that all suffix sets are finite.

4.5 Definition. For any (finite) suffix set M let the *closure* $\overline{M}$ of M be defined by $\overline{M} = \{\mu_0 \vee \nu_1 \vee \cdots \vee \nu_n \neq \top \mid \mu_0 \in M, \nu_i \in \mathit{Pref}(M), n \geq 0\}$. □

It can be shown that $\overline{M}$ is actually the *unique minimal closed suffix set* containing M. We also remark that for any $\overline{\mu} \in \overline{M}$ there are $\mu_0 \in M$, $\nu_\mu \in \mathit{Pref}(\mu)$ for each $\mu \in M$, $\mu \neq \mu_0$, s.t. $\overline{\mu}$ may be written as $\overline{\mu} = \mu_0 \vee \bigvee_{\mu \in M \setminus \{\mu_0\}} \nu_\mu$. This is because for any $\nu, \mu \in \Sigma^*$ with $\nu \trianglelefteq \mu$ (or $\nu \in \mathit{Pref}(\mu)$, in particular) we have that $\nu \vee \mu = \mu$. Hence $\overline{M}$ must be finite (on the basis of our general assumption that M is finite). Also observe that $M \subseteq \overline{M}$ and that any $\overline{\mu} \in \overline{M}$ is an instance of some $\mu_0 \in M$.

The closure operation has the usual properties justifying its name: *extensionality* ($M \subseteq \overline{M}$), *monotonicity* ($N \subseteq M \Rightarrow \overline{N} \subseteq \overline{M}$) and *idempotence* ($\overline{\overline{M}} = \overline{M}$).

The following lemma provides us with a useful recursive algorithm to compute closures. For a suffix set M and $\alpha \in \Sigma$ we let $M/\alpha = \{\mu \in \Sigma^* \mid \alpha\mu \in M\}$ be the suffix set obtained by "factoring out" the head symbol α.[3]

4.6 Lemma. Let $M \subseteq \Sigma^*$ be a suffix set.

a) $M \subseteq \{\varepsilon\} \Rightarrow \overline{M} = M$.

b) $M \not\subseteq \{\varepsilon\} \Rightarrow \overline{M} = \bigcup_{\alpha\in\Sigma} \alpha\overline{M_\alpha}$ where

$$\begin{aligned} M_\omega &= M/\omega \\ M_\alpha &= \begin{cases} M/\alpha \cup \omega^{\#\alpha} M/\omega & M/\alpha \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \end{aligned}$$

$\forall \, \alpha \neq \omega$.

[3]For completeness, let us note that we assume the following precedence of operations: first /, then concatenation, then $\wedge$ and $\vee$, and finally the set operations $\cup$ and $\setminus$.

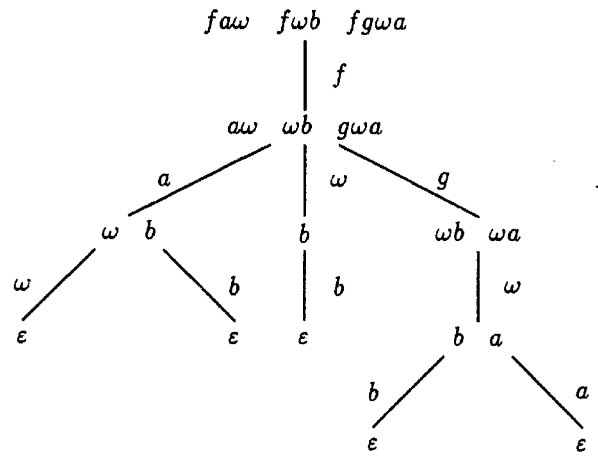

Figure 2: Computation of closed pattern set.

Proof: a) is clear from Definition 4.5. For b), we can show that $\overline{M_\alpha} = \overline{M}/\alpha \;\forall\, \alpha \in \Sigma$ using Lemma 4.3. □

Note that a computation of $\overline{M}$ following Lemma 4.6 always terminates (given a finite suffix set $M \subseteq \Sigma^*$) since each step corresponding to 4.6b) reduces the maximum size $k_M = \max_{\mu\in\overline{M}} |\mu|$ of the closure element suffixes under computation.

4.7 Example. Consider again the pattern set $P = \{fa\omega, f\omega b, fg\omega a\}$ of Example 3.3. The computation of $\overline{P}$ is depicted in fig. 2. We have that

$$\overline{P} = f\overline{P/f} = f\overline{\{a\omega, \omega b, g\omega a\}} =: f\overline{M},$$

where

$$\begin{aligned} \overline{M} &= a\overline{M_a} \cup \omega\overline{M_\omega} \cup g\overline{M_g} \\ &= a\overline{\{\omega, b\}} \cup \omega\overline{\{b\}} \cup g\overline{\{\omega b, \omega a\}} \\ &= \{a\omega, ab, \omega b, g\omega b, g\omega a\}. \end{aligned}$$

□

To compute closures incrementally by combining existing closures, we can make use of the extensionality, monotonicity and idempotence properties stated above. Suppose we want to compute $\overline{M \cup N}$ for some suffix set $M \cup N$, and have already constructed $\overline{M}$ and $\overline{N}$. We then have that $\overline{M \cup N} \subseteq \overline{\overline{M} \cup \overline{N}} \subseteq \overline{\overline{M \cup N}} = \overline{M \cup N}$ and hence $\overline{M \cup N} = \overline{\overline{M} \cup \overline{N}}$. Because $\overline{M}$ and $\overline{N}$ are already closed, to construct $\overline{\overline{M} \cup \overline{N}}$ we need only consider "mixed" closure elements $\mu \vee \nu$ s.t. $\mu \in \mathit{Pref}(\overline{M})$, $\nu \in \mathit{Pref}(\overline{N})$, as specified in the following lemma.

4.8 Lemma. Let $M \cup N$ be a suffix set s.t. M and N are closed, i.e. $\overline{M} = M$, $\overline{N} = N$. Then:

a) If $M = \emptyset$, $N = \emptyset$, or $M = N = \{\varepsilon\}$, then $\overline{M \cup N} = M \cup N$.

b) If $M \cup N \not\subseteq \{\varepsilon\}$, then $\overline{M \cup N} = \bigcup_{\alpha\in\Sigma} \alpha(\overline{M_\alpha \cup N_\alpha})$ where $M_\omega = M/\omega$, $N_\omega = N/\omega$,

$$M_\alpha = \begin{cases} M/\alpha & M/\alpha \neq \emptyset \\ \omega^{\#\alpha} M/\omega & M/\alpha = \emptyset \neq M/\omega, N/\alpha \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

$$N_\alpha = \begin{cases} N/\alpha & N/\alpha \neq \emptyset \\ \omega^{\#\alpha} N/\omega & N/\alpha = \emptyset \neq N/\omega, M/\alpha \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

$\forall\, \alpha \neq \omega$.

Proof: Apply Lemma 4.6. □

Note that if M is closed then M/α, $\alpha \in \Sigma$ is so, too. This makes it possible to apply 4.8 recursively. Also note that by 4.8a), we may stop applying 4.8b) as soon as either M or N becomes empty.

4.9 Example. In particular; Lemma 4.8 can be used to *add* new patterns to an existing closure as follows. For (closed) suffix sets M and N let $\nabla(M,N) = \overline{M \cup N}$. Given $\overline{P}$ and $Q \subseteq T_\Sigma$, compute $\overline{Q}$ and then $\nabla(\overline{P},\overline{Q})$. Note that if Q is a singleton pattern set $Q = \{q\}$ then $\overline{Q} = Q$ and hence $\overline{P \cup Q} = \nabla(\overline{P}, Q)$.

For instance, consider $P = \{fa\omega, f\omega b\}$ ($\Rightarrow \overline{P} = \{fa\omega, f\omega b, fab\}$) and $Q = \{fg\omega a\}$. Then

$$\nabla(\{fa\omega, f\omega b, fab\}, \{fg\omega a\}) = f\nabla(\{a\omega, \omega b, ab\}, \{g\omega a\}) =: fM$$

where

$$\begin{aligned} M &= a\nabla(\{\omega, b\}, \emptyset) \cup \omega\nabla(\{b\}, \emptyset) \cup g\nabla(\{\omega b\}, \{\omega a\}) \\ &= \{a\omega, ab, \omega b\} \cup g\omega\nabla(\{b\}, \{a\}) \\ &= \{a\omega, ab, \omega b\} \cup g\omega b\nabla(\{\varepsilon\}, \emptyset) \cup g\omega a\nabla(\emptyset, \{\varepsilon\}) \\ &= \{a\omega, ab, \omega b, g\omega b, g\omega a\}. \end{aligned}$$

Hence $\overline{P \cup Q} = \{fa\omega, fab, f\omega b, fg\omega b, fg\omega a\}$. □

We conclude this section with a technique to *delete* patterns from closures. Given some suffix set M, the corresponding closure $\overline{M}$ and $N \subseteq M$, we want to compute $\overline{M \setminus N}$. Our solution is based on the idea of generating closure elements induced by $M \setminus N$ "on the fly." By this means we may identify the members of $\overline{M} \setminus (\overline{M \setminus N})$ which have to be deleted from $\overline{M}$, according to the identity

$$\overline{M \setminus N} = \overline{M} \setminus (\overline{M} \setminus (\overline{M \setminus N})).$$

4.10 Lemma. Let $N \subseteq M \subseteq \Sigma^*$ be suffix sets. Then

$$\overline{M} \setminus (\overline{M \setminus N}) = \begin{cases} \emptyset & N = \emptyset \\ \overline{M} & M \setminus N = \emptyset \\ \bigcup_{\alpha \in \Sigma} \alpha(\overline{M}/\alpha \setminus (\overline{M_\alpha \setminus N_\alpha})) & \text{otherwise} \end{cases}$$

where $M_\omega = M/\omega$, $N_\omega = N/\omega$,

$$\begin{aligned} M_\alpha &= \begin{cases} M/\alpha \cup \omega^{\#\alpha} M/\omega & \overline{M}/\alpha \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \\ N_\alpha &= \begin{cases} N/\alpha \cup \omega^{\#\alpha} N/\omega & M/\alpha \setminus N/\alpha \neq \emptyset \\ M_\alpha & \text{otherwise} \end{cases} \end{aligned}$$

$\forall\, \alpha \neq \omega$.

Proof: Apply Lemma 4.6. □

Note that $N_\alpha \subseteq M_\alpha$ and, by Lemma 4.6b), $\overline{M}/\alpha = \overline{M_\alpha}\ \forall\, \alpha \in \Sigma$ which makes it possible to apply 4.10 recursively.

4.11 Example. Consider $P = \{fa\omega, f\omega b, fg\omega a\}$ ($\overline{P} = \{fa\omega, f\omega b, fg\omega a, fab, fg\omega b\}$) and suppose we want to delete $f\omega b$ from the closure.

Let $\Delta(\overline{M}, M, N) = \overline{M} \setminus (\overline{M \setminus N})$. We may compute $\Delta(\overline{P}, P, \{f\omega b\})$ recursively following Lemma 4.10. The computation is depicted in fig. 3.

In the computation tree, the elements to the left of the bar denote the members of $\overline{M}$, where the elements of $\overline{M} \setminus M$ are marked with an asterisk and the members of M are unmarked; the elements to the right of the bar denote the members of N; and the sets at the leaves of the tree, under the bar, denote the final closures computed when either N or $M \setminus N$ becomes empty, in which case $\Delta(\overline{M}, M, N) = \emptyset$ or $\Delta(\overline{M}, M, N) = \overline{M}$, respectively.

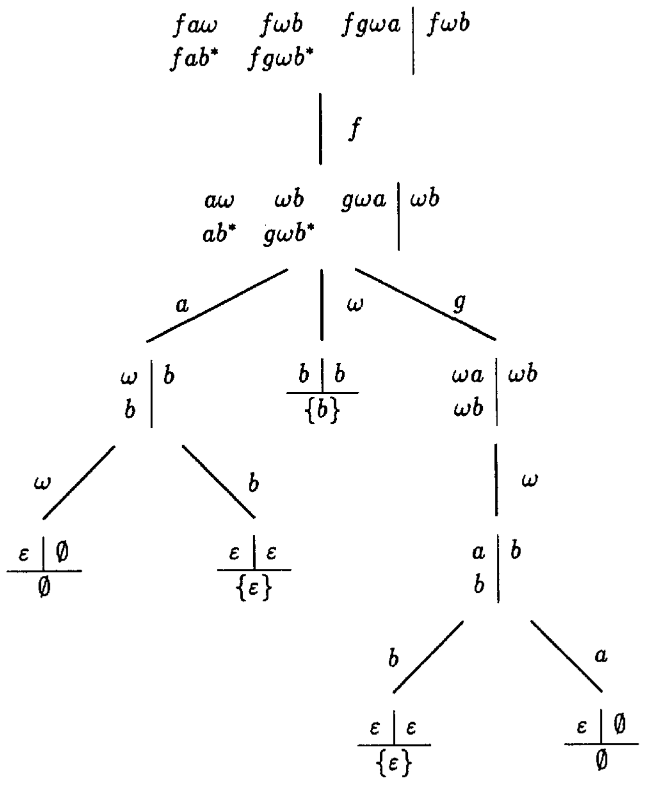

Figure 3: Pattern deletion.

For instance, we have that $\Delta(\overline{P}, P, \{f\omega b\}) = f\Delta(\overline{M}, M, N)$ where $M = \{a\omega, \omega b, g\omega a\}$, $\overline{M} = M \cup \{ab, g\omega b\}$ and $N = \{\omega b\}$. Following the edge labelled a we find that $\Delta(\overline{M}/a, M_a, N_a) = \Delta(\{\omega, b\}, \{\omega, b\}, \{b\}) = \omega\Delta(\{\varepsilon\}, \{\varepsilon\}, \emptyset) \cup b\Delta(\{\varepsilon\}, \{\varepsilon\}, \{\varepsilon\}) = \omega\emptyset \cup b\{\varepsilon\} = \{b\}$.

The final result is $\Delta(\overline{P}, P, \{f\omega b\}) = \{fab, f\omega b, fg\omega b\}$ (read the paths from the root to the leaves labelled with nonempty result sets) and hence $\overline{P \setminus \{f\omega b\}} = \overline{P} \setminus \Delta(\overline{P}, P, \{f\omega b\}) = \{fa\omega, fg\omega a\}$. □

5 Canonical TA Construction

We now are ready to state our main result and show how to effectively construct a canonical TA for any given finite pattern set P. Let $L(P) = \{q \in T_\Sigma \mid q \geq p \text{ for some } p \in P\}$ be the term language of all instances of P. The basic procedure to construct a canonical TA accepting $L(P)$ is as follows.

Construct the closure $\overline{P}$ of P, using the methods outlined in the previous section. We then have that $P \subseteq \overline{P} \subseteq L(P)$, and $\overline{P}$ is finite. Hence we may construct a deterministic FA A accepting $\overline{P}$ which also is a w.d. TA accepting $L(\overline{P}) = L(P)$. We show below that A is canonical.

It is worth noting that once the pattern set closure has been constructed, to derive the corresponding canonical TA from it is a trivial task. In fact, the closed pattern set can be stored using a tree-like representation which immediately gives the state diagram of the canonical TA (compare fig. 2 on page and fig. 5 below!); such a tree-like representation will also be helpful to implement the recursive algorithms of Lemmas 4.6, 4.8 and 4.10.

5.1 Proposition. Let $P \subseteq T_\Sigma$ be a finite pattern set and A a deterministic FA accepting $\overline{P}$. If A is *reduced* (i.e. for any non-final state s of A there is at least one final state reachable from s), then A is a canonical TA accepting $L(P)$.

Proof: A is a w.d. TA accepting $L(P)$ by construction. We have to show that A is canonical.

$$\begin{tikzcd}
 & s_1 \arrow[r, "\overline{\mu}_1 \leq \mu_1"] & \overline{s}_1 \\
s \arrow[ur, "\omega \leq u"] \arrow[dr, "\alpha"'] & & \\
 & s_2 \arrow[r, "\tilde{\mu}_2"] \arrow[dr, "\omega^n\overline{\mu}_1 \leq u_1\cdots u_n\mu_1"'] & \tilde{s}_2 \\
 & & \overline{s}_2
\end{tikzcd}$$

Figure 4: Proof of Proposition 5.1.

$$\begin{tikzcd}
 & & & s_3 & \\
 & & s_2 \arrow[ur, "\omega"] \arrow[r, "b"] & s_4 & \\
s_0 \arrow[r, "f"'] & s_1 \arrow[ur, "a"] \arrow[r, "\omega"'] \arrow[dr, "g"'] & s_5 \arrow[r, "b"'] & s_6 & \\
 & & s_7 \arrow[r, "\omega"'] & s_8 \arrow[r, "a"'] \arrow[dr, "b"'] & s_9 \\
 & & & & s_{10}
\end{tikzcd}$$

Figure 5: Canonical TA for $P = \{fa\omega, f\omega b, fg\omega a\}$.

Without loss of generality we assume that A is of the form $S = Pref(\overline{P})$, $s_0 = \varepsilon$, $F = \overline{P}$, $\delta = \{s\alpha \to s' \mid s, s' \in S, s' = s\alpha\}$ discussed in Example 3.3.[4]

Assume a state $s \in S$ and input $\mu \in \Sigma^*$ s.t. $s\mu \to s_1\mu_1 \xrightarrow{*} \overline{s}_1 \in F$, but $s\mu \overset{c}{\not\to} s_1\mu_1$. Then $\mu = u\mu_1$, $u = \alpha u_1 \cdots u_n$, $\alpha \in \Sigma_n \setminus \{\omega\}$, $u_i \in T_\Sigma$, and $s\omega \to s_1 \in \delta$ (i.e. $s_1 = s\omega$), while there is another transition $s\mu \xrightarrow{c} s_2 u_1 \cdots u_n\mu_1$ by some $s\alpha \to s_2 \in \delta$.

We have that $s_2 = s\alpha$ is the prefix of some final state $\tilde{s}_2 = s\alpha\tilde{\mu}_2 \in F$ (for some $\tilde{\mu}_2 \in \Sigma^*$). Let $\overline{s}_1 = s_1\overline{\mu}_1 = s\omega\overline{\mu}_1$ for some $\overline{\mu}_1 \in \Sigma^*$. Since $F = \overline{P}$ is closed, $\overline{s}_2 = \overline{s}_1 \vee s\alpha$ also is a final state of A. We have that $\overline{s}_2 = s\omega\overline{\mu}_1 \vee s\alpha = s(\omega\overline{\mu}_1 \vee \alpha) = s_2\omega^n\overline{\mu}_1$.

The situation is depicted in fig. 4.

Note that for any $s, s' \in S$ with $s' = s\mu'$ we have that $s\mu \xrightarrow{*} s'$ iff $\mu \geq \mu'$. Hence $s_1\mu_1 \xrightarrow{*} \overline{s}_1 \Rightarrow \mu_1 \geq \overline{\mu}_1 \Rightarrow u_1 \cdots u_n\mu_1 \geq \omega^n\overline{\mu}_1 \Rightarrow s\mu \xrightarrow{c} s_2u_1 \cdots u_n\mu_1 \xrightarrow{*} \overline{s}_2 \in F$.

We have shown that for any "critical" situation of the form

$$s_2\mu_2 \xleftarrow{c} s\mu \to s_1\mu_1 \xrightarrow{*} \overline{s}_1 \in F$$

there exists another transition chain $s_2\mu_2 \xrightarrow{*} \overline{s}_2$ leading to a final state $\overline{s}_2$. It is now easy to show, by induction on the transition relation $\to$, that A is canonical. □

5.2 Example. Consider again the pattern set $P = \{fa\omega, f\omega b, fg\omega a\}$ with $\#f = 2$, $\#g = 1$, $\#a = \#b = 0$. The closure of P is $\overline{P} = P \cup \{fab, fg\omega b\}$ (cf. Example 4.7). The corresponding canonical TA is depicted in fig. 5. □

The following lemma is useful in determining the actual patterns which have been matched in the final states of a canonical TA.

5.3 Lemma. Let A be the canonical TA constructed for some finite pattern set P, obtained through the prefix construction, and for any $s \in F = \overline{P}$ let the *match set* $P_s \subseteq P$ be defined by $P_s = \{p \in P \mid p \leq s\}$.

Then $q \geq p \in P$ iff $s_0q \xrightarrow{c*} s \in F$ s.t. $p \in P_s$.

[4]Note that if the proposition holds for this automaton A, then also for any other reduced deterministic FA A' accepting $\overline{P}$, since then there always is a surjective FA homomorphism $h : A \mapsto A'$, implying that A' essentially behaves excactly like A (though A' may identify "equivalent" states of A).

Proof: If $s_0 q \xrightarrow{c*} s \in F$, then q is an instance of s, and hence of any $p \in P_s$.

For the opposite direction, let $q \geq p \in P$. By 5.1, $s_0 q \xrightarrow{c*} s$ for some $s \in F$. We have that $q \geq s$ and $q \geq p$ and hence $q \geq s \vee p \in \overline{P}$. Now it is easy to see that since $s_0 q \xrightarrow{c*} s$ and $q \geq s \vee p \geq s$ we also have that $s_0 q \xrightarrow{c*} s \vee p$. But then $s \vee p = s$, i.e. $p \leq s$ and thus $p \in P_s$. □

5.4 Example. For our sample pattern set $P = \{fa\omega, f\omega b, fg\omega a\}$, the match sets associated with final states are as follows (cf. fig. 5):

$$\begin{array}{lll} s_3 : fa\omega & s_6 : f\omega b & s_9 : fg\omega a \\ s_4 : fa\omega, f\omega b & & s_{10} : f\omega b \end{array}$$

□

6 Complexity

In this section we briefly present some results concerning the complexity of the canonical TA technique. We assume that A is the canonical TA obtained from the pattern set P as discussed in Section 5 (using the prefix construction). We will derive bounds for both the maximum matching time of A (which is $\max_{\overline{p} \in \overline{P}} |\overline{p}|$, assuming that each move of the TA – in particular, the skipping of subtrees – may be carried out in constant time) and the number of final states of A (which is $|\overline{P}|$).

We first consider the sizes of closure patterns $\overline{p} \in \overline{P}$. It is easy to show that for any $p, q \in T_\Sigma$ and $\nu \in \mathit{Pref}(q)$ s.t. $p \vee \nu \neq \top$, we have that $|p \vee \nu| \leq |p| + |q|$ and $h(p \vee \nu) \leq \max\{h(p), h(q)\}$. Since any $\overline{p} \in \overline{P}$ is the join of some $p \in P$ with some prefixes of other patterns in P, it follows that

$$|\overline{p}| \leq \sum_{p \in P} |p| \;\forall\, \overline{p} \in \overline{P}. \tag{1}$$

and

$$h(\overline{p}) \leq \max_{p \in P} h(p) \;\forall\, \overline{p} \in \overline{P}, \tag{2}$$

i.e. $|\overline{p}| = O(r^h)$ where r is the maximum arity of the symbols in Σ and h is the maximum pattern height.

(1) is essentially the same result as that for the matching technique proposed in [11]. (2) tells us, in particular, that the maximum matching time of A is independent of the number of different patterns.

Bounds for $|\overline{P}|$ can be derived by counting the maximum number of different joins of pattern prefixes. Since any $\overline{p} \in \overline{P}$ may be written as $\overline{p} = \bigvee_{p \in P} \nu_p$ s.t. $\nu_p \in \mathit{Pref}(p)$, $p \in P$, we have that

$$|\overline{P}| \leq \prod_{p \in P} (|p| + 1). \tag{3}$$

Again, this parallels the corresponding result in [11]. Tighter bounds for $|\overline{P}|$ can be obtained by further restricting the number of different combinations ν_p, $p \in P$ to consider, depending on the structure of the pattern set. For any pair $p, q \in P$ let $n_{p,q}$ be the number of different joins $p \vee \nu \notin \{p, \top\}$ for $\nu \in \mathit{Pref}(q)$. Let k_p be the number of $n_{p,q} > 0$, $q \in P$, and let n_p be the average of all $n_{p,q} > 0$ (i.e. $n_p = 0$ if $k_p = 0$, and $n_p = \frac{1}{k_p} \sum_{n_{p,q} > 0} n_{p,q}$ otherwise). For each $p \in P$ we then have that the number of different joins $p \vee \bigvee_{q \neq p} \nu_q \in \overline{P}$ is $\leq \prod_{q \in P} (n_{p,q} + 1) \leq (n_p + 1)^{k_p}$. Hence:

$$|\overline{P}| \leq \sum_{p \in P} (n_p + 1)^{k_p}. \tag{4}$$

For instance, for the pattern set $P = \{fa\omega, f\omega b, fg\omega a\}$ used as a running example throughout this paper, we have that $\prod_{p \in P} (|p|+1) = 80$, but $(n_{fa\omega}+1)^{k_{fa\omega}} + (n_{f\omega b}+1)^{k_{f\omega b}} + (n_{fg\omega a}+1)^{k_{fg\omega a}} = 2^1 + 2^2 + 1^0 = 7$. In fact, (4) indicates that if the k_p's are bounded (i.e., there is a bound on the maximum number of patterns with nontrivial overlapping prefixes), the size of $\overline{P}$ is linear in $|P|$ and polynomial w.r.t. the n_p's, which in turn are always bounded by the size of the longest pattern.

All bounds derived in this section are sharp (asymptotically), see [5] for details. This means that there are always pathological examples of pattern sets which produce excessively large canonical TA's, and that the bound (4), in general, is no better than (3). Up to now, we have not been able to find a more specific measure for the actual size of closures which can be determined from the given pattern set P easily. We think, however, that the k_p parameters of (4) might be useful in characterizing certain classes of "good-natured" pattern sets in terms of syntactic criteria.

Some final remarks about the closure construction algorithms sketched out in Section 4 are in order. For simplicity, let us assume that all basic operations performed in these algorithms (such as test for emptiness, computation of M/α, etc.) may be carried out in constant time.

We then have that the recursive closure computation following Lemma 4.6 needs time proportional to $|\mathit{Pref}(\overline{M})|$. The construction of $\nabla(M,N) = \overline{M \cup N}$ according to 4.8 may need time $O(|\mathit{Pref}(\nabla(M,N))|)$, in the worst case, and the computation of $\Delta(\overline{M},M,N) = \overline{M} \setminus \overline{M \setminus N}$ following 4.10 needs time $O(|\mathit{Pref}(\Delta(\overline{M},M,N))|)$ (assuming that in each step we only consider those $\alpha \in \Sigma$ for which $N_\alpha \neq \emptyset$).

For 4.10 we may conclude that only those positions in closure elements need to be considered that are part of at least one element of $\overline{M}$ which is to be deleted; in this sense, 4.10 is efficient. In the case of 4.8, it is not as easy to tell the effects of a change, since we may rescan large parts of M and N which in fact may fail to produce even a single new closure element. Another difficulty is that the complexity not only depends on the "sizes" of M and N but also on the overlaps between elements of M and N. Somehow restricting the possible number of these overlaps, in a manner similar to the derivation of (4) above, might prove to be a useful approach to the problem.

7 Conclusion

In this paper we presented a new technique for left-to-right tree pattern matching, which is useful in the implementation of programming languages based on term rewriting. Our technique is comparable in efficiency with existing techniques used in functional language implementations, such as the one described in [11].

The technique is based on a new approach to the construction of left-to-right tree matching automata. The theoretical background is the novel concept of *prefix unification* which extends the normal unification of terms to term prefixes. In our framework, *canonical* matching automata admitting a deterministic matching strategy are produced by computing a *closure* of the pattern set which is determined by unifying pattern prefixes.

We showed how closures can be computed recursively by simultaneously unifying pattern prefixes, and incrementally by combining existing closures and deleting patterns from a closure.

There are several directions for future research. First of all, a detailed analysis of the incremental closure construction algorithms still needs to be done. One might then consider to apply these algorithms to Knuth-Bendix completion or similar applications involving dynamically changing pattern sets. Up to now we have only used the canonical TA matching technique to implement a simple prototype of a term rewrite system compiler without incremental features.

Secondly, there remains the question of determining *minimal* canonical TA's. Given some finite pattern set P, we may always construct a "reduced" pattern set $P_0 \subseteq P$ s.t. $L(P_0) = L(P)$ and P_0 consists only of the "most general" patterns in P (i.e., if $p, q \in P_0$ s.t. $p \leq q$ then already $p = q$). There is strong evidence that the canonical TA constructed from P_0, in some sense, is a "smallest" canonical TA accepting $L(P)$. More precisely, it seems quite reasonable that *any* weakly deterministic TA accepting $L(P)$, as an FA, must accept at least $\overline{P}_0$ in order to be canonical. Together with FA state minimization techniques this could probably be turned into a theorem characterizing the minimal canonical TA's for a given pattern set P, and a corresponding procedure to effectively construct such minimal TA's.

Finally, from the theoretical point of view, prefix unification and closure construction might be useful concepts which should be explored further. In particular, it might be interesting to extend prefix unification and closure construction to nonlinear patterns and to the subterm matching problem (find *all* matches with subterms of the subject term).

References

[1] Augustsson, Lennart: Compiling pattern matching. *Proc. Functional Programming Languages and Architectures '85.* Berlin (etc.): Springer, 1985, pp. 369–381. (Lecture Notes in Computer Science 201).

[2] Birkhoff, G.: *Lattice theory.* New York: American Mathematical Society, 1948.

[3] Christian, Jim: Fast Knuth-Bendix completion: summary. *Proc. Rewriting Techniques and Applications '89.* Berlin (etc.): Springer, 1989, pp. 551–555. (Lecture Notes in Computer Science 355).

[4] Gécseg, Ferenc; Magnus Steinby: *Tree automata.* Budapest: Akadémia Kiadó, 1984.

[5] Gräf, Albert: *Efficient pattern matching for term rewriting.* Johannes Gutenberg-Universität Mainz, 1990. (Technical Report 3/90).

[6] Hemerik, C.; J.P. Katoen: Bottom-up tree acceptors. *Science of Computer Programming*, **13**, 1989, pp. 51–72.

[7] Hoffmann, C.M.; M.J. O'Donnell: Pattern matching in trees. *Journal of the ACM*, **29**, 1, 1982, pp. 68–95.

[8] Knuth, D.E.; P.B. Bendix: Simple word problems in universal algebras. *Proc. of the Conference on Computational Problems in Abstract Algebra 1967.* Oxford: Pergamon Press, 1970, pp. 263–298.

[9] Laville, A.: Implementation of lazy pattern matching algorithms. *Proc. European Symposium on Programming '88.* Berlin (etc.): Springer, 1988, pp. 298–316. (Lecture Notes in Computer Science 300).

[10] Peyton Jones, Simon L.: *The implementation of functional programming languages.* Englewood Cliffs, N.J. (etc.): Prentice Hall, 1987. (International Series in Computer Science).

[11] Schnoebelen, Ph.: Refined compilation of pattern matching for functional languages. *Science of Computer Programming*, 11, 1988, pp. 133–159.

[12] Toyama, Yoshihito: Fast Knuth-Bendix completion with a term rewriting system compiler. *Information Processing Letters*, **32**, 1989, pp. 325–328.