

A Set Automaton to Locate All Pattern Matches in a Term

Rick Erkens
r.j.a.erkens@tue.nl

Jan Friso Groote
J.F.Groote@tue.nl

June 30, 2021

Abstract

Term pattern matching is the problem of finding all pattern matches in a subject term, given a set of patterns. Finding efficient algorithms for this problem is an important direction for research [19]. We present a new set automaton solution for the term pattern matching problem that is based on match set derivatives where each function symbol in the subject pattern is visited exactly once. The algorithm allows for various traversal patterns over the subject term and is particularly suited to search the subject term in parallel.

1 Introduction

Given a set of term patterns and a subject term, we are interested in the *subterm matching problem*, which is to find all locations in the subject term where a pattern matches. In term rewriting this corresponds to the act of finding all redexes. Typically, the matching operation must be performed for many subject terms using the same pattern set, which makes it desirable that matching is efficient. The costs of preprocessing the pattern set is less important as it is only done once.

The subterm pattern matching problem should not be confused with the *root (pattern) matching problem*. In the latter, only the matches at a specific position in the subject term are needed. There are many solutions to the root matching problem that are designed to efficiently deal with sets of patterns [19]. Moreover these solutions have been compared in a the practical setting of theorem proving [18]. A solution for the root matching problem can be applied to solve the subterm matching problem by applying it to every position in a subject term. But this solution can be expensive as many function symbols in the subject term will be inspected multiple times.

In contrast to the root matching problem, efficient solutions to the subterm matching problem are generally restricted to only a single pattern, and not to a set as is common in term rewriting. More seriously, they avoid the use of an automaton construction and process both the pattern and the subject term, which is expensive if the matching problem needs to be solved for a huge number of subject terms. Existing solutions for pattern sets are reductions from stringpath matching, which requires the resulting stringpaths to be merged in order to yield a conclusive answer. The algorithm that we propose is a mixture of an automaton and the match set approach. It is explicitly formulated for an arbitrary number of patterns, operates directly on the subject term in a top-down fashion, and directly outputs pattern-position pairs instead of stringpath matches.

We present a solution using a so-called *set automaton*. In a set automaton intermediate results are stored in a set and these stored results can be processed independently using the same automaton. This is similar to a pushdown automaton where intermediate results are stored on a stack to be processed at a later moment. A set automaton allows for massive parallel processing. This is interesting given the prediction that the next boost in computing comes from developing algorithms that are more parallel in nature [17].

Given a pattern set \mathcal{L} , we construct a deterministic automaton that prescribes a traversal of subject terms t . The automaton is executed at some position p in t , initially at the root. In each state a next transition is chosen based on the function symbol f in t at a prescribed position, which is a sub-position of p . Every function symbol of t is only inspected once. Each transition is labelled with zero or more outputs of the form $\ell@p'$, announcing a match of pattern ℓ at some position p' in the subject term.

Each transition ends in a set of next state/position pairs that must be processed further. In case the resulting set always consists of one single state/position, the set automaton behaves as an ordinary automaton. The order in which the resulting state/position pairs need to be processed is undetermined, hence the name *set automaton*. In a sequential implementation a stack or queue could be used to store these pairs giving depth-first or breadth-first strategies. But more interestingly, the new state/position pairs can be taken up by independent processors, exploring the subject term t in parallel. Note that also when running in parallel the algorithm adheres to its main asset, namely that every function symbol of t will only be inspected once.

The set automaton is generated by taking function symbol/position derivatives of match goal sets, similar to how Brzozowski derivatives work for regular expressions [2]. The derivatives are partitioned into independent classes, giving rise to the set of next states. By shifting the match goal sets back, the relative displacement through the subject term is derived allowing to calculate the position where the next state must be evaluated. This keeps the automaton finite.

The paper is organized as follows. After some preliminaries we informally discuss an example set automaton that matches associativity patterns in Section 3. Section 4 is dedicated to the set automaton construction. In Sections 5 we show that the construction is a well-defined and terminating procedure, and in Section 6 we prove that the obtained set automaton is indeed a correct and efficient solution to the subterm matching problem. In Section 7 we discuss the complexity of applying a set automaton and briefly discuss some preliminary experiments on the size of set automata. Lastly in Section 8 we share our thoughts on future work.

1.1 Related work

Many solutions for the subterm pattern matching problem focus on the time complexity or benchmarking of matching *one* pattern against *one* subject term. See for example [4, 8, 22, 9]. These methods are typically inefficient if there is a large pattern *set*, and the subject terms that need to be matched against the pattern set outnumber the subject term size and pattern size by orders of magnitude. Especially in model checking tools that use term rewriting to manipulate data [3, 10], the pattern set size is usually a fixed parameter whereas the amount of terms that need to be rewritten blows up according to state space explosion. A better solution is to preprocess the pattern set into an automaton-like data structure. Even though the preprocessing step is usually expensive, the size of the pattern set is removed as a parameter from the time complexity of the matching time. This makes the subterm matching problem efficiently solvable against a vast number of subject terms. To our knowledge, our approach is the first

top-down solution of this kind, that achieves this efficiency.

A literature study on related solutions is found in the taxonomy of [6, 5]. Hoffmann and O'Donnell [16] convert a pattern into a set of stringpaths, after which they create an Aho-Corasick automaton [1] that accepts this set of stringpaths. Cleophas, Hemerik and Zwaan report that this algorithm is closely related to their algorithm, which constructs a tree automaton from a single pattern [7]. In [5], Algorithm 6.7.9, there is a version of this algorithm that supports multiple patterns. The disadvantage of both approaches is that a subject term is scanned for matching stringpaths, rather than term pattern matches. In order to yield a conclusive answer to the term pattern matching problem, it is required to keep track which stringpaths match for every pattern, at every position in the subject term. Our set automata are built directly on the pattern set, which allows us to output pattern-position pairs directly and avoid the postprocessing step of merging stringpath matches.

Flouri et al. create a push-down automaton in [11] from a single pattern. This approach is very similar to the construction of our set automaton in the sense that match-sets are used in the automaton construction. This yields the same complexity as Hoffman and O'Donnell's bottom-up algorithm [16].

The notation and the fact that set automaton states are labelled with positions, have much in common with Adaptive Pattern Matching Automata [21], which form a solution to the root pattern matching problem.

2 Preliminaries

A signature is a sequence of disjoint, finite sets of function symbols $\mathbb{F}_0, \mathbb{F}_1, \dots, \mathbb{F}_n$ where \mathbb{F}_i consists of function symbols of arity i . We denote the arity of f by $\#f$. The set of constants is \mathbb{F}_0 , the entire signature is defined by $\mathbb{F} = \bigcup_{i=0}^n \mathbb{F}_i$ and the set of non-constants is denoted by $\mathbb{F}_{>0} = \bigcup_{i=1}^n \mathbb{F}_i$. Let $\mathbb{T}(\mathbb{F})$ be the set of terms over \mathbb{F} , defined as the smallest set that contains the variable ω , every constant, and for all $f \in \mathbb{F}_{>0}$, whenever $t_1, \dots, t_{\#f} \in \mathbb{T}(\mathbb{F})$, then also $f(t_1, \dots, t_{\#f}) \in \mathbb{T}(\mathbb{F})$. The set of closed terms $\mathbb{T}_C(\mathbb{F})$ is defined similarly, but without the clause $\omega \in \mathbb{T}_C(\mathbb{F})$. Since we only deal with linear patterns, that is, patterns in which no variable occurs twice, it is unnecessary to distinguish between the terms $f(x)$ and $f(y)$. Therefore we only use one variable ω .

A pattern over the signature \mathbb{F} is a term in $\mathbb{T}(\mathbb{F}) \setminus \{\omega\}$. We use ℓ to range over patterns. A pattern is typically the 'left-hand side' of a rewrite rule. Given a pattern $\ell = f(t_1, \dots, t_n)$, its head symbol is given by $\text{hd}(\ell) = f$. A pattern set is a finite, non-empty set of patterns. Throughout this paper we use an arbitrary pattern set denoted by \mathcal{L} .

A position is a list of positive natural numbers. We use \mathbb{P} to denote the set of all positions and we use ϵ to denote the empty list; it is referred to as the *root position*. Given two positions p, q their concatenation is denoted by $p.q$. The root position acts as a unit with respect to concatenation.

To alleviate the notation, we often denote a pair (x, p) in some set $X \times \mathbb{P}$ by $x@p$ so that the pair may be read as 'x at position p'. The term domain function $\mathcal{D} : \mathbb{T}(\mathbb{F}) \rightarrow \mathcal{P}(\mathbb{P})$ maps a term to a set of positions. That is, $\mathcal{D}(\omega) = \{\epsilon\}$, for all $a \in \mathbb{F}_0$ we have $\mathcal{D}(a) = \{\epsilon\}$, and for all $f \in \mathbb{F}_n$ with $n > 0$ we have $\mathcal{D}(f(t_1, \dots, t_n)) = \{\epsilon\} \cup \bigcup_{i \leq n} \{i.p \mid p \in \mathcal{D}(t_i)\}$.

Given a term t and a position $p \in \mathcal{D}(t)$, the subterm of t at position p is denoted by $t[p]$. A pattern ℓ matches term t on position p iff for all $p' \in \mathcal{D}(\ell)$ such that $\ell[p'] \neq \omega$ we have that $\text{hd}(t[p.p']) = \text{hd}(\ell[p'])$.

Let $\text{sub}(t)$ be the subpatterns of t , given by $\{t[p] \mid p \in \mathcal{D}(t) \text{ and } t[p] \neq \omega\}$. Since ω

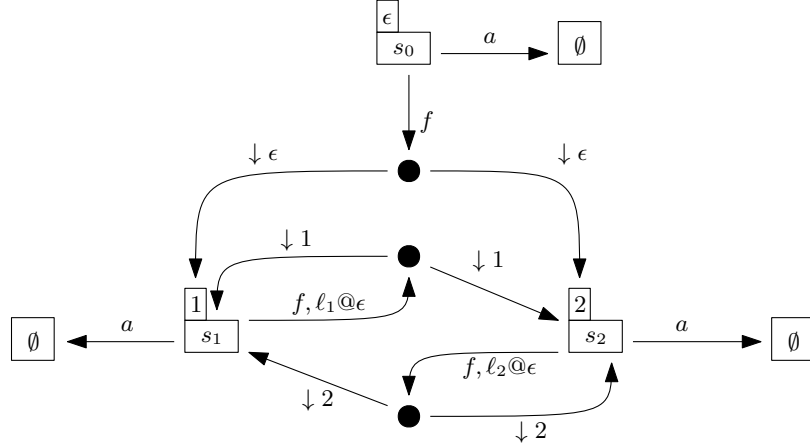


Figure 1: A set automaton for the associativity patterns.

is not a pattern, it is excluded from this set on purpose. We extend \mathcal{D} and sub to sets of terms by pointwise union. That is, $\mathcal{D}(\mathcal{L}) = \bigcup_{\ell \in \mathcal{L}} \mathcal{D}(\ell)$, and similarly for sub .

3 An example set automaton

In this section we informally discuss the example set automaton in Figure 1. It solves the term matching problem for the associativity patterns $\ell_1 = f(f(\omega, \omega), \omega)$ and $\ell_2 = f(\omega, f(\omega, \omega))$. We work in a setting with one binary function symbol f and one constant a .

We explain this automaton by applying it to the term $t = f(f(a, f(a, a)), a)$. The evaluation is done in a semi-top-down fashion. That is, in order to inspect position $p.i$ we need to have inspected position p before. We execute the automaton given a state and a *position pointer* p , which is initially state s_0 at the root position. The automaton tells us which position in t to inspect, which pattern matches are given as an output at which positions, and it tells at which state/position pairs the evaluation of the automaton must be continued.

The initial state s_0 is labelled with the root position in the box on top of it. This means that we have to inspect the function symbol in t at position ϵ relative to the position pointer p . Since the position pointer is initially ϵ , we inspect the head symbol at $t[\epsilon.\epsilon]$ which is f . There are two f -transitions from state s_0 in the automaton, which have been depicted graphically as an f -labelled arrow, going to a black dot with two outgoing arrows. If a match is found, the transition is labelled with $\ell@p'$ to indicate that pattern ℓ matches at position p' relative to the position pointer p . In this case, no such label is present on the f -labelled transition. Therefore no pattern match is reported. Furthermore, the arrows from the black dots are labelled with a relative displacement p'' indicating that the next state must be evaluated at position pointer $p.p''$. In this case, the displacement annotation $\downarrow \epsilon$ prescribes that we continue the evaluation at position pointer $\epsilon.\epsilon$. The two transitions for f go to states s_1 and s_2 indicating that both states must be evaluated independently at position ϵ . This can be done in parallel, but for simplicity we do a sequential traversal and continue in state s_1 .

We are in state s_1 and the position pointer is still ϵ . The state label of s_1 is 1, so we look at position 1 relative to the position pointer. In term $t = f(f(a, f(a, a)), a)$ we observe $\text{hd}(t[\epsilon.1]) = f$, so we take *both* f -transitions from s_1 . The arrow labelled by f ,

is accompanied by the label $\ell_1 @ \epsilon$. This means that we announce a match for pattern ℓ_1 at position ϵ relative to the position pointer. Since the position pointer is still ϵ , we announce that ℓ_1 matches t at position ϵ . From the black dot there are two outgoing arrows with the label $\downarrow 1$. This means that we continue in states s_1 and s_2 with the position pointer changed to $\epsilon.1$.

Continuing the evaluation in state s_2 at position pointer 1, we find the state label 2 on top. So, we inspect t at position 2 relative to the position pointer and find that $\text{hd}(t[1.2]) = f$. We again follow both outgoing f -transitions. First we announce a match for pattern ℓ_2 at position ϵ relative to the position pointer, so we get that t matches ℓ_2 at position 1. Following the arrows from the bottom black dot, we continue the evaluation in s_1 and s_2 with position pointer 1.2.

Now the following state/position pairs still remain to be evaluated: s_2 at position pointer ϵ , s_1 at 1, and s_1 and s_2 both at position pointer 1.2. Inspecting t at each position $p.L(s)$ where p is the position pointer and $L(s)$ is the state label, we find the constant a . Following any a -transition, the evaluation ends up in the final state, denoted by \emptyset , which means that no new state/positions pairs need to be added for evaluation.

The algorithm provides the following answer to the question “at which positions do the patterns $\ell_1 = f(f(\omega, \omega), \omega)$ and $\ell_2 = f(\omega, f(\omega, \omega))$ match the term $t = f(f(a, f(a, a)), a)$?”. The pattern ℓ_1 matches t at the root position and ℓ_2 matches t at position 1. Observe that the algorithm inspected every position of t exactly once. The construction of the automaton guarantees this efficiency, even though at every inspection occurrence of a symbol f two independent evaluations of the automaton were started.

4 Automaton construction

We describe how to create a set automaton based on *position-/function symbol derivatives*. To this end we first formally define the automaton, and in particular, what kind of information should be encoded by states.

The sets of *match obligations* MO and *match announcements* MA are respectively defined by

$$MO = \mathcal{P}(\text{sub}(\mathcal{L}) \times \mathbb{P}) \setminus \{\emptyset\} \quad MA = \mathcal{L} \times \mathbb{P}.$$

A *match goal* is a match obligation paired with a match announcement. To limit the amount of parentheses, we often denote a match goal, i.e. a pair in $MO \times MA$, by $\ell_1 @ p_1, \dots, \ell_n @ p_n \rightarrow \ell @ p$. Such a match goal should be read as: “in order to announce a match for pattern ℓ at position p , we are obliged to observe the (sub)pattern ℓ_i on position p_i , for all $1 \leq i \leq n$ ”. We denote the positions of a match obligation mo by $\text{pos}(mo)$, defined by $\text{pos}(mo) = \{p \in \mathbb{P} \mid (t, p) \in mo\}$.

A set automaton for the pattern set \mathcal{L} is a tuple $(S, s_0, L, \delta, \eta)$ where

- $S \subseteq \mathcal{P}(MO \times MA) \setminus \{\emptyset\}$ is a finite set of states;
- $s_0 \in S$ is the initial state;
- $L : S \rightarrow \mathbb{P}$ is a state labelling function;
- $\delta : S \times \mathbb{F} \rightarrow \mathcal{P}(S \times \mathbb{P})$ is a transition function;
- $\eta : S \times \mathbb{F} \rightarrow \mathcal{P}(\mathcal{L} \times \mathbb{P})$ is an output function.

The empty set serves as a final state, but it has no outgoing transitions and no output. Furthermore, a match goal of the form $\ell @ p \rightarrow \ell @ p$ is called *fresh*, and a match goal of the form $mo \rightarrow \ell @ \epsilon$ is called a *root goal*.

Example 4.1. Consider the pattern $\ell = f(f(\omega, g(\omega)), g(\omega))$. Figure 2 is a set automaton for the singleton pattern set $\{\ell\}$. It serves as a running example throughout this section and the next. The state labels are given in the small boxes on the top left of every state, and on the top right of every state there is an identifier. We have $L(s_0) = \epsilon$ and $L(s_3) = 1.2$. Formally we have $\delta(s_3, f) = \{(s_0, 1.1), (s_1, 1.2)\}$, which is depicted graphically as an f -labelled arrow going to the black dot, with two outgoing position-labelled arrows to s_0 and s_2 . The only non-empty output set is $\eta(s_3, g) = \{\ell@{\epsilon}\}$. For all other state/symbol pairs (s, h) we have $\eta(s, h) = \emptyset$. The final state \emptyset has two incoming transitions. For graphical purposes it is displayed twice.

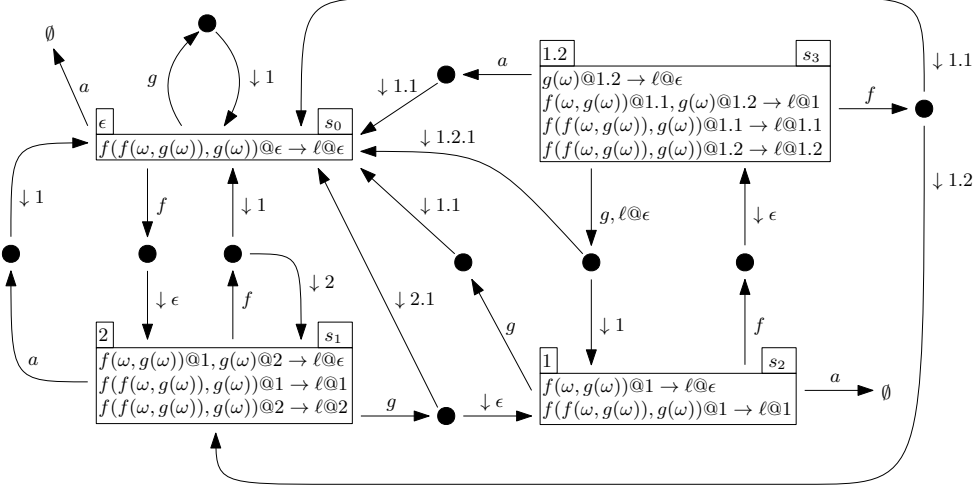


Figure 2: A set automaton for $\ell = f(f(\omega, g(\omega)), g(\omega))$.

4.1 Initial state

Let \mathcal{L} be a pattern set. We construct the automaton $M = (S, s_0, L, \delta, \eta)$ by starting with the initial state. It is labelled with the root position and its match goals are all possible fresh root goals:

$$s_0 = \{\ell@{\epsilon} \rightarrow \ell@{\epsilon} \mid \ell \in \mathcal{L}\} \quad \text{and} \quad L(s_0) = \epsilon.$$

4.2 Function symbol-position derivatives

To determine the transition relation, we introduce function symbol-position derivatives. This terminology is borrowed from Brzozowski derivatives of regular expressions [2]. From a state s with $L(s) = p$, and a symbol f , we determine the f - p -derivative of s by computing the reduced match obligations of s and adding the fresh match goal $\ell@p.i \rightarrow \ell@p.i$ for every argument i of f and every pattern $\ell \in \mathcal{L}$. Based on observing function symbol f at position p , the match obligation $\ell_1@p_1, \dots, \ell_n@p_n$ can be altered in one of four ways.

- $p = p_1$, $n = 1$ and $\ell_1 = f(\omega, \dots, \omega)$. Then $f@p$ is the last observation that was needed, so the obligation is fulfilled. The match announcement paired with this obligation is presented as a pattern match.

- $p = p_i$ for some i and $\text{hd}(\ell_i) \neq f$. Then $f@p$ contradicts with an expected observation, so the match obligation is discarded.
- $p \neq p_i$ for all i . Then $f@p$ is unrelated, so the obligation remains unchanged by this observation.
- otherwise $p = p_i$ for some i and $\text{hd}(\ell_i) = f$, but $f@p$ is only one of the many expected observations. Then $\ell_i@p_i$ is removed and the arguments of ℓ_i are added as new match obligations.

Formally, the mapping $\text{reduce} : MO \times \mathbb{F} \times \mathbb{P} \rightarrow MO \cup \{\emptyset\}$ alters the match obligation mo after the observation $f@p$ by

$$\begin{aligned} \text{reduce}(mo, f, p) = & \{\ell@q \in mo \mid q \neq p\} \cup \\ & \{\ell[i]@p.i \mid \ell@p \in mo \wedge 1 \leq i \leq \#f \wedge \ell[i] \neq \omega\}. \end{aligned}$$

Using the mapping reduce , we can define the f -derivative of state s by

$$\begin{aligned} \text{deriv}(s, f) = & \text{unchanged} \cup \text{reduced} \cup \text{fresh}, \text{ where} \\ \text{unchanged} = & \{mo \rightarrow ma \in s \mid L(s) \notin \text{pos}(mo)\} \\ \text{reduced} = & \{\text{reduce}(mo, f, L(s)) \rightarrow ma \mid mo \rightarrow ma \in s \wedge \\ & \exists \ell[\ell@L(s) \in mo \wedge \text{hd}(\ell) = f] \wedge \text{reduce}(mo, f, L(s)) \neq \emptyset\} \\ \text{fresh} = & \{\ell@L(s).i \rightarrow \ell@L(s).i \mid \ell \in \mathcal{L} \wedge 1 \leq i \leq \#f\} \end{aligned}$$

Example 4.2. Recall the pattern $\ell = f(f(\omega, g(\omega)), g(\omega))$ and the set automaton in Figure 2. Consider state s_1 . The parts of $\text{deriv}(s_1, g)$ are computed as follows:

$$\begin{aligned} \text{unchanged} = & \{f(f(\omega, g(\omega)), g(\omega))@1 \rightarrow \ell@1\} \\ \text{reduced} = & \{f(\omega, g(\omega))@1 \rightarrow \ell@e\} \\ \text{fresh} = & \{f(f(\omega, g(\omega)), g(\omega))@2.1 \rightarrow \ell@2.1\}. \end{aligned}$$

Note that the goal $f(f(\omega, g(\omega)), g(\omega))@2 \rightarrow \ell@2$ disappears completely since there is a mismatch with the expected symbol g at position 2.

4.3 Derivative partitioning

One application of deriv creates new match obligations with strictly lower positions. Repeated application of deriv therefore results in an automaton with an infinite amount of states. To solve this problem we take two more steps after computing the derivative. First, we partition the derivative into independent equivalence classes. Then, in every equivalence class, we lower the positions of all match goals as much as possible. These two measures suffice to create a finite set automaton.

Note from Example 4.2 that the derivative has two match obligations at position 1, and one match obligation at position 2.1. To obtain an efficient matching algorithm, it is important that goals with overlapping positions stay together to obtain an efficient matching algorithm. Conversely, sets of goals that are independent from each other can be separated to form a new state with fewer match goals. When evaluating a set automaton this creates the possibility of exploring parts of the subject term independently.

Given a finite subset of match obligations $X \subseteq MO$, define the *direct dependency relation* R on X for all $mo_1, mo_2 \in X$ by $mo_1 R mo_2$, iff $\text{pos}(mo_1) \cap \text{pos}(mo_2) \neq \emptyset$.

Note that R is reflexive (since MO excludes the empty set) and symmetric. But R is not transitive, since for the obligations

$$mo_1 = \{t_1@1\} \quad mo_2 = \{t_1@1, t_2@2\} \quad mo_3 = \{t_2@2\}$$

we have $mo_1 R mo_2 R mo_3$, but not $mo_1 R mo_3$. Denote the *dependency relation* on X by \sim_X , defined as the transitive closure of R . Two match obligations are said to be *dependent* iff $mo_1 \sim_X mo_2$. We extend \sim_X to match goals by $(mo_1 \rightarrow ma_1) \sim_X (mo_2 \rightarrow ma_2)$ iff $mo_1 \sim_X mo_2$. The subscript X is mostly omitted if the set is clear from the context, but note that it is necessary to define this relation separately on every state. Defining it on the set of all match obligations will simply result in the full relation $MO \times MO$.

To determine the outgoing transitions we partition $\text{deriv}(s, f)$ into equivalence classes with respect to dependency \sim on the match obligations. Each equivalence class then corresponds to a new state. The set of equivalence classes of the derivative is denoted by $[\text{deriv}(s, f)]_\sim$. We use the letter K to range over equivalence classes.

Example 4.3. Consider the computed g -derivative in Example 4.2. Partitioning yields

$$\begin{aligned} K_1 &= \{f(f(\omega, g(\omega)), g(\omega))@1 \rightarrow \ell@1f(\omega, g(\omega))@1 \rightarrow \ell@e\} \\ K_2 &= \{f(f(\omega, g(\omega)), g(\omega))@2.1 \rightarrow \ell@2.1\} \end{aligned}$$

Example 4.4. Consider the f -derivative of s_2 , which is exactly s_3 . Note that the goals $g(\omega)@1.2 \rightarrow \ell@e$ and $f(f(\omega, g(\omega)), g(\omega))@1.1 \rightarrow \ell@1.1$ are not directly dependent, but the goal $f(\omega, g(\omega))@1.1, g(\omega)@1.2 \rightarrow \ell@1$ is directly dependent to both goals. Therefore we obtain a singleton partition.

4.4 Lifting the positions of classes

Partitioning into smaller states is not enough to obtain a finite state machine since the positions of match goals are increasing. As the last part of the construction, we shorten the positions of every equivalence class. This can be done due to the following observation. Suppose that we are looking at term t on position ϵ . If all match goals say something about position 1 or lower, we can remove the prefix 1 everywhere, and start to look at term t from position 1. Inspecting position $1.p$ from the root is the same as inspecting p from position 1.

Let $\text{pos}_{MA}(K)$ denote the positions of the match announcements of K . We want to ‘lift’ every position in every goal of K by the greatest common prefix of $\text{pos}_{MA}(K)$, which we denote by $\text{gcp}(\text{pos}_{MA}(K))$. To ease the notation we write $\text{gcp}(K)$ instead of $\text{gcp}(\text{pos}_{MA}(K))$. Since all positions in a state are of the form $\text{gcp}(K).p'$, we can replace them by p' . Define $\text{lift}(s)$ by $\text{lift}(s) = \{(\text{lift}(mo), \ell@p') \mid (mo, \ell@\text{gcp}(s).p') \in s\}$ where $\text{lift}(mo) = \{\ell@p' \mid \ell@\text{gcp}(s).p' \in mo\}$.

This concludes the construction of the transition relation. For a state s and a function symbol f , we fix $\delta(s, f) = \{(\text{lift}(K), \text{gcp}(K)) \mid K \in [\text{deriv}(s, f)]_\sim\}$. Note that $\text{gcp}(K)$ is also recorded in each transition since it tells us how to traverse the term.

Example 4.5. Continuing in Example 4.3, we compute the greatest common prefix and corresponding transition for the two equivalence classes. For K_1 we have $\text{gcp}(K_1) = \text{gcp}(\{1, \epsilon\}) = \epsilon$. Then $\text{lift}(K_1) = K_1 = s_2$, and therefore $(s_2, \epsilon) \in \delta(s_1, g)$. Class K_2 has one goal with $\text{gcp}(K_2) = \text{gcp}(\{2.1\}) = 2.1$. Then $\text{lift}(K_2) = \{f(f(\omega, g(\omega)), g(\omega))@e \rightarrow \ell@e\}$, which yields the transition $(s_0, 2.1) \in \delta(s_1, g)$.

4.5 Output patterns

The output patterns after an f -transition are simply the match announcements that accompany the match obligations that reduce to \emptyset :

$$\eta(s, f) = \{ma \in MA \mid f(\omega, \dots, \omega)@L(s) \rightarrow ma \in s\}.$$

Example 4.6. Consider state s_3 in Figure 2. The goal $g(\omega)@1.2 \rightarrow \ell@e$ can be completed upon observing g at position 1.2, so we fix $\eta(s_3, g) = \{\ell@e\}$.

4.6 Position labels

For every state s there must be a position label $L(s)$ in order to construct the transitions from s . It makes sense to only choose a position from one of the match obligations. We demand the extra constraint that this position should be part of a root match goal. The construction guarantees that every state has a root goal, which we prove in detail in the next section. Similar to Adaptive Pattern Matching Automata [21], there might be multiple positions available to choose from. Any of such positions can be chosen in the construction of the automaton, but this position needs to be fixed when s is created.

4.7 Summary

The following is a summary of the construction of the set automaton.

- $s_0 = \{\ell@e \rightarrow \ell@e \mid \ell \in \mathcal{L}\};$
- $\delta(s, f) = \{(\text{lift}(K), \text{gcp}(K)) \mid K \in [\text{deriv}(s.f)]_{\sim}\};$
- $\eta(s, f) = \{ma \in MA \mid f(\omega, \dots, \omega)@L(s) \rightarrow ma \in s\};$ and
- $L(s)$ can be any $p \in \text{pos}(mo)$ for some root match goal $mo \rightarrow \ell@e \in s$.

5 Validity of the construction

In order to see that the construction algorithm of the set automaton works we need to know whether the following two properties hold. Firstly, it is necessary that $L(s)$ is a position in the match obligation of some root goal, but it is not immediately clear that every state has a root goal. Secondly, the algorithm needs to terminate. In this section we show that these properties are valid.

First we need some extra preliminaries. In the previous section we used $\text{gcp}(P)$ to denote the greatest common prefix in a set of positions. This is a lattice construct that requires more elaboration to do proofs.

Definition 5.1 (Position join-semilattice). Position p is said to be below position q , denoted by $p \leq q$, iff there is a position q' such that $p = q.q'$. Position p is strictly below q , denoted by $p < q$, if in addition $q' \neq \epsilon$. This definition makes the structure (\mathbb{P}, \leq) a join-semilattice. That is, \leq is reflexive, transitive and antisymmetric, and for each finite, non-empty set of positions P there is a unique join $\bigvee P$, which satisfies $p \leq \bigvee P$ for all $p \in P$ and whenever $p \leq r$ for all $p \in P$ then also $\bigvee P \leq r$. We call this join the greatest common prefix $\text{gcp}(P)$. We denote the join of two positions p and q by $p \vee q$. Two positions p, q are comparable if $p \leq q$ or $q \leq p$.

Proposition 5.2. The following properties hold for (sets of) positions.

- For all $p, q, r \in \mathbb{P}$ we have $p.q \leq p.r \Leftrightarrow q \leq r$;
- For all $p \in \mathbb{P}$, for all $i \in \mathbb{N}^+$ we have $p \not\leq p.i$;
- For all $p, q, r \in \mathbb{P}$, if $p \leq q$ and $p \leq r$ then q and r are comparable;
- For all $p, q \in \mathbb{P}$, if p and q are comparable then $p \vee q = p$ or $p \vee q = q$; and
- For all finite $P, Q \subseteq \mathbb{P}$ we have $\text{gcp}(P \cup Q) = \text{gcp}(P) \vee \text{gcp}(Q)$.

Lastly, consider the straightforward notion of reachable state. A state s is reachable if there is a sequence of transitions to it from s_0 . That is, s_0 is reachable and whenever s is reachable and $(s', p) \in \delta(s, f)$, then s' is also reachable. The following claims are useful in many places of the correctness proof.

Proposition 5.3. Let s be a reachable state.

- For all goals $\ell_1 @ p_1, \dots, \ell_n @ p_n \rightarrow \ell @ p$ in s we have that $p_i \leq p$ for all i .
- For all distinct $p, q \in \text{pos}_{MO}(s)$ the positions p and q are incomparable.
- For all distinct $p, q \in \text{pos}_{MO}(\text{deriv}(s, f))$ the positions p and q are incomparable.

First, we show that every reachable state always has an available root goal. By definition of the transition function, the positions of all match goals in a class K get shortened by $\text{gcp}(K)$ after partitioning. The partitioning allows us to show that $\text{gcp}(K)$ is always in $\text{pos}_{MA}(K)$.

Lemma 5.4. Let s be a reachable state. Then for all $f \in \mathbb{F}$, if $K \in [\text{deriv}(s, f)]_{\sim}$ then there is a goal $mo \rightarrow \ell @ \text{gcp}(K)$ in K .

The details of the proof can be found in the appendix; we give a sketch here. The proof is by induction on the size of K . The base case is trivial, and if $|K| \geq 2$ then K can be split into two non-empty classes with a dependency between them. By using Propositions 5.2 and 5.3, and the induction hypothesis we can show that one of the two smaller classes has a goal of the right form.

Corollary 5.5. Every reachable state has a root goal.

Next, we show that the construction terminates. There are two key observations to termination. Firstly, the lift operation always shortens the positions of derivative partitions with respect to \leq . Secondly, every state label is a match obligation position of some root goal in that state. This allows us to prove that reachable states can only have match positions in some finite set.

Lemma 5.6. Let N be the largest arity of any function symbol in \mathbb{F} , and define the set of reachable positions by $\mathcal{R} = \{p \in \mathbb{P} \mid \exists q, r, i : q \in \mathcal{D}(\mathcal{L}) \wedge r \in \mathbb{P} \wedge 1 \leq i \leq N \wedge r.p = q.i\}$. Then for all reachable states s we have that $\text{pos}_{MO}(s) \subseteq \mathcal{R}$.

The proof can be found in the appendix. Intuitively, since there are only finitely many state labels, the longest position in any match obligation is of the form $L(s).i$ where i is bounded by N .

Corollary 5.7. There are finitely many reachable states.

6 Correctness of the evaluation

The informal evaluation that was discussed in Section 3 describes how to apply an automaton M to a subject term. Formally this procedure can be defined by the mapping $\text{eval}_M : S \times \mathbb{P} \times \mathbb{T}(\mathbb{F}) \rightarrow \mathcal{P}(\mathcal{L} \times \mathbb{P})$ given by

$$\text{eval}_M(s, p, t) = \{\ell @ p.q \mid \ell @ q \in \eta(s, f)\} \cup \bigcup_{(s', p') \in \delta(s, f)} \text{eval}(s', p.p', t)$$

where $f = \text{hd}(t[p.L(s)])$. Finding all pattern matches in a term t is the invocation of $\text{eval}_M(s_0, \epsilon, t)$. The desired correctness property can then be stated as follows:

$$\text{eval}_M(s_0, \epsilon, t) = \{\ell @ p \in \mathcal{L} \times \mathbb{P} \mid \ell \text{ matches } t \text{ at } p\}.$$

This property cannot be shown by a straightforward structural induction on t . In this section we take a detour and prove an equivalent correctness claim. The proof is sketched as follows. First, we add explicit structure to the evaluation by computing an evaluation tree $ET_M(t)$ of a term t . We prove a one-to-one correspondence between the nodes of $ET_M(t)$ and t . It follows that this method of pattern matching is efficient in the sense that every position of t is inspected exactly once. Soundness and completeness is shown at the end of the section.

6.1 Evaluation trees

Definition 6.1. An *evaluation tree* for an automaton $M = (S, s_0, L, \delta, \eta)$ is a tuple (N, \rightarrow) where $N \subseteq S \times \mathbb{P}$ is a set of nodes, and $\rightarrow \subseteq N \times N$ is a set of directed edges. With a closed term t we associate an evaluation tree $ET_M(t) = (N, \rightarrow)$ defined as the smallest evaluation tree such that

- there is a root $(s_0, \epsilon) \in N$; and
- whenever $(s, p) \in N$ and $\text{hd}(t[p.L(s)]) = f$ then for every $(s', p') \in \delta(s, f)$ there is an edge $(s, p) \rightarrow (s', p.p')$ with $(s', p') \in N$.

The successors of a node n are given by $\text{Suc}(n) = \{n' \in N \mid n \rightarrow n'\}$.

Example 6.2. Figure 3 shows the term $t = f(g(a), f(f(a, g(a)), g(a)))$ and its evaluation tree $ET_M(t)$, given the set automaton M of Figure 2. There is a one-to-one correspondence between the positions of t and the nodes of the evaluation tree.

We prove that $ET_M(t)$ indeed corresponds to t in general. To this end, we define for every node the set of positions that still has to be inspected. That is, the set of work that still has to be done.

Definition 6.3. Define the mapping $\mathcal{W} : N \rightarrow \mathcal{P}(\mathcal{D}(t))$ by

$$\mathcal{W}(s, p) = \{p.q \in \mathcal{D}(t) \mid \exists r : r \in \text{pos}_{MO}(s) \wedge q \leq r\}.$$

By definition of s_0 we have $\mathcal{W}(s_0, \epsilon) = \mathcal{D}(t)$. Intuitively this makes sense, since at the beginning of the evaluation, no work is done and all the positions still have to be inspected. The mapping \mathcal{W} fixes a correspondence between an evaluation tree and the strict subset ordering $(\mathcal{D}(t), \subset)$. This follows from the following lemma. A detailed proof can be found in the appendix.

Lemma 6.4. Let $ET_M(t) = (N, \rightarrow)$ and consider an arbitrary node $(s, p) \in N$. Then

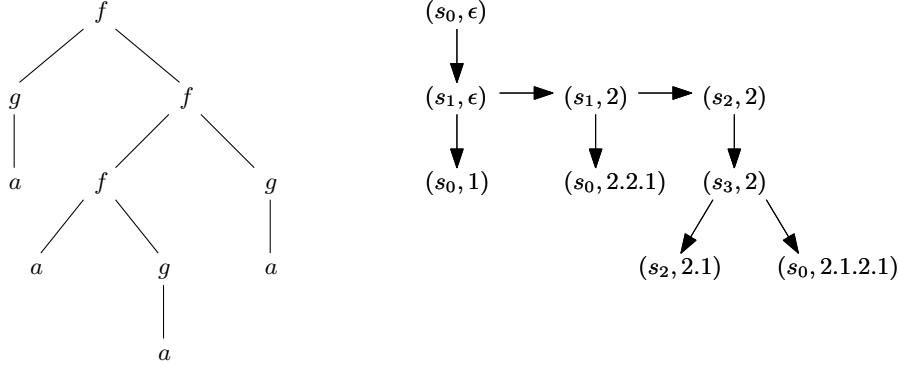


Figure 3: The term $t = f(g(a), f(f(a, g(a)), g(a)))$ on the left and its evaluation tree $ET_M(t)$ on the right.

1. For all successors $(s', p.p') \in \text{Suc}(s, p)$ we have that $p.L(s) \notin \mathcal{W}(s', p.p')$.
2. For all distinct successors $(s_1, p.p_1), (s_2, p.p_2) \in \text{Suc}(s, p)$ the sets $\mathcal{W}(s_1, p.p_1)$ and $\mathcal{W}(s_2, p.p_2)$ are disjoint.
3. We have that $\mathcal{W}(s, p) = \{p.L(s)\} \cup \bigcup_{n \in \text{Suc}(s, p)} \mathcal{W}(n)$.

By combining these properties, we get the following two corollaries.

Corollary 6.5. For all terms t , we have that $ET_M(t) = (N, \rightarrow)$ is a finite tree.

Corollary 6.6. Define $\varphi : N \rightarrow \mathcal{D}(t)$ by $\varphi(s, p) = p.L(s)$. Then φ is a bijection.

It follows that the evaluation of a term terminates, and every position is inspected exactly once. Whenever an evaluation tree node has multiple outgoing edges, it means that parallelism is possible. This parallelism preserves the efficiency of no observation being made twice.

6.2 Soundness and completeness

First, consider the following evaluation function that takes an evaluation tree node and traverses it until a leaf node is reached.

Definition 6.7. Given $ET_M(t) = (N, \rightarrow)$, define $\text{eval}_M : N \rightarrow \mathcal{P}(\mathcal{L} \times \mathbb{P})$ by

$$\text{eval}_M(s, p) = \{\ell @ p.q \mid \ell @ q \in \eta(s, \text{hd}(t[p.L(s)]))\} \cup \bigcup_{(s', p.p') \in \text{Suc}(s, p)} \text{eval}_M(s', p.p').$$

By Corollary 6.6, applying eval on the initial state from the root position is the same as retrieving the output at every level of the evaluation tree.

$$\text{eval}_M(s_0, \epsilon) = \bigcup_{(s, p) \in N} \{\ell @ p.q \mid \ell @ q \in \eta(s, \text{hd}(t[p.L(s)]))\}. \quad (1)$$

Theorem 6.12 (Correctness). For all closed terms t ,

$$\text{eval}_M(s_0, \epsilon) = \{\ell @ p \in \mathcal{L} \times \mathcal{D}(t) \mid \ell \text{ matches } t \text{ at } p\}.$$

We show both inclusions at the end of this section. The inclusion from left to right is the soundness claim. When the evaluation yields an output, then it is indeed a correct match. The inclusion from right to left is the completeness claim. When some pattern matches at some position, then the evaluation will output it at some point.

To understand soundness, consider that match goals carry history. Intuitively, a match goal $a@1, b@2 \rightarrow f(a, b)@e$ has a history of having seen f already. A state with this goal can only be reached by evaluating a term with symbol f . This notion can be formalised as follows.

Definition 6.8. The *history of an evaluation tree node* (s, p) *respects* t iff for all goals $mo \rightarrow \ell@q \in s$, for all $r \in \mathcal{D}(\ell)$ such that $\ell[r] \neq \omega$, if there is some $r' \in \text{pos}(mo)$ with $r \not\leq r'$ then $\text{hd}(t[p.r]) = \text{hd}(\ell[r])$.

With this definition, the following invariant is the key to soundness. A proof can be found in the appendix.

Lemma 6.9. Let $ET_M(t) = (N, \rightarrow)$. The history of every node (s, p) respects t .

To understand completeness, observe that upon taking derivatives a fresh match obligation is added for every new position. The partitioning then takes care of grouping the fresh goals with other goals that have the same positions.

Proposition 6.10. Whenever a state has a match obligation on position p , then it has the fresh match goal $\ell@p \rightarrow \ell@p$ for all $\ell \in \mathcal{L}$ as well.

The following invariant connects to Proposition 6.10. Intuitively, if a term matches pattern ℓ at position $p.q$, and the evaluation tree reaches a state with some goal $mo \rightarrow \ell@q$ is a match announcement, then this announcement belongs to some goal in some state visited by eval , until it is given as an output. A detailed proof can be found in the appendix.

Lemma 6.11. If ℓ matches t at $p.q$ and there is a node (s, p) and a match goal $mo \rightarrow \ell@q \in s$ then either $\ell@q \in \eta(s, \text{hd}(t[p.L(s)]))$ or there is a node $(s', p.p') \in \text{Suc}(s, p)$ such that s' has some goal $mo' \rightarrow \ell@q'$ with $q = p'.q'$.

Theorem 6.12 (Correctness). For all closed terms t ,

$$\text{eval}_M(s_0, \epsilon) = \{\ell@p \in \mathcal{L} \times \mathcal{D}(t) \mid \ell \text{ matches } t \text{ at } p\}.$$

Proof. As mentioned before, we show soundness and completeness.

- \subseteq By Equation 1 it suffices to show that for all nodes (s, p) , whenever $\ell@q \in \eta(s, \text{hd}(t[p.L(s)]))$ then ℓ matches t at $p.q$. Consider that $\text{hd}(t[p.L(s)]) = f$. By definition of η , see Section 4.5, we have $f(\omega, \dots, \omega)@L(s) \rightarrow \ell@q \in s$. By Lemma A.1, the history of node (s, p) respects t . Then for all positions $r \in \mathcal{D}(\ell)$ with $\ell[r] \neq \omega$ and $r \neq L(s)$ we have that $\text{hd}(t[p.r]) = \text{hd}(\ell[r])$. From the additional observation $\text{hd}(\ell[L(s)]) = f = \text{hd}(t[p.L(s)])$ and Proposition 5.3 it follows that ℓ matches t at $p.q$.
- \supseteq Consider that ℓ matches t at p . By Corollary 6.6, consider the node $\varphi^{-1}(p) = (s, q)$. By definition of φ we have $q.L(s) = p$. Since $L(s) \in \text{pos}_{MO}(s)$, the fresh goal $\ell@L(s) \rightarrow \ell@L(s)$ is s by Proposition 6.10. Then the repeated application of Lemma 6.11 yields a node $(s', q.q')$ such that s' has some goal $mo' \rightarrow \ell@r$ with $L(s) = q'.r$ and $\ell@r \in \eta(s', \text{hd}(t[q.q'.L(s')]))$. Then $\ell@q.q'.r \in \text{eval}(s', q.q')$ by definition of eval . Since $q.q'.r = q.L(s) = p$ it follows that $\ell@p \in \text{eval}(s', q.q')$. By Equation 1 we conclude $\ell@p \in \text{eval}(s_0, \epsilon)$.

□

Specification	Signature size	Amount of patterns	Amount of states
int	22	50	27
pos	15	46	45
nat	37	91	117
fset	15	28	23
set	20	40	24
list	16	26	24
bool	9	27	14
bag	29	44	32
fbag	18	30	25
real	30	31	31

Table 1: The set automaton sizes for parts of the default mCRL2 specification

7 Complexity and automaton size

Given an automaton M of pattern set \mathcal{L} , the matching algorithm $\text{eval}_M(s_0, t)$ runs in $O(d(n+m))$ time where n is the number of function symbols in t , and m is the amount of pattern matches in t , and d is the maximal depth of any pattern in \mathcal{L} . The factor d is due to the fact that observing a function symbol on position $L(s)$ takes $|L(s)|$ time in general.

The size of a set automaton is exponential in the worst case, which is not surprising due to similar observations concerning the root pattern matching problem. Gräf observed that a left-to-right pattern matching automaton is exponentially large in the worst case [13]. Sekar et al. observed that adaptive pattern matching automata are exponentially big in the worst case as well, although a good traversal can reduce the automaton size exponentially in some cases [21].

However, practical experiments with pattern sets show that the automaton size is small, which is in line with other forms of automaton based matching. We generated set automata to match the left hand sides of rewrite systems used in mCRL2 [14, Appendix B], see Table 1. In almost all cases the amount of states in the set automaton does not exceed the number of patterns.

The degree of freedom in the choice of state labels strongly influences the set automaton size. Consider for example the set of terms $\{t_n\}_{n \in \mathbb{N}}$ given by $t_0 = \omega$ and $t_{n+1} = f(t_n, g(\omega))$. The set automaton in Example 4.1 is generated for pattern set $\{t_2\}$. We found that the choice of state labels influences the automaton size by a quadratic factor. By choosing the right-most available position one obtains an automaton of size $2n$ for the pattern set $\{t_n\}$. A left-most strategy yields an automaton of size $n^2 + n$ for $\{t_n\}$.

8 Future work

The original motivation for this work is to construct a high performance term rewriter suited for parallel processing, which can both work on a single large term as well as on many small terms, repeatedly. This means that the matching effort must be minimal, which is provided by the automaton, and it also requires that the subject term is not transformed before matching commences. To enable term rewriting, our matching algorithm must still be extended with term rewriting along lines set out in [15]. We want to employ that we know the structure of the right-hand side of a rewrite step, minimizing inspecting known parts of a newly constructed term. Fokkink et al. have a

similar approach in [12], based on Hoffmann and O'Donnell's algorithm from [16].

Our algorithm has freedom in the position of the function symbol to be selected, as well as in the next state/position pair that the evaluator chooses. It is interesting to see whether with knowledge about the distribution of function symbols in subject terms, this freedom can be exploited to construct a most efficient set automaton. For instance, we may want to generate the first match as quickly as possible. This is particularly interesting in combination with rewriting where some sub-terms do not have to be inspected as they will be removed by the rewriting rules.

Observe that the algorithm as it stands does not employ non-linear patterns in line with matching algorithms such as [20]. But in term rewriting non-linear patterns do occur and therefore an extension to support them is desired. An extension that provides all matches in a setting where some symbols are known to be associative and/or commutative would also be interesting.

References

- [1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.
- [2] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
- [3] Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mCRL2 toolset for analysing concurrent systems - improvements in expressivity and usability. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2019.
- [4] Cédric Chauve. Tree pattern matching for linear static terms. In Alberto H. F. Laender and Arlindo L. Oliveira, editors, *String Processing and Information Retrieval, 9th International Symposium, SPIRE 2002, Lisbon, Portugal, September 11-13, 2002, Proceedings*, volume 2476 of *Lecture Notes in Computer Science*, pages 160–169. Springer, 2002.
- [5] Loek G. Cleophas. *Tree algorithms: two taxonomies and a toolkit*. PhD thesis, Eindhoven University of Technology, 2008.
- [6] Loek G. Cleophas and Kees Hemerik. Taxonomies of regular tree algorithms. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic, August 31 - September 2, 2009*, pages 146–159. Prague Stringology Club, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2009.
- [7] Loek G. Cleophas, Kees Hemerik, and Gerard Zwaan. Two related algorithms for root-to-frontier tree pattern matching. *Int. J. Found. Comput. Sci.*, 17(6):1253–1272, 2006.
- [8] Richard Cole, Ramesh Hariharan, and Piotr Indyk. Tree pattern matching and subset matching in deterministic $O(n \log^3 n)$ -time. In Robert Endre Tarjan and

- Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 245–254. ACM/SIAM, 1999.
- [9] Moshe Dubiner, Zvi Galil, and Edith Magen. Faster tree pattern matching. *J. ACM*, 41(2):205–213, 1994.
 - [10] Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The Maude LTL model checker. *Electron. Notes Theor. Comput. Sci.*, 71:162–187, 2002.
 - [11] Tomás Flouri, Costas S. Iliopoulos, Jan Janousek, Borivoj Melichar, and Solon P. Pissis. Tree template matching in ranked ordered trees by pushdown automata. *J. Discrete Algorithms*, 17:15–23, 2012.
 - [12] Wan J. Fokkink, Jasper Kamperman, and Pum Walters. Within arm’s reach: Compilation of left-linear rewrite systems via minimal rewrite systems. *ACM Trans. Program. Lang. Syst.*, 20(3):679–706, 1998.
 - [13] Albert Gräf. Left-to-right tree pattern matching. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 1991.
 - [14] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.
 - [15] Christoph M. Hoffmann and Michael J. O’Donnell. Interpreter generation using tree pattern matching. In Alfred V. Aho, Stephen N. Zilles, and Barry K. Rosen, editors, *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*, pages 169–179. ACM Press, 1979.
 - [16] Christoph M. Hoffmann and Michael J. O’Donnell. Pattern matching in trees. *J. ACM*, 29(1):68–95, 1982.
 - [17] Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. There’s plenty of room at the top: What will drive computer performance after Moore’s law? *Science*, 368(6495), 2020.
 - [18] Robert Nieuwenhuis, Thomas Hillenbrand, Alexandre Riazanov, and Andrei Voronkov. On the evaluation of indexing techniques for theorem proving. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2001.
 - [19] I. V. Ramakrishnan, R. C. Sekar, and Andrei Voronkov. Term indexing. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1853–1964. Elsevier and MIT Press, 2001.
 - [20] R. Ramesh and I. V. Ramakrishnan. Nonlinear pattern matching in trees. *J. ACM*, 39(2):295–316, 1992.
 - [21] R. C. Sekar, R. Ramesh, and I. V. Ramakrishnan. Adaptive pattern matching. *SIAM J. Comput.*, 24(6):1207–1234, 1995.

- [22] Jan Trávníček, Jan Janoušek, Bořivoj Melichar, and Loek Cleophas. On modification of Boyer-Moore-Horspool's algorithm for tree pattern matching in linearised trees. *Theoretical Computer Science*, 830:60–90, 2020.

A Appendix

A.1 Proof for Lemma 5.4

Lemma 5.4. Let s be a reachable state. Then for all $f \in \mathbb{F}$, if $K \in [\text{deriv}(s, f)]_{\sim}$ then there is a goal $mo \rightarrow \ell @ \text{gcp}(K)$ in K .

Proof. By induction. If $|K| = 1$ then the claim follows trivially. If $|K| \geq 2$, then by virtue of \sim being a transitive closure, we can partition K into subsets $K_1, K_2 \subseteq K$ such that

- there are match goals $mo_1 \rightarrow \ell_1 @ p_1 \in K_1$ and $mo_2 \rightarrow \ell_2 @ p_2 \in K_2$ such that $\text{pos}(mo_1) \cap \text{pos}(mo_2) \neq \emptyset$; and
- partitioning K_1 and K_2 with respect to \sim_{K_1} and \sim_{K_2} respectively yields K_1 and K_2 .

Pick a position p with $p \in \text{pos}(mo_1)$ and $p \in \text{pos}(mo_2)$. By Proposition 5.3 we know that $p \leq p_1$ and $p \leq p_2$. By the induction hypothesis there are two match goals $mo'_1 \rightarrow \ell'_1 @ \text{gcp}(K_1) \in K_1$ and $mo'_2 \rightarrow \ell'_2 @ \text{gcp}(K_2) \in K_2$. By properties of gcp , it follows that $p_1 \leq \text{gcp}(K_1)$ and $p_2 \leq \text{gcp}(K_2)$. By transitivity we get $p \leq \text{gcp}(K_1)$ and $p \leq \text{gcp}(K_2)$. Then by Proposition 5.2, $\text{gcp}(K_1)$ and $\text{gcp}(K_2)$ are comparable. Since

$$\text{gcp}(K) = \text{gcp}(\text{pos}_{MA}(K_1) \cup \text{pos}_{MA}(K_2)) = \text{gcp}(\text{pos}_{MA}(K_1)) \vee \text{gcp}(\text{pos}_{MA}(K_2)),$$

by adding the syntactic sugar for $\text{gcp}(K_1)$ and $\text{gcp}(K_2)$, and by applying position properties, it follows that $\text{gcp}(K) = \text{gcp}(K_1)$ or $\text{gcp}(K) = \text{gcp}(K_2)$. Since both match goals $mo'_1 \rightarrow \ell'_1 @ \text{gcp}(K_1) \in K_1$ and $mo'_2 \rightarrow \ell'_2 @ \text{gcp}(K_2) \in K_2$ are in K , we conclude the proof. \square

A.2 Proof for Lemma 5.6

Lemma 5.6. Let N be the largest arity of any function symbol in \mathbb{F} , and define the set of reachable positions by $\mathcal{R} = \{p \in \mathbb{P} \mid \exists q, r, i : q \in \mathcal{D}(\mathcal{L}) \wedge r \in \mathbb{P} \wedge 1 \leq i \leq N \wedge r.p = q.i\}$. Then for all reachable states s we have that $\text{pos}_{MO}(s) \subseteq \mathcal{R}$.

Proof. The initial state easily satisfies the claim. We show that the claim is an invariant over the production of a transition.

Let s be a reachable state and suppose that $\text{pos}_{MO}(s) \subseteq \mathcal{R}$. Let $f \in \mathbb{F}$ and consider that $(s', p') \in \delta(s, f)$. By definition $s' = \text{lift}(K)$ and $p' = \text{gcp}(K)$ for some $K \in [\text{deriv}(s, f)]_{\sim}$. We have to show that $\text{pos}_{MO}(\text{lift}(K)) \subseteq \mathcal{R}$.

Consider some position $p \in \text{pos}_{MO}(\text{lift}(K))$. By definition of deriv and lift , we have that $\text{gcp}(K).p \in \text{pos}_{MO}(K)$. Observe that \mathcal{R} is upward closed under the position prefix ordering \leq . That is, whenever $x \in \mathcal{R}$ and $x \leq y$ then $y \in \mathcal{R}$. Therefore we can ignore $\text{gcp}(K)$; it suffices to show that $p \in \mathcal{R}$.

If p is the position of an unchanged pair in some match obligation of K , then $p \in \mathcal{R}$ by assumption. If p is a position in a changed pair of some fresh or reduced match obligation, then it suffices to show that $L(s).i \in \mathcal{R}$ for all $i \leq \#f$. By construction, $L(s)$ is the position of a root goal in s . Therefore, $L(s) \in \mathcal{D}(\mathcal{L})$. Since $\#f \leq N$, we have that $i \leq N$ as well. Hence, $L(s).i \in \mathcal{R}$. \square

A.3 Proof for Lemma 6.4

Lemma 6.4. Let $ET_M(t) = (N, \rightarrow)$ and consider an arbitrary node $(s, p) \in N$. Then

1. For all successors $(s', p.p') \in \text{Suc}(s, p)$ we have that $p.L(s) \notin \mathcal{W}(s', p.p')$.
2. For all distinct successors $(s_1, p.p_1), (s_2, p.p_2) \in \text{Suc}(s, p)$ the sets $\mathcal{W}(s_1, p.p_1)$ and $\mathcal{W}(s_2, p.p_2)$ are disjoint.
3. We have that $\mathcal{W}(s, p) = \{p.L(s)\} \cup \bigcup_{n \in \text{Suc}(s, p)} \mathcal{W}(n)$.

Proof. Consider that $\text{hd}(t[p.L(s)]) = f$. By construction of M and $ET_M(t)$, we can characterise the successors of node (s, p) by

$$\text{Suc}(s, p) = \{(\text{lift}(K), p.\text{gcp}(K)) \in S \times \mathbb{P} \mid K \in [\text{deriv}(s, f)]_{\sim}\}. \quad (2)$$

1. Towards a contradiction, using Equation 2, pick an equivalence class $K \in [\text{deriv}(s, f)]_{\sim}$ and assume that $p.L(s) \in \mathcal{W}(\text{lift}(K), p.\text{gcp}(K))$. By definition of \mathcal{W} , there is a pair $\ell @ p$ in some match obligation in $\text{lift}(K)$ and some $q \leq p$ such that $p.L(s) = p.\text{gcp}(K).q$. From the position properties it follows that $L(s) = \text{gcp}(K).q$. From $q \leq p$ it follows that $\text{gcp}(K).q \leq \text{gcp}(K).p$.

Since $\ell @ p$ is part of a match obligation in $\text{lift}(K)$, by definition $\ell @ \text{gcp}(K).p$ is part of a match obligation in K . Since $K \subseteq \text{deriv}(s, f)$, there are two possibilities.

- If $\ell @ \text{gcp}(K).p = \ell @ L(s).i$ then it is part of a reduced or fresh match goal. Then $\text{gcp}(K).p = L(s).i$ for some index $1 \leq i \leq \#f$. But then by

$$L(s) = \text{gcp}(K).q \leq \text{gcp}(K).p = L(s).i,$$

we have $L(s) \leq L(s).i$, which contradicts Proposition 5.2.

- Otherwise $\ell @ \text{gcp}(K).p$ is also part of a match obligation in s . But since $L(s) \leq \text{gcp}(K).p$ and $L(s) \in \text{pos}_{MO}(s)$, it must be that $L(s) = \text{gcp}(K).p$ by Proposition 5.3. Then, by definition of **reduce** it cannot be that $\ell @ \text{gcp}(K).p$ is a match obligation of K , a contradiction.

2. By Equation 2, let $K_1, K_2 \in [\text{deriv}(s, f)]_{\sim}$ such that $s_1 = \text{lift}(K_1)$ and $s_2 = \text{lift}(K_2)$, and $p_1 = \text{gcp}(K_1)$ and $p_2 = \text{gcp}(K_2)$.

Towards a contradiction, pick a position q such that that $q \in \mathcal{W}(\text{lift}(K_1), p.\text{gcp}(K_1))$ and $q \in \mathcal{W}(\text{lift}(K_2), p.\text{gcp}(K_2))$. By definition of \mathcal{W} there are pairs $\ell_1 @ q_1$ and $\ell_2 @ q_2$ that are part of some match obligation in $\text{lift}(K_1)$ and $\text{lift}(K_2)$ respectively, and there are two positions $q'_1 \leq q_1$ and $q'_2 \leq q_2$ such that $q = p.\text{gcp}(K_1).q'_1$ and $q = p.\text{gcp}(K_2).q'_2$. Then it follows that $\text{gcp}(K_1).q'_1 = \text{gcp}(K_2).q'_2$,

By definition of **lift**, the pairs $\ell_1 @ \text{gcp}(K_1).q_1$ and $\ell_2 @ \text{gcp}(K_2).q_2$ are part of some match obligation in K_1 and K_2 respectively. But then from $\text{gcp}(K_1).q'_1 \leq \text{gcp}(K_1).q_1$ and $\text{gcp}(K_1).q'_1 \leq \text{gcp}(K_2).q_2$ it must be that $\text{gcp}(K_1).q_1$ and $\text{gcp}(K_2).q_2$ are comparable. Since $\ell_1 @ \text{gcp}(K_1).q_1$ and $\ell_2 @ \text{gcp}(K_2).q_2$ are both elements of $\text{deriv}(s, f)$, by Proposition 5.3 it follows that $\text{gcp}(K_1).q_1 = \text{gcp}(K_2).q_2$, which violates the assumption that K_1 and K_2 are distinct equivalence classes.

3. Let $\text{hd}(t[p.L(s)]) = f$. By Equation 2 we should show that

$$\mathcal{W}(s, p) = \{p.L(s)\} \cup \bigcup_{K \in [\text{deriv}(s, f)]_{\sim}} \mathcal{W}(\text{lift}(K), p.\text{gcp}(K)).$$

We prove both inclusions.

- ⊇ For the singleton set, it follows from $L(s) \in \text{pos}_{MO}(s)$ and the definition of \mathcal{W} that $p.L(s) \in \mathcal{W}(s, p)$. For the big union, consider some $K \in [\text{deriv}(s, f)]_{\sim}$ and a position $p.\text{gcp}(K).q \in \mathcal{W}(\text{lift}(K), p.\text{gcp}(K))$. By definition of \mathcal{W} there is a pair $\ell@r$ which is part of some match obligation in $\text{lift}(K)$ such that $q \leq r$. Then $\ell@\text{gcp}(K).r \in mo'$ with mo' a match obligation in K .
 From $K \in [\text{deriv}(s, f)]_{\sim}$ there are two cases. If $\ell@\text{gcp}(K).r$ is in some match obligation in s , then $p.\text{gcp}(K).q \in \mathcal{W}(s, p)$ by virtue of $\text{gcp}(K).q \leq r$ and $r \in \text{pos}_{MO}(s)$. Otherwise, $\text{gcp}(K).r = L(s).i$ for some $i \leq \#f$ and $\ell@\text{gcp}(K).r$ is part of a fresh or reduced match obligation. Since $L(s) \in \text{pos}_{MO}(s)$ there is a pair $\ell'@L(s)$ in s . Then $p.\text{gcp}(K).q \in \mathcal{W}(s, p)$ because $\text{gcp}(K).q \leq \text{gcp}(K).r = L(s).i \leq L(s)$.
- ⊆ Let mo be a match obligation in s , let $\ell@r \in mo$ and consider a position q with $q \leq r$. We have to show that $p.q \in \{p.L(s)\}$ or there is a $K \in [\text{deriv}(s, f)]_{\sim}$ with $p.q \in \mathcal{W}(\text{lift}(K), p.\text{gcp}(K))$. It suffices to distinguish two cases.
- In the case $L(s) \neq r$, then $\ell@r$ is a pair in some match obligation mo in $\text{deriv}(s, f)$. Then there is an equivalence class K such that $\ell@r$ is in some match obligation of K . Then $r = \text{gcp}(K).r'$ for some r' and $\ell@r'$ is in the match obligation $\text{lift}(mo')$ of the state $\text{lift}(K)$.
 We have to show that $p.q \in \mathcal{W}(\text{lift}(K), p.\text{gcp}(K))$. From $q \leq r$ and $r = \text{gcp}(K).r'$ we get that $q = \text{gcp}(K).r'.r''$ for some r'' . Then $p.q = p.\text{gcp}(K).r'.r''$. By definition of \mathcal{W} and from $\ell@r'$ being a match obligation in $\text{lift}(K)$, it follows that $p.q \in \mathcal{W}(\text{lift}(K), p.\text{gcp}(K))$.
 - In the case $r = L(s)$ then $p.q \leq p.L(s)$. If $q = r = L(s)$ then $p.q = p.L(s)$, which is in the singleton set $\{p.L(s)\}$. Otherwise, $q < L(s)$. Then there is an index i such that $q \leq L(s).i$. Since $p.q \in \mathcal{D}(t)$ and $\text{hd}(t[p.L(s)]) = f$ it must be that $i \leq \#f$. Then by definition of $\text{deriv}(s, f)$ there is a fresh match obligation $\ell@L(s).i \rightarrow \ell@L(s).i$ in K . By definition of lift we have that $\text{gcp}(K).r = L(s).i$ for some r and $\ell@r$ is a match obligation in $\text{lift}(K)$. Then the proof obligation follows by $p.q \leq p.L(s).i = p.\text{gcp}(K).r \in \mathcal{W}(\text{lift}(K), \text{gcp}(K))$.

□

A.4 Proof for Lemma A.1

Lemma A.1. Let $ET_M(t) = (N, \rightarrow)$. The history of every node (s, p) respects t .

Proof. The history of (s_0, ϵ) trivially respects t . Consider a node (s, p) whose history respects t , and let $f = \text{hd}(t[p.L(s)])$. Consider a successor $(\text{lift}(K), p.\text{gcp}(K)) \in \text{Suc}(s, p)$ for some $K \in [\text{deriv}(s, f)]_{\sim}$. We show that the history of $(\text{lift}(K), p.\text{gcp}(K))$ respects t as well.

Following the definition of deriv , we only look at the reduced match goals in $\text{lift}(K)$. By definition those are match goals $mo \rightarrow ma$ with some pair $f(t_1, \dots, t_n)@L(s)$. The history of the unchanged goals respects t by assumption and fresh match goals have no history. Suppose that $mo' \rightarrow \ell@p$ is a reduced match goal. Then $mo' = \text{reduce}(mo, f, L(s))$ with $mo \rightarrow ma \in s$. By definition of reduce we have that

$$mo' = \{\ell@q \in mo \mid q \neq L(s)\} \cup \{\ell[i]@L(s).i \mid \ell@L(s) \in mo \wedge 1 \leq i \leq \#f \wedge \ell[i] \neq \omega\}.$$

For all unchanged pairs $\ell@q$ with $q \neq L(s)$ we do not have to prove anything. If $\ell'@L(s)$ is a pair in mo then $\ell'[i]@L(s).i$ is a pair in mo' for all i with $\ell'[i] \neq \omega$. Hence,

$L(s) \in \mathcal{D}(\ell)$, $\ell[L(s)] \not\leq r$ and $L(s) \not\leq r$ for all goals $\ell'@p \in mo'$. So, $\text{hd}(t[p.L(s)]) = \text{hd}(\ell[L(s)])$. \square

A.5 Proof of Lemma 6.11

Lemma 6.11. If ℓ matches t at $p.q$ and there is a node (s, p) and a match goal $mo \rightarrow \ell@q \in s$ then either $\ell@q \in \eta(s, \text{hd}(t[p.L(s)]))$ or there is a node $(s', p.p') \in \text{Suc}(s, p)$ such that s' has some goal $mo' \rightarrow \ell@q'$ with $q = p'.q'$.

Proof. Suppose that $\text{hd}(t[p.L(s)]) = f$. We distinguish two cases.

- If $mo = \{f(\omega, \dots, \omega)@L(s)\}$ then $\text{reduce}(mo, f, L(s)) = \emptyset$. By construction $\ell@q \in \eta(s, \text{hd}(t[p.L(s)]))$, as needed to conclude.
- Otherwise, let $mo' = mo$ if $L(s) \notin \text{pos}(mo)$ and $mo' = \text{reduce}(mo, f, L(s))$ if $L(s) \in \text{pos}(mo)$. Note that if $L(s) \in \text{pos}(mo)$ then $\text{reduce}(mo, f, L(s))$ is not empty. Then $mo' \rightarrow \ell@q \in K$ for some $K \in [\text{deriv}(s, f)]_{\sim}$. By construction $\text{lift}(mo') \rightarrow \ell@q' \in \text{lift}(K)$ for some q' such that $q = \text{gcp}(K).q'$. Then $(\text{lift}(K), p.\text{gcp}(K))$ is the node that we are looking for.

\square