Locating Matches of Tree Patterns in Forests

Andreas Neumann and Helmut Seidl

Department of Computer Science, University of Trier, Germany
neumann,seidl@psi.uni-trier.de

Abstract. We deal with matching and locating of patterns in forests of variable arity. A pattern consists of a structural and a contextual condition for subtrees of a forest, both of which are given as tree or forest regular languages. We use the notation of constraint systems to uniformly specify both kinds of conditions. In order to implement pattern matching we introduce the class of pushdown forest automata. We identify a special class of contexts such that not only pattern matching but also locating all of a forest's subtrees matching in context can be performed in a single traversal. We also give a method for computing the reachable states of an automaton in order to minimize the size of transition tables.

1 Introduction

In Standard Generalized Markup Language (SGML) [Gol90] documents are represented as trees. A node in a document tree may have arbitrarily many children, independent of the symbol at that node. A sequence of documents or subdocuments is called a forest. A main task in document transformation and information retrieval is locating one or all of a forest's subtrees matching a pattern. A pattern is made up of a structural condition on the subtrees themselves and a contextual condition on their location in the forest, relative to its siblings and ancestors. Due to the potentially enormous size of documents, it is highly desirable to perform pattern localization "on the fly" during one pass through the document. Here, we present a solution for an interesting subclass of contextual conditions. To the best of our knowledge, no single-pass localization algorithm has been suggested so far.

A structural condition is given by a regular language over trees or forests. Tree regular languages have first been characterized (under the name of recognizable sets of pseudoterms) as projections of the derivation trees of context free grammars by [Tha67]. The author also defines a class of so-called pseudo-automata which accept the class of tree regular languages. Restricting himself to ranked trees, [Bra69] gives a slightly different definition of finite tree automata. Under the name bilanguages, [PQ68] define tree regular languages to be homomorphic images of tree-local sets. [Tak75] shows that a string representation of forest regular languages using pairs of parentheses indexed by a node's symbol is a nest language, i.e., a special case of context-free languages. [Mur96] uses pointed tree representations, as introduced by [Pod92], for specifying contextual conditions in patterns. The author then gives a method for locating matching subtrees in two passes, namely a bottom-up and a top-down traversal of the tree. The idea is similar to [BKR96]: here the bottom-up traversal is preceded by a top-down traversal, the so-called guide, labeling each node with information about its upper context. In contrast to our automaton, however, the guide may not take into account information about left siblings.

In this paper we use constraint systems as a uniform method for specifying both structural and contextual conditions in patterns. We introduce the class of pushdown forest automata, and show how they can be used for efficiently recognizing forest regular languages: during a depth-first, left-to-right traversal of the forest, each node is visited twice: once before and once immediately after processing its children. This is closely related to SGML parsing, where each node a is represented by a pair of markers $<a>$ and $$ enclosing the representation of its children. We identify an interesting subclass of contextual conditions, namely those referring only to nodes that are visited earlier in a left-to-right depth-first traversal. This class allows, e.g., searching for list items that are nested deeper than a certain level n, or locating all captions in figures. We show that the decision whether a subtree satisfies both the structural and this special class of contextual condition can already be taken when its root is visited for the second time. We also give a method for computing the reachable states of the constructed automaton.

[Tak75] already proposed so called P-tracers as a special class of pushdown automata for accepting the string representation of tree regular languages. Her construction, however, is non-deterministic, and she does not provide a method for making it deterministic. [SGYM98] use the string representation of ranked trees for the purposes of code generation. They use LR-parsing, and exploit the property that even in the non-deterministic case moves on the stack are directed by the tree structure. Thus alternative runs can be tracked simultaneously. LR-parsing is mainly a bottom-up strategy: the pushdown is used for determining possible points for reductions, i.e. whether the right hand side of a production is satisfied. In our case these points are determined by the tree structure without having to look at the pushdown. Instead, our algorithm uses the pushdown for gathering information about the part of forest visited so far. It uses this information for selecting sensible expansions when descending to a node's children, and is therefore more closely related to recursive descent parsing. A different class of pushdown tree automata was given in [Mor94]: a two-way automaton can arbitrarily change its direction from bottom-up to top-down and vice versa and thus visit each node several times. As opposed to our approach, a transition proceeds from a node to all of its children simultaneously. It is not clear how this can be efficiently implemented by a left-to-right traversal.

The remainder of this paper is organized as follows: the next section introduces our notion of trees and forests and defines constraint systems and patterns. In Section 3 we present the class of pushdown forest automata which are shown to be well suited for implementing constraint systems in the following section. Section 5 demonstrates how matches of a pattern can be located. Section 6 explains how to compute the reachable states of an automaton, and the last section discusses extensions to the pattern language.

2 Trees, Forests and Patterns

Various notations have been proposed for specifying tree regular languages, e.g. by F-local sets or F-grammars. In this paper we use constraint systems for convenient specification of forest regular languages. Let us first formalize the notion of trees and forests.

Definition 1: Let Σ be an alphabet. The sets $\mathcal{T}_\Sigma$ of *trees* t and $\mathcal{F}_\Sigma$ of *forests* f over Σ are given by the following grammar (in contrast to graph theory, a forest is a sequence rather than an unordered set of trees):

$$t \ ::= \ a\langle f\rangle, a \in \Sigma \qquad\qquad f \ ::= \ \varepsilon \ \mid \ tf$$

For brevity, we often omit the ε in a forest $t_1 \dots t_n\varepsilon$. For the remainder we refer to a fixed alphabet Σ unless explicitly stated otherwise. Thus we can omit the subscript Σ and simply write $\mathcal{T}$ and $\mathcal{F}$.

A forest language over Σ is a subset of $\mathcal{F}_\Sigma$. Constraint systems provide a convenient way of denoting forest languages:

Definition 2: A *constraint system* over Σ is a quadruple $C = (X, Y, y_\sharp, S)$, where X and Y are sets of *tree variables* and *forest variables* such that $X \cap Y = \emptyset$. $y_\sharp \in Y$ is the *start variable* of C, and S is a set of constraints[1] of the form

$$\begin{array}{ll} y \sqsupseteq \varepsilon & \text{with } y \in Y, \\ y \sqsupseteq xy_1 & \text{with } y, y_1 \in Y \text{ and } x \in X, \text{ or} \\ x \sqsupseteq a\langle y\rangle & \text{with } x \in X, y \in Y \text{ and } a \in \Sigma. \end{array}$$

We use the following naming convention: $x, x_1, \dots$ are always tree variables in X, whereas $y, y_1, \dots$ denote forest variables in Y. Variables from either X or Y are named z. When the context is clear, we abbreviate $z \sqsupseteq r \in S$ to $z \sqsupseteq r$.

In order to define the meaning of constraint systems, we need the concept of substitutions:

Definition 3: A *substitution* $\sigma : X \cup Y \to 2^{\mathcal{T}} \cup 2^{\mathcal{F}}$ assigns sets of trees to tree variables and sets of forests to forest variables. We can now define the meaning function $[\![.]\!]$ for right hand sides of constraints w.r.t. an input substitution σ:

$$[\![\varepsilon]\!]\,\sigma = \{\varepsilon\} \qquad\qquad [\![a\langle y\rangle]\!]\,\sigma = \{a\langle f\rangle \mid f \in \sigma y\}$$
$$[\![xy]\!]\,\sigma = \{tf \mid t \in \sigma x, f \in \sigma y\}$$

The least solution $[\![C]\!]$ of a constraint system C is obtained by fixpoint iteration, i.e., in terms of a sequence of substitutions $\sigma_0, \sigma_1, \dots$ The value of σ_{i+1} for variable z is obtained by evaluating all right hand sides for z w.r.t. σ_i:

$$\sigma_0\ z = \emptyset \qquad\qquad \sigma_{i+1}\ z = \bigcup_{z \sqsupseteq r} [\![r]\!]\,\sigma_i \qquad\qquad [\![C]\!]\ z = \bigcup_{i>0} \sigma_i\ z$$

[1] Note that what we consider as a constraint corresponds closely to a rule of a grammar. The constraint notation, however, has the advantage of allowing more easily for generalizations (c.f. Section 7).

The *language* of C is $\mathcal{L}_C = [\![C]\!]\ y_\sharp$. It turns out that the languages described by constraint systems coincide with the *forest regular languages* in [Tak75].

Example 1: Suppose that $\Sigma = \{a, b\}$ and consider constraint system $C = (\{x_1, x_\top\}, \{y_\sharp, y_\top\}, y_\sharp, S)$ with constraints:

$$\begin{array}{llll} x_\top \sqsupseteq a\langle y_\top\rangle & y_\top \sqsupseteq x_\top y_\top & y_\sharp \sqsupseteq x_1 y_\top & x_1 \sqsupseteq a\langle y_\top\rangle \\ x_\top \sqsupseteq b\langle y_\top\rangle & y_\top \sqsupseteq \varepsilon & y_\sharp \sqsupseteq x_\top y_\sharp & \end{array}$$

First note that $[\![C]\!]\ x_\top = \mathcal{T}$ and $[\![C]\!]\ y_\top = \mathcal{F}$, and thus x_1 represents all trees with root a. Therefore $\mathcal{L}_C$ is the language of all forests one of whose trees has a at its root. For the remainder of this paper we assume that each constraint system implicitly contains variables $x_\top$ and $y_\top$ with appropriate constraints to make them represent $\mathcal{T}$ and $\mathcal{F}$. $\square$

In document processing we are often not interested in whether a whole forest f satisfies a constraint system C. We rather want to locate *subtrees* of f satisfying a structural condition and being in a specified context. A context can be viewed as a forest that contains a hole $\bullet$ indicating the position of the desired subtree: e.g., the forest $a\langle\varepsilon\rangle\bullet b\langle\varepsilon\rangle$ is a context at the second position of a forest with three trees, the first and third one having symbols a and b and no children. A contextual condition is a set of contexts. Here we are interested in contextual conditions which can be tested just by inspecting ancestors and left siblings of $\bullet$. These are the nodes visited before $\bullet$ during a depth-first, left-to-right traversal of a forest.

Definition 4: The sets $\mathcal{T}^\circ_\Sigma$ of *pointed trees* t° and $\mathcal{F}^\circ_\Sigma$ of *pointed forests* f° over Σ are defined as follows:

$$t^\circ \ ::= \ \bullet \ \mid \ a\langle f^\circ\rangle, a \in \Sigma \qquad\qquad f^\circ \ ::= \ t^\circ f \ \mid \ t f^\circ$$

Note that similar to the definition of [Pod92], a pointed tree or forest contains exactly one hole. Again we often omit subscript Σ and simply use $\mathcal{T}^\circ$ and $\mathcal{F}^\circ$. The hole serves as a fill-in for a tree: for a pointed forest f° and a tree t, we denote by f°/t the tree obtained by replacing the $\bullet$ in f° with t.

Contextual conditions are described by an extended form of constraint systems:

Definition 5: A context system is a tuple $C^\circ = (X, X^\circ, Y, Y^\circ, y^\circ_\sharp, S^\circ)$, such that X, X°, Y, Y° are mutually disjoint, $y^\circ_\sharp \in Y^\circ$ and S is a set of constraints such that:

1) For $y^\circ \in Y^\circ$ and $y^\circ \sqsupseteq r$, either $r = x^\circ y_\top$ with $x^\circ \in X^\circ \cup \{\bullet\}$, or $r = x y^\circ_1$ with $x \in X$ and $y^\circ_1 \in Y^\circ$;
2) for $x^\circ \in X^\circ$ and $x^\circ \sqsupseteq r$, r is of the form $a\langle y^\circ\rangle$, with $y^\circ \in Y^\circ$;
3) for $z \in X \cup Y$ and $z \sqsupseteq r$, r is as in Def. 2 (and thus contains no variables from X° or Y°).

X° and Y° are called *context* variables. Naming conventions are as for the non-pointed case, except that context variables have a superscript $^\circ$. The meaning $[\![C^\circ]\!]$ of a context system is defined as for constraint systems with the addition:

$$[\![\bullet y_\top]\!]\ \sigma = \{\bullet f \mid f \in \mathcal{F}\}$$

The language of C° is $\mathcal{L}_{C^\circ} = [\![C^\circ]\!]\, y_\sharp^\circ$. Note that 1)-3) ensure that each $f \in \mathcal{L}_{C^\circ}$ contains exactly one hole. Furthermore, only $y_\top$ can occur right to a context variable. We can thus only specify *upper left contexts*.

Example 2: Let $\Sigma = \{a, b, c\}$ and $C^\circ = (\{x_\top, x\}, \{x_1^\circ\}, \{y_\top\}, \{y_\sharp^\circ, y_1^\circ, y_2^\circ\}, y_\sharp^\circ, S^\circ)$ where S° consists of the constraints:

$$\begin{array}{llll} y_\sharp^\circ \sqsupseteq x_\top y_\sharp^\circ & x_1^\circ \sqsupseteq b\langle y_1^\circ \rangle & y_1^\circ \sqsupseteq x_1^\circ y_\top & x \sqsupseteq c\langle y_\top \rangle \\ y_\sharp^\circ \sqsupseteq x_1^\circ y_\top & y_1^\circ \sqsupseteq x_\top y_\sharp^\circ & y_1^\circ \sqsupseteq x y_2^\circ & y_2^\circ \sqsupseteq \bullet y_\top \end{array}$$

C° requires $\bullet$ to be the immediate right sibling of a node labeled c which is the first child of node labeled b all of whose ancestors are b. □

We are now ready to define our notion of a pattern:

Definition 6: A *pattern system* is a pair $\Pi = (C^\circ, x_\sharp)$ such that $C^\circ = (X, X^\circ, Y, Y^\circ, y_\sharp^\circ, S^\circ)$ is a context system and $x_\sharp \in X$ is a distinguished *target variable*. The language $\mathcal{L}_\Pi$ is the set of all forests $f = f^\circ/t$ with $f^\circ \in \mathcal{L}_{C^\circ}$ and $t \in [\![C^\circ]\!] x_\sharp$. Then we also say that t *matches* $x_\sharp$ *in* f *with context* $y_\sharp^\circ$. Note that substituting $x_\sharp$ for $\bullet$ in each right hand side in S° yields a constraint system C_Π with $\mathcal{L}_\Pi = \mathcal{L}_{C_\Pi}$. The language of a pattern system is therefore a forest regular language.

Example 3: Consider again context system C° from Example 2. Adding a target variable $x_\sharp$ and a constraint $x_\sharp \sqsupseteq a\langle y_\top \rangle$ yields a pattern Π. $\mathcal{L}_\Pi$ consists of all forests containing a tree labeled a as the immediate right sibling of a tree labeled c which has no left siblings and all of whose ancestors are labeled b. □

3 Pushdown Forest Automata

How can we implement constraint systems with automata? A common approach (e.g., [Tha67], [Bra69], [Mur96]) are finite state bottom-up forest automata with a transition relation $\alpha : \Sigma \times 2^{Q^*} \times Q$ such that R is regular for all $(a, R, q) \in \alpha$. In other words, a node is assigned a state by checking the word of states assigned to its children for membership in a string regular language R over Q. Representation of R is usually by the set of all transitions (a, w, q) with $w \in R$. This has the major disadvantage that the automaton may have infinitely many transitions.

Our approach is to represent R by a finite string automaton with a separate state space $Q_\mathcal{F}$ integrated into the forest automaton. This has the effect that each subtree of the forest becomes marked with a state in $Q_\mathcal{F}$ during a run of the automaton. These marks can be used to determine whether a subtree is at a position that might satisfy a contextual condition.

As a further enhancement we adopt the depth-first, left-to-right visiting order of SGML parsers as the traversing strategy of the automaton: each node is visited twice, once before and once after its children. Thus we can, in a single traversal, first pass some finite information down to a node's children, modify it during the children's traversal and then hand it back to the parent node. We even go one step further equipping the automaton with a pushdown: only an interesting part of the information is handed to the children while the current state is stacked onto the pushdown. This stacked information is later combined with the new information returned from the children. Note that our traversing strategy is the same as used for evaluation of L-attributed grammars (e.g. [MW95]).

Definition 7: A *(non-deterministic) pushdown forest automaton* (PFA) is a tuple $P = (Q_{\mathcal{T}}, Q_{\mathcal{F}}, I, F, \mathit{Entry}, \mathit{Exit}, \mathit{Comb})$, where $Q_{\mathcal{T}}$ and $Q_{\mathcal{F}}$ are sets of *tree states* and *forest states*, $I \subseteq Q_{\mathcal{F}}$ and $F \subseteq Q_{\mathcal{F}}$ are sets of *start states* and *final states*, $\mathit{Entry} : \Sigma \to Q_{\mathcal{F}} \times Q_{\mathcal{F}}$ is the *entry function*, $\mathit{Exit} : \Sigma \to Q_{\mathcal{F}} \times Q_{\mathcal{T}}$ is the *exit function*, and $\mathit{Comb} \subseteq Q_{\mathcal{F}} \times Q_{\mathcal{T}} \times Q_{\mathcal{F}}$ is the *transition relation*. As a shorthand for $\mathit{Entry}\ a$ and $\mathit{Exit}\ a$ we use Entry_a and Exit_a.
Now we define two transition relations $\delta^P_{\mathcal{T}} \subseteq Q_{\mathcal{F}} \times \mathcal{T} \times Q_{\mathcal{T}}$ and $\delta^P_{\mathcal{F}} \subseteq Q_{\mathcal{F}} \times \mathcal{F} \times Q_{\mathcal{F}}$ (when the context is clear we omit superscript P):

$$
\begin{array}{rl}
(q, \varepsilon, q) \in \delta_{\mathcal{F}} & \text{for all } q; \\
(q_1, tf, q_2) \in \delta_{\mathcal{F}} & \text{iff there are } p \in Q_{\mathcal{T}}, q \in Q_{\mathcal{F}} \text{ such that} \\
 & \quad (q_1, t, p) \in \delta_{\mathcal{T}}, (q_1, p, q) \in \mathit{Comb} \text{ and } (q, f, q_2) \in \delta_{\mathcal{F}}. \\
(q, a\langle f \rangle, p) \in \delta_{\mathcal{T}} & \text{iff there are } q_0, q_1 \in Q_{\mathcal{F}} \text{ with } (q, q_0) \in \mathit{Entry}_a, \\
 & \quad (q_0, f, q_1) \in \delta_{\mathcal{F}}, \text{ and } (q_1, p) \in \mathit{Exit}_a.
\end{array}
$$

The *language* of P is $\mathcal{L}_P = \{f \mid (q, f, q_1) \in \delta_{\mathcal{F}} \text{ for some } q \in I, q_1 \in F\}$. P is called deterministic (DPFA) if I is a singleton, $\mathit{Entry}_a : Q_{\mathcal{F}} \to Q_{\mathcal{F}}$ and $\mathit{Exit}_a : Q_{\mathcal{F}} \to Q_{\mathcal{T}}$ are functions for all a and Comb is also a function: $Q_{\mathcal{F}} \times Q_{\mathcal{T}} \to Q_{\mathcal{F}}$. In this case $\delta_{\mathcal{T}} : Q_{\mathcal{F}} \times \mathcal{T} \to Q_{\mathcal{T}}$ and $\delta_{\mathcal{F}} : Q_{\mathcal{F}} \times \mathcal{F} \to Q_{\mathcal{F}}$ are functions as well.

Note that Comb and $Q_{\mathcal{F}}$ form a finite string automaton over $Q_{\mathcal{T}}$, with the difference that initial and accepting states are determined by Entry and Exit. When the automaton reaches a node of the input forest, the current forest state is stacked and then passed to the children through Entry_a. The forest state obtained by traversing the children is then transformed into a tree state via the Exit function, which is combined with the stacked forest state via Comb. The result is passed to the right sibling or if there is no such sibling, to the parent node via Exit.
It is worth mentioning that this is close to an idea of [Tak75]: there forests are uniquely mapped to a string representation using pairs of parentheses marked with symbols from Σ. The image of a forest regular language is then a *nest language*, i.e. a special class of context free languages which can therefore be parsed by non-deterministic pushdown string automata.
As opposed to string automata theory, we succeed in constructing for our non-deterministic pushdown forest automata equivalent deterministic ones.

Theorem 1: For each PFA P there is a DPFA D such that $\mathcal{L}_P = \mathcal{L}_D$.

The proof is by subset construction. The conventional approach is to use as states the subsets of $Q_{\mathcal{T}}$ and $Q_{\mathcal{F}}$. But this does not suffice: after the Exit_a step, a forest state q may only be combined with a tree state p if p was obtained starting from an initial state in $\mathit{Entry}_a\ q$ (for the same q!). This relationship is not determinable with sets of tree or forest states. Instead we need sets of pairs (q, p) as tree states and sets of pairs (q, q_1) as forest states, indicating that state p or q_1 was obtained as a consequence of $\mathit{Entry}_a\ q$. The details of the construction are omitted for brevity. □

A special case of PFAs are those where $\mathit{Entry}_a\, q = I$ for all a and q. These automata correspond closely to the finite state bottom-up automata mentioned at the beginning of this section and are therefore called *bottom-up forest automata* (BFAs). Although BFAs do not use the pushdown, they accept the same languages as PFAs:

Lemma 1: For each PFA P there is a BFA B with $\mathcal{L}_P = \mathcal{L}_B$.

We only give the idea of the proof: B simulates P's $\mathit{Entry}_a\, q$ step by guessing a and q non-deterministically. As forest states, B has triples (q, a, q_1) indicating that q_1 was reached due to guessing q and a. *Exit* can then check whether a was guessed correctly, yielding a tree state (q, p) meaning that p was obtained as a consequence of guessing q. The *Comb* function can then check whether q was the correct guess. □

We conclude that the pushdown does add nothing to expressiveness as compared to bottom-up automata. It may, however, add to succinctness of representation: it can drastically reduce the size of the automaton in the deterministic case:

Lemma 2: There is a sequence of languages $\mathcal{L}^1, \mathcal{L}^2, \ldots$ such that for all $n > 0$, there is a DPFA P accepting $\mathcal{L}^n$ with $O(n)$ states whereas each DBFA B accepting $\mathcal{L}^n$ has at least $O(2^n)$ states.

Proof: let $\mathcal{L}^n$ be the language of all unary trees having symbol a at the node of depth n. For $n > 0$, $\mathcal{L}^n$ is accepted by $P = (\{-1, 0\}, \{-1, 0, \ldots, n, \infty\}, \{1\}, \{0\}, \mathit{Entry}, \mathit{Exit}, \mathit{Comb})$ with

$$\begin{array}{rcll@{\qquad}rcll}
\mathit{Entry}_b\, i &=& i{+}1, & 1 \le i < n, b \in \Sigma & \mathit{Exit}_b\, q &=& 0, & b \in \Sigma, q \in \{0, \infty\} \\
\mathit{Entry}_a\, n &=& \infty & & \mathit{Exit}_b\, q &=& -1, & b \in \Sigma, q \notin \{0, \infty\} \\
\mathit{Entry}_b\, n &=& -1, & b \ne a, b \in \Sigma & \mathit{Comb}(q, 0) &=& 0, & q \in \{1, \ldots, n, \infty\} \\
\mathit{Entry}_b\, \infty &=& \infty, & b \in \Sigma & \mathit{Comb}(p, q) &=& -1, & p \in \{0, -1\} \\
\mathit{Entry}_b\, q &=& -1, & q \in \{0, -1\}, b \in \Sigma & & & & \text{or } q = -1
\end{array}$$

Here, -1 is an error state, whereas 0 indicates that the first tree of a forest has been successfully traversed. For $1 \le i \le n$, state i means that level i has just been entered, and ∞ represents a depth greater than n. Now it is easy to see that P accepts $\mathcal{L}^n$, and it has $n{+}5 = O(n)$ states. Now suppose that there is a DBFA B with initial state q_I accepting $\mathcal{L}^n$. Then, for deciding whether a tree t is in $\mathcal{L}^n$, $\delta_{\mathcal{T}}(q_I, t)$ must denote whether t has symbol a at depth n. But for deciding whether $b\langle t\rangle \in \mathcal{L}^n$, it must contain that information also for depth $n{-}1$, and analogously for $n{-}2, \ldots, 1$. The smallest domain capable of expressing this information consists of all subsets of $\{1, \ldots, n\}$, and there are 2^n such sets. Thus, B must have at least 2^n tree states. □

We conclude that for some languages, our pushdown automata are exponentially more succinct than, e.g., the bottom-up automata of [Tha67] or the LR-automata of [SGYM98]. Observe that the decisive point in the last example is the PFA's awareness of the left upper context. Another approach to enhancing BFAs is to provide knowledge only about the parent node's symbol. This leads to a subclass of PFAs where $\mathit{Entry}_a\, q$ is equal for all q and thus depends only on a. They are related to the approach of [BMW91] for trees of fixed arity. Here an n-ary transition function: $Q^n \times \Sigma \to Q$ is simulated by a finite automaton with states Q_a for each element a of Σ, requiring a unary transition function: $Q_a \times Q \to Q_a$ only. The initial state for a is a weak version of our Entry_a. Though this class of automata is more succinct than BFAs, it still requires $\Omega(2^n)$ states for the above example in the deterministic case.

4 The Construction

In [NS98] a one-to-one correspondence between constraint systems and BFAs is established. Especially it is shown that for each constraint system an equivalent DBFA can be constructed, that has $2^{Y\times Y}$ as forest states. A BFA, however, is not aware of the upper context and therefore unsuited for checking contextual conditions. Therefore we will now give an algorithm for constructing, for a given constraint system C, an equivalent DPFA D_C with the same set of forest states $2^{Y\times Y}$. We then give a modification of the base method that computes just the reachable states of the automaton. Later on we shall see that our method is also suited for verifying contextual conditions.

Definition 8: Let $C = (X, Y, y_\sharp, S)$ be a constraint system. The DPFA D_C is defined as $(2^X, 2^{Y\times Y}, \{q_\sharp\}, F, \mathit{Entry}, \mathit{Exit}, \mathit{Comb})$, where

$$\begin{aligned}
q_\sharp &= \{(y_\sharp, y_\sharp)\}\\
F &= \{q \mid (y_\sharp, y_0) \in q \text{ for some } y_0 \text{ with } y_0 \sqsupseteq \varepsilon\}\\
\mathit{Entry}_a\, q &= \{(y, y) \mid (y_1, y_2) \in q, y_2 \sqsupseteq xy_3, \text{ and } x \sqsupseteq a\langle y\rangle\}\\
\mathit{Exit}_a\, q &= \{x \mid x \sqsupseteq a\langle y\rangle \text{ and there is a } (y, y_0) \in q \text{ with } y_0 \sqsupseteq \varepsilon\}\\
\mathit{Comb}(q, p) &= \{(y, y_2) \mid (y, y_1) \in q, y_1 \sqsupseteq xy_2 \text{ and } x \in p\}
\end{aligned}$$

Assume the automaton wants to verify that $a\langle f\rangle$ belongs to the language of variable x (i.e., $[\![C]\!]\, x$). Then it needs to verify that f is in the language of some y with $x \sqsupseteq a\langle y\rangle$. In order to show this, it maintains, during traversal of f, pairs (y, y_1). The first component simply records y, whereas the second component is a variable to be verified for the remaining part of f. Thus, the entry function produces pairs of the form (y, y). Accordingly, if after complete traversal of f a pair (y, y_0) with $y_0 \sqsupseteq \varepsilon$ is obtained, then $a\langle f\rangle$ belongs to the languages of all x with $x \sqsupseteq a\langle y\rangle$. This is reflected by the *Exit*-function.

Theorem 2: For each constraint system C, $\mathcal{L}_{D_C} = \mathcal{L}_C$.

We omit the proof and instead illustrate the construction by an example:

Example 4: Consider constraint system C from Ex. 1. Then D_C has tree states $Q_\mathcal{T} = 2^{\{x_1, x_\top\}}$ and forest states $Q_\mathcal{F} = 2^{\{y_\sharp, y_\top\}^2}$. The start state is $\{\{(y_\sharp, y_\sharp)\}\}$, and the final states are all that contain $(y_\sharp, y_\top)$.
A run of D_C for $f = b\langle\varepsilon\rangle a\langle b\langle\varepsilon\rangle\rangle b\langle\varepsilon\rangle$ is illustrated in Fig. 1. Let us explain some steps of this run: node a is entered with forest state $1 = \{(y_\sharp, y_\sharp)\}$. In order to proceed to its child, the *Entry* function is applied:

$$\mathit{Entry}_a\, 1 = \{(y, y) \mid (y_1, y_2) \in 1, y_2 \sqsupseteq xy_3, x \sqsupseteq a\langle y\rangle\} = \{(y_\top, y_\top)\} = 2$$

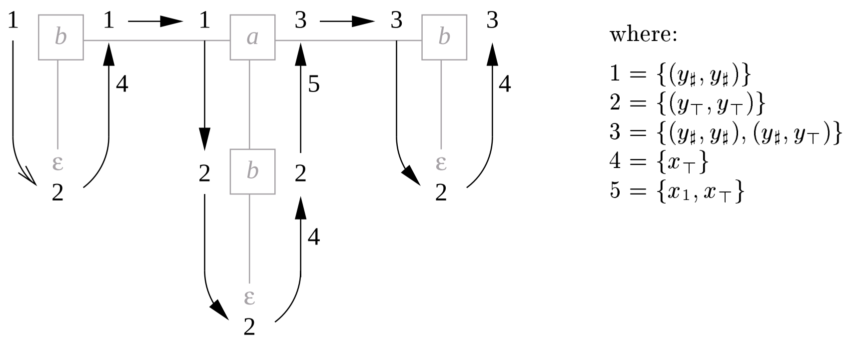

Fig. 1. A run of D_C from Ex. 4 for $f = b\langle\varepsilon\rangle a\langle b\langle\varepsilon\rangle\rangle b\langle\varepsilon\rangle$.

Node b is thus entered with state 2. This state is reproduced through inspecting its (zero) children, then fed into the *Exit*-function when returning to node a:

$$\mathit{Exit}_a\, 2 = \{x \mid x \sqsupseteq a\langle y\rangle, (y, y_0) \in 2, y_0 \sqsupseteq \varepsilon\} = \{x \mid x \sqsupseteq a\langle y_\top\rangle\} = \{x_1, x_\top\} = 5$$

The so obtained tree state 5 is now combined with the previously stacked forest state $1 = \{(y_\sharp, y_\sharp)\}$:

$$\begin{aligned} \mathit{Comb}(1,5) &= \{(y, y_2) \mid (y, y_1) \in 1, y_1 \sqsupseteq x y_2, x \in 5\} \\ &= \{(y_\sharp, y_2) \mid y_\sharp \sqsupseteq x y_2, x \in 5\} = \{(y_\sharp, y_\sharp), (y_\sharp, y_\top)\} = 3 \end{aligned}$$

This forest state 3 is preserved by traversing the last subtree $b\langle\varepsilon\rangle$. Since 3 is a final state, D_C accepts f. □

5 Locating Matches

The previous section showed how we can use DPFAs for accepting the language of a constraint system and thus also of a pattern. But how can we, for a pattern system $(C^\circ, x_\sharp)$, *locate* subtrees t that match target variable $x_\sharp$ in a forest f? We will show that we recognize t by looking at the input and output state of the $\delta_\mathcal{T}$-call for t. We can further observe that if these states indicate a match, then the whole run of the automaton is always successful.
Let us first define the notion of a labeling produced by a run of a DPFA:

Definition 9: Let $D = (Q_\mathcal{T}, Q_\mathcal{F}, \{q_\sharp\}, F, \mathit{Entry}, \mathit{Exit}, \mathit{Comb})$ be a DPFA and f_0 be a forest. Then the *labeling* λ of f_0 by D is a mapping from subtrees and subforests of f_0 to forest states of D, assigning each subtree or subforest[2] the input state of the corresponding call of $\delta_\mathcal{T}$ or $\delta_\mathcal{F}$ during the run of D on f_0. Formally speaking,

- $\lambda f_0 = q_\sharp$;
- if $\lambda t = q$ and $t = a\langle f\rangle$, then $\lambda f = \mathit{Entry}_a\, q$;
- if $\lambda f = q$ and $f = t f_1$, then $\lambda t = q$ and $\lambda f_1 = \mathit{Comb}(q, \delta_\mathcal{T}(q, t))$.

[2] More precisely, we should rather speak of *occurrences* of subtrees and subforests.

We can now formulate the main result of this paper:

Theorem 3: Let $\Pi = (C^\circ, x_\sharp)$ be a pattern system. We obtain a constraint system C by substituting $x_\sharp$ for all occurrences of $\bullet$ in C°. Now let D_C be defined as in the previous section and, for a fixed forest f_0, let λ be the labeling of f_0 produced by D_C. Then for all subtrees t of f_0 the following two statements are equivalent:

(1) t matches $x_\sharp$ in f_0;

(2) there are $y, y_1 \in Y$ such that $(y, y_1) \in \lambda t$, $y_1 \sqsupseteq x_\sharp y_\top$, and $x_\sharp \in \delta_{\mathcal{T}}(\lambda t, t)$.

The proof is omitted. $\square$

We conclude that during a run of D_C, we can identify candidates t fulfilling the context of a pattern by looking at λt. Then it only remains to verify that t satisfies target variable $x_\sharp$, i.e. $x_\sharp \in \delta_{\mathcal{T}}(\lambda t, t)$. In that case we have detected a match of the pattern. Therefore, in a single run, we not only determine whether a forest contains a match, but we can also locate all matching subtrees. Thus we can find all matches of a pattern in an SGML document during the parse process, without the need for constructing a copy of the document in memory.

6 Computing Reachable States

We have presented a method for constructing an equivalent DPFA for a given constraint system. The set of forest states of D_C is $2^{Y \times Y}$ which is still large. In most cases, however, a considerable subset of states is not reachable from the automaton's start state. An implementation of D_C will therefore compress the transition tables by storing transitions for the reachable states only. In order to describe reachable states we make up a system of inclusion constraints for each $q \in Q_{\mathcal{F}}$:

$$\begin{aligned} r_{\mathcal{T}}[q] &\supseteq \{\mathit{Exit}_a\, q_1 \mid q_1 \in r_{\mathcal{F}}[\mathit{Entry}_a\, q], a \in \Sigma\} \\ r_{\mathcal{F}}[q] &\supseteq \{q\} \\ r_{\mathcal{F}}[q] &\supseteq \{\mathit{Comb}(q_1, p) \mid q_1 \in r_{\mathcal{F}}[q] \text{ and } p \in r_{\mathcal{T}}[q_1]\} \end{aligned}$$

Variable $r_{\mathcal{F}}[q]$ describes the set of all forest states reachable from q at the same level, i.e., $q_1 \in r_{\mathcal{F}}[q]$ iff there is an f such that $q_1 = \delta_{\mathcal{F}}(q, f)$. Similarly, $r_{\mathcal{T}}[q]$ describes the set of all tree states reachable from q by consuming exactly one tree, i.e., $p \in r_{\mathcal{T}}[q]$ iff $p = \delta_{\mathcal{T}}(q, t)$ for some t.

In order to compute $r_{\mathcal{F}}[q]$ for some state q, $r_{\mathcal{F}}[q_1]$ and $r_{\mathcal{T}}[q_1]$ must be computed for all q_1 reachable from $\mathit{Entry}_a\, q$. In order to compute these, all states reachable from $\mathit{Entry}_a\, q_1$ have to be considered, and so on. Therefore, computing a partial least solution for all $r_{\mathcal{F}}[q]$ and $r_{\mathcal{T}}[q]$ necessary to compute $r_{\mathcal{F}}[q_\sharp]$ for the start state $q_\sharp$ involves exactly the reachable states of D_C. [LH92] propose *local solvers* for computing such a solution. Because the only operation required for solving the system is set union, *differential local solvers* as in [FS98] can also be applied. The interested reader may consult that paper for efficient algorithms.

Example 5: Consider automaton D_C from Ex. 4. Running a local solver yields:

$r_{\mathcal{F}}[1] = \{1,3\}$, $r_{\mathcal{F}}[3] = \{3\}$, $r_{\mathcal{F}}[2] = \{2\}$, and $r_{\mathcal{T}}[1] = r_{\mathcal{T}}[2] = r_{\mathcal{T}}[3] = \{5,4\}$. I.e., states $1,\ldots,5$ from Fig. 1 are the only reachable ones among the twenty states of D_C. □

7 Extensions

We have suggested a mechanism for specifying forest regular languages, namely constraint systems. This formalism has the advantage of allowing for convenient extensions. For instance, we can add conjunctions:

$$x \sqsupseteq a\langle \sqcap M\rangle \qquad \text{with } M \subseteq Y \text{ and} \qquad [\![\, a\langle \sqcap M\rangle]\!]\,\sigma = \bigcap_{y\in M} [\![a\langle y\rangle]\!]\,\sigma$$

On the automaton side this introduces the concept of *alternation*: Exit_a is extended to be a subset of $2^{Q_{\mathcal{F}}} \times Q_{\mathcal{T}}$ (instead of $Q_{\mathcal{F}} \times Q_{\mathcal{T}}$), and $\delta_{\mathcal{T}}$ is redefined:

$$\begin{aligned}(q, a\langle f\rangle, p) \in \delta_{\mathcal{T}} \ \text{ iff there is an } M \subseteq Q_{\mathcal{F}} \text{ with } (M,p) \in \mathit{Exit}_a \text{ and for}\\ \text{all } q_1 \in M \text{ there is a } q_0 \in \mathit{Entry}_a\, q \text{ with } (q_0, f, q_1) \in \delta_{\mathcal{F}}.\end{aligned}$$

Since alternation can be removed by a subset construction similar to the one removing non-determinism, the construction of D_C can also handle conjunctions, provided we change the definition of *Entry* and *Exit*:

$$\begin{aligned}\mathit{Entry}_a\, q &= \{(y,y) \mid (y_1,y_2) \in q, y_2 \sqsupseteq xy_3, x \sqsupseteq a\langle \sqcap M\rangle \text{ and } y \in M\}\\ \mathit{Exit}_a\, q &= \{x \mid x \sqsupseteq a\langle \sqcap M\rangle \text{ and for } y \in M, (y,y_0) \in q \text{ with } y_0 \sqsupseteq \varepsilon\}\end{aligned}$$

But we have to be careful: conjunctions may only occur in constraints for non-context variables. Otherwise locating matches becomes complicated.
Interestingly enough, it is also complicated to allow conjunctions in constraints for forest variables, like $y \sqsupseteq \sqcap_{i=1}^{n} x_i y_i$. Carried over to automata, this would imply alternation in traversing a forest from left to right. Removing this kind of alternation results in a state space that is doubly exponential – provided we preserve the direction of traversal [CKS81]. In order to do with single exponential blowup (needed anyway for removing non-determinism) we would have to change direction of traversal. Clearly, this is contrary to our intention of running the automaton during one left-to-right parse of a document tree.

8 Conclusion

We presented constraint systems for denoting regular languages of trees and forests of variable arity and used them for uniformly specifying both structural and contextual conditions in patterns. Then we introduced the class of forest pushdown automata which we showed to be well-suited for recognizing forest regular languages. We identified an interesting subset of contextual conditions and showed how PFAs can be used for locating all subtrees of a forest that match a pattern. For this purpose a single traversal through the forest is sufficient, and the decision whether a subtree matches can be made in-place, i.e. at the time of visiting that subtree. We gave an algorithm for computing the reachable states of pushdown forest automata and observed that this can considerably reduce the

size of the transition tables.
It remains to be investigated whether we can extend our method to deal with arbitrary contexts. Future work includes integrating these methods into a document processing system.

Acknowledgements: We thank Priti Shankar for kindly providing us with a preliminary version of his paper [SGYM98].

References

[BKR96] M. Biehl, N. Klarlund, and T. Rauhe. Algorithms for guided tree automata. In *First International Workshop on Implementing Automata (WIA'96)*, LNCS 1260. Springer, 1996.

[BMW91] J. Börstler, U. Möncke, and R. Wilhelm. Table Compression for Tree Automata. *ACM TOPLAS*, 13(3):295–314, 1991.

[Bra69] W.S. Brainerd. Tree Generating Regular Systems. *Information and Control*, 14:217–231, 1969.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[FS98] C. Fecht and H. Seidl. Propagating Differences: An Efficient New Fixpoint Algorithm for Distributive Constraint Systems. In *ESOP '98*, LNCS 1381, pages 90–104. Springer, 1998.

[Gol90] C.F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[LH92] B. LeCharlier and P. Van Hentenryck. A Universal Top-Down Fixpoint Algorithm. Technical Report CS-92-25, Brown University, Providence, 1992.

[Mor94] E. Moriya. On two-way tree automata. *IPL*, 50:117–121, 1994.

[Mur96] M. Murata. Transformations of Trees and Schemas by Patterns and Contextual Conditions. In C. Nicolas and D. Wood, editors, *Principles of Document Processing (PODP'96)*, LNCS 1293, pages 153–169. Springer, 1996.

[MW95] D. Maurer and R. Wilhelm. *Compiler Design*. Addison-Wesley, 1995.

[NS98] A. Neumann and H. Seidl. Locating Matches of Tree Patterns in Forests. Technical Report 98-08, Mathematik/Informatik, Universität Trier, 1998.

[Pod92] A. Podelski. A Monoid Approach to Tree Automata. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 41–56. North Holland, 1992.

[PQ68] C. Pair and A. Quere. Définition et Etude des Bilangages Réguliers. *Information and Control*, 13:565–593, 1968.

[SGYM98] P. Shankar, A. Gantait, A.R. Yuvaraj, and M. Madhavan. A New Algorithm for Linear Regular Tree Pattern Matching. Submitted to TCS, 1998.

[Tak75] M. Takahashi. Generalizations of Regular Sets and their Application to a Study of Context-Free Languages. *Information and Control*, 27:1–36, 1975.

[Tha67] J.W. Thatcher. Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory. *JCSS*, 1:317–322, 1967.