REWRITING SYSTEMS OVER UNRANKED TREES

von
Alexandra Spelten
Matrikelnummer 224295

Diplomarbeit
im Studiengang Informatik

vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften der
Rheinisch-Westfälischen Technischen Hochschule Aachen

20. September 2006

Betreuer: Dr. rer. nat. Christof Löding

angefertigt am
LEHRSTUHL FÜR INFORMATIK VII
Logik und Theorie diskreter Systeme
Univ.-Prof. Dr. Dr.h.c. Wolfgang Thomas

Abstract

Finite graphs constitute an important tool in various fields of computer science. In order to transfer the theory of finite graphs at least partially to infinite systems, a finite representation of infinite systems is needed. Rewriting systems form a practical model for the finite representation of infinite graphs. Among attractive subclasses of rewriting systems is the class of ground tree rewriting systems over ranked trees, which is known to have good algorithmic properties.

We investigate these algorithmic properties for two kinds of rewriting systems over unranked trees.

For the first introduced rewriting formalism, we define a reduction to ranked (binary) trees via an encoding and also to standard ground tree rewriting, and we show that the generated classes of transition graphs coincide.

For the second introduced rewriting formalism over unranked trees using subtree rewriting combined with flat prefix rewriting, we obtain strictly more transition graphs, and we show that the reachability problem over such graphs is still decidable. Here, a flat prefix rewrite rule substitutes a prefix of the word derived from the front of a subtree of height 1.

However, as opposed to standard ground tree rewriting systems, this decidability result fails for both formalisms when the transition graphs are restricted by a deterministic top down tree automaton.

Contents

Introduction

The theory of finite graphs is an important tool in several different fields of computer science. Initially, finite graphs were employed as a pure data structure, but were soon discovered for the behavioral description of programs or processes. With programs translated to graphs and desired properties translated to logic formulae, algorithms were developed to automatically check the satisfaction of a formula over a graph, thus constituting the foundation for the theory of automatic verification and symbolic model checking (cf. e.g. [Eme81, CE81]). This approach was followed and enhanced by many researchers (cf. e.g. [BVW94, Eme96]).

Due to the employment of theoretically unbounded data structures in programs, such as stacks, counters, queues, etc., the present model had to be extended to cover infinite systems. In order to be able to transfer some of the established methods for finite graphs and allow algorithmic approaches to tasks like verification of infinite state systems, classes of infinite graphs have to be given effectively, i.e. they need to be represented by some finite object.

Many classes of finitely representable infinite graphs have been described recently (for a survey on infinite graphs cf. [Tho02]). One task in the development of a theory of infinite graphs is to identify classes of infinite graphs where elementary problems like reachability are decidable. The reachability problem "Given a finite representation of an infinite graph G and two vertices u, u' of G, is there a path from u to u'?" in automatic verification translates to checking whether a system can reach an undesirable state or configuration. The paradigm of rewriting systems (cf. e.g. [DJ90]) constitutes a practical model for the finite representation of infinite graphs.

Word rewriting systems are among the most general formalisms found in computer science and build a unifying framework to compare such heterogeneous concepts as runs of finite automata, transducers, pushdown automata, or even Turing machines. On the one hand, most of the interesting properties of rewriting systems are undecidable, while on the other hand, several interesting results were obtained on special kinds of rewriting systems, thus one obtains decidable properties of fragments of theories or of subclasses of rewriting systems.

In recent years, many classes of finitely representable infinite graphs have been studied. For the simplest ones, it is possible to automatically verify expressive logics, while for the most complex classes, hardly anything can be determined (for instance it is not difficult to define classes of graphs that can encode undecidable problems such as the halting problem for Turing machines via the reachability problem).

The research on infinite graphs was started with the analysis of pushdown graphs (cf. [MS85]). When considering a pushdown graph, not the recognized language of the pushdown automaton is of interest, but the transition system

that is generated. This transition system forms an infinite graph whose vertices are configuration words (control state and stack content), and the edges are induced by a transition of the pushdown automaton, thus corresponding to the application of a finite set of prefix rewrite rules for the configuration words. Based on a fact proven by Büchi in 1964, that the set of all reachable configurations of a pushdown automaton forms a regular language (cf. [Büc64]), Muller and Schupp provided a proof that the monadic second order logic is decidable for the class of pushdown graphs (cf. [MS85]). Since the monadic second order logic contains temporal logic (cf. [Eme90]), the model checking for pushdown graphs and properties expressed in temporal logic is decidable. In [EHRS00], more efficient algorithms for solving e.g. reachability problems are given.

A variety of possibilities to define other classes of graphs emerges from taking rewriting as the basic principle for the generation of infinite graphs. One way to define a more general class of graphs than the class of pushdown graphs is to generalize the rewriting relation. Caucal defined the class of prefix recognizable graphs, which is also generated by word rewriting, but allows regular languages instead of single words in the rewrite rules (cf. [Cau96]). The monadic second order logic remains decidable; even the formalisms of interpretation and unfolding in alternating application preserve the decidability of the monadic second order theory, yielding the "pushdown hierarchy" (cf. [Cau02]).

A further generalization is accomplished by employing finite word transducers to define the edge relations of graphs. For synchronous transducers, this approach yields the class of automatic graphs (cf. [BG00]), while rational graphs are derived from applying asynchronous transducers (cf. [Mor99]). The formalisms to create these kinds of graphs are very strong (both classes contain e.g. the transition graphs of Turing machines), and thus even simple reachability problems on these graphs are undecidable. Only the first order theory remains decidable for automatic graphs (cf. [BG00]), while the theory of this logic is undecidable for the class of rational graphs (cf. [Mor99]).

A common point of all of the above mentioned families of infinite graphs is their definition as rewriting systems over words. Thus, another natural approach for a generalization is to employ rewriting on structures other than words. In tree (or term) rewriting formalisms, the basic objects of the rewriting systems are ranked trees, as already considered in [Bra69]. Thereby, the internal structure of the trees is not the main point of interest, but the different rewriting operations that can be applied on trees. Consequently, the vertices of the generated infinite graphs are ranked trees, while the edge relation is induced by (simple) tree operations. Considering the transitive closure of these operations, one exploits a generalization of Büchi's theorem proven by Brainerd, that the sets of trees generated by a regular system are exactly those accepted by (finite) tree automata (cf. [Bra69]). Tree automata and recognizable tree languages proved to be a powerful tool in the theory of rewriting systems (cf. e.g. [GS84, DJ90, GT95, CDG+97]). One of the main motivations of tree automata were decision prob-

lems (cf. e.g. [Rab77, Tho90]), and the popular algebraic and algorithmic frames of rationality and finite automata associate decision algorithms with logical specifications. Therefore, the interest lies in tree operations preserving the regularity of sets of trees, so that the transitive closure of these operations can be represented by tree automata.

Among attractive subclasses of rewriting systems, an interesting and practical subclass is made up by the simplest form of rewriting systems, the ground rewriting systems (for an extensive analysis cf. [Löd03]). Ground means that no variables occur in the rewrite rules, thus in ground tree (or term) rewriting systems, only explicitly specified subtrees can be replaced by other explicitly specified subtrees. Löding studied the structure of the transition graphs of ground tree rewriting systems and their relation to other classes of infinite graphs, in particular to pushdown graphs (cf. [Löd02a, Löd03]). Furthermore, in [Löd02b, Löd03], Löding investigated several variants of the reachability problem for this class of graphs and defined a decidable logic (which is in some sense a maximal fragment of the temporal logic CTL*, cf. [Eme90]) for the class of (regular) ground tree rewriting graphs.

The structure of ground tree rewriting graphs constitutes an interesting field of research because ground tree rewriting systems allow the specification of grid structures, which are automatic but not prefix recognizable. On the other hand, there are prefix recognizable graphs that cannot be generated by ground tree rewriting systems. The geometric properties of ground tree rewriting graphs were investigated in [Löd02a, Col02, Löd03], and the relationship of this class of graphs with other well known classes was determined.

As reachability problems constitute an important part of verification, the analysis in [Löd02b, Löd03] concentrated on reachability problems, which play a central role in the specification of system properties. Apart from earlier results of the decidability of confluence (cf. [DHLT90]), first order theory (cf. [DT90]), and some reachability problems (cf. [CG90]), Löding showed the decidability of the model checking problem for a temporal logic with reachability and recurrence operators, which are needed e.g. for the formalization of fairness properties. Furthermore, the analysis was widened for the case of *regular* ground tree rewriting systems, which is a rewriting principle with regular sets of trees on each side of the rewrite rules (represented by tree automata) as opposed to single trees.

For many applications, the modelling of system states, messages, or data by ranked trees is not the most intuitive approach, since every symbol is of a fixed arity. With this formalism the direct modelling of associative operations such as parallel composition of processes (cf. [May00, BT03]) is not possible. As a model for such applications, trees of unbounded arity are a natural choice. In brief, unranked trees are finite labeled trees where nodes can have an arbitrary but finite number of children, and no fixed rank is associated to any label. Based on the work of Löding (cf. [Löd02a, Löd02b, Löd03]), new rewriting systems over unranked trees are introduced and the aims of this thesis are the following.

- Investigate a possible propagation of the idea of ground tree rewriting systems to the case of unranked trees.
- Find new classes of infinite graphs generated by rewriting systems over unranked trees that have decidable properties.

The development of unranked tree automata was initiated in [BMW01], based on early work of Pair and Quéré (cf. [PQ68]) and Takahashi (cf. [Tak75]). As semi-structured documents in XML or HTML form unranked trees, several different applications of unranked trees and unranked tree automata in the field of XML research can be found (cf. e.g. [Nev02a]).

In order to investigate whether results for ground tree rewriting systems are transferable, the direct adaption of this rewriting principle is not of interest. When starting from a fixed initial tree and applying a finite set of rewrite rules with constant trees, the resulting trees are of bounded branching and hence can be traced to the case of ground tree rewriting over ranked trees. Another natural approach to handle unbounded branching of unranked trees is to encode unranked trees as binary trees. Using this formalism, it is shown that there is a class of rewriting systems over unranked trees, which will be called partial subtree rewriting systems, that generates the same class of infinite graphs as ground tree rewriting systems over ranked trees.

However, encodings are problematic as they alter locality and path properties. This means that this approach blurs a decisive point, namely the separation of unboundedness which is derived from the arbitrary hierarchy levels (represented by the height of the tree) from the unboundedness of the number of data on the same hierarchy level. In most applications in the database field, the second aspect is the decisive point and requires a direct consideration (cf. e.g. [Nev02b]). This lack of separation is exposed when considering deterministic top down tree automata on both classes of rewriting systems. This kind of automaton considers the complete successor sequence of a node before deciding on the state to assign to each successor node. For any kind of encoding of unranked trees into ranked ones, it fails to capture the complete successor sequence of the unranked tree in the encoding (which is spread onto different levels in the ranked tree). This results in a different outcome of the reachability problem on the transition graph restricted to the tree language of a deterministic top down tree automaton. For ground tree rewriting systems over ranked trees this problem is decidable (which is derived from results in [Col02]), while it is shown to be undecidable for partial subtree rewriting systems.

Therefore, another class of rewriting systems over unranked trees is defined, which will be called subtree and flat prefix rewriting systems, and which combines ground tree rewriting with prefix word rewriting on the flat front of a tree. Here, a flat prefix rewrite rule can only be applied to the word derived as the successor sequence of a node that is the root of a subtree of height 1. With this approach related to ground tree rewriting, a new class of infinite graphs is obtained, which has a decidable reachability problem. Furthermore, analogous to regular

ground tree rewriting systems over ranked trees, a regular variant of subtree and flat prefix rewriting systems is considered.

Outline

Following this introduction, Chapter 1 – Preliminaries introduces the general terminology and definitions used throughout this thesis. This includes standard notations for graphs, ranked and unranked trees, hedges as introduced by Courcelle (cf. e.g. [Cou78, Cou89]), and substitutions. Two encoding formalisms for the transition from unranked trees to binary ones are compared (for further alternatives cf. e.g. [Nev02b, CNT04, Suc02]). The rewriting systems introduced are ground tree rewriting systems and regular ground tree rewriting systems as analyzed in [Löd03]. The relationship of the classes of infinite graphs generated by these two formalisms is similar to the one of pushdown and prefix recognizable graphs (cf. [Cau96]). Furthermore, the notation for transition graphs as well as automata on words and on trees is fixed (for a survey on tree automata cf. [CDG+97]). After a brief introduction to deterministic Turing machines, symbolic model checking and reachability problems are discussed, and the chapter concludes with some known results over (regular) ground tree rewriting systems.

In Chapter 2 – Partial Subtree Rewriting Systems, the approach of encoding unranked trees as ranked ones yields the rewriting principle of partial subtree rewriting systems. Partial subtree rewriting systems operate with single trees on each side of the rewrite rules, but as opposed to ground tree rewriting systems, these trees are not ground but allow the utilization of one variable. This variable is restricted to a certain position in the tree, such that the specified partial tree matches a subtree in the encoding, and the variable can be substituted by a hedge. Section 2.1 shows how to construct a ground tree rewriting system over ranked trees for a given partial subtree rewriting system and vice versa, such that the generated transition graphs coincide. Based on this class equivalence, results for partial subtree rewriting systems are discussed in Section 2.2.

Chapter 3 – Subtree and Flat Prefix Rewriting Systems introduces two other models of rewriting systems over unranked trees, the subtree and flat prefix rewriting systems and the regular subtree and flat prefix rewriting systems. Both introduce a new kind of rewriting principle; they allow to substitute a prefix of the word derived from the successor sequence of a node via concatenation. The application of these rewrite rules is restricted to the flat front of a tree, i.e. all nodes of the sequence are leaves (in other words, the subtree obtained at this node is of height 1). Regular subtree and flat prefix rewriting systems additionally allow regular sets of trees on each side of the rewrite rules as opposed to single trees in subtree and flat prefix rewriting systems. Section 3.1 provides the proof that the class of transition graphs of subtree and flat prefix rewriting systems strictly includes the class of transition graphs of partial subtree rewrit-

ing systems (which coincides with the class of transition graphs of ground tree rewriting systems over ranked trees). Section 3.2 shows that reachability for the class of transition graphs of regular subtree and flat prefix rewriting systems is decidable. The proof follows the well known concept of saturation, which has to be refined due to the different nature of the two kinds of rewrite rules.

Chapter 4 – Undecidability Results reveals a limitation of all introduced rewriting systems. If a transition graph of any of the rewriting systems is restricted by a deterministic top down tree automaton, the reachability problem on this graph is undecidable. Even though the classes of transition graphs of partial subtree rewriting systems and ground tree rewriting systems over ranked trees coincide, this result shows that when taking the inner structure of the trees into account, encoding unranked trees as ranked ones results in losing structural information.

Chapter 5 – Conclusion closes with a brief recapitulation and points to some possible topics for further research.

Acknowledgments

I would like to thank all people who contributed to this thesis in various ways. In particular, I would like to thank Prof. Wolfgang Thomas and Dr. Christof Löding for suggesting this topic and for their great support.

I would like to thank Prof. Erich Grädel for his kind readiness to review this thesis.

Furthermore, I would like to thank my advisor Dr. Christof Löding for his numerous comments and substantial suggestions; he proposed most of the present results.

I am also grateful to Jan-Henrik Altenbernd for many helpful discussions and his careful proofreading. The numerous comments of Philipp Böckers, Andreas Hermanns, and Kai Sauerzapfe also uncovered several mistakes and inaccuracies. Finally, I would like to thank Kai Sauerzapfe for his support, encouragement, and patience with me while I was writing this thesis.

Chapter 1

Preliminaries

In this chapter, the notations, basic definitions and terminology are formalized that were introduced informally in the introduction. Thereby, standard expressions are employed as introduced e.g. in [CDG+97].

Alphabets, Terms, and Languages

For a nonempty set S of *symbols*, let S^* denote the set of all finite sequences (*words*) over S, and S^+ the set of all nonempty finite sequences over S. For $w \in S^*$, say $w = a_1 \cdots a_n$ for some $n \in \mathbb{N}$ and $a_i \in S$ for all $1 \leq i \leq n$, let $|w| := n$ denote the *length* of w, and $w(i)$ denote the i-th letter of w, i.e. $w(i) = a_i$. The *empty word* is denoted by ε with $|\varepsilon| = 0$. A *language* over a set S is a subset of S^*.

The *concatenation* of words u and v, denoted by $u \cdot v$ or shortly uv, is obtained by appending v to u, i.e. uv is the word $u(1) \cdots u(|u|)v(1) \cdots v(|v|)$. Moreover, the concatenation of two languages K and L is defined by $KL := \{uv \mid u \in K, v \in L\}$. For a language L, define $L^0 := \{\varepsilon\}$, $L^{i+1} := L^i \cdot L$, and $L^* := \bigcup_{i \geq 0} L^i$. L^* is called the *Kleene closure* of L. The set $RE(S)$ of *regular expressions* is built up inductively from $\emptyset$, ε, and all symbols $s \in S$ by application of concatenation, Kleene closure, and union.

The word $u \in S^*$ is called a *prefix* of $w \in S^*$, if there is a word $v \in S^*$ such that $w = u \cdot v$. A language $L \subseteq S^*$ is *prefix closed* iff for each $w \in L$, it holds that every prefix of w is also in L.

For the set $\mathbb{N}$ of natural numbers, i.e. the set of positive integers, the prefix ordering on $\mathbb{N}^*$ is denoted by $\preceq$. That is, $x \preceq y$ for $x, y \in \mathbb{N}^*$ iff there is a $z \in \mathbb{N}^*$ such that $y = xz$, and $x \prec y$ if $z \in \mathbb{N}^+$. For $x = x(1) \cdots x(m) \in \mathbb{N}^*$ define $x \oplus i := x(1) \cdots x(m-1)\, \underbrace{x(m) + i}$ for $i \in \mathbb{N}$.

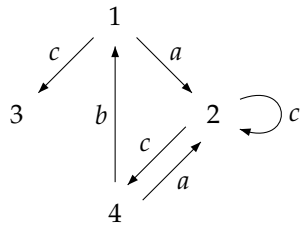
A *ranked alphabet* is a nonempty finite set Σ of symbols, where each symbol a is assigned a *rank* (or *arity*) $rk(a) \in \mathbb{N}$. Symbols of the same rank i are collected in the set $\Sigma_i := \{a \in \Sigma \mid rk(a) = i\}$, and the ranked alphabet Σ can be denoted by listing all symbols with their ranks, or as

$$\Sigma := \bigcup_{i=0}^{k} \Sigma_i \quad (\text{with } k = max\{rk(a) \mid a \in \Sigma\}).$$

Let $\mathcal{X}$ be a set of constants (arity 0) called *variables* disjoint from Σ_0. The set $T_\Sigma(\mathcal{X})$ of *terms* over the ranked alphabet Σ and the set of variables $\mathcal{X}$ is defined inductively by

- $\Sigma_0 \subseteq T_\Sigma(\mathcal{X})$,
- $\mathcal{X} \subseteq T_\Sigma(\mathcal{X})$, and

Figure 1.1 Graph G from Example 1.1.



- $a \in \Sigma_i$ and $t_1, \ldots, t_i \in T_\Sigma(\mathcal{X})$ for some $i \geq 1$, then $a(t_1, \ldots, t_i) \in T_\Sigma(\mathcal{X})$.

A term $t \in T_\Sigma(\mathcal{X})$ is called *linear* if each variable of $\mathcal{X}$ occurs at most once in t. If $\mathcal{X} = \emptyset$, then $T_\Sigma(\mathcal{X})$ is also written as T_Σ, and $t \in T_\Sigma$ is called a *ground term*.

An *unranked* alphabet is a nonempty finite set Σ of symbols without any kind of arity.

Graphs

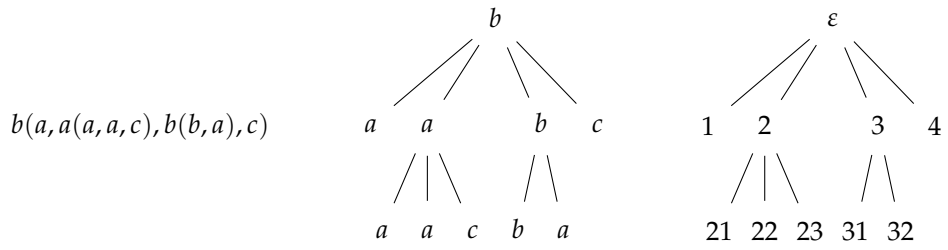
A *countable, directed, edge labeled graph* G is of the form $G = (V, E, \Gamma)$, where V is a countable set of *vertices*, $E \subseteq V \times \Gamma \times V$ is the set of *edges (transitions)* between the vertices, and Γ is the *edge label (transition) alphabet*. If the edge labels are not of interest or if graphs without edge labels are considered, Γ is omitted, the graph is called *unlabeled*, and the edge relation is $E \subseteq V \times V$. Throughout this thesis, whenever a graph is mentioned, the graph is a countable, directed, edge labeled graph if not stated otherwise.

A *path* is a nonempty (possibly infinite) sequence $\pi = v_0, \ldots, v_m$ of vertices of a graph with $(v_i, v_{i+1}) \in E$ for all $0 \leq i \leq m-1$. $|\pi|$ denotes the *length* of π and is defined as $|\pi| = m$. For two vertices u and u', there is a path from u to u' in G, if there is a path $\pi = v_0, \ldots, v_m$ in G with $v_0 = u$ and $v_m = u'$.

Two graphs $G = (V, E, \Gamma)$ and $G' = (V', E', \Gamma)$ are *isomorphic* iff there is a bijection $f : V \to V'$ with $(v_1, \sigma, v_2) \in E \Leftrightarrow (f(v_1), \sigma, f(v_2)) \in E'$ for all $v_1, v_2 \in V$ and all $\sigma \in \Gamma$. In this thesis, graphs do not need to be distinguished up to isomorphism, i.e. two graphs are considered identical iff they are isomorphic.

Example 1.1. The graph $G = (V, E, \Gamma)$ with
$V = \{1, 2, 3, 4\}$,
$E = \{(1, a, 2), (1, c, 3), (2, c, 2), (2, c, 4), (4, b, 1), (4, a, 2)\}$, and
$\Gamma = \{a, b, c\}$
is depicted in Figure 1.1. There are infinitely many paths from vertex 1 to vertex 4, e.g. $\pi_1 = 1, 2, 4$, $\pi_2 = 1, 2, 2, 4$, with $|\pi_1| = 2$ and $|\pi_2| = 3$. ◂

Figure 1.2 Tree t from Example 1.2.



(a) Term notation. (b) Graphical notation. (c) Domain dom_t of t.

Trees

A *(finite) tree* over Σ is a mapping $t : dom_t \to \Sigma$ with finite set $dom_t \subseteq \mathbb{N}^*$ (the *domain* of t) such that

- dom_t is prefix closed (note that this implies $\varepsilon \in dom_t$),
- $dom_t \neq \emptyset$,
- for each $x \in \mathbb{N}^*$ and $i \in \mathbb{N}$: if $xi \in dom_t$, then $xj \in dom_t$ for all $j \leq i$, and
- for ranked Σ: if $x0, \ldots, x\,\underline{i-1} \in dom_t$ and $xi \notin dom_t$, then $t(x) \in \Sigma_i$.

A tree over an unranked alphabet is called an *unranked tree* while a tree over a ranked alphabet is a *ranked tree*. For both, the elements of dom_t are called the *nodes* of t (to clearly separate them from *vertices* of a graph). The *height* of a tree $t \in T_\Sigma$ is denoted by $ht(t) := max\{|x| \mid x \in dom_t\}$. For $x, y \in dom_t$, x is the *predecessor* of y and y a *successor* of x iff $y = xi$ for some $i \in \mathbb{N}$. If x has no predecessor (i.e. $x = \varepsilon$), then x is called the *root* of t. If x has no successors, i.e. $xi \notin dom_t$ holds for all $i \in \mathbb{N}$, then x is called a *leaf* of t, and $front(t)$ denotes the word that is derived from the front of t by reading all leaves from left to right. If a tree t is of height 1, the word derived from the front of t is called a *flat front*, denoted by $flatfront(t)$. The n-th right sibling y of a node x is calculated by $y = x \oplus n$ for $n \in \mathbb{N}$ and $y \in dom_t$. If the root of a finite tree t is labeled by $a \in \Sigma$ and has k successors at which the subtrees $t_1, \ldots, t_k$ are rooted, then the term $a(t_1, \ldots, t_k)$ is equivalent to t. The set of all finite trees (i.e. trees with finite domain) over Σ is denoted by T_Σ.

The *subtree* $t_{\downarrow x}$ of t, with node $x \in dom_t$ as root, is obtained by taking all nodes that have x as prefix, and removing the prefix x from all these nodes while keeping the labels. This is formalized as follows: for $x \in \mathbb{N}^*$, define $xdom_t = \{xy \in \mathbb{N}^* \mid y \in dom_t\}$ and $x^{-1}dom_t = \{y \in \mathbb{N}^* \mid xy \in dom_t\}$. For $x \in dom_t$, the subtree $t_{\downarrow x}$ of t at node x is the tree with domain $dom_{t_{\downarrow x}} = x^{-1}dom_t$ and $t_{\downarrow x}(y) = t(xy)$.

For a set $\mathcal{X}_n = \{X_1, \ldots, X_n\}$ of variables, a linear term $C \in T_\Sigma(\mathcal{X}_n)$ is called a *context*. The term in T_Σ obtained from C by replacing variable X_i by t_i for each $1 \leq i \leq n$, $t_i \in T_\Sigma$ is denoted by $C[t_1, \ldots, t_n] = C[X_1|t_1, \ldots, X_n|t_n]$. If $t = C[X|t_{\downarrow x}]$ for subtree $t_{\downarrow x}$ at node $x \in dom_t$, this is written shortly as $t = C[t_{\downarrow x}]$.

Example 1.2. Let t be an unranked tree over $\Sigma = \{a, b, c\}$ with $dom_t = \{\varepsilon, 1, 2, 21, 22, 23, 3, 31, 32, 4\}$ and $t(x) = \begin{cases} a & \text{for } x = 1, 2, 21, 22, 32, \\ b & \text{for } x = \varepsilon, 3, 31, \\ c & \text{for } x = 23, 4. \end{cases}$

The term and the graphical notation of t as well as the domain dom_t are depicted in Figure 1.2. The height of t is $ht(t) = 2$. Position 3 is the predecessor of the leaves 31 and 32 and $front(t) = aaacbac$. The subtree $t_{\downarrow 3}$ of t is the tree $t_{\downarrow 3} = b(b, a)$, which has a flat front: $flatfront(t_{\downarrow 3}) = ba$. The second right sibling of node 21 with $t(21) = a$ is calculated by $21 \oplus 2 = 23$ with $t(23) = c$. ◀

Hedges

Hedges were initially introduced by Courcelle (cf. [Cou78, Cou89]) as finite ordered sequences of trees. Formally, a *hedge* h over Σ is a mapping $h : dom_h \to \Sigma$ with $dom_h \subseteq \mathbb{N}^*$ and the following inductive definition

- the empty sequence ε is a hedge;
- if h is a hedge and $a \in \Sigma$, then $a(h)$ is a hedge;
- if g, h are hedges, then gh is a hedge (where $\varepsilon h = h\varepsilon = h$).

The *domain* dom_h of a hedge h is defined inductively as follows

- for the empty hedge ε: $dom_\varepsilon = \emptyset$,
- for a hedge h consisting of a single tree t: $dom_h = \{1 \cdot x \mid x \in dom_t\}$,
- if g, h are hedges, for the hedge gh:
  $dom_{gh} = dom_g \cup \{\underbrace{w(1) + \|g\|}_{} \cdot w(2) \cdots w(|w|) \mid w \in dom_h\}$,

where $\|g\|$ denotes the *width* of a hedge, which is defined inductively by

- $\|\varepsilon\| = 0$,
- $\|a(g)\| = 1$,
- $\|gh\| = \|g\| + \|h\|$.

Consequently, a tree is a hedge of width 1. Note that since the domains of trees of a hedge $h = t_1, \ldots, t_n$ are numbered from 1 to n, the domain of a hedge consisting of a single tree is not equal to the domain of the tree. In order to extract the tree t_i from hedge h, the leading cipher needs to be deleted, i.e. $dom_{t_i} = \{\varepsilon\} \cup \{w(2) \cdots w(|w|) \mid w \in dom_h \wedge w(1) = i\}$, and $t_i(x) = h(ix)$.

Example 1.3. Consider the hedge $h = t_1, t_2 =$  , c of width $\| h \| = 2$. The domain of h is $dom_h = \{1, 11, 12, 2\}$ with $h(1) = h(11) = b$, $h(12) = a$, and $h(2) = c$. The tree $t_1 = b(b, a)$ is extracted from h by deleting the leading cipher and one obtains $dom_{t_1} = \{\varepsilon, 1, 2\}$ with $t_1(\varepsilon) = t_1(1) = b$, and $t_1(2) = a$. ◂

Substitutions

To replace a subtree of a given tree by another tree or by a hedge, the notion of *substitution* is introduced. A substitution is a pair of a node $x \in \mathbb{N}^*$ and a tree $s \in T_\Sigma$ or a hedge $s = s_1 \cdots s_m$, written as $[x|s]$. A substitution $[x|s]$ can be applied to a tree t if $x \in dom_t$.

For a tree $s \in T_\Sigma$, the result $t' = t[x|s]$ of a substitution applied to t is the tree t' with domain $dom_{t'} = (dom_t \setminus x dom_{t_{\downarrow x}}) \cup x dom_s$ and

$$t'(y) = \begin{cases} t(y) & \text{if } y \in dom_t \setminus x dom_{t_{\downarrow x}}, \\ s(z) & \text{if } y = xz \text{ with } z \in dom_s. \end{cases}$$

This means that the subtree $t_{\downarrow x}$ in t is replaced by s. For instance, in Example 1.2, the substitution $[2|c(c)]$ replaces the subtree $t_{\downarrow 2} = a(a, a, c)$ with the tree $c(c)$, which results in the tree $t' = t[2|c(c)] = b(a, c(c), b(b, a), c)$.

For a nonempty hedge $s = s_1 \cdots s_m$, the result $t' = t[x|s]$ of a substitution applied to t (with node x having n right siblings) is the tree t' with domain

$$\begin{aligned} dom_{t'} = (dom_t \setminus \bigcup_{i=0}^{n} (x \oplus i) dom_{t_{\downarrow x \oplus i}}) \\ \cup \{(x \oplus (i-1))w \mid iw \in dom_{s_i},\ 1 \le i \le m,\ w \in \mathbb{N}^*\} \\ \cup \{(x \oplus (i+m-1))w \mid w \in dom_{t_{\downarrow x \oplus i}},\ 1 \le i \le n,\ w \in \mathbb{N}^*\}. \end{aligned}$$

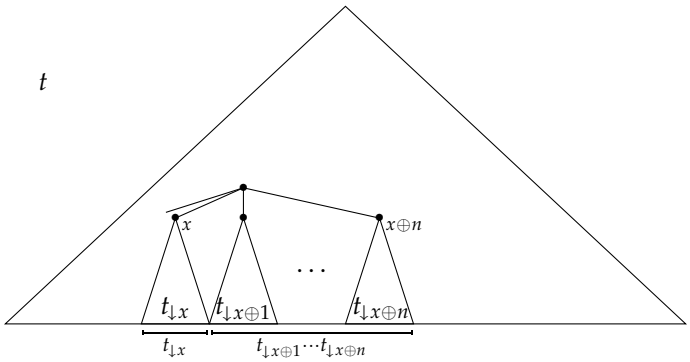
When inserting the hedge s, subtree $t_{\downarrow x}$ is replaced by s_1, trees $s_2, \ldots, s_m$ are inserted next to s_1, and the subtrees of the n right siblings of node x are shifted to the right by $m - 1 = \| s \| - 1$ positions. For the new tree t' it holds that

$$t'(y) = \begin{cases} t(y) & \text{if } y \in dom_t \setminus \bigcup_{i=0}^{n} (x \oplus i) dom_{t_{\downarrow x \oplus i}}, \\ s(iw) & \text{if } y = (x \oplus (i-1))w, 1 \le i \le m, \\ t((x \oplus i)w) & \text{if } y = (x \oplus (i+m-1))w, 1 \le i \le n. \end{cases}$$

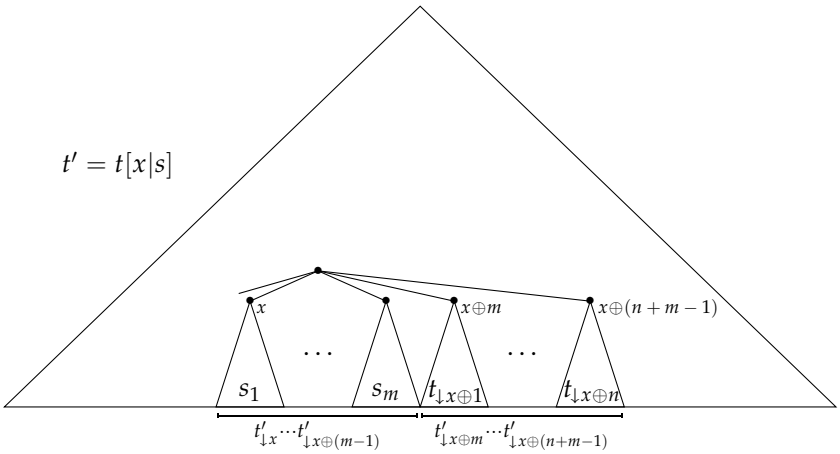
This is depicted in Figure 1.3.

If $m = 0$, i.e. the empty hedge $s = \varepsilon$ is inserted, t' results by shifting the n right siblings of node x to the left by 1, thus deleting $t_{\downarrow x}$.

Figure 1.3 Substitution $t' = t[x|s]$ for hedge $s = (s_1 \cdots s_m)$.



(a) Unranked tree t.



(b) Tree $t' = t[x|s]$.

Encodings for Unranked Trees

Unranked trees can be encoded into binary ones in several ways, two of which will be introduced here.

1. The first child next sibling encoding as introduced in [Nev02b].
2. The extension operator encoding as introduced in [CNT04].

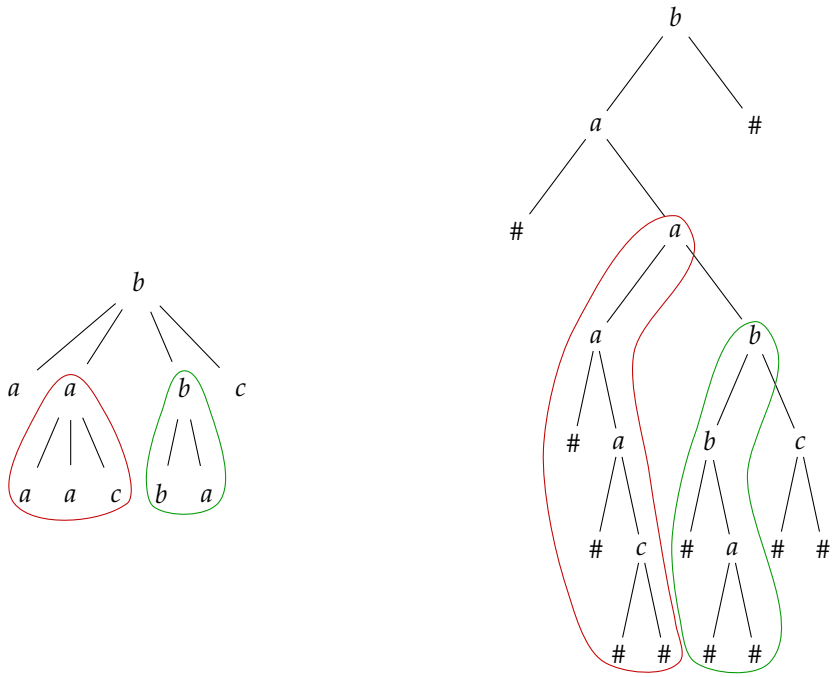
Algorithm 1.1 Algorithm to obtain FCNS-encoding t' of t.

procedure FIRST CHILD NEXT SIBLING(dom_t, $t : dom_t \to \Sigma$)
  $dom_s := dom_t$
  $dom_{s'} := \emptyset$
  $s' := \emptyset$
  while $dom_s \neq \emptyset$ **do**
    take next node $x = x(1) \cdots x(n)$ out of dom_s
    ▷ *encode first child of x as first child, next child of x as right child of its former left sibling*

$$enc(x) = \begin{cases} \varepsilon & \text{if } x = \varepsilon \text{; i.e. } n = 0 \\ enc(x(1) \cdots x(n-1))1 & \text{if } x(n) = 1; n \geq 1 \\ enc(x(1) \cdots x(n-1)\,\underline{x(n)-1})2 & \text{if } x(n) > 1; n \geq 1 \end{cases}$$

    $dom_{s'} := dom_{s'} \cup \{enc(x)\}$
    $s'(enc(x)) := t(x)$
    $dom_s := dom_s \setminus \{x\}$
  end while
  ▷ *fill unused leaves with # symbol*
  while $\exists (x(1) \cdots x(n-1)1 \in dom_{s'} \wedge x(1) \cdots x(n-1)2 \notin dom_{s'})$ **do**
    $dom_{s'} := dom_{s'} \cup \{x(1) \cdots x(n-1)2\}$
    $s'(x(1) \cdots x(n-1)2) := \#$
  end while
  while $\exists (x(1) \cdots x(n-1)2 \in dom_{s'} \wedge x(1) \cdots x(n-1)1 \notin dom_{s'})$ **do**
    $dom_{s'} := dom_{s'} \cup \{x(1) \cdots x(n-1)1\}$
    $s'(x(1) \cdots x(n-1)1) := \#$
  end while
  while $\exists (x \in dom_{s'} \wedge s'(x) \neq \# \wedge \nexists (x1, x2 \in dom_{s'}))$ **do**
    $dom_{s'} := dom_{s'} \cup \{x1, x2\}$
    $s'(x1) := \#$
    $s'(x2) := \#$
  end while
  return $dom_{t'} := dom_{s'}$, $t' := s'$
end procedure

Figure 1.4 FCNS-encoding of Example 1.4.



(a) Unranked tree t.

(b) Ranked encoding t'.

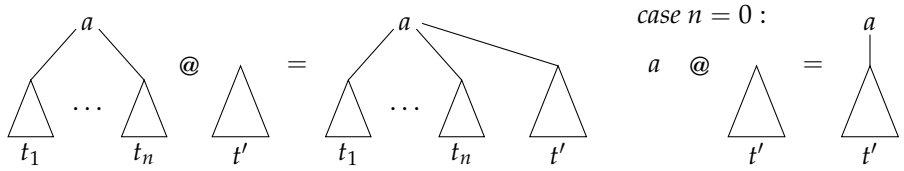
First Child Next Sibling Encoding. For a given tree $t : dom_t \to \Sigma$, with $dom_t \subseteq \mathbb{N}^*$ and an unranked alphabet Σ, the encoding t' of t is obtained by applying Algorithm 1.1.

In Line 7, an unranked tree is encoded into a binary one by appending the first child of a node x as its left child in the resulting binary tree. A right child of a node in the encoding corresponds to its former right sibling in the unranked tree. Thus, if a node has more than one child, each right sibling is encoded as a right child of its former right sibling. Unused positions are filled with the symbol #.

Since the encoding function (cf. Line 7 of Algorithm 1.1) is injective, the decoding of t' can be obtained by setting $dec(x) := enc^{-1}(x)$ (with $dec(x) := \emptyset$ for $t'(x) = \#$) as decoding function.

Example 1.4. Consider the tree t from Example 1.2. Applying Algorithm 1.1, one obtains the tree t' with $\Sigma' = \Sigma'_2 \cup \Sigma'_0$, where $\Sigma'_2 = \{a, b, c\}$, $\Sigma'_0 = \{\#\}$,
$dom_{t'} = \{\varepsilon, 1, 11, 12, 121, 1211, 1212, 12121, 12122, 121221, 121222, 122, 1221, 12211,$
$12212, 122121, 122122, 1222, 12221, 12222, 2\}$, and

Figure 1.5 Application of the extension operator @ according to [CNT04].



$$t'(x) = \begin{cases} a & \text{for } x = 1, 12, 121, 1212, 12212, \\ b & \text{for } x = \varepsilon, 122, 1221, \\ c & \text{for } x = 12122, 1222, \\ \# & \text{else,} \end{cases}$$

which is depicted in Figure 1.4. Note that neither the subtree structure nor the hierarchy levels are preserved by this encoding. ◀

Extension Operator Encoding. Let $@ : T_\Sigma \times T_\Sigma \to T_\Sigma$ be the extension operator for trees. A tree t is extended by t' by adjoining t' as the next sibling of the last child of t: $a(t_1, \ldots, t_n) @ t' = a(t_1, \ldots, t_n, t')$, respectively for case $n = 0$: $a @ t' = a(t')$, which is depicted in Figure 1.5. Note that every unranked tree can be generated uniquely from trees of height 0 using the extension operator: $a(t_1, \ldots, t_n) = [\underbrace{(\cdots(}_{n-1} a @ t_1) @ t_2) \cdots @ t_n]$, and thus, this formalism can be used as an encoding. Note that the tree extension @ is neither associative nor commutative, consequently, if a term is denoted without correct parentheses, it is assumed to be left associative, i.e. the extension operator notation of $a(t_1, \ldots, t_n)$ is shortly written as $a @ t_1 @ \cdots @ t_n$.

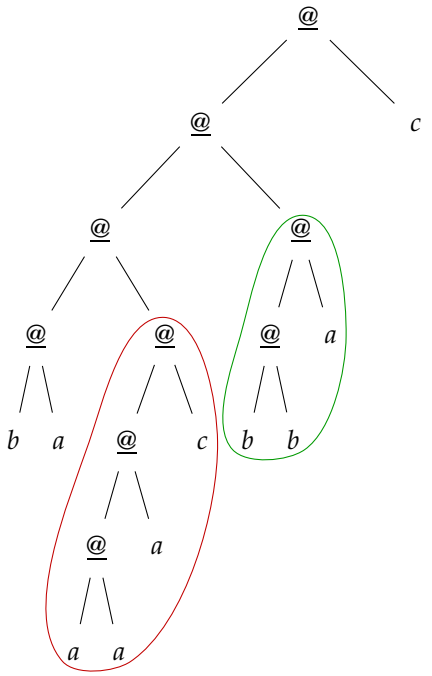
Furthermore, by application of the extension operator trees can also be extended by hedges. Given a tree $t = a(t_1, \ldots, t_n)$ and a hedge $h = s_1 \cdots s_m$, $t @ h$ is defined as $t @ h := a(t_1, \ldots, t_n) @ (s_1 \cdots s_m) = a(t_1, \ldots, t_n) @ s_1 @ \cdots @ s_m = a(t_1, \ldots, t_n, s_1, \ldots, s_m)$. In the context of this thesis, the extension operator @ is transformed from taking multiple arguments in the unranked case into taking two arguments only.

To encode unranked trees over Σ as ranked ones, let $\Sigma_{\underline{@}} := \Sigma_2 \cup \Sigma_0$ with a single binary symbol $\underline{@}$, and $\Sigma_2 := \{\underline{@}\}$, $\Sigma_0 := \Sigma$. Thus, unranked trees correspond precisely to ranked constructions with respect to the function $c_{tree} : T_\Sigma \to T_{\Sigma_{\underline{@}}}$, which satisfies for all unranked trees t_1, t_2 and symbols $a \in \Sigma$

$$c_{tree}(t_1 @ t_2) = \underline{@}\,(c_{tree}(t_1), c_{tree}(t_2)) \text{ and } c_{tree}(a) = a.$$

Hierarchy levels are again altered, but contrary to the first child next sibling encoding, the extension operator encoding preserves the subtree structure of the original tree to some extent, i.e. every subtree in the unranked tree is mapped to

Figure 1.6 Extension operator encoding of Example 1.5.



a subtree in the encoding (however, for the reverse direction from encoding to unranked tree, this claim fails). This will be useful for the first approach towards rewriting systems over unranked trees (cf. Chapter 2 – Partial Subtree Rewriting Systems).

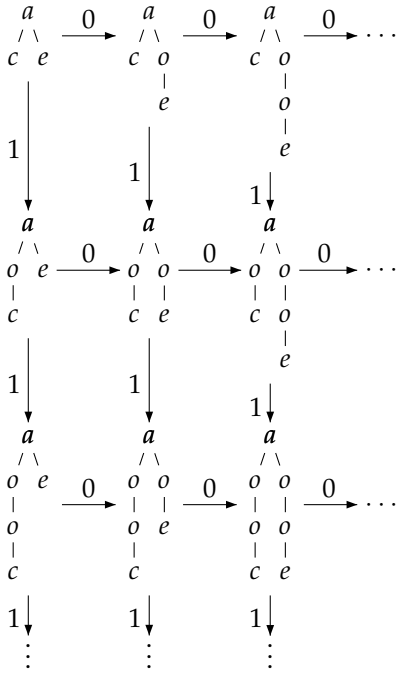
Example 1.5. The notation for the tree from Figure 1.4(a), using the extension operator, is $b \mathbin{@} a \mathbin{@} (a \mathbin{@} a \mathbin{@} a \mathbin{@} c) \mathbin{@} (b \mathbin{@} b \mathbin{@} a) \mathbin{@} c$, or $\Big[\Big((b \mathbin{@} a) \mathbin{@} \Big(((a \mathbin{@} a) \mathbin{@} a) \mathbin{@} c\Big)\Big) \mathbin{@} ((b \mathbin{@} b) \mathbin{@} a)\Big] \mathbin{@} c$, respectively. The corresponding encoded binary tree $t' \in T_{\Sigma_{\underline{@}}}$ with term notation

$$t' = \underline{@}\Big[\underline{@}\Big(\underline{@}\Big(\underline{@}\,(b,a), \underline{@}\,\big(\underline{@}\,(\underline{@}\,(a,a),a),c\big)\Big), \underline{@}\,\big(\underline{@}\,(b,b),a\big)\Big), c\Big]$$

is depicted in Figure 1.6. Note that the subtrees $a(a,a,c)$ and $b(b,a)$ of the unranked tree are mapped to subtrees in the encoding. ◀

Proposition 1.6. [CNT04] *c_{tree} is a bijective mapping between the set of unranked and the set of ranked trees.*

Figure 1.7 Transition graph $G_{\mathcal{R}}$ of the ground tree rewriting system $\mathcal{R}$ of Example 1.7.



Ground Tree Rewriting and Transition Graphs

A *ground tree rewriting system* over ranked trees is a tuple $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$, with ranked alphabet $\Sigma = (\Sigma_i)_{i=0,\dots,k}$, transition alphabet Γ, finite set R of rules of the form $s \overset{\sigma}{\hookrightarrow} s'$ with $s, s' \in T_\Sigma$, and initial tree $t_{in} \in T_\Sigma$. The set R defines the kind of substitutions that are compatible with $\mathcal{R}$ as follows.

A substitution $[x|s']$ is $(\mathcal{R}, \sigma)$-applicable to a tree $t \in T_\Sigma$ if there is a rule $s \overset{\sigma}{\hookrightarrow} s' \in R$ and a node $x \in dom_t$ with $s = t_{\downarrow x}$. It is $\mathcal{R}$-applicable to t if there is a $\sigma \in \Gamma$ such that it is $(\mathcal{R}, \sigma)$-applicable to t. The following notation is applied

- $t \xrightarrow[\mathcal{R}]{\sigma} t'$ if there is an $(\mathcal{R}, \sigma)$-applicable substitution $[x|s']$ such that $t[x|s'] = t'$,
- $t \xrightarrow[\mathcal{R}]{} t'$ iff there is $\sigma \in \Gamma$ with $t \xrightarrow[\mathcal{R}]{\sigma} t'$, and
- $\xrightarrow[\mathcal{R}]{*}$ for the transitive and reflexive closure of $\xrightarrow[\mathcal{R}]{}$.

$T(\mathcal{R}) = \{t \in T_\Sigma \mid t_{in} \to^*_{\mathcal{R}} t\}$ denotes the tree language that is generated by $\mathcal{R}$, while $\mathfrak{R}^{gt}$ denotes the set of all ground tree rewriting systems.

Rewriting systems of this kind have already been considered in [Bra69] with focus on the generated tree language $T(\mathcal{R})$ of a ground tree rewriting system $\mathcal{R}$. In [Löd03], the focus is on the graph structure induced on $T(\mathcal{R})$ by the rewriting relation, which will be the focus of this thesis for rewriting systems over un-

ranked trees. This structure is a directed edge labeled graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}}, \Gamma)$, with $V_{\mathcal{R}} = T(\mathcal{R})$, and $(t, \sigma, t') \in E_{\mathcal{R}}$ iff $t \to_{\mathcal{R}}^{\sigma} t'$. $G_{\mathcal{R}}$ is called the *transition graph* of the ground tree rewriting system $\mathcal{R}$. Note that the set $V_{\mathcal{R}}$ is defined as the set of trees that are reachable from t_{in} by repeated application of the rewrite rules. The class of transition graphs of ground tree rewriting systems is denoted by GTRG.

Example 1.7. Consider the ground tree rewriting system $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$, with

- $\Sigma = \Sigma_2 \cup \Sigma_1 \cup \Sigma_0$, with $\Sigma_2 = \{a\}$, $\Sigma_1 = \{o\}$, $\Sigma_0 = \{c, e\}$,
- $\Gamma = \{0, 1\}$,
- $R = \{c \overset{1}{\hookrightarrow} \begin{smallmatrix} o \\ | \\ c \end{smallmatrix}, e \overset{0}{\hookrightarrow} \begin{smallmatrix} o \\ | \\ e \end{smallmatrix}\}$, and
- $t_{in} = \begin{smallmatrix} & a & \\ / & & \backslash \\ c & & e \end{smallmatrix}$.

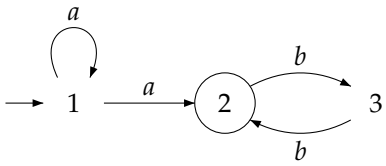
The generated transition graph $G_{\mathcal{R}} = (T(\mathcal{R}), \underset{\mathcal{R}}{\longrightarrow})$ is depicted in Figure 1.7. Since the two rewrite rules can be applied independently to the left respectively to the right branch of each tree, $\mathcal{R}$ generates the two-dimensional infinite grid, whose monadic second order theory is known to be undecidable (cf. e.g. [See72]). ◀

Automata on Words and on Unranked Trees

A *nondeterministic finite automaton (NFA)* on words over a set S is of the form $\mathcal{A} = (P, S, p_0, \Delta, F)$ with nonempty finite state set P, finite input alphabet S, initial state $p_0 \in P$, transition relation $\Delta \subseteq P \times S \times P$, and set of final states $F \subseteq P$.

A *run* of $\mathcal{A}$ from $p \in P$ via a word $w = w(1) \cdots w(|w|)$ to $p' \in P$ is a sequence $\tilde{p}_0, \ldots, \tilde{p}_{|w|}$ of states with $\tilde{p}_0 = p$, $(\tilde{p}_i, w(i+1), \tilde{p}_{i+1}) \in \Delta$ for $0 \leq i < |w|$, and $\tilde{p}_{|w|} = p'$. If such a run exists, it is denoted by $\mathcal{A} : p \xrightarrow{w} p'$. $\mathcal{A}$ *accepts* w iff $\mathcal{A} : p_0 \xrightarrow{w} p'$ for some $p' \in F$. The language recognized by $\mathcal{A}$ is denoted by $L(\mathcal{A}) = \{w \in S^* \mid \mathcal{A} \text{ accepts } w\}$. The class of languages recognized by NFAs over S is exactly the class of languages that can be defined by regular expressions over S (cf. e.g. [HMU01]), which is called the class of *regular languages*, denoted by $REG(S)$.

Example 1.8. Consider the NFA $\mathcal{A} = (P, S, p_0, \Delta, F)$ over the set $S = \{a, b\}$ with state set $P = \{1, 2, 3\}$, transition relation $\Delta = \{(1, a, 1), (1, a, 2), (2, b, 3), (3, b, 2)\}$, and final state set $F = \{2\}$.



The language of $\mathcal{A}$ contains exactly those words over S that start with any positive number of a's which can be followed by an even number (possibly 0) of b's. This is denoted with regular expressions as follows: $L(\mathcal{A}) = a^*a(bb)^* = a^+(bb)^*$. ◀

The theory of finite tree automata arises as a straightforward extension of the theory of finite word automata when words are viewed as unary terms. Only a brief introduction to finite tree automata is provided in this thesis, for extensive analyses cf. e.g. [GS84, CDG+97] for automata on ranked trees and [CLT05] for automata on unranked trees.

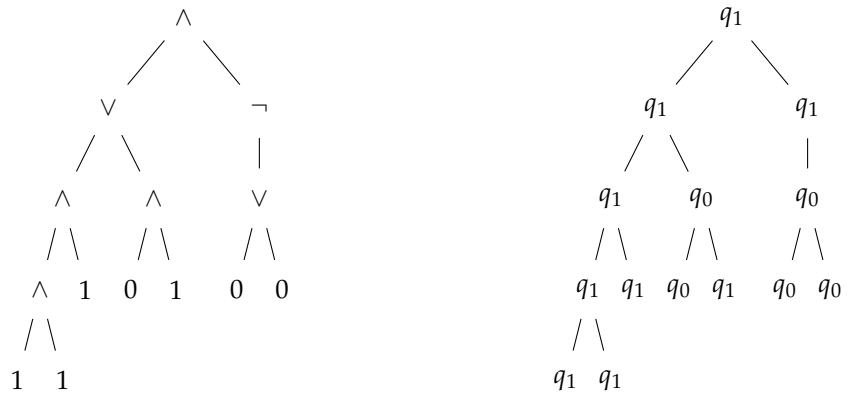
Considering a graphical notation of a tree with the root symbol at the top (cf. e.g. Figure 1.2(b)), a *nondeterministic bottom up tree automaton (N↑TA)* on ranked trees starts its computation at the bottom of the tree on the leaves and moves upward. Formally, an N↑TA over a ranked alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, F)$ with nonempty finite state set Q, final state set $F \subseteq Q$, and transition relation $\Delta \subseteq \bigcup_{k=0}^{n} Q^k \times \Sigma_k \times Q$ for $\Sigma_k \in \Sigma$. For $k = 0$ the transitions $\Sigma_0 \times Q$ are starting pairs which assign states to the leaves of a tree. A run of $\mathcal{A}$ on a ranked tree t is a function $\rho : dom_t \to Q$ such that for each node $x \in dom_t$, if $t(x) \in \Sigma_k$, then $(\rho(x1), \ldots, \rho(xk), t(x), \rho(x)) \in \Delta$. If $\rho(\varepsilon) = q$ for some run ρ of $\mathcal{A}$ on t, this is denoted by $t \to^*_{\mathcal{A}} q$. Consequently, if the run of $\mathcal{A}$ over a subtree $t_{\downarrow x}$ results in $\rho(x) = q$, this is denoted by $t \to^*_{\mathcal{A}} t[x|q]$. For $Q' \subseteq Q$ write $t \to^*_{\mathcal{A}} Q'$ if $t \to^*_{\mathcal{A}} q$ for some $q \in Q'$. A run ρ is accepting if $\rho(\varepsilon) \in F$, and a tree t is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on t. The language recognized by $\mathcal{A}$ is denoted by $T(\mathcal{A}) := \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. *Regular languages of ranked trees* are those that can be accepted by N↑TAs on ranked trees.

In order to define finite automata on finite unranked trees, a simple generalization of tree automata over ranked alphabets can be applied. The transitions are expanded to the form (L_{aq}, a, q), with L_{aq} being a regular language over the state set of the automaton, which can be defined e.g. by an NFA.

A *nondeterministic bottom up tree automaton (N↑TA)* on unranked trees is of the form $\mathcal{A} = (Q, \Sigma, \Delta, F)$ over an unranked alphabet Σ, with Q, $F \subseteq Q$ as for ranked trees and a finite set of transitions $\Delta \subseteq REG(Q) \times \Sigma \times Q$, where $REG(Q)$ can be given e.g. by an NFA over Q. A run of $\mathcal{A}$ on a tree t is a function $\rho : dom_t \to Q$ such that for each node $x \in dom_t$ with k successors $x1, \ldots, xk$, there is a transition $(L_{t(x)\rho(x)}, t(x), \rho(x)) \in \Delta$ with $\rho(x1) \cdots \rho(xk) \in L_{t(x)\rho(x)}$. The language L_{aq} is given for each $q \in Q$ and $a \in \Sigma$ by an NFA $\mathcal{C}_{aq}$ over Q with $L(\mathcal{C}_{aq}) = L_{aq}$. Note that two transitions (L_{aq}, a, q) and (L'_{aq}, a, q) can be collapsed into $(L_{aq} \cup L'_{aq}, a, q)$, and if x is a leaf labelled with a, there must be a transition $(L_{aq}, a, q) \in \Delta$ with $\varepsilon \in L_{aq}$. The notations $t \to^*_{\mathcal{A}} q$, $t \to^*_{\mathcal{A}} t[x|q]$, and $t \to^*_{\mathcal{A}} Q'$ as well as the definition of an accepting run are analogous to the ranked case. The language recognized by $\mathcal{A}$ is denoted by $T(\mathcal{A}) := \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. *Regular languages of unranked trees* are those that can be accepted by N↑TAs on unranked trees.

For upcoming constructions of tree automata, it will be useful to allow a change of states in the automaton without reading a letter from the unranked input alphabet Σ. In order to realize this, the model of ε-N↑TAs is introduced.

Figure 1.8 A run of the N↑TA $\mathcal{A}$ on t of Example 1.9.



(a) Ranked tree t.

(b) Accepting run of $\mathcal{A}$ on t.

An *ε-N↑TA* on unranked trees is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, F)$ with Q, Σ, F as for N↑TAs, and where $\Delta \subseteq (REG(Q) \times \Sigma \times Q) \cup (Q \times Q)$ is the finite transition relation. The only difference to N↑TAs is that transitions of the form (q, q') for $q, q' \in Q$ are allowed. Transitions of this kind are called ε-transitions. Note that analogous to the ranked case, adding ε-transitions does not change the expressiveness of the N↑TA; they can always be eliminated via a standard construction (for each ε-transition $(q, q') \in \Delta$ the transitions $(L_{aq'}, a, q')$ with $(L_{aq}, a, q) \in \Delta$ are added), but they are useful in some constructions and simplify some proofs.

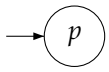
Note that by reversing the transitions of N↑TAs (for ranked and unranked trees alike), and regarding the final states as initial states, one obtains a tree automaton that proceeds in a dual way to the corresponding N↑TA. The tree automaton starts its computation at the root of the tree and works its way downwards, and is thus called a *nondeterministic top down tree automaton (N↓TA)*. Both N↑TAs and N↓TAs recognize the same class of tree languages.

Example 1.9. Consider the N↑TA $\mathcal{A} = (Q, \Sigma_r, \Delta, F)$ over the ranked alphabet $\Sigma_r = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ with $\Sigma_0 = \{0, 1\}$, $\Sigma_1 = \{\neg\}$, and $\Sigma_2 = \{\wedge, \vee\}$ and state set $Q = \{q_0, q_1\}$, final state set $F = \{q_1\}$, and transition relation
$\Delta = \{(0, q_0), (q_0, \neg, q_1), (q_0, q_0, \wedge, q_0), (q_0, q_1, \wedge, q_0), (q_1, q_0, \wedge, q_0), (q_1, q_1, \wedge, q_1),$
$(1, q_1), (q_1, \neg, q_0), (q_0, q_0, \vee, q_0), (q_0, q_1, \vee, q_1), (q_1, q_0, \vee, q_1), (q_1, q_1, \vee, q_1)\}$.
The trees over Σ_r correspond to Boolean expressions, and N↑TA $\mathcal{A}$ accepts the set of trees representing Boolean expressions that evaluate to true. A tree $t \in T_{\Sigma_r}$ corresponding to the Boolean expression $(((1 \wedge 1) \wedge 1) \vee (0 \wedge 1)) \wedge \neg(0 \vee 0) = 1$ and the accepting run of $\mathcal{A}$ on t are depicted in Figure 1.8.

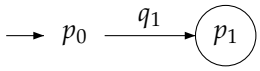
Consider the N↑TA $\mathcal{B} = (Q, \Sigma, \Delta', F)$ over the unranked alphabet $\Sigma = \{0, 1, \neg, \wedge, \vee\}$, and Q, F as for $\mathcal{A}$ with transition relation
$\Delta' = \{(L_{0q_0}, 0, q_0), (L_{\neg q_0}, \neg, q_0), (L_{\wedge q_0}, \wedge, q_0), (L_{\vee q_0}, \vee, q_0),$
$(L_{1q_1}, 1, q_1), (L_{\neg q_1}, \neg, q_1), (L_{\wedge q_1}, \wedge, q_1), (L_{\vee q_1}, \vee, q_1)\}$,

and $L_{0_{q_0}} = L_{1_{q_1}} = \{\varepsilon\}, L_{\neg_{q_0}} = \{q_1\}, L_{\neg_{q_1}} = \{q_0\}, L_{\wedge_{q_0}} = q_1^* q_0 (q_0 + q_1)^*, L_{\wedge_{q_1}} = q_1^+, L_{\vee_{q_0}} = q_0^+, L_{\vee_{q_1}} = q_0^* q_1 (q_0 + q_1)^*$. Exemplary, the NFAs for $L_{0_{q_0}}$, $L_{\neg_{q_0}}$, and $L_{\wedge_{q_0}}$ are given:

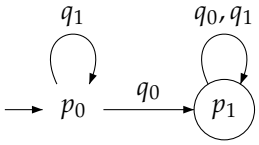
- NFA $\mathcal{C}_{0_{q_0}}$ with $L(\mathcal{C}_{0_{q_0}}) = L_{0_{q_0}} = \{\varepsilon\}$;



- NFA $\mathcal{C}_{\neg_{q_0}}$ with $L(\mathcal{C}_{\neg_{q_0}}) = L_{\neg_{q_0}} = \{q_1\}$;



- NFA $\mathcal{C}_{\wedge_{q_0}}$ with $L(\mathcal{C}_{\wedge_{q_0}}) = L_{\wedge_{q_0}} = q_1^* q_0 (q_0 + q_1)^*$.



It is not difficult to see that the only NFAs applicable at the leaves are $\mathcal{C}_{0_{q_0}}$ and $\mathcal{C}_{1_{q_1}}$ since these are the only NFAs that accept the empty word. Thus, only leaves labeled with 0 or 1 can start a run of $\mathcal{B}$. Since $\mathcal{C}_{\neg_{q_0}}$ and $\mathcal{C}_{\neg_{q_1}}$ only accept the words q_1 respectively q_0, a node with label $\neg$ can only have one successor. It is not difficult to see that N↑TA $\mathcal{B}$ also accepts Boolean expressions that evaluate to true, but allows the overloading of the operators $\wedge$ and $\vee$. ◀

The deterministic version of an N↑TA on unranked trees, a *deterministic bottom up tree automaton (D↑TA)*, has an additional semantic restriction to guarantee determinism: For each $a \in \Sigma$ and all $q, q' \in Q$ it holds that if there are transitions (L, a, q) and (L', a, q'), then $L \cap L' = \emptyset$. Each N↑TA can be transformed into an equivalent D↑TA by the standard subset construction (cf. e.g. [BMW01]). As an alternative to the semantic restriction, one can also define a transition function that syntactically enforces determinism (cf. e.g. [CLT05]).

For a *deterministic top down tree automaton (D↓TA)* on unranked trees, the concept from ranked trees is adapted with the result that, depending on its current state, the current label, and number of successors of the current node, the automaton will assign states to the successors of the current node deterministically, i.e. for each tuple of current state, current label, and number of successors there is exactly one sequence of states that can be assigned to the successors. Analogous to the ranked case, D↓TAs are less expressive than N↓TAs (N↑TAs, D↑TAs), since intuitively, tree properties specified by deterministic top down tree automata can depend on path properties only.

In this thesis a variant of D↓TAs is studied, which is slightly more expressive than the standard definition. These automata can also take the labels of the successors of the current node into account before deciding which states to assign to the successors. Thus, for each tuple of current state, current label, and label sequence of successors of the current node, there is exactly one sequence of states that can be assigned to the successors. Note that whenever a D↓TA is mentioned without further description, this model is meant. One way to formalize this prop-

erty is to use the paradigm of bimachines. Since the intuition is sufficient for the purposes of this thesis, the formal notation is omitted. A more comprehensive introduction to tree automata on unranked trees and the formal notations can be found in e.g. [CLT05].

Deterministic Turing Machines

An extensive introduction to Turing machines is given e.g. in [HMU01]. Here, only a brief introduction to Turing machines will be provided. A *deterministic Turing machine (DTM)* is a tuple $M = (Q, \Sigma, \Gamma, q_{in}, q_h, \delta)$ consisting of

- a finite set Q of states,
- a finite input alphabet Σ,
- a finite tape alphabet Γ with $\Sigma \subseteq \Gamma$ and $\Gamma \cap Q = \emptyset$, containing a $\sqcup$ symbol, with $\sqcup \notin \Sigma$,
- an initial state $q_{in} \in Q$,
- a unique halting state q_h, and
- a transition function $\delta \subseteq (Q \setminus \{q_h\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

In the context of this thesis, the working tape of the deterministic Turing machine is assumed to be infinite to the left side only. Note that a semi-infinite tape suffices to ensure the complete expressiveness of a deterministic Turing machine (cf. e.g. [HMU01]). The transition function is specified completely.

A *configuration* κ of M is a word $\kappa = a_1 \cdots a_n q b_1 \cdots b_m$ with $a_i, b_j \in \Gamma$ for $1 \leq i \leq n$ and $1 \leq j \leq m$, and $q \in Q$. A configuration κ can be reduced to its minimal relevant part by deleting all $\sqcup$ symbols on the left and right ends of κ, resulting in $red(\kappa) = a_k \cdots a_n q b_1 \cdots b_\ell$ with $a_k \neq \sqcup \neq b_\ell$. Two configurations κ, κ' are equivalent iff $red(\kappa) = red(\kappa')$. In the following, equivalent configurations are identified. The symbol to the right of the current state is said to be in the *working cell* of the Turing machine tape. If there is no symbol of Σ left in the working cell, a $\sqcup$ symbol is added. Adding $\sqcup$ symbols to the left or right side of the tape inscription always results in an equivalent configuration, so if there are no symbols to either side of the working cell, it is assumed that a $\sqcup$ symbol is appended to the appropriate side of the tape inscription. The *successor configuration* of $\kappa = a_1 \cdots a_n q b_1 \cdots b_m$ is

- $\kappa' = a_1 \cdots a_{n-1} q' a_n c b_2 \cdots b_m$ for the left transition $\delta(q, b_1) = (q', c, L)$, and
- $\kappa' = a_1 \cdots a_n c q' b_2 \cdots b_m$ for the right transition $\delta(q, b_1) = (q', c, R)$.

If κ' is the successor configuration of κ, this is denoted by $\kappa \vdash \kappa'$. $\vdash^*$ denotes the transitive and reflexive closure of $\vdash$.

Symbolic Model Checking and Reachability Problems

In general, symbolic model checking is the task to decide whether a given structure fulfills a given property (cf. [Eme96]). Formally, the model checking problem can be stated as follows: given a desired property, expressed as a temporal logic formula φ (often given in CTL, LTL, or CTL*, cf. [Eme90]), and a structure M from a class of specified structures with initial state s, decide if $M, s \models \varphi$, i.e. "M with starting vertex s is a model of φ".

In the theory of algorithmic verification, the classical framework for modeling systems is given by finite transition systems. This classical idea was then extended to cover infinite systems to allow algorithmic approaches to tasks like verification of infinite state systems. These systems have to be given effectively, i.e. by some finite object. The formalism of tree (term) rewriting systems representing the corresponding transition graphs meets this requirement. In this framework most of the system analysis problems of model checking reduce to (various kinds of) reachability problems on these models. For the specification of infinite sets of trees, the regular sets and thus finite tree automata are employed.

The basic reachability problems (denoted with the corresponding CTL* formulae) include:

One step reachability ($\varphi = EX$): Given a rewriting system $\mathcal{R}$, a vertex t, and a regular set T of vertices of $G_\mathcal{R}$, is there a direct successor of t that is in T?

Reachability ($\varphi = EF$): Given a rewriting system $\mathcal{R}$, a vertex t, and a regular set T of vertices of $G_\mathcal{R}$, is there a path from t to a vertex $t' \in T$?

Constrained reachability ($\varphi = EU$): Given a rewriting system $\mathcal{R}$, a vertex t, and regular sets T_1, T_2 of vertices of $G_\mathcal{R}$, is there a path starting in t that only visits vertices of T_1 until it eventually reaches a vertex $t' \in T_2$?

Recurrence ($\varphi = EGF$): Given a rewriting system $\mathcal{R}$, a vertex t, and a regular set T of vertices of $G_\mathcal{R}$, is there a path starting in t that infinitely often visits T?

All of the above mentioned problems can also be considered in their universal version, requiring the respective property for all paths starting in t.

Universal one step reachability ($\varphi = AX$): Given a rewriting system $\mathcal{R}$, a vertex t, and a regular set T of vertices of $G_\mathcal{R}$, are all direct successors of t in T?

Universal reachability ($\varphi = AF$): Given a rewriting system $\mathcal{R}$, a vertex t, and a regular set T of vertices of $G_\mathcal{R}$, do all paths starting in t eventually reach vertices of T?

Figure 1.9 Decidability results for ground tree rewriting systems (cf. [Löd03]).

one step reachability (EX)	decidable
reachability (EF)	decidable
constrained reachability (EU)	undecidable
recurrence (EGF)	decidable
universal one step reachability (AX)	decidable
universal reachability (AF)	undecidable
universal constrained reachability (AU)	undecidable
universal recurrence (AGF)	undecidable

Universal constrained reachability ($\varphi = AU$): Given a rewriting system $\mathcal{R}$, a vertex t, and regular sets T_1, T_2 of vertices of $G_\mathcal{R}$, do all paths starting in t visit only vertices of T_1 until it eventually reaches vertices of T_2?

Universal recurrence ($\varphi = AGF$): Given a rewriting system $\mathcal{R}$, a vertex t, and a regular set T of vertices of $G_\mathcal{R}$, do all paths starting in t infinitely often visit T?

These problems have been investigated intensively in the past for various kinds of rewriting systems. The well studied ground tree rewriting systems over ranked trees (cf. e.g. [Bra69, CDGV94, DT90, Löd02b, CDG+97]; for an extensive survey on ground tree rewriting systems confer [Löd03]) build the foundation of this research over unranked trees.

Table 1.9 gives an overview of proven results for ground tree rewriting systems (cf. [Löd03]) for the basic reachability problems introduced above.

Note that all sets of trees are defined to be regular sets of trees, i.e. they are recognizable by N↑TAs, D↑TAs, or N↓TAs. *Deterministic top down* tree automata are strictly less expressive, and if the set T_1 of the constrained reachability problem is given by a D↓TA, the problem is again decidable. This result follows from the research in [Col02], and will be briefly explained in the following. To not confuse this problem with the "classical" constrained reachability, it will be denoted as the reachability problem over $G_\mathcal{R}|_{T(\mathcal{A})} = (T(\mathcal{A}), \rightarrow_\mathcal{R})$, the restriction of the transition graph $G_\mathcal{R}$ of a ground tree rewriting system $\mathcal{R}$ to the tree language $T(\mathcal{A})$ for a D↓TA $\mathcal{A}$.

The reachability problem over $G_\mathcal{R}|_{T(\mathcal{A})} = (T(\mathcal{A}), \rightarrow_\mathcal{R})$ for a ground tree rewriting system $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$ and a D↓TA $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ can be reduced to the reachability problem EF over a ground tree rewriting system $\mathcal{R}' = (\Sigma \times Q, \Gamma, R', t'_{in})$. The main idea is to construct $\mathcal{R}'$ such that the transition function δ of $\mathcal{A}$ is already simulated by the rewrite rules. Thus, $\mathcal{R}'$ operates on trees whose node labels are pairs over $\Sigma \times Q$, starting with the initial state at the root of the initial tree: $t'_{in}(\varepsilon) = (t_{in}(\varepsilon), q_0)$. Since $\mathcal{A}$ is deterministic, the node labels of the

initial tree can be determined uniquely, and conform to the transition function δ: for a node x with m successors, and $t'_{in}(x) = (t_{in}(x), q)$, the Q components of the node labels of the successors are derived from $\delta_m(t_{in}(x), q) = (q_1, \ldots, q_m)$, resulting in $t'_{in}(x1) = (t_{in}(x1), q_1), \ldots, t'_{in}(xm) = (t_{in}(xm), q_m)$. Additionally, the trees over $\Sigma \times Q$ are defined such that only $\mathcal{A}$-accepting pairs are valid at the front. For a rewrite rule $s_1 \overset{\sigma}{\hookrightarrow} s_2 \in R$, the substitution for $t_{\downarrow x} = s_1$ is initialized with $t'(x, q) := (s_2(\varepsilon), q)$, and proceeds analogously to the technique described above. Again, since $\mathcal{A}$ is deterministic, the rules can be supplemented uniquely for the respective state q at the subtree where they are applied.

Hence, the vertex set of $\mathcal{R}'$ consists of trees of $T(\mathcal{A})$ only, and $\mathcal{R}'$ is a ground tree rewriting system over $\Sigma \times Q$. Consequently, the reachability problem over $G_{\mathcal{R}}|_{T(\mathcal{A})}$ for ground tree rewriting system $\mathcal{R}$ and D$\downarrow$TA $\mathcal{A}$ is reduced to the reachability problem EF over the ground tree rewriting system $\mathcal{R}'$, and thus, the reachability problem over $G_{\mathcal{R}}|_{T(\mathcal{A})}$ is decidable.

Chapter 2

Partial Subtree Rewriting Systems

As mentioned in the introduction, ground tree rewriting systems have a decidable first order logic with a reachability predicate. The interest of this chapter is to transfer this practical aspect to rewriting systems over unranked trees. In Section 2.1 partial subtree rewriting systems are defined, and it is shown that these generate the same class of transition graphs as ground tree rewriting systems. In Section 2.2 results for ground tree rewriting systems are tested for transferability.

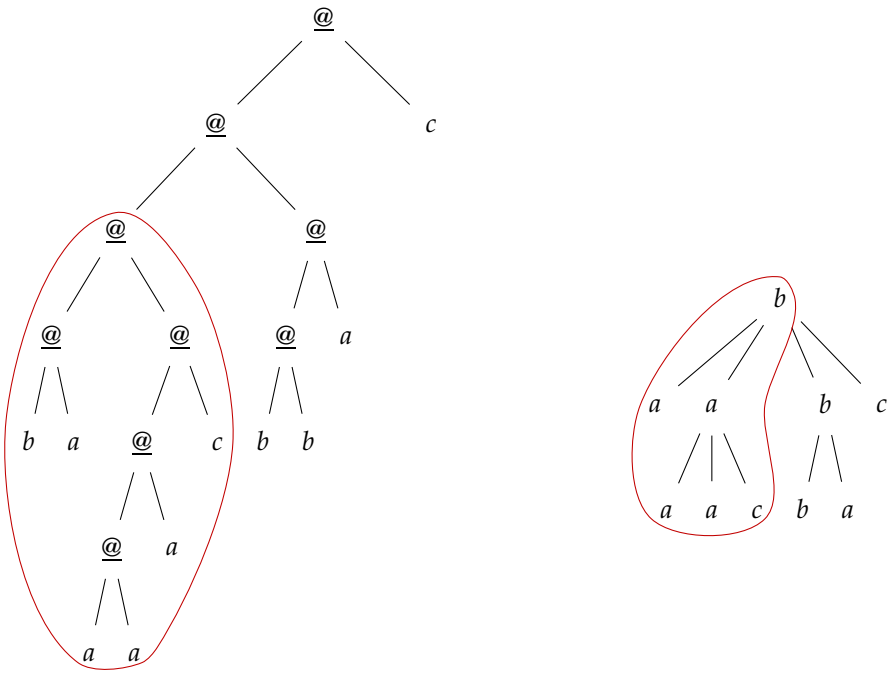
2.1 Decidability Result via Ground Tree Rewriting over Encodings

A natural approach to transfer results for ground tree rewriting systems to unranked trees is to encode unranked trees as ranked ones. Unranked trees that result from applying finitely many subtree rewrite rules with constant trees, starting from a fixed initial tree, would be of bounded branching, hence this could be traced back to the case of ground tree rewriting over ranked trees. Thus, the direct transfer of the ground tree rewriting principle is not of interest and new rewriting techniques need to be defined.

In Chapter 1 – Preliminaries, two encodings were introduced: the first child next sibling encoding as well as the extension operator encoding. The rewriting relation of ground tree rewriting systems consists of subtree substitutions, i.e. every tree of the vertex set of a transition graph of a ground tree rewriting system is derived by repeated subtree substitution from the initial tree. Since the first child next sibling encoding does not preserve the subtree structure of unranked trees, it is not suitable for this purpose. The extension operator encoding however preserves the subtree structure to some extent: subtrees in unranked trees are mapped to subtrees in the encoding. In the encoded tree however not all subtrees are mapped to subtrees in the unranked tree, but rather to "partial" subtrees including the root of the subtree. The remaining parts of the subtrees are hedges on the right side, which are not altered when applying ground tree rewrite rules to the extension operator encoding of unranked trees (cf. Figure 2.1(b), the remaining hedge is $b(b, a), c$). With some technical effort however, this requirement can be met, and for every rewriting system of this kind over unranked trees one can construct a ground tree rewriting system that generates an isomorphic transition graph.

It will then be shown that conversely for every ground tree rewriting system, one can construct a rewriting system of the type mentioned above over unranked

Figure 2.1 Illustration of the partial subtree corresponding to a proper subtree in the extension operator encoding.



(a) Ranked extension operator encoding t'.

(b) Unranked tree t.

trees that generates an isomorphic transition graph, and thus, the rewriting systems generate the same class of infinite graphs.

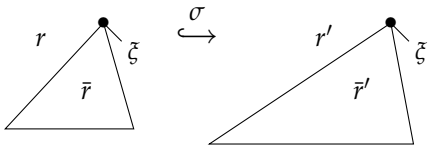
First the set $T_{\Sigma,\xi}$ over an unranked alphabet Σ and a variable ξ is defined, which corresponds exactly to the above mentioned partial subtree of the unranked tree which is to be substituted according to the rewriting relation. The trees of $T_{\Sigma,\xi}$ contain a variable ξ, which may only occur at level 1 as a leaf and as the rightmost child of the root. If ξ is substituted by the unchanged hedge, the subtree substitution of ground tree rewriting systems over the extension operator encodings is realized.

Definition 2.1. The set $T_{\Sigma,\xi}$ is the set of all unranked trees over Σ with one occurrence of the variable ξ as leaf and rightmost child of the root:

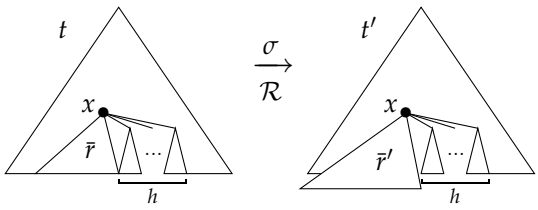
$$T_{\Sigma,\xi} := \Big\{ t \in T_\Sigma(\{\xi\}) \mid \exists y \in dom_t \big[t(y) = \xi \wedge \neg\exists z (y \neq z \wedge t(z) = \xi) \\ \wedge \neg\exists z (y \prec z) \wedge \neg\exists z (z = y + 1) \wedge |y| = 1 \big] \Big\}.$$

Since usually the variable ξ is to be substituted for $t \in T_{\Sigma,\xi}$, denote $t[\xi|s]$ shortly as $t[s]$ for hedge s over Σ, respectively for tree $s \in T_\Sigma$. Note that any tree $t \in T_{\Sigma,\xi}$ can be written as $t = \bar{t} \mathbin{@} \xi$ with $\bar{t} \in T_\Sigma$.

Figure 2.2 Rewrite rules of $\mathcal{R} \in \mathfrak{R}^{ps}$ of Definition 2.2.



(a) Rule $r \overset{\sigma}{\hookrightarrow} r' \in R$ with $r, r' \in T_{\Sigma,\xi}$.



(b) Application of rewrite rule $r \overset{\sigma}{\hookrightarrow} r' \in R$.

Definition 2.2. A *partial subtree rewriting system* over unranked trees in T_Σ is of the form $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$, with an unranked alphabet Σ, a transition alphabet Γ, a finite set of rules R, and an initial tree t_{in}. R consists of subtree rewrite rules over trees of $T_{\Sigma,\xi}$ as defined above of the form: $r \overset{\sigma}{\hookrightarrow} r'$ with $r, r' \in T_{\Sigma,\xi}$ as depicted in Figure 2.2(a).

t' is derived from t ($t \to^*_{\mathcal{R}} t'$), if there are a node $x \in dom_t$, a hedge h over Σ, and a rule $r \overset{\sigma}{\hookrightarrow} r' \in R$, such that $r[h] = t_{\downarrow x}$, and there is a substitution $[x|r'[h]]$ such that $t[x|r'[h]] = t'$ (cf. Figure 2.2(b)).

The set of all partial subtree rewriting systems is denoted by $\mathfrak{R}^{ps}$, and the class of transition graphs of partial subtree rewriting systems is denoted by PSRG.

With these definitions it will be shown that partial subtree rewriting systems over unranked trees and ground tree rewriting systems over ranked trees generate equal transition graphs up to isomorphism.

Theorem 2.3. *Partial subtree rewriting systems generate the same class of transition graphs as ground tree rewriting systems.*

PROOF. First, it will be shown that for every partial subtree rewriting system $\mathcal{R}$ over unranked trees, one can construct a ground tree rewriting system $\mathcal{S}$ that generates an isomorphic transition graph.

For $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$ construct $\mathcal{S} = (\Sigma_{\underline{@}}, \Gamma, S, t'_{in})$ with ranked alphabet $\Sigma_{\underline{@}} = \Sigma_2 \cup \Sigma_0$ and $\Sigma_2 = \{\underline{@}\}$, $\Sigma_0 = \Sigma$. The initial tree t'_{in} of $\mathcal{S}$ is the extension operator encoding of t_{in} of $\mathcal{R}$, i.e. $t'_{in} := c_{tree}(t_{in})$. For the set S of rules, define for every rule $\bar{r} \,@\, \xi \overset{\sigma}{\hookrightarrow} \bar{r}' \,@\, \xi \in R$ a rule $s \overset{\sigma}{\hookrightarrow} s' \in S$ with $s = c_{tree}(\bar{r})$ and $s' = c_{tree}(\bar{r}')$.

With the vertex sets of transition graphs $G_{\mathcal{R}}$ and $G_{\mathcal{S}}$ consisting only of trees that are reachable from the respective initial tree, it suffices to show for the correctness of this construction that

1. for every $t, t' \in T_\Sigma$ it holds that $t \xrightarrow[\mathcal{R}]{\sigma} t' \Rightarrow c_{tree}(t) \xrightarrow[\mathcal{S}]{\sigma} c_{tree}(t')$, and
2. for every $\tau, \tau' \in T_{\Sigma_{@}}$ it holds that $\tau \xrightarrow[\mathcal{S}]{\sigma} \tau' \Rightarrow c_{tree}^{-1}(\tau) \xrightarrow[\mathcal{R}]{\sigma} c_{tree}(\tau')$,

i.e. $(t, \sigma, t') \in R$ iff $(c_{tree}(t), \sigma, c_{tree}(t')) \in S$.

1. $t \xrightarrow[\mathcal{R}]{\sigma} t' \Rightarrow c_{tree}(t) \xrightarrow[\mathcal{S}]{\sigma} c_{tree}(t')$

   Since the extension operator encoding of the unranked tree t maps subtrees of t to subtrees in $c_{tree}(t)$, the context C of $t_{\downarrow x}$ remains unchanged, that is $c_{tree}(t) = c_{tree}(C[t_{\downarrow x}]) = c_{tree}(C)[c_{tree}(t_{\downarrow x})]$, and similarly $c_{tree}(t') = c_{tree}(C[t'_{\downarrow x}])$ $= c_{tree}(C)[c_{tree}(t'_{\downarrow x})]$.

   $$\begin{array}{ccc} t = C[t_{\downarrow x}] & \xrightarrow[\mathcal{R}]{\sigma} & t' = C[t'_{\downarrow x}] \quad \text{unranked} \\ \downarrow c_{tree} & & \downarrow c_{tree} \\ c_{tree}(C[t_{\downarrow x}]) & \overset{\sigma}{\underset{\mathcal{S}}{\dashrightarrow}} & c_{tree}(C[t'_{\downarrow x}]) \\ = c_{tree}(C)[c_{tree}(t_{\downarrow x})] & & = c_{tree}(C)[c_{tree}(t'_{\downarrow x})] \quad \text{ranked} \end{array}$$

   Thus, it suffices to show that $c_{tree}(t_{\downarrow x}) \xrightarrow[\mathcal{S}]{\sigma} c_{tree}(t'_{\downarrow x})$.

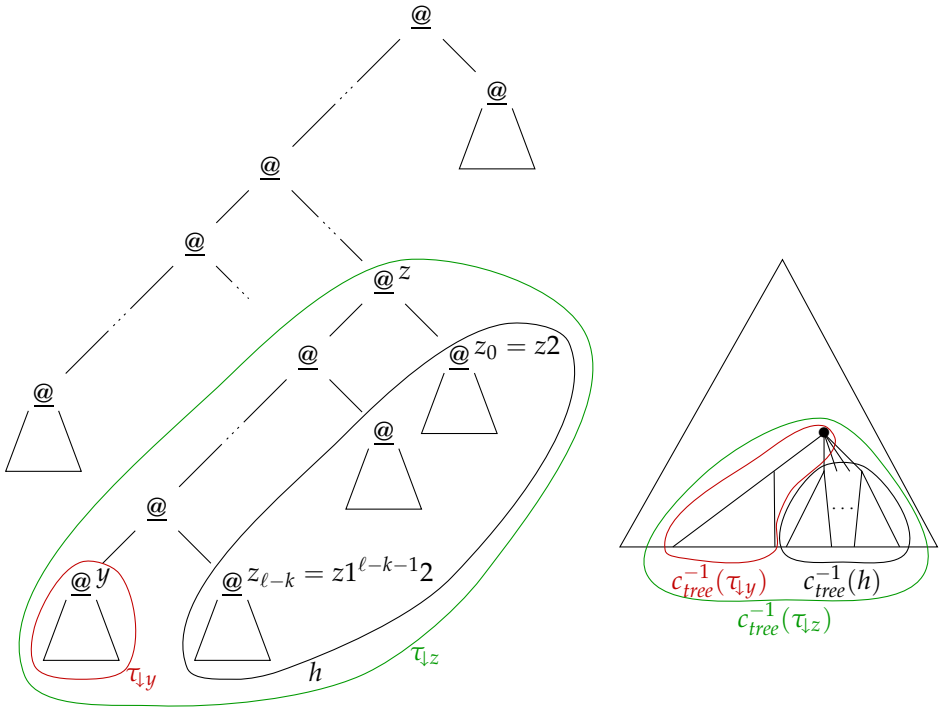
   With $t \to_{\mathcal{R}}^{\sigma} t'$ there exist a rule $r \overset{\sigma}{\hookrightarrow} r' \in R$ with $r, r' \in T_{\Sigma,\xi}$, a hedge $h = s_1 @ \cdots @ s_m$ over Σ, and a substitution $[x|r'[h]]$ for some $x \in dom_t$, such that $t_{\downarrow x} = r[h]$ and $t'_{\downarrow x} = r'[h]$. According to the construction, there is a rule $s \overset{\sigma}{\hookrightarrow} s' \in S$ with $s = c_{tree}(\bar{r})$ and $s' = c_{tree}(\bar{r}')$. Hence,

   $$\begin{array}{rl} & c_{tree}(t_{\downarrow x}) = c_{tree}(r[h]) = c_{tree}(\bar{r} @ h) \\ = & c_{tree}(\bar{r} @ s_1 @ \cdots @ s_m) \\ = & c_{tree}(\bar{r}) @ c_{tree}(s_1 @ \cdots @ s_m) \\ = & s @ c_{tree}(s_1 @ \cdots @ s_m) \\ \xrightarrow[\mathcal{S}]{\sigma} & s' @ c_{tree}(s_1 @ \cdots s_m) \\ = & c_{tree}(\bar{r}') @ c_{tree}(s_1 @ \cdots @ s_m) \\ = & c_{tree}(\bar{r}' @ s_1 @ \cdots @ s_m) \\ = & c_{tree}(\bar{r}' @ h) = c_{tree}(r'[h]) = c_{tree}(t'_{\downarrow x}). \end{array}$$

2. $\tau \xrightarrow[\mathcal{S}]{\sigma} \tau' \Rightarrow c_{tree}^{-1}(\tau) \xrightarrow[\mathcal{R}]{\sigma} c_{tree}^{-1}(\tau')$

   With $\tau \to_{\mathcal{S}}^{\sigma} \tau'$ there exist a rule $s \overset{\sigma}{\hookrightarrow} s' \in S$ and some $y \in dom_\tau$ with $\tau_{\downarrow y} = s$ and $\tau'_{\downarrow y} = s'$. According to the construction, there is a corresponding rule $r \overset{\sigma}{\hookrightarrow} r' \in R$ with $r = \bar{r} @ \xi$, $r' = \bar{r}' @ \xi$, and a suitable hedge h with $r[h] = \bar{r} @ h$ and $r'[h] = \bar{r}' @ h$, where $s = c_{tree}(\bar{r})$ and $s' = c_{tree}(\bar{r}')$. Two cases are to be distinguished.

Figure 2.3 Illustration of the location of $\tau_{\downarrow z} = \tau_{\downarrow y} @ h$ and of $c_{tree}^{-1}(\tau_{\downarrow z})$.



(a) Ranked encoding τ.

(b) Unranked tree $t = c_{tree}^{-1}(\tau)$.

Case (a). $y = \varepsilon$ or $y = y_1 \cdots y_\ell$ with $y_i \in \{1,2\}$ for $1 \leq i \leq \ell - 1, \ell \in \mathbb{N}$ and $y_\ell = 2$; i.e. y is a right child of an @ node. According to the definition, the right argument of the extension operator corresponds to a subtree, which is appended as the rightmost child of the root of the tree, i.e. $\tau_{\downarrow y}$ is mapped to an entire subtree in the unranked tree. In this case, hedge h is unified with the empty hedge, and it holds that $r[h] = \bar{r} @ h = \bar{r}$ and $r'[h] = \bar{r}' @ h = \bar{r}'$. Since the context C of $\tau_{\downarrow y}$ again remains unchanged, and since $c_{tree}^{-1}(\tau) = c_{tree}^{-1}(C[\tau_{\downarrow y}]) = c_{tree}^{-1}(C)[c_{tree}^{-1}(\tau_{\downarrow y})]$ and $c_{tree}^{-1}(\tau') = c_{tree}^{-1}(C[\tau'_{\downarrow y}]) = c_{tree}^{-1}(C)[c_{tree}^{-1}(\tau'_{\downarrow y})]$, it suffices to show that $c_{tree}^{-1}(\tau_{\downarrow y}) \xrightarrow[\mathcal{R}]{\sigma} c_{tree}^{-1}(\tau'_{\downarrow y})$.

$$\begin{array}{lcl} c_{tree}^{-1}(\tau_{\downarrow y}) & = & c_{tree}^{-1}(s) = c_{tree}^{-1}(c_{tree}(\bar{r})) = \bar{r} = r[h] \\ & \xrightarrow[\mathcal{R}]{\sigma} & r'[h] = \bar{r}' = c_{tree}^{-1}(c_{tree}(\bar{r}')) = c_{tree}^{-1}(s') \\ & = & c_{tree}^{-1}(\tau'_{\downarrow y}) \end{array}$$

Case (b). $y = y_1 \cdots y_\ell$ with $y_\ell = 1$. According to the definition of $c_{tree}(t)$, a left child of an @ node always contains the root of a subtree of the corresponding unranked tree t. Hence, in order to get a node in τ that corresponds to an *entire* subtree in $c_{tree}^{-1}(\tau)$, one needs to find a right child z of an @ node, with

$z \preceq y$. Thus, take node

$$z = \begin{cases} \varepsilon & \text{for } y_i = 1 \text{ for all } 1 \leq i \leq \ell, \\ y_1 \cdots y_k & \text{else, with } k = max\{i \mid y_i = 2\}. \end{cases}$$

Assign the hedge $h = s_0 \cdots s_{\ell-k-1}$ as the decoding of all subtrees of τ whose roots are right children between z and y, i.e. $s_i := c^{-1}_{tree}(\tau_{\downarrow z_i})$ with $z_i = z1^i2$ for $0 \leq i \leq \ell - k - 1$ (where 1^i denotes $1 \cdots 1$ with $|1 \cdots 1| = i$, cf. Figure 2.3), and $\tau_{\downarrow z} = \big(\cdots((\tau_{\downarrow y} @ \tau_{\downarrow z_{\ell-k-1}}) @ \tau_{\downarrow z_{\ell-k-2}}) \cdots\big) @ \tau_{\downarrow z_0}$. Thus, the context C of $\tau_{\downarrow z}$ again remains unchanged, so $c^{-1}_{tree}(\tau) = c^{-1}_{tree}(C[\tau_{\downarrow z}]) = c^{-1}_{tree}(C)[c^{-1}_{tree}(\tau_{\downarrow z})]$ and $c^{-1}_{tree}(\tau') = c^{-1}_{tree}(C[\tau'_{\downarrow z}]) = c^{-1}_{tree}(C)[c^{-1}_{tree}(\tau'_{\downarrow z})]$ hold, and one obtains:

$$\begin{aligned}
& c^{-1}_{tree}(\tau_{\downarrow z}) = c^{-1}_{tree}(\tau_{\downarrow y} @ \tau_{\downarrow z_{\ell-k-1}} @ \cdots @ \tau_{\downarrow z_0}) \\
= \;& c^{-1}_{tree}(\tau_{\downarrow y}) @ c^{-1}_{tree}(\tau_{\downarrow z_{\ell-k-1}}) @ \cdots @ c^{-1}_{tree}(\tau_{\downarrow z_0}) \\
= \;& c^{-1}_{tree}(c_{tree}(\bar{r})) @ s_{\ell-k-1} @ \cdots @ s_0 \\
= \;& \bar{r} @ s_{\ell-k-1} @ \cdots @ s_0 \\
= \;& r[h] \\
\xrightarrow[\mathcal{R}]{\sigma} \;& r'[h] \\
= \;& \bar{r}' @ s_{\ell-k-1} @ \cdots @ s_0 \\
= \;& c^{-1}_{tree}(c_{tree}(\bar{r}')) @ s_{\ell-k-1} @ \cdots @ s_0 \\
= \;& c^{-1}_{tree}(\tau'_{\downarrow y}) @ c^{-1}_{tree}(\tau_{\downarrow z_{\ell-k-1}}) @ \cdots @ c^{-1}_{tree}(\tau_{\downarrow z_0}) \\
= \;& c^{-1}_{tree}(\tau'_{\downarrow y} @ \tau_{\downarrow z_{\ell-k-1}} @ \cdots @ \tau_{\downarrow z_0}) = c^{-1}_{tree}(\tau'_{\downarrow z}).
\end{aligned}$$

It remains to be shown that for a ground tree rewriting system $\mathcal{S}$, one can construct a partial subtree rewriting system $\mathcal{R}$ that generates an isomorphic transition graph. Since ranked trees can be viewed as unranked trees, and since each symbol has a unique rank, the construction of $\mathcal{R}$ is straightforward.

Let $\mathcal{S} = (\Sigma_r, \Gamma, S, t_{in})$ be a ground tree rewriting system over a ranked alphabet Σ_r. Construct a partial subtree rewriting system $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$ with $\Sigma = \Sigma_r$ (the ranks of the symbols are simply omitted), and the same initial tree t_{in}. For each rule $s \overset{\sigma}{\hookrightarrow} s' \in S$, define $r \overset{\sigma}{\hookrightarrow} r' \in R$ with $r, r' \in T_{\Sigma,\xi}$ such that $\bar{r} = s$ and $\bar{r}' = s'$.

Since each symbol in Σ_r has a unique rank, the transferred rewrite rules of the partial subtree rewriting system $\mathcal{R}$ cannot alter the unranked trees in any other way than the ground tree rewriting system $\mathcal{S}$, thereby preserving the ranks. Thus, in each rewriting step the variable ξ can only be substituted by the empty hedge. With the same initial tree for both rewriting systems, it is not difficult to see that the transition graph $G_{\mathcal{R}}$ of $\mathcal{R}$ is isomorphic to the transition graph $G_{\mathcal{S}}$ of $\mathcal{S}$. ■

With this equivalence between the classes of transition graphs, several results for ground tree rewriting systems can be transferred to partial subtree rewriting systems. However, since an encoding of unranked trees as ranked ones does lose some structural information, it will be shown that there are still concepts which cannot be transferred.

2.2 Properties of Partial Subtree Rewriting Systems

In the previous section it was shown that the classes of transition graphs of partial subtree rewriting systems over unranked trees and of ground tree rewriting systems over ranked trees coincide. This means that by disregarding the inner structure of the vertices (unranked vs. ranked trees), the transition graphs are of identical structure. Hence, known properties of ground tree rewriting graphs can be transferred to partial subtree rewriting graphs with the restriction that the inner structure of the trees is not taken into account. This includes the basic reachability problems which were introduced in Chapter 1 – Preliminaries, with an overview of the decidability results of [Löd03] in Table 1.9.

Furthermore, the regularity of sets of trees under pre* and post* is preserved; an algorithm from [Löd02b], based on earlier results (cf. [CDGV94] for term rewriting systems and [EHRS00] for pushdown systems), calculating the set of trees from which one can reach the specified set T of trees, can be transferred directly. To obtain the set post* of T, the same algorithm can be applied for the reversed rewriting system, i.e. the left and right hand sides of the rules are swapped.

In addition, the first order theory over ground tree rewriting systems is decidable (cf. [DT90]), which is again transferable to partial subtree rewriting systems. The first order theory is composed of Boolean combinations of local properties (cf. [Gai82]), but does not have any formalism to express temporal properties. Therefore it is considered together with a reachability predicate (i.e. the rewriting relation $\rightarrow_{\mathcal{R}}^*$) in order to further categorize classes of infinite graphs with respect to their decidable properties. Thus, decidable problems for partial subtree rewriting systems include:

- preservation of the regularity of sets of trees under pre* and post* (cf. [Löd02b, Löd03, CDGV94, EHRS00]),
- first order theory (cf. [DT90, CDG$^+$97]),
- one step reachability (EX) (cf. [Löd03, CDG$^+$97] for ground tree rewriting systems),
- reachability (EF) (cf. [Bra69, Löd03, CDG$^+$97]),
- universal one step reachability (AX) (cf. [Löd03]).

The undecidability results for ground tree rewriting systems, which do not rely on the structure of the trees, can also be transferred. From [Löd03], one obtains the following undecidable reachability problems for partial subtree rewriting systems:

- constrained reachability (EU),
- universal reachability (AF),
- universal constrained reachability (AU),

- universal recurrence (AGF).

However, as opposed to ground tree rewriting systems, certain concepts are not transferable. From [Col02] it follows that when restricting the transition graphs of ground tree rewriting systems to the tree language $T(\mathcal{A})$ of a *deterministic top down tree automaton* $\mathcal{A}$, a formalism which is not strong enough to capture the full class of regular tree languages, the reachability problem remains decidable (in contrast to the undecidability of constrained reachability for a regular set of trees, cf. Chapter 1 – Preliminaries).

For the undecidability proof for partial subtree rewriting systems, refer to Chapter 4 – Undecidability Results.

Chapter 3

Subtree and Flat Prefix Rewriting Systems

In the previous chapter the impact of ground tree rewriting systems over an encoding of unranked trees was studied and transferred to the corresponding rewriting formalism over unranked trees, the partial subtree rewriting systems, resulting in the same class of transition graphs.

In order to obtain a new class of transition graphs, the approach of subtree and flat prefix rewriting systems does not employ any kind of encoding, but rather operates directly over unranked trees. However, since the aim is to transfer results for ground tree rewriting systems, this new formalism is reminiscent of ground tree rewriting as it also employs subtree rewrite rules. Subtree rewriting by itself does not suffice though, since the resulting trees, starting from a fixed initial tree with application of finitely many rewrite rules with constant trees, have a bounded branching factor and thus could be traced back to ground tree rewriting over ranked trees. Therefore, prefix rewrite rules are introduced to enable an unbounded number of successors. The application of these rules is restricted to the flat front of a tree, i.e. the object to be rewritten can be conceived as a word over the unranked alphabet. With this restriction it is ensured that the nodes that are replaced do not have any successors, and thus, there are no issues arising about successor placement in the new tree.

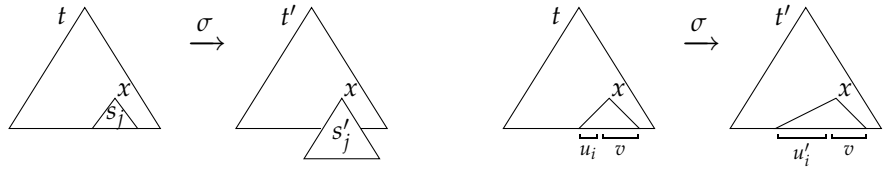
Analogous to regular ground tree rewriting systems, regular subtree and flat prefix rewriting systems are introduced that incorporate regular sets of trees respectively words in the rewrite rules. After defining the rewriting systems formally, a transition graph of a subtree and flat prefix rewriting system which is not in the class GTRG of transition graphs of ground tree rewriting systems is introduced in Section 3.1. Thus subtree and flat prefix rewriting systems define a new class of infinite graphs. In Section 3.2 it is shown that the reachability problem for regular subtree and flat prefix rewriting systems is decidable, but as mentioned in Chapter 1 – Preliminaries, an undecidability result for subtree and flat prefix rewriting systems is also given in Chapter 4 – Undecidability Results.

Definition 3.1. A *subtree and flat prefix rewriting system* over unranked trees in T_Σ is of the form $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$, with a finite unranked alphabet Σ, a finite transition alphabet Γ, an initial tree t_{in}, and a finite set R of rules of two types:

1. subtree substitution (cf. Figure 3.1(a))
   with rules of the form $r_j : s_j \overset{\sigma}{\hookrightarrow} s'_j$ for $j \in J$, $s_j, s'_j \in T_\Sigma$, $\sigma \in \Gamma$, and

2. flat prefix substitution at the flat front of the tree (cf. Figure 3.1(b))
   with rules of the form $r_i : u_i \overset{\sigma}{\hookrightarrow} u'_i$ for $i \in I$, $u_i, u'_i \in \Sigma^+$, $\sigma \in \Gamma$,

Figure 3.1 Rewriting rules of $\mathcal{R} \in \mathfrak{R}^{sfp}$ of Definition 3.1.



(a) Subtree substitution.

(b) Flat prefix substitution.

where the set $I \subseteq \{1, \ldots, |R|\}$ denotes the indices for flat prefix rewrite rules, and set $J \subseteq \{1, \ldots, |R|\}$ denotes the indices for subtree rewrite rules, with $I \cap J = \emptyset$ and $I \cup J = \{1, \ldots, |R|\}$. The set of all subtree and flat prefix rewriting systems is denoted by $\mathfrak{R}^{sfp}$, and the class of transition graphs of subtree and flat prefix rewriting systems is denoted by SFPRG.

t' is derived from t ($t \rightarrow^{\sigma}_{\mathcal{R}} t'$) by applying a subtree rewrite rule $r_j : s_j \overset{\sigma}{\hookrightarrow} s'_j$, if there are a node $x \in dom_t$ with $t_{\downarrow x} = s_j$ and a substitution $[x|s'_j]$ such that $t[x|s'_j] = t'$ (cf. Figure 3.1(a)).

t' is derived from t, by applying a flat prefix rewrite rule $r_i : u_i \overset{\sigma}{\hookrightarrow} u'_i$, if there are a node $x \in dom_t$ with $ht(t_{\downarrow x}) = 1$ and $flatfront(t_{\downarrow x}) = u_i v$, a tree $s \in T_\Sigma$ with $ht(s) = 1$ and $s(\varepsilon) = t(x)$, $flatfront(s) = u'_i v$, and a substitution $[x|s]$ such that $t[x|s] = t'$ for some $v \in \Sigma^*$ (cf. Figure 3.1(b)).

Analogous to the definition of subtree and flat prefix rewriting systems, *regular* subtree and flat prefix rewriting systems are defined with regular sets of trees respectively words in the rewrite rules (similar to the relation between pushdown graphs and prefix recognizable graphs, respectively ground tree rewriting systems and regular ground tree rewriting systems). Naturally, subtree and flat prefix rewriting systems are regular subtree and flat prefix rewriting systems with singleton sets of trees respectively words in the rewrite rules. Thus, the class SFPRG of transition graphs of subtree and flat prefix rewriting systems is included in the class of transition graphs of regular subtree and flat prefix rewriting systems.

Definition 3.2. A *regular subtree and flat prefix rewriting system* over unranked trees in T_Σ is of the form $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$ with Σ, Γ, t_{in} as for subtree and flat prefix rewriting systems, and R is a finite set of rules of two types:

1. subtree substitution
   with rules of the form $r_j : T_j \overset{\sigma}{\hookrightarrow} T'_j$ for $j \in J$, regular sets $T_j, T'_j \in T_\Sigma$, $\sigma \in \Gamma$, and

2. flat prefix substitution at the flat front of the tree
   with rules of the form $r_i : L_i \overset{\sigma}{\hookrightarrow} L'_i$ for $i \in I$, regular sets $L_i, L'_i \subseteq \Sigma^+$, $\sigma \in \Gamma$.

Sets I and J denote the indices for flat prefix respectively subtree rewrite rules analogously to the case of subtree and flat prefix rewriting systems. The set of all regular subtree and flat prefix rewriting systems is denoted by $\mathfrak{R}^{rsfp}$, and the class of transition graphs of regular subtree and flat prefix rewriting systems is denoted by RSFPRG.

t' is derived from t ($t \rightarrow^{\sigma}_{\mathcal{R}} t'$), by applying a subtree rewrite rule $r_j : T_j \overset{\sigma}{\hookrightarrow} T'_j$, if there are a node $x \in dom_t$ with $t_{\downarrow x} = s_j \in T_j$ and a substitution $[x|s'_j]$ for some $s'_j \in T'_j$ such that $t[x|s'_j] = t'$.

t' is derived from t, by applying a flat prefix rewrite rule $r_i : L_i \overset{\sigma}{\hookrightarrow} L'_i$, if there are a node $x \in dom_t$ with $ht(t_{\downarrow x}) = 1$ and $flatfront(t_{\downarrow x}) = u_i v$ for some $u_i \in L_i$, a tree $s \in T_\Sigma$ with $ht(s) = 1$, $s(\varepsilon) = t(x)$, $flatfront(s) = u'_i v$ for some $u'_i \in L'_i$, and a substitution $[x|s]$ such that $t[x|s] = t'$ for some $v \in \Sigma^*$.

The number of successors of a node in a ranked tree is bounded. Therefore, any encoding of an unranked tree into a ranked one naturally maps siblings to different levels in the encoded tree. Due to this alteration of hierarchy levels, it is not possible to simulate flat prefix substitution on an unranked tree at a node's successor sequence on the corresponding encoded ranked tree. Thus, the analysis of properties of subtree and flat prefix rewriting systems cannot take place on encodings, but needs to be shown directly on unranked trees.

3.1 Relation of Transition Graph Classes PSRG and SFPRG

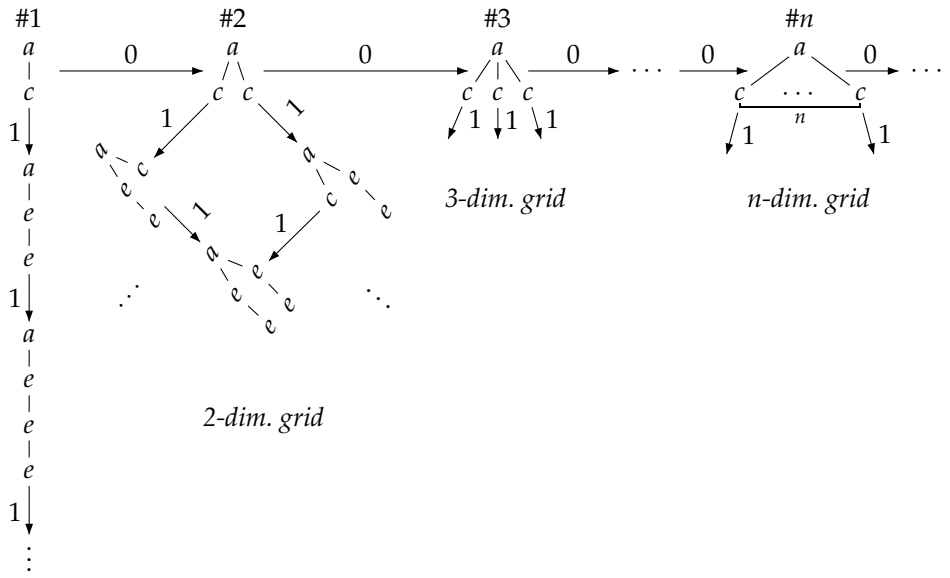
In order to determine a classification of the transition graphs of subtree and flat prefix rewriting systems with respect to expressiveness and decidability results, the class SFPRG is compared to the class PSRG of transition graphs of partial subtree rewriting systems. Since the classes PSRG and GTRG coincide, the more intuitive approach via an example and a comparison to ground tree rewriting systems is taken.

Consider the subtree and flat prefix rewriting system $\mathcal{R}_0 = (\Sigma, \Gamma, R, t_{in})$ with

- $\Sigma = \{a, c, e\}$,
- $\Gamma = \{0, 1\}$,
- $R = \{r_1 : c \overset{1}{\hookrightarrow} \begin{smallmatrix} e \\ | \\ e \end{smallmatrix}, r_2 : e \overset{1}{\hookrightarrow} \begin{smallmatrix} e \\ | \\ e \end{smallmatrix}, r_3 : c \overset{0}{\hookrightarrow} cc\}$, $\quad I = \{3\}, J = \{1, 2\}$, and
- $t_{in} = \begin{smallmatrix} a \\ | \\ c \end{smallmatrix}$.

The transition graph G_0 of $\mathcal{R}_0$ is depicted in Figure 3.2. The crucial point is that the flat prefix 0-transition r_3 can only be applied if the tree t at the corresponding vertex in G_0 has a subtree $t_{\downarrow x}$ of height 1 with $flatfront(t_{\downarrow x}) = cw$ for

Figure 3.2 Transition graph G_0 of subtree and flat prefix rewriting system R_0.



$w \in \Sigma^*$. This is only the case for the trees at the vertices of the horizontal line on top in Figure 3.2 (which will be referred to by enumeration from left to right, starting with vertex #1 at t_{in}). At the same time, n different 1-transitions are applicable at vertex #n. All of these lead to an n-dimensional grid in which no flat prefix 0-transitions are possible. This means for vertex #n that executing any one of the n 1-transitions blocks the execution of a 0-transition. In an equivalent transition graph of a ground tree rewriting system, this would correspond to rewriting a subtree s satisfying a left hand side of a 0-rule in the ground tree rewriting system.

Lemma 3.3. *The transition graph G_0 of the subtree and flat prefix rewriting system $\mathcal{R}_0$ cannot be generated by a ground tree rewriting system.*

PROOF. Towards a contradiction, assume that for a ground tree rewriting system $\mathcal{S} = (\Sigma, \Gamma, S, t_{in})$ with ranked alphabet Σ, $\Gamma = \{0, 1\}$, a finite set S of rules (say $|S| = k$), and a ranked tree t_{in} over Σ, it holds that $G_{\mathcal{S}} = G_0$. Furthermore, let $h := max\{ht(s) \mid s \text{ is the left hand side of a rule in } S\}$.

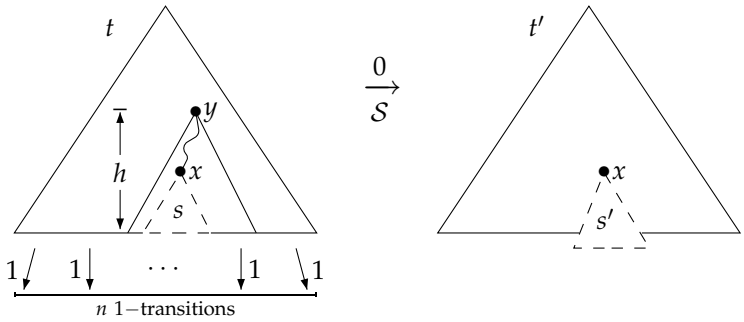
Consider the tree t in $G_{\mathcal{S}}$ at vertex #n. Let $x \in dom_t$, $t_{\downarrow x} = s$, and a rule $s_\ell : s \overset{0}{\hookrightarrow} s'$ such that

$$t = t[x|s] \underset{\mathcal{S}}{\overset{0}{\rightarrow}} t[x|s'] = t',$$

i.e. t' is the tree at vertex #$n+1$, and rule s_ℓ realizes the 0-transition from vertex #n to #$n+1$ in $G_{\mathcal{S}}$ (cf. Figure 3.3).

In order to get a transition graph isomorphic to G_0, there must be n 1-transitions at vertex #n, all of which lead to a tree preventing the execution of rule s_ℓ.

Figure 3.3 Trees t at vertex $\#n$ and t' at vertex $\#n+1$ of $G_{\mathcal{S}}$.



Therefore all 1-transitions have to rewrite $t_{\downarrow x} = s$.

Consider node $y \in dom_t$ with $y \preceq x$, and $ht(t_{\downarrow y}) = h$. Node y is chosen such that $t_{\downarrow y}$ contains $t_{\downarrow x}$ and is of the same height as the highest tree of a left hand side of the rewrite rules in S. Thus, the n rewrite rules (with label 1) that alter $t_{\downarrow x}$ have to be applied at nodes in $dom_{t_{\downarrow y}}$. More precisely, only rewrite rules applied at nodes in $dom_{t_{\downarrow x}} \subseteq dom_{t_{\downarrow y}}$ and at nodes on the path from y to x alter $t_{\downarrow x} = s$. Let the number of these nodes be m. With k transitions in $\mathcal{S}$, this makes at most $m \cdot k$ replacements which alter $t_{\downarrow x} = s$. With a selection of $n > m \cdot k$ this is a contradiction. ■

Thus, the subtree and flat prefix rewriting system $\mathcal{R}_0$ over unranked trees has a transition graph G_0 which cannot be generated by a ground tree rewriting system over ranked trees. On the other hand, every ground tree rewriting system can always be conceived as a subtree and flat prefix rewriting system with subtree rewrite rules only. With the same initial tree and the same subtree rewrite rules, omitting the ranks of the symbols does not provide more substitution possibilities. Since the classes of transition graphs of ground tree rewriting systems GTRG and of partial subtree rewriting systems PSRG are equivalent, one obtains the following.

Proposition 3.4. *The class PSRG of transition graphs of partial subtree rewriting systems is strictly included in the class SFPRG of transition graphs of subtree and flat prefix rewriting systems.*

Since subtree and flat prefix rewriting systems are special cases of regular subtree and flat prefix rewriting systems (namely with the restriction of singleton sets in the rewrite rules), the class SFPRG is also included in the class RSFPRG of transition graphs of regular subtree and flat prefix rewriting systems.

Thus, since partial subtree rewriting systems can be conceived as a special case of subtree and flat prefix rewriting systems, the undecidability results for partial subtree rewriting systems carry over to subtree and flat prefix rewriting

systems as well as to regular subtree and flat prefix rewriting systems. These include the following reachability problems:

- constrained reachability (EU),
- universal reachability (AF),
- universal constrained reachability (AU),
- universal recurrence (AGF).

Additionally, since this is the case for ground tree rewriting systems, the monadic second order logic of subtree and flat prefix rewriting systems (and thus of regular subtree and flat prefix rewriting systems) is undecidable. This is also obtained directly from the transition graph G_0 of $\mathcal{R}_0$, since G_0 includes the two-dimensional grid, whose monadic second order logic is undecidable (as proven e.g. in [See72]).

3.2 Reachability with Saturation

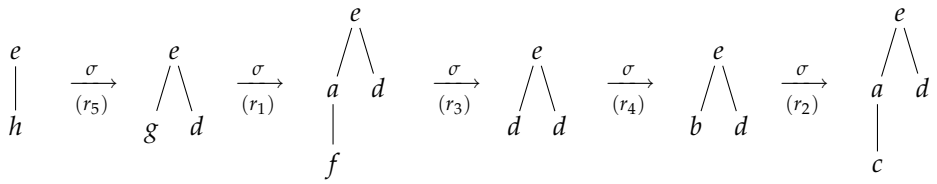
It will be shown that for regular subtree and flat prefix rewriting systems, the reachability problem (EF) over the transition graph is decidable. Since subtree and flat prefix rewriting systems are a special case of regular subtree and flat prefix rewriting systems, the decidability of the reachability problem for subtree and flat prefix rewriting systems follows immediately from the decidability result for the regular subtree and flat prefix rewriting systems.

In order to show that the reachability problem for a regular subtree and flat prefix rewriting system $\mathcal{R} \in \mathfrak{R}^{rsfp}$, a vertex t_{in} and a regular set T of vertices in $G_{\mathcal{R}}$ is decidable, a proof from [Löd03] is adapted in a straightforward way: with the set $\mathrm{pre}^*_{\mathcal{R}}(T) = \{t \in T_\Sigma \mid t \rightarrow^*_{\mathcal{R}} t' \text{ for } t' \in T\}$, it is checked whether t_{in} belongs to $\mathrm{pre}^*_{\mathcal{R}}(T)$. The algorithm given in the proof, which successively constructs an ε-N$\uparrow$TA $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ with $T(\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}) = \mathrm{pre}^*_{\mathcal{R}}(T)$, is similar to the well known saturation algorithm which e.g. solves the reachability problem for semi-monadic linear rewriting systems over ranked trees (cf. [CDGV94]).

The decisive point of the proof is that the rewrite rules of a regular subtree and flat prefix rewriting system $\mathcal{R}$ can be simulated by adding transitions to an N$\uparrow$TA that recognizes the union of the target set T and all trees that correspond to a left hand side of the rewrite rules. Due to the different natures of the two types of rules of regular subtree and flat prefix rewriting systems, the "classical" saturation algorithm has to be refined to distinguish the rules.

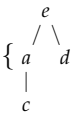
For a flat prefix rewrite rule, the saturation is realized by adding ε-transitions on the level of word automata that recognize the sequence of labels of the successors of a node. To ensure that the subtree at the according node is of height 1, the transition relation consists of two NFAs for each pair (a, q) of $a \in \Sigma$ and

Figure 3.4 A possible path from t to T in $G_{\mathcal{R}}$ of Example 3.5.



$q \in Q$. One of these operates on the flat front only, and this is the one which is being saturated.

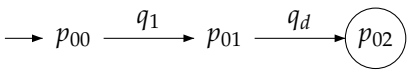
For subtree rewrite rules, the saturation is realized by adding ε-transitions on the level of tree automata. Thereby, one case requires further treatment: if there is a tree of height 0 on the left hand side of a subtree rewrite rule, it is necessary to add supplementary transitions to the word automata which are to be saturated in order to obtain the complete set $\text{pre}^*_{\mathcal{R}}(T)$. This idea will first be illustrated by a small example before the formal notation and proof are given.

Example 3.5. For $\Sigma = \{a, b, c, d, e, f, g, h\}$, consider the target set $T = \{$  $\}$,

the tree $t = e(h)$, and the regular subtree and flat prefix rewriting system $\mathcal{R} = (\Sigma, \{\sigma\}, R, e(d))$ with $R = \{r_1 : \{g\} \overset{\sigma}{\hookrightarrow} \{a(f)\}, r_2 : \{b\} \overset{\sigma}{\hookrightarrow} \{a(c)\}, r_3 : \{a(f)\} \overset{\sigma}{\hookrightarrow} \{d\}, r_4 : \{d\} \overset{\sigma}{\hookrightarrow} \{b, gd\}, r_5 : \{h\} \overset{\sigma}{\hookrightarrow} \{gd\}\}$ with index sets $I = \{4, 5\}$ and $J = \{1, 2, 3\}$.

A possible path from t to T in $G_{\mathcal{R}}$ is depicted in Figure 3.4. It will be shown that an ε-N$\uparrow$TA $\mathcal{B}_k$ can be constructed successively by saturating an ε-N$\uparrow$TA $\mathcal{B}_0$ with $T(\mathcal{B}_0) = T$, such that $t \in T(\mathcal{B}_k)$.

Let the regular target set T be given by the N$\uparrow$TA $\mathcal{A} = (\{q_c, q_d, q_1, q_2\}, \Sigma, \Delta_{\mathcal{A}}, \{q_2\})$ with $\Delta_{\mathcal{A}} = \{(L(\mathcal{C}_{cq_c}), c, q_c), (L(\mathcal{C}_{dq_d}), d, q_d), (L(\mathcal{C}_{aq_1}), a, q_1), (L(\mathcal{C}_{eq_2}), e, q_2)\}$. NFAs $\mathcal{C}_{cq_c}$ and $\mathcal{C}_{dq_d}$ simply recognize the empty word, i.e. assign state q_c to leaf c respectively q_d to leaf d. Furthermore, let $L(\mathcal{C}_{aq_1}) = \{q_c\}$. The only NFA of $\mathcal{A}$ that will be used explicitly in this example is $\mathcal{C}_{eq_2}$ with $L(\mathcal{C}_{eq_2}) = \{q_1 q_d\}$:

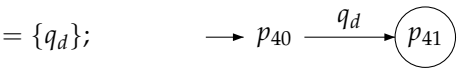


For each subtree rewrite rule r_j with $j \in J$ of the regular subtree and flat prefix rewriting system $\mathcal{R}$, N$\uparrow$TAs $\mathcal{A}_j$ and $\mathcal{A}'_j$ are constructed that recognize the left respectively the right hand side of rule r_j. The N$\uparrow$TAs $\mathcal{A}_2$ and $\mathcal{A}'_2$ are given exemplary:

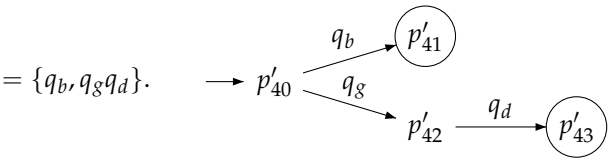
- N$\uparrow$TA $\mathcal{A}_2 = (\{q_b\}, \Sigma, \Delta_2, \{q_b\})$ with $\Delta_2 = \{(\{\varepsilon\}, b, q_b)\}$, i.e. $T(\mathcal{A}_2) = \{b\}$;
- N$\uparrow$TA $\mathcal{A}'_2 = (\{q_c, q'_{21}\}, \Sigma, \Delta'_2, \{q'_{21}\})$
  with $\Delta'_2 = \{(\{\varepsilon\}, c, q_c), (\{q_c\}, a, q'_{21})\}$, i.e. $T(\mathcal{A}'_2) = \{a(c)\}$.

The N↑TAs $\mathcal{A}_1, \mathcal{A}'_1, \mathcal{A}_3, \mathcal{A}'_3$ are constructed analogously. Note that each leaf $m \in \Sigma$ is assigned the state q_m by the tree automata. To facilitate the notation, let $Q_\Sigma := \{q_m \mid m \in \Sigma\}$. Furthermore, for $u \in \Sigma^*$ define $\underline{q}_u := q_{u(1)} \cdots q_{u(|u|)}$, thus $\underline{q}_u \in Q_\Sigma^*$, and $Q_{L_i} := \{\underline{q}_u \mid u \in L_i\}$. In order to allow interaction between the tree and word automata, i.e. the application of both subtree and flat prefix rewrite rules, the NFAs $\mathcal{A}_i$ respectively $\mathcal{A}'_i$ for flat prefix rewrite rule $r_i : L_i \overset{\sigma}{\hookrightarrow} L'_i$ with $i \in I$ are constructed such that they recognize not the regular languages L_i respectively L'_i, but the regular languages Q_{L_i} respectively $Q_{L'_i}$. The NFAs $\mathcal{A}_4$ and $\mathcal{A}'_4$ are given exemplary (the construction of NFAs $\mathcal{A}_5, \mathcal{A}'_5$ proceeds analogously):

- NFA $\mathcal{A}_4$ with $L(\mathcal{A}_4) = \{q_d\}$;

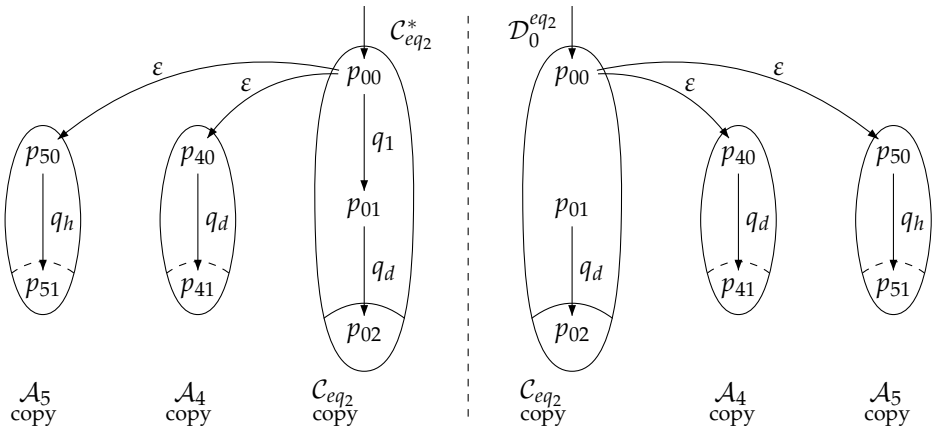


- NFA $\mathcal{A}'_4$ with $L(\mathcal{A}'_4) = \{q_b, q_g q_d\}$.



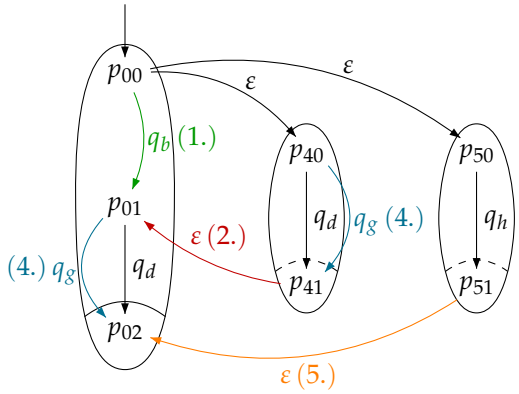
With these automata, the ε-N↑TA $\mathcal{B}_0 = (Q_\mathcal{B}, \Sigma, \Delta_0^\mathcal{B}, F_\mathcal{A})$ is constructed that recognizes the target set $T = T(\mathcal{A})$, and additionally has paths that correspond to each tree respectively each prefix of the left sides of the rewrite rules. Thus, $Q_\mathcal{B}$ is the union of the state set of $\mathcal{A}$ and all state sets of $\mathcal{A}_j$ for $j \in J$, while the set $F_\mathcal{A}$ of final states remains unchanged. As for $\Delta_0^\mathcal{B}$, the transitions of the $\mathcal{A}_j$ are inherited, but the transitions of $\mathcal{A}$ have to be altered in order to realize the recognition of the left hand sides of the flat prefix rewrite rules and $\Delta_0^\mathcal{B}$ is defined as $\Delta_0^\mathcal{B} := \bigcup_{j\in J} \Delta_j \cup \{(L(\mathcal{C}^*_{aq}) \cup L(\mathcal{D}_0^{aq}), a, q) \mid (L(\mathcal{C}_{aq}), a, q) \in \Delta_\mathcal{A}\}$.

For each NFA $\mathcal{C}_{aq}$, two ε-NFAs $\mathcal{C}^*_{aq}$ and $\mathcal{D}_0^{aq}$ are constructed. Both ε-NFAs consist of a copy of $\mathcal{C}_{aq}$ as well as a copy of the set of all $\mathcal{A}_i$ for $i \in I$, extended by an ε-transition from the initial state of the $\mathcal{C}_{aq}$ copy to each initial state of the $\mathcal{A}_i$ copies, while the set of final states remains unchanged. The NFAs $\mathcal{D}_0^{aq}$ are restricted to Q_Σ and can therefore only be applied at the flat front of a tree. As an example, consider $\mathcal{C}^*_{eq_2}$ and $\mathcal{D}_0^{eq_2}$:



Thus, the newly constructed ε-NFAs additionally have paths corresponding to the prefixes of all $L(\mathcal{A}_i)$ (which however do *not* lead to final states). It is not difficult to see that $\mathcal{C}^*_{eq_2}$ only differs from $\mathcal{D}_0^{eq_2}$ in the additional edge (p_{00}, q_1, p_{01}) with $q_1 \notin Q_\Sigma$. Consequently, for all constructed ε-NFAs, it holds that $L(\mathcal{D}_0^{aq}) \subseteq L(\mathcal{C}^*_{aq})$ and $L(\mathcal{C}^*_{aq}) = L(\mathcal{C}_{aq})$. To ensure that flat prefix rewrite rules are only applied at the flat front of a tree, the ε-NFAs $\mathcal{D}_0^{aq}$ are the ones that are saturated, while the ε-NFAs $\mathcal{C}^*_{aq}$ remain unchanged.

Starting with the ε-N$\uparrow$TA $\mathcal{B}_0$ that recognizes the target set $T(\mathcal{B}_0) = T$, the saturation has to traverse the rewrite rules backwards and in reverse order. As the complete saturation of $\mathcal{B}_0$ would exceed the scope of this example, only some selected steps will be illustrated such that the resulting automaton recognizes all trees of the derivation depicted in Figure 3.4. Moreover, of all ε-NFAs only $\mathcal{D}_k^{eq_2}$ will be considered. While the addition of the different transitions is described below, the resulting NFA $\mathcal{D}_5^{eq_2}$ is the following.



1. The first derivation to traverse backwards is $\overset{e}{\underset{b}{/}\ \underset{d}{\backslash}} \xrightarrow[\mathcal{R}]{\sigma} \overset{e}{\underset{a}{/}\ \underset{d}{\backslash}}$ with subtree $\overset{a}{|}\ c$ rewrite rule r_2. Since $e(a(c), d) \in T = T(\mathcal{B}_0)$, a transition to be added needs to realize that subtree b is also accepted in the place of subtree $a(c)$. It holds that $\mathcal{B}_0 : b \to q_b$ and $\mathcal{B}_0 : a(c) \to^* q_1$ (recall that the $\mathcal{A}$-transitions were inherited). Therefore, ε-transition (q_b, q_1) is added.

   Note that now a tree is accepted that has a new flat front bd. To enable flat prefix rewriting at this flat front, it needs to be accepted by $\mathcal{D}_0^{eq_2}$, as opposed to only being accepted by $q_b \to^\varepsilon q_1$ and $\mathcal{C}^*_{eq_2} : p_{00} \to^{q_1} p_{01} \to^{q_d} p_{02}$ (since $q_1 \notin Q_\Sigma$). Thus, the supplementary transition (p_{00}, q_b, p_{01}) is added to $\mathcal{D}_0^{eq_2}$, yielding $\mathcal{B}_1$ and $\mathcal{D}_1^{eq_2}$ with $q_b q_d \in L(\mathcal{D}_1^{eq_2})$.

2. In the path in Figure 3.4, the next derivation to be traversed backwards is $\overset{e}{\underset{d}{/}\ \underset{d}{\backslash}} \xrightarrow[\mathcal{R}]{\sigma} \overset{e}{\underset{b}{/}\ \underset{d}{\backslash}}$ with flat prefix rewrite rule r_4. Thus the acceptance of the prefix $d \in L_4$ instead of the prefix $b \in L'_4$ needs to be enabled. Therefore, consider the accepting run $p_{00} \to^{q_b} p_{01} \to^{q_d} p_{02}$ on the flat front bd of $\mathcal{D}_1^{eq_2}$.

Accepting dd is realized by adding the ε-transition (p_{41}, p_{01}) that starts from the copy of the final state of $\mathcal{A}_4$, p_{41}, to the state that was reached via the prefix b, p_{01}. This way, the automata $\mathcal{B}_2$ and $\mathcal{D}_2^{eq_2}$ are obtained with $q_d q_d \in L(\mathcal{D}_2^{eq_2})$ by the run $p_{00} \to^{\varepsilon} p_{40} \to^{q_d} p_{41} \to^{\varepsilon} p_{01} \to^{q_d} p_{02}$.

3. The third derivation is $\begin{smallmatrix} & e & \\ a & & d \\ f & & \end{smallmatrix} \xrightarrow[\mathcal{R}]{\sigma} \begin{smallmatrix} & e & \\ d & & d \end{smallmatrix}$ with subtree rewrite rule r_3. This time it needs to be realized that the subtree $a(f)$ is accepted in places where subtree d already is accepted. With $\mathcal{B}_2 : a(f) \to^* q_{31}$ (transitions inherited from $\mathcal{A}_3$, the state names are chosen analogous to the construction of $\mathcal{A}_2$ shown above) and $\mathcal{B}_2 : d \to q_d$, the ε-transition (q_{31}, q_d) is added to $\mathcal{B}_2$. Since this rewriting step did not yield a new flat front, no supplementary transitions need to be added to any of the NFAs (for consistency though, the indices are increased regardlessly, i.e. $\mathcal{D}_3^{eq_2} = \mathcal{D}_2^{eq_2}$).

4. The next derivation is again a subtree rewriting step: $\begin{smallmatrix} & e & \\ g & & d \end{smallmatrix} \xrightarrow[\mathcal{R}]{\sigma} \begin{smallmatrix} & e & \\ a & & d \\ f & & \end{smallmatrix}$ with subtree rewrite rule r_1. Analogous to (1.), the ε-transition (q_g, q_{31}) is added to $\mathcal{B}_3$. However, in (3.), the ε-transition (q_{31}, q_d) was added to the automaton, and thus there also is a derivation $\mathcal{B}_3 : a(f) \to^* q_d$. Thus, also the transition (q_g, q_d) needs to be added to $\mathcal{B}_3$, and analogous to (1.), the automaton needs to ensure that at any flat front, q_g can be read instead of q_d. Therefore, the supplementary transitions (p_{40}, q_g, p_{41}) and (p_{01}, q_g, p_{02}) need to be inserted into $\mathcal{D}_3^{eq_2}$ additionally. Therefore it holds that $q_g q_d \in L(\mathcal{D}_4^{eq_2})$.

5. The last derivation in Figure 3.4 is the flat prefix rewriting step $\begin{smallmatrix} & e & \\ g & & d \end{smallmatrix} \xrightarrow[\mathcal{R}]{\sigma} \begin{smallmatrix} e \\ | \\ h \end{smallmatrix}$ with $e(h) = t$ and flat prefix rewrite rule r_5. Analogous to (2.), the ε-transition (p_{51}, p_{02}) is added to $\mathcal{D}_4^{eq_2}$, thus yielding $\mathcal{B}_5$ and $\mathcal{D}_5^{eq_2}$ with $q_h \in L(\mathcal{D}_5^{eq_2})$.

With all added transitions, $t = e(h)$ is accepted by $\mathcal{B}_5$ with the run

$$\mathcal{B}_5 : e(h) \to e(q_h) \to q_2 \in F_{\mathcal{A}}.$$

Note that $\mathcal{B}_5$ does not recognize the complete set $\text{pre}^*(T)$ since several other saturation steps are possible. If the full saturation is achieved in k steps, $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}}$ is set to $\mathcal{B}_k$, and it holds that $T(\mathcal{A}_{\text{pre}^*_{\mathcal{R}}}) = \text{pre}^*(T)$, which will be proven next. ◀

For the formal correctness, several constructive factors have to be respected:

- The tree automata have to start all of their runs by assigning to each leaf $x \in dom_t$ the state $q_{t(x)} \in Q_\Sigma$.
- Apart from Q_Σ, the state sets of the tree automata need to be pairwise disjoint.

- Since the ε-NFAs $\mathcal{C}^*_{aq}$ are not altered throughout the saturation, it suffices that they share one copy of the NFAs $\mathcal{A}_i$.
- Apart from the shared $\mathcal{A}_i$ copy of the ε-NFAs $\mathcal{C}^*_{aq}$, all word automata also need to have pairwise disjoint state sets.

All of these aspects are realized in the following constructions and in the initialization of ε-N↑TA $\mathcal{B}_0$ (cf. Algorithm 3.1). To clarify the employed notation, the indices $a \in \Sigma$, $q \in Q_\Sigma \dot\cup Q_\mathcal{A}$, $m \in \{0\} \dot\cup I$, $n \in \mathbb{N}$ of state $p^\mathcal{D}_{aqmn}$ denote the following:

- if $m = 0$, then $p^\mathcal{D}_{aq0n}$ is located in the $\mathcal{C}_{aq}$ copy of ε-NFA $\mathcal{D}^{aq}_k$ (for all k) (analogously, state $p^\mathcal{C}_{aq0n}$ is located in the $\mathcal{C}_{aq}$ copy of $\mathcal{C}^*_{aq}$), and
- if $m \in I$, then $p^\mathcal{D}_{aqmn}$ is located in the $\mathcal{A}_m$ copy of ε-NFA $\mathcal{D}^{aq}_k$ (for all k).

As the ε-NFAs $\mathcal{C}^*_{aq}$ share one copy of the NFAs $\mathcal{A}_i$, these state sets do not need indices a and q; therefore their states are denoted p_{in} for $i \in I$, $n \in \mathbb{N}$.

To facilitate the notation, a state $p^\mathcal{D}$ refers to a state $p^\mathcal{D}_{aqmn}$ in $P^\mathcal{D}_{aq}$ for some $m, n \in \mathbb{N}$ when possible and when $a \in \Sigma$ and $q \in Q_\Sigma \dot\cup Q_\mathcal{A}$ are clear from the context. At the same time, state $p^\mathcal{C}$ refers to the corresponding state $p^\mathcal{C}_{aqmn}$ respectively p_{mn} in

Algorithm 3.1 Construction of ε-N↑TA $\mathcal{B}_0$.

procedure INITIALIZE_$\mathcal{B}_0(\mathcal{A}, \{\mathcal{A}_i \mid i \in I\}, \{\mathcal{A}_j \mid j \in J\})$
▷ *Notation as in Algorithm 3.2.*
  $Q_\mathcal{B} := Q_\mathcal{A} \dot\cup Q_\Sigma \dot\cup \dot\bigcup_{j\in J} Q_j$
  for all $\mathcal{C}_{aq} = (P_{aq}, Q_\mathcal{A} \dot\cup Q_\Sigma, p_{aq0}, \Delta_{aq}, F_{aq})$ with $(L(\mathcal{C}_{aq}), a, q) \in \Delta_\mathcal{A}$ **do**
    construct ε-NFA $\mathcal{C}^*_{aq} := (P^\mathcal{C}_{aq}, Q_\mathcal{A} \dot\cup Q_\Sigma, p^\mathcal{C}_{aq00}, \Delta^\mathcal{C}_{aq}, F^\mathcal{C}_{aq})$ with
      $P^\mathcal{C}_{aq} := \{p^\mathcal{C}_{aq0n} \mid p_{aqn} \in P_{aq}\} \dot\cup \dot\bigcup_{i\in I} P_i$
      $\Delta^\mathcal{C}_{aq} := \{(p^\mathcal{C}_{aq0n}, \tilde q, p^\mathcal{C}_{aq0\lambda}) \mid (p_{aqn}, \tilde q, p_{aq\lambda}) \in \Delta_{aq}\} \dot\cup \dot\bigcup_{i\in I} \Delta_i$
        $\dot\cup \dot\bigcup_{i\in I} \{(p^\mathcal{C}_{aq00}, p_{i0}) \mid p_{i0} \in P_i \text{ (initial state of } \mathcal{A}_i)\}$
      $F^\mathcal{D}_{aq} := \{p^\mathcal{C}_{aq0n} \mid p_{aqn} \in F_{aq}\}$
    let $\Delta^\mathcal{C}_{aq}\big|_{Q_\Sigma} = \Delta^\mathcal{C}_{aq} \setminus \{(p^\mathcal{C}_{aq0n}, q_1, p^\mathcal{C}_{aq0\lambda}) \mid (p^\mathcal{C}_{aq0n}, q_1, p^\mathcal{C}_{aq0\lambda}) \in \Delta^\mathcal{C}_{aq} \wedge q_1 \in Q_\mathcal{A}\}$
    construct ε-NFA $\mathcal{D}^{aq}_0 := (P^\mathcal{D}_{aq}, Q_\Sigma, p^\mathcal{D}_{aq00}, \Delta^{aq}_0, F^\mathcal{D}_{aq})$ with
      $P^\mathcal{D}_{aq} := \{p^\mathcal{D}_{aqmn} \mid p^\mathcal{C}_{aqmn} \in P^\mathcal{C}_{aq} \vee p_{mn} \in P^\mathcal{C}_{aq}\}$
      $\Delta^{aq}_0 := \{(p^\mathcal{D}_{aqmn}, q_b, p^\mathcal{D}_{aq\mu\lambda}) \mid (p^\mathcal{C}_{aqmn}, q_b, p^\mathcal{C}_{aq\mu\lambda}) \in \Delta^\mathcal{C}_{aq}\big|_{Q_\Sigma}$
        $\vee (p_{mn}, q_b, p_{\mu\lambda}) \in \Delta^\mathcal{C}_{aq}\big|_{Q_\Sigma}\}$
      $F^\mathcal{D}_{aq} := \{p^\mathcal{D}_{aqmn} \mid p^\mathcal{C}_{aqmn} \in F^\mathcal{C}_{aq}\}$
  end for
  $\Delta^\mathcal{B}_0 := \{(L(\mathcal{D}^{aq}_0) \cup L(\mathcal{C}^*_{aq}), a, q) \mid (L(\mathcal{C}_{aq}), a, q) \in \Delta_\mathcal{A}\} \dot\cup \Delta_\Sigma \dot\cup \dot\bigcup_{j\in J} \Delta_j$
  return ε-N↑TA $\mathcal{B}_0 := (Q_\mathcal{B}, \Sigma, \Delta^\mathcal{B}_0, F_\mathcal{A})$
end procedure

Algorithm 3.2 Algorithm to calculate $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ with $T(\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}) = \mathrm{pre}^*_{\mathcal{R}}(T)$.

Given:

- Rewriting system $\mathcal{R} \in \mathfrak{R}^{rsfp}$ with $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$
- N↑TA $\mathcal{A} = (Q_{\mathcal{A}} \dot\cup Q_\Sigma, \Sigma, \Delta_{\mathcal{A}} \dot\cup \Delta_\Sigma, F_{\mathcal{A}})$ with $T(\mathcal{A}) = T$
  $\Delta_{\mathcal{A}} \subseteq \{(L(\mathcal{C}_{aq}), a, q) \mid a \in \Sigma, q \in Q_{\mathcal{A}}\}$
  with NFAs $\mathcal{C}_{aq} = (P_{aq}, Q_{\mathcal{A}} \dot\cup Q_\Sigma, p_{aq0}, \Delta_{aq}, F_{aq})$
- for $j \in J$ (subtree rewrite rule), $r_j : T_j \overset{\sigma}{\hookrightarrow} T'_j$:
  N↑TA $\mathcal{A}_j = (Q_j \dot\cup Q_\Sigma, \Sigma, \Delta_j \dot\cup \Delta_\Sigma, F_j)$ with $T(\mathcal{A}_j) = T_j$
  N↑TA $\mathcal{A}'_j = (Q'_j \dot\cup Q_\Sigma, \Sigma, \Delta'_j \dot\cup \Delta_\Sigma, F'_j)$ with $T(\mathcal{A}'_j) = T'_j$
- for $i \in I$ (flat prefix rewrite rule), $r_i : L_i \overset{\sigma}{\hookrightarrow} L'_i$:
  NFA $\mathcal{A}_i = (P_i, Q_\Sigma, p_{i0}, \Delta_i, F_i)$ with $L(\mathcal{A}_i) = Q_{L_i}$
  NFA $\mathcal{A}'_i = (P'_i, Q_\Sigma, p'_{i0}, \Delta'_i, F'_i)$ with $L(\mathcal{A}'_i) = Q_{L'_i}$

procedure SATURATION($\mathcal{R}$, $\mathcal{A}$,$\{\mathcal{A}_i \mid i \in I\}$,$\{\mathcal{A}'_i \mid i \in I\}$,$\{\mathcal{A}_j \mid j \in J\}$,$\{\mathcal{A}'_j \mid j \in J\}$)
  INITIALIZE_$\mathcal{B}_0$($\mathcal{A}$, $\{\mathcal{A}_i \mid i \in I\}$, $\{\mathcal{A}_j \mid j \in J\}$)
  $k := 0$
  while $\Big[\big[\exists$ subtree rewrite rule $r_\ell : T_\ell \overset{\sigma}{\hookrightarrow} T'_\ell \in R$ with $\ell \in J$ and a tree
  $s' \in T(\mathcal{A}'_\ell)$ with $s' \xrightarrow[\mathcal{B}_k]{*} \hat q$ and $(F_\ell \times \{\hat q\}) \not\subseteq \Delta^{\mathcal{B}}_k\big]$ $(\star)$
  or $\big[\exists$ flat prefix rewrite rule $r_\ell : L_\ell \overset{\sigma}{\hookrightarrow} L'_\ell \in R$ with $\ell \in I$
  and words $u', v \in \Sigma^*$ with $\underline{q}_{u'} \in L(\mathcal{A}'_\ell)$,
  $\mathcal{D}^{aq}_k : p^{\mathcal{D}}_{aq00} \xrightarrow{q_{u'}} \hat p \xrightarrow{q_v} p \in F^{\mathcal{D}}_{aq}$,
  and $(\{p^{\mathcal{D}}_{aq\ell m} \mid p_{\ell m} \in F_\ell\} \times \{\hat p\}) \not\subseteq \Delta^{aq}_k\big]\Big]$ $(\star\star)$
  do
    if $(\star)$ **then**
      for all $\mathcal{D}^{aq}_k$ **do**
        $\Delta^{aq}_{k+1} := \Delta^{aq}_k \cup \{(p^{\mathcal{D}}_{aqmn}, q_b, p^{\mathcal{D}}_{aq\mu\lambda}) \mid q_b \in F_\ell \cap Q_\Sigma \wedge$
        $\big[(p^{\mathcal{C}}_{aqmn}, \hat q, p^{\mathcal{C}}_{aq\mu\lambda}) \in \Delta^{\mathcal{C}}_{aq} \vee (p_{mn}, \hat q, p_{\mu\lambda}) \in \Delta^{\mathcal{C}}_{aq}\big]\}$
        $\mathcal{D}^{aq}_{k+1} := (P^{\mathcal{D}}_{aq}, Q_\Sigma, p^{\mathcal{D}}_{aq00}, \Delta^{aq}_{k+1}, F^{\mathcal{D}}_{aq})$
      end for
      $\Delta^{\mathcal{B}}_{k+1} := \Delta^{\mathcal{B}}_k \cup (F_\ell \times \{\hat q\}) \cup \{(L(\mathcal{D}^{aq}_{k+1}) \cup L(\mathcal{C}^*_{aq}), a, q)\}$
      $\setminus \{(L(\mathcal{D}^{aq}_k) \cup L(\mathcal{C}^*_{aq}), a, q)\}$
    else if $(\star\star)$ **then**
      $\Delta^{aq}_{k+1} := \Delta^{aq}_k \cup (\{p^{\mathcal{D}}_{aq\ell m} \mid p_{\ell m} \in F_\ell\} \times \{\hat p\})$
      $\mathcal{D}^{aq}_{k+1} := (P^{\mathcal{D}}_{aq}, Q_\Sigma, p^{\mathcal{D}}_{aq00}, \Delta^{aq}_{k+1}, F^{\mathcal{D}}_{aq})$
      $\Delta^{\mathcal{B}}_{k+1} := \Delta^{\mathcal{B}}_k \cup \{(L(\mathcal{D}^{aq}_{k+1}) \cup L(\mathcal{C}^*_{aq}), a, q)\} \setminus \{(L(\mathcal{D}^{aq}_k) \cup L(\mathcal{C}^*_{aq}), a, q)\}$
    end if
    $\mathcal{B}_{k+1} := (Q_{\mathcal{B}}, \Sigma, \Delta^{\mathcal{B}}_{k+1}, F_{\mathcal{A}})$
    $k := k + 1$
  end while
  return $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}} := \mathcal{B}_k$
end procedure

$P^{\mathcal{C}}_{aq}$ for the same a, q, m, n. Note that by construction of $P^{\mathcal{C}}_{aq}$, there cannot be states $p^{\mathcal{C}}_{aqmn} \in P^{\mathcal{C}}_{aq}$ and $p_{mn} \in P^{\mathcal{C}}_{aq}$ for the same m at the same time.

In Algorithm 3.1, the ε-N↑TA $\mathcal{B}_0 = (Q_{\mathcal{B}}, \Sigma, \Delta^{\mathcal{B}}_0, F_{\mathcal{A}})$ is initialized. In Line 5, the restriction of Δ^*_{aq} to Q_Σ is defined, which is then taken as the transition relation of the ε-NFA $\mathcal{D}^{aq}_0$ (with respect to renaming for disjoint state sets). This restriction ensures that the state set of the $\mathcal{D}^{aq}_0$ is of the same structure as the corresponding state set of the $\mathcal{C}^*_{aq}$, and that the $\mathcal{D}^{aq}_0$ can only read words of the flat front of a tree, i.e. only words over Q_Σ.

Algorithm 3.2 distinguishes the rewrite rules analogous to Example 3.5. The three different types of transitions that are added in the construction of $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ are depicted in Figure 3.5.

For a subtree rewrite rule (case $(\star)$), Algorithm 3.2 checks in Line 5 whether there is a subtree $t_{\downarrow x}$ corresponding to a tree from the right hand side of a subtree rewrite rule that is reduced to a state $\hat{q} \in Q_{\mathcal{B}}$ in $\mathcal{B}_k$. If this is the case for a tree of say subtree rewrite rule r_ℓ (which can be verified by the corresponding N↑TA $\mathcal{A}'_\ell$), an ε-transition from each state $q \in F_\ell$ to $\hat{q}$ is added to $\Delta^{\mathcal{B}}_k$ (cf. Line 11 and Figure 3.5(a)).

If the left hand side of rule r_ℓ contains a tree of height 0 (which corresponds to the test $q_b \in F_\ell \cap Q_\Sigma$, cf. Line 8), then for each transition $(p^{\mathcal{C}}_1, \hat{q}, p^{\mathcal{C}}_2)$ in $\mathcal{C}^*_{aq}$, a corresponding $(p^{\mathcal{D}}_1, q_b, p^{\mathcal{D}}_2)$-transition is added to the $\mathcal{C}_{aq}$ copy in $\mathcal{D}^{aq}_k$ (cf. Figure 3.5(b)) to enable potential flat prefix rewriting before the tree of height 0 was rewritten, thus yielding $\mathcal{D}^{aq}_{k+1}$ and $\mathcal{B}_{k+1}$. Note that while adding this transition changes $L(\mathcal{D}^{aq}_k)$, it does not change the accepted tree language (cf. Lemma 3.6).

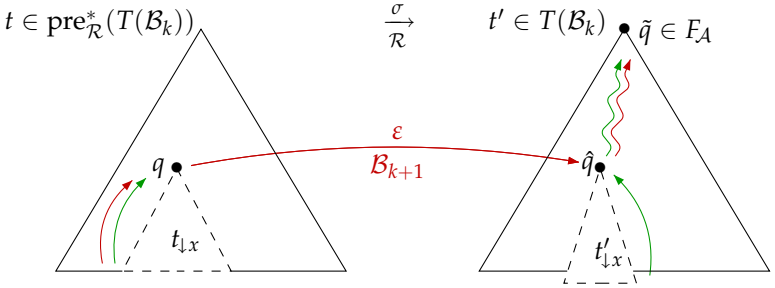
For a flat prefix rewrite rule (case $(\star\star)$), Algorithm 3.2 checks in Line 5 whether there is a word over Σ with a prefix from the right hand side of a flat prefix rewrite rule, and with the prefix being mapped to a state $\hat{p} \in P^{\mathcal{D}}_{aq}$ for an NFA $\mathcal{D}^{aq}_k$ of $\mathcal{B}_k$. If this is the case for a word of say flat prefix rewrite rule r_ℓ (which can be verified by the corresponding NFA $\mathcal{A}'_\ell$), an ε-transition from each state of F^{aq}_ℓ to $\hat{p}$ is added to Δ^{aq}_k, yielding Δ^{aq}_{k+1} and thus $\mathcal{B}_{k+1}$ (cf. Line 13 and Figure 3.5(c)).

With these techniques, $\mathcal{B}_{k+1}$ pretends to have read a tree respectively a prefix of the right hand side of either the subtree or the flat prefix rewrite rule r_ℓ, thus simulating the rule r_ℓ and additionally accepting all trees which can be rewritten by r_ℓ to a tree which is already accepted. By iterating this principle, the resulting automaton $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ will finally accept all trees which can be rewritten by the regular subtree and flat prefix rewriting system $\mathcal{R}$ to trees from T, thus all trees in $\mathrm{pre}^*_{\mathcal{R}}(T)$.

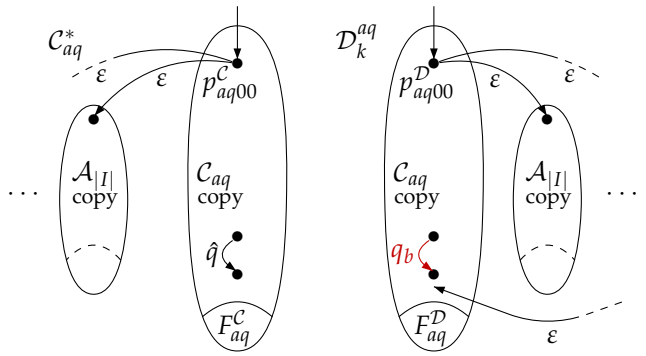
In order to show that the process described above yields a solution to the reachability problem, i.e. correctly calculates the set $\mathrm{pre}^*_{\mathcal{R}}(T)$, the above mentioned preparations as well as the following claims need to be shown:

1. Algorithm 3.2 terminates.
2. $T(\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}) = \mathrm{pre}^*_{\mathcal{R}}(T)$.

Figure 3.5 The idea of the reachability construction of Algorithm 3.2.

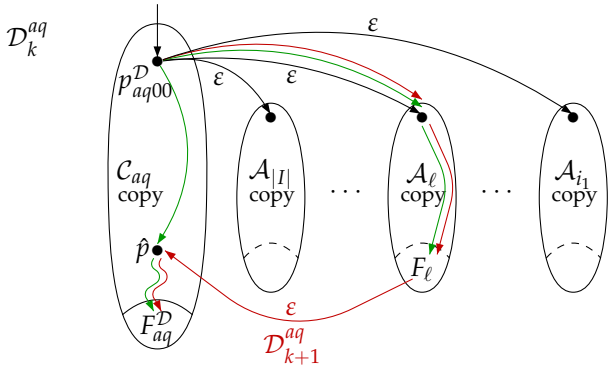


(a) Saturation for subtree rewrite rules on the level of tree automata.



With $b \in T_\ell, s' \in T'_\ell$ and $s' \xrightarrow[\mathcal{B}_k]{*} \hat{q}$: $(p^{\mathcal{C}}_{aqk\ell}, \hat{q}, p^{\mathcal{C}}_{aqmn}) \in \Delta^{\mathcal{C}}_{aq} \Rightarrow (p^{\mathcal{D}}_{aqk\ell}, q_b, p^{\mathcal{D}}_{aqmn}) \in \Delta^{aq}_{k+1}$

(b) If applicable, supplement for subtree rewrite rule $r_\ell : T_\ell \overset{\sigma}{\hookrightarrow} T'_\ell$ on the level of word automata.



$\underline{q}_{u'}\underline{q}_v \in L(\mathcal{D}^{aq}_k) : p^{\mathcal{D}}_{aq00} \xrightarrow{\underline{q}_{u'}} \hat{p} \xrightarrow{\underline{q}_v} p \in F^{\mathcal{D}}_{aq}$ with $\underline{q}_{u'} \in L(\mathcal{A}'_\ell)$
↑: runs of automaton $\mathcal{D}^{aq}_k$ ↑: new accepting $\mathcal{D}^{aq}_{k+1}$-run on $\underline{q}_u\underline{q}_v$

(c) Saturation for flat prefix rewrite rules on the level of word automata.

Preparations. To facilitate the notation for the following frequently used expressions, let $\Delta_\Sigma := \{(L(\mathcal{C}_{aq_a}), a, q_a) \mid a \in \Sigma\}$ with $\mathcal{C}_{aq_a} = (\{p_{aq_a0}\}, Q_\mathcal{A} \dot\cup Q_\Sigma, p_{aq_a0}, \emptyset, \{p_{aq_a0}\})$. The NFAs $\mathcal{C}_{aq_a}$ recognize the empty word, i.e. with Δ_Σ, every leaf with label a is assigned the state q_a. Given a regular subtree and flat prefix rewriting system $\mathcal{R} = (\Sigma, \Gamma, R, t_{in})$ and the regular set $T \subseteq T_\Sigma$, construct the following automata:

- Without loss of generality, assume that the regular target set T is given by an N↑TA $\mathcal{A}^\sharp = (Q_\mathcal{A}, \Sigma, \Delta^\sharp, F^\sharp_\mathcal{A})$ with $T(\mathcal{A}^\sharp) = T$, $\Delta^\sharp \subseteq \{(L(\mathcal{C}^\sharp_{aq}), a, q) \mid a \in \Sigma, q \in Q_\mathcal{A}\}$ and NFAs $\mathcal{C}^\sharp_{aq} = (P_{aq}, Q_\mathcal{A}, p_{aq0}, \Delta^\sharp_{aq}, F^\sharp_{aq})$, where $P_{aq} = \{p_{aqk} \mid 0 \leq k < |P_{aq}|\}$. Furthermore, the NFAs $\mathcal{C}^\sharp_{aq}$ are constructed such that no transitions lead back into the respective initial state p_{aq0}. This can always be accomplished using standard techniques described e.g. in [HMU01].

  In order to realize that all leaves of the trees in T are assigned states in Q_Σ, a new N↑TA $\mathcal{A}$ is constructed that inherits all transitions of $\mathcal{A}^\sharp$, has the transitions of Δ_Σ in addition, and realizes that the same runs of $\mathcal{C}^\sharp_{aq}$ are possible for runs that assign the Q_Σ-states to the leaves.

  Formally, construct N↑TA $\mathcal{A} = (Q_\mathcal{A} \dot\cup Q_\Sigma, \Sigma, \Delta_\mathcal{A} \dot\cup \Delta_\Sigma, F_\mathcal{A})$ with $\Delta_\mathcal{A} := \{(L(\mathcal{C}_{aq}), a, q) \mid (L(\mathcal{C}^\sharp_{aq}), a, q) \in \Delta^\sharp\}$ and $\mathcal{C}_{aq} = (P_{aq}, Q_\Sigma \dot\cup Q_\mathcal{A}, p_{aq0}, \Delta_{aq}, F_{aq})$ such that $\Delta_{aq} = \Delta^\sharp_{aq} \cup \{(p_{aqk}, q_b, p_{aq\ell}) \mid \exists b \in \Sigma, \exists q' \in Q_\mathcal{A}\big((p_{aqk}, q', p_{aq\ell}) \in \Delta^\sharp_{aq} \wedge \varepsilon \in L(\mathcal{C}^\sharp_{bq'})\big)\}$. With $F_{aq} := F^\sharp_{aq} \setminus \{p_{aq0}\}$, it is ensured that the NFAs $\mathcal{C}_{aq}$ do not accept the empty word (recall that no transition leads back to the initial state p_{aq0}, thus removing p_{aq0} from the set of final states does not change the languages in any other way), and thus, any run of $\mathcal{A}$ has to start with transitions of Δ_Σ. The set $F_\mathcal{A}$ of final states is expanded such that if there is a tree $t \in T$ with $ht(t) = 0$, it is accepted by a run that assigns the state $q_{t(\varepsilon)} \in Q_\Sigma$ to t, hence $F_\mathcal{A} := F^\sharp_\mathcal{A} \cup \{q_a \mid q_a \in Q_\Sigma \wedge a \in T(\mathcal{A}^\sharp)\}$. It is not difficult to see that $T(\mathcal{A}) = T$, and that for each $t \in T$, there exists an accepting run of $\mathcal{A}$ on t that assigns the state $q_{t(x)} \in Q_\Sigma$ to each leaf x of t.

- For subtree rewrite rules $r_j : T_j \overset{\sigma}{\hookrightarrow} T'_j$ with $j \in J$:
  - Since the left hand side of rule r_j consists of the regular set T_j of trees, and with the construction of $\mathcal{A}$ shown above, it is not difficult to construct N↑TAs $\mathcal{A}_j = (Q_j \dot\cup Q_\Sigma, \Sigma, \Delta_j \dot\cup \Delta_\Sigma, F_j)$ with $T(\mathcal{A}_j) = T_j$ with the following properties: the state sets Q_j are pairwise disjoint, and moreover, disjoint from the state set $Q_\mathcal{A}$ of $\mathcal{A}$, and for each $t \in T_j$ there is an accepting run of $\mathcal{A}_j$ on t that assigns the state $q_{t(x)} \in Q_\Sigma$ to each leaf x of t.
  - Analogously, construct N↑TAs $\mathcal{A}'_j = (Q'_j \dot\cup Q_\Sigma, \Sigma, \Delta'_j \dot\cup \Delta_\Sigma, F'_j)$ for the right hand side of rule r_j, with $T(\mathcal{A}'_j) = T'_j$ and with analogous properties as the $\mathcal{A}_j$.

- For flat prefix rewrite rules $r_i : L_i \overset{\sigma}{\hookrightarrow} L'_i$ with $i \in I$,

- let L_i be given by an NFA $\mathcal{A}_i^\sharp = (P_i^\sharp, \Sigma, p_0, \Delta_i^\sharp, F_i^\sharp)$ with $L(\mathcal{A}_i^\sharp) = L_i$. As the leaves of trees in T_Σ are assigned states in Q_Σ by the N↑TAs, NFAs $\mathcal{A}_i$ are constructed that recognize the languages Q_{L_i}, and for technical reasons have pairwise disjoint state sets.

  Formally, construct NFAs $\mathcal{A}_i := (P_i, Q_\Sigma, p_{i0}, \Delta_i, F_i)$ with $P_i := \{p_{i\ell} \mid p_\ell \in P_i^\sharp\}$, $\Delta_i := \{(p_{ik}, q_a, p_{i\ell}) \mid (p_k, a, p_\ell) \in \Delta_i^\sharp\}$, and $F_i := \{p_{i\ell} \mid p_\ell \in F_i^\sharp\}$. It is not difficult to see that $L(\mathcal{A}_i) = Q_{L_i}$, i.e. $\mathcal{A}_i$ recognizes the flat prefix that makes up the left hand side of rule r_i.

- let L_i' be given by an NFA $\mathcal{A}_i'^\sharp = (P_i'^\sharp, \Sigma, p_0', \Delta_i'^\sharp, F_i'^\sharp)$ with $L(\mathcal{A}_i'^\sharp) = L_i'$. Construct NFAs $\mathcal{A}_i' := (P_i', Q_\Sigma, p_{i0}', \Delta_i', F_i')$ with $L(\mathcal{A}_i') = Q_{L_i'}$ for the right hand side of flat prefix rewrite rule r_i analogously to the case above.

Note that the state sets of the NFAs are pairwise disjoint. For the N↑TAs, the sets Q_j, Q_j', and $Q_\mathcal{A}$ are also pairwise disjoint for all $j \in J$.

1. Algorithm 3.2 terminates, since the state set $Q_\mathcal{B}$ of the N↑TAs $\mathcal{B}_k$ and the state sets $P_{aq}^\mathcal{D}$ of the NFAs $\mathcal{D}_k^{aq}$ are not altered, and consequently only finitely many transitions can be added.

2. In order to show $T(\mathcal{A}_{\text{pre}_\mathcal{R}^*}) = \text{pre}_\mathcal{R}^*(T)$, five lemmata will be utilized.

   Before the two language inclusions are treated, it will be shown that if a tree t is reduced to a state $\tilde{q} \in Q_\mathcal{B}$ by N↑TA $\mathcal{B}_k$, then t can be rewritten by $\mathcal{R}$ to a tree s which can be deduced to state $\tilde{q}$ by $\mathcal{B}_0$. This is expressed in Lemma 3.6 part *(a)*. Since flat prefix rewriting is only allowed at the flat front of the tree, an NFA-run over a successor sequence is split into two parts: the first one allows the usage of ε-transitions since flat prefix rewriting may have occurred, and the second one does not. Parts *(b)* and *(c)* of Lemma 3.6 are the supplementary statements for these two parts of an NFA-path.

   Moreover, for the subsequently added q_b-transitions it can be shown that if $(p_{aqmn}^\mathcal{D}, q_b, p_{aq\mu\lambda}^\mathcal{D})$ was added to Δ_k^{aq} for $(p_{aqmn}^\mathcal{C}, \hat{q}, p_{aq\mu\lambda}^\mathcal{C}) \in \Delta_{aq}^\mathcal{C}$ (cf. Line 8 of Algorithm 3.2, and Figure 3.5(b)), and it holds that $\hat{q} \notin Q_\Sigma$ (i.e. the q_b-transition is not in an $\mathcal{A}_i$ copy of $\mathcal{D}_k^{aq}$, but in the $\mathcal{C}_{aq}$ copy of $\mathcal{D}_k^{aq}$), it is *not* possible to traverse an ε-transition after such a q_b-transition:

   Since $(p_{aqmn}^\mathcal{D}, q_b, p_{aq\mu\lambda}^\mathcal{D})$ is only added to Δ_k^{aq} for $(p_{aqmn}^\mathcal{C}, \hat{q}, p_{aq\mu\lambda}^\mathcal{C}) \in \Delta_{aq}^\mathcal{C}$ or $(p_{mn}, \hat{q}, p_{\mu\lambda}) \in \Delta_{aq}^\mathcal{C}$ (cf. Line 8 of Algorithm 3.2), if $\hat{q} \notin Q_\Sigma$, i.e. $\hat{q} \in Q_\mathcal{A}$, the case $(p_{mn}, \hat{q}, p_{\mu\lambda})$ cannot be fulfilled and thus the q_b-transition was added for $(p_{aqmn}^\mathcal{C}, \hat{q}, p_{aq\mu\lambda}^\mathcal{C}) \in \Delta_{aq}^\mathcal{C}$. As $p_{aqmn}^\mathcal{C}$ and $p_{aq\mu\lambda}^\mathcal{C}$ are states of the $\mathcal{C}_{aq}$ copy of $\mathcal{C}_{aq}^*$, $p_{aqmn}^\mathcal{D}$ and $p_{aq\mu\lambda}^\mathcal{D}$ are located in the $\mathcal{C}_{aq}$ copy of $\mathcal{D}_k^{aq}$ (for all k). The added ε-transitions however are of the form $(p_{aqik}^\mathcal{D}, \hat{p})$ for $p_{ik} \in F_i$, i.e. $p_{aqik}^\mathcal{D}$ is in an $\mathcal{A}_i$ copy of $\mathcal{D}_k^{aq}$. The only transitions leading to the $\mathcal{A}_i$ copies of $\mathcal{D}_k^{aq}$ start from the initial state $p_{aq00}^\mathcal{D}$ of $\mathcal{D}_k^{aq}$. According to the construction, no transitions lead back to $p_{aq00}^\mathcal{D}$, thus no $p_{aqik}^\mathcal{D}$ is reachable from $p_{aq\mu\lambda}^\mathcal{D}$, and therefore,

no ε-transition can be traversed after an inserted $(p^{\mathcal{D}}_{aqmn}, q_b, p^{\mathcal{D}}_{aq\mu\lambda})$-transition which was added for $(p^{\mathcal{C}}_{aqmn}, \hat{q}, p^{\mathcal{C}}_{aq\mu\lambda})$ with $\hat{q} \notin Q_\Sigma$.

To facilitate the notation, for hedges $h = s_1 \cdots s_n$ and $h' = s'_1 \cdots s'_n$, let $h \to^*_{\mathcal{R}} h'$ denote $s_1 \to^*_{\mathcal{R}} s'_1, \ldots, s_n \to^*_{\mathcal{R}} s'_n$, and analogously, for $\underline{q} = q_1 \cdots q_n$, let $h \to^*_{\mathcal{B}_k} \underline{q}$ denote $s_1 \to^*_{\mathcal{B}_k} q_1, \ldots, s_n \to^*_{\mathcal{B}_k} q_n$.

With these preparations, the proof can be continued as follows:

Lemma 3.6. *For each ε-N↑TA $\mathcal{B}_k$ computed in Algorithm 3.2, each $t \in T_\Sigma$, each $\underline{q}_u \in Q^*_\Sigma$, and each $q_c \in Q_\Sigma$ the following holds:*

(a) If $t \xrightarrow[\mathcal{B}_k]{*} \tilde{q} \in Q_{\mathcal{B}}$, then there is a tree $s \in T_\Sigma$, such that $t \xrightarrow[\mathcal{R}]{*} s$ and $s \xrightarrow[\mathcal{B}_0]{*} \tilde{q}$.

(b) If $\mathcal{D}^{aq}_k : p^{\mathcal{D}}_{aq00} \xrightarrow{\underline{q}_u} p^{\mathcal{D}}$, then there is a hedge h, such that $u \xrightarrow[\mathcal{R}]{*} h$, $h \xrightarrow[\mathcal{B}_0]{*} \underline{q} \in (Q_\Sigma \cup Q_{\mathcal{A}})^*$, and $\mathcal{C}^*_{aq} : p^{\mathcal{C}}_{aq00} \xrightarrow{\underline{q}} p^{\mathcal{C}}$.

(c) If $(p^{\mathcal{D}}_1, q_c, p^{\mathcal{D}}_2) \in \Delta^{aq}_k$, then there is a tree $s \in T_\Sigma$, such that $c \xrightarrow[\mathcal{R}]{*} s$, $s \xrightarrow[\mathcal{B}_0]{*} q \in (Q_\Sigma \cup Q_{\mathcal{A}})$, and there is $(p^{\mathcal{C}}_1, q, p^{\mathcal{C}}_2) \in \Delta^{\mathcal{C}}_{aq}$.

PROOF. The claims will be proven by induction on the number k of saturation steps in Algorithm 3.2. To exploit dependencies and reuse results, the lemma will be shown in reverse order.

Case $k = 0$. If $(p^{\mathcal{D}}_1, q_c, p^{\mathcal{D}}_2) \in \Delta^{aq}_0$, choose tree $s := c$. Then it holds that $c \to^*_{\mathcal{B}_0} q_c$ since $c \in \Sigma$ and $\Delta_\Sigma \subseteq \Delta^{\mathcal{B}}_0$. Since $\mathcal{D}^{aq}_0$ is the restriction of $\mathcal{C}^*_{aq}$ to Q_Σ, it holds that $(p^{\mathcal{C}}_1, q_c, p^{\mathcal{C}}_2) \in \Delta^{\mathcal{C}}_{aq}$ and part *(c)* is proven.
Analogously, for part *(b)* let $\mathcal{D}^{aq}_0 : p^{\mathcal{D}}_{aq00} \to^{\underline{q}_u} p^{\mathcal{D}}$. With hedge $h := u$ it holds that $h \to^*_{\mathcal{B}_0} q_{u(1)} \cdots q_{u(|u|)}$, since $u \in \Sigma^*$ and $\Delta_\Sigma \subseteq \Delta^{\mathcal{B}}_0$. Since $\mathcal{D}^{aq}_0$ is the restriction of $\mathcal{C}^*_{aq}$ to Q_Σ, it holds that $\mathcal{C}^*_{aq} : p^{\mathcal{C}}_{aq00} \to^{\underline{q}_u} p^{\mathcal{C}}$.
With $t \to^*_{\mathcal{B}_0} \tilde{q} \in Q_{\mathcal{B}}$, part *(a)* of the claim holds for $s := t$.

Case $k \to k+1$. Assume that the claim holds for k. According to Line 5 of Algorithm 3.2, two cases are to be distinguished:

1. $\mathcal{B}_{k+1}$ was derived from $\mathcal{B}_k$ because $(\star)$ applied.
2. $\mathcal{B}_{k+1}$ was derived from $\mathcal{B}_k$ because $(\star\star)$ applied.

In both cases, a second induction on the number κ of inserted transitions used in the derivation of *(b)* $\mathcal{D}^{aq}_{k+1} : p^{\mathcal{D}}_{aq00} \to^{\underline{q}_u} p^{\mathcal{D}}$ respectively *(a)* $t \to^*_{\mathcal{B}_{k+1}} \tilde{q}$ needs to be applied. For both cases, if $\kappa = 0$, it holds that $\mathcal{D}^{aq}_k : p^{\mathcal{D}}_{aq00} \to^{\underline{q}_u} p^{\mathcal{D}}$ respectively $t \to^*_{\mathcal{B}_k} \tilde{q}$, and the claim holds by induction on k. To show *Case* $\kappa \to \kappa + 1$ and part *(c)*, the two cases are considered separately:

> *Case* 1. $(\star)$ applied, i.e. $\mathcal{B}_{k+1}$ was derived from $\mathcal{B}_k$ because a subtree rewrite rule could be applied. That means, in general two different kinds of rules were added to $\mathcal{B}_k$: rules of the form $(q, \hat{q}) \in (Q_{\mathcal{B}} \cap F_\ell) \times Q_{\mathcal{B}}$ (cf. Line 11 of Algorithm 3.2, and Figure 3.5(a)) and rules of the form

$(p_1^{\mathcal{D}}, q_b, p_2^{\mathcal{D}}) \in P_{aq}^{\mathcal{D}} \times (Q_\Sigma \cap F_\ell) \times P_{aq}^{\mathcal{D}}$ (cf. Line 8 of Algorithm 3.2, and Figure 3.5(b)) for $\ell \in J$.

For part (c) of Lemma 3.6, transitions of $(Q_{\mathcal{B}} \cap F_\ell) \times Q_{\mathcal{B}}$ are irrelevant. Therefore consider a transition $(p_1^{\mathcal{D}}, q_b, p_2^{\mathcal{D}}) \in \Delta_{k+1}^{aq} \setminus \Delta_k^{aq}$. Since the transition was added to $\mathcal{D}_k^{aq}$, there is a rule $T_\ell \overset{\sigma}{\hookrightarrow} T'_\ell \in R$ with $b \in T(\mathcal{A}_\ell)$, $s' \in T(\mathcal{A}'_\ell)$ and a transition $(p_1^{\mathcal{C}}, \hat{q}, p_2^{\mathcal{C}}) \in \Delta_{aq}^{\mathcal{C}}$ for $s' \to_{\mathcal{B}_k}^* \hat{q}$ (cf. Lines 5 and 8). With the induction assumption of part *(a)* there is a tree $s \in T_\Sigma$ with $s' \to_{\mathcal{R}}^* s$ and $s \to_{\mathcal{B}_0}^* \hat{q}$. Thus, $b \to_{\mathcal{R}} s' \to_{\mathcal{R}}^* s$, $s \to_{\mathcal{B}_0}^* \hat{q}$, and $(p_1^{\mathcal{C}}, \hat{q}, p_2^{\mathcal{C}}) \in \Delta_{aq}^{\mathcal{C}}$.

For part (b) of Lemma 3.6, transitions of $(Q_{\mathcal{B}} \cap F_\ell) \times Q_{\mathcal{B}}$ are again irrelevant. For the case that transitions of the form $(p_\mu^{\mathcal{D}}, q_{b_\mu}, p_{\mu+1}^{\mathcal{D}}) \in \Delta_{k+1}^{aq} \setminus \Delta_k^{aq}$ were used, consider a run

$$\mathcal{D}_{k+1}^{aq} : p_{aq00}^{\mathcal{D}} \xrightarrow{\underline{q}_{w_1}} p_2^{\mathcal{D}} \xrightarrow{q_{b_2}} \tilde{p}_2^{\mathcal{D}} \xrightarrow{\underline{q}_{w_2}} p_3^{\mathcal{D}} \xrightarrow{q_{b_3}} \tilde{p}_3^{\mathcal{D}} \xrightarrow{\underline{q}_{w_3}} \cdots \xrightarrow{\underline{q}_{w_{m-1}}} p_m^{\mathcal{D}} \xrightarrow{q_{b_m}} \tilde{p}_m^{\mathcal{D}} \xrightarrow{\underline{q}_{w_m}} p_{m+1}^{\mathcal{D}} \xrightarrow{q_{b_{m+1}}} p_{m+2}^{\mathcal{D}} \xrightarrow{q_{b_{m+2}}} \cdots \xrightarrow{q_{b_{m+n}}} p^{\mathcal{D}}$$

with

- $\mathcal{D}_k^{aq} : \tilde{p}_\mu^{\mathcal{D}} \to^{\underline{q}_{w_\mu}} p_{\mu+1}^{\mathcal{D}}$ for $1 \le \mu \le m$, with $\tilde{p}_1^{\mathcal{D}} = p_{aq00}^{\mathcal{D}}$,
- $(p_\mu^{\mathcal{D}}, q_{b_\mu}, \tilde{p}_\mu^{\mathcal{D}}) \in \Delta_{k+1}^{aq} \setminus \Delta_k^{aq}$ for $2 \le \mu \le m$, and
- $(p_\mu^{\mathcal{D}}, q_{b_\mu}, p_{\mu+1}^{\mathcal{D}}) \in \Delta_{k+1}^{aq}$ for $m+1 \le \mu \le m+n$, $p_{m+n+1}^{\mathcal{D}} = p^{\mathcal{D}}$.

Thus, there is a hedge $wb_{m+1} \cdots b_{m+n}$ with $w = w_1 b_2 w_2 b_3 \cdots w_{m-1} b_m w_m$. In the preparations of Lemma 3.6, it was shown that for inserted $(p_\mu^{\mathcal{D}}, q_{b_\mu}, p_{\mu+1}^{\mathcal{D}})$-transitions with $(p_\mu^{\mathcal{C}}, \hat{q}, p_{\mu+1}^{\mathcal{C}}) \in \Delta_{aq}^{\mathcal{C}}$ and $\hat{q} \notin Q_\Sigma$, it is *not* possible to traverse an ε-transition after such a transition. Note though, that on every segment of the run that is of the form $\mathcal{D}_k^{aq} : \tilde{p}_\mu^{\mathcal{D}} \to^{\underline{q}_{w_\mu}} p_{\mu+1}^{\mathcal{D}}$ for $1 \le \mu \le m$, a traversal of ε-transitions is possible. Furthermore, for all $(p_\mu^{\mathcal{D}}, q_{b_\mu}, \tilde{p}_\mu^{\mathcal{D}}) \in \Delta_{k+1}^{aq} \setminus \Delta_k^{aq}$, $1 \le \mu \le m$, there is a transition $(p_\mu^{\mathcal{C}}, q_{c_\mu}, \tilde{p}_\mu^{\mathcal{C}}) \in \Delta_{aq}^{\mathcal{C}}$ with $q_{c_\mu} \in Q_\Sigma$, i.e. $c_\mu \in \Sigma$.

This means according to part *(c)* of Lemma 3.6, that with $(p_\mu^{\mathcal{D}}, q_{b_\mu}, \tilde{p}_\mu^{\mathcal{D}}) \in \Delta_{k+1}^{aq}$ there exist trees $d_\mu \in T_\Sigma$ with $b_\mu \to_{\mathcal{R}}^* d_\mu$ and $d_\mu \to_{\mathcal{B}_0}^* q_{c_\mu} \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})$. As the only transitions applicable at leaves in $\mathcal{B}_0$ are transitions of Δ_Σ, it holds that $d_\mu = c_\mu$, and with $(p_\mu^{\mathcal{C}}, q_{c_\mu}, \tilde{p}_\mu^{\mathcal{C}}) \in \Delta_{aq}^{\mathcal{C}}$, there also are transitions $(p_\mu^{\mathcal{D}}, q_{c_\mu}, \tilde{p}_\mu^{\mathcal{D}}) \in \Delta_0^{aq}$ for $2 \le \mu \le m$. Therefore, $wb_{m+1} \cdots b_{m+n} \to_{\mathcal{R}}^* vb_{m+1} \cdots b_{m+n}$, with $w = w_1 b_2 w_2 b_3 \cdots w_{m-1} b_m w_m$ and $v = w_1 c_2 w_2 c_3 \cdots w_{m-1} c_m w_m$, and there exists a run

$$\mathcal{D}_{k+1}^{aq} : p_{aq00}^{\mathcal{D}} \xrightarrow{\underline{q}_v} p_{m+1}^{\mathcal{D}} \xrightarrow{q_{b_{m+1}}} p_{m+2}^{\mathcal{D}} \xrightarrow{q_{b_{m+2}}} \cdots \xrightarrow{q_{b_{m+n}}} p^{\mathcal{D}}$$
$$\text{with } \mathcal{D}_k^{aq} : p_{aq00}^{\mathcal{D}} \xrightarrow{\underline{q}_v} p_{m+1}^{\mathcal{D}},\ \underline{q}_v \in Q_\Sigma^*.$$

The induction assumption of part *(b)* yields a hedge h' with $v \to_{\mathcal{R}}^* h'$, $h' \to_{\mathcal{B}_0}^* \underline{q}' \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})^*$, and $\mathcal{C}_{aq}^* : p_{aq00}^{\mathcal{C}} \to^{\underline{q}'} p_{m+1}^{\mathcal{C}}$.

The transitions $(p^{\mathcal{D}}_{\mu}, q_{b_\mu}, p^{\mathcal{D}}_{\mu+1}) \in \Delta^{aq}_{k+1}$ with $m+1 \leq \mu \leq m+n$ are either from Δ^{aq}_{k} or newly inserted and from $\Delta^{aq}_{k+1} \setminus \Delta^{aq}_{k}$. Since part *(c)* has already been shown, for both cases it holds that there exist trees $s_\mu \in T_\Sigma$ with $b_\mu \to^*_{\mathcal{R}} s_\mu$, $s_\mu \to^*_{\mathcal{B}_0} \tilde{q}_\mu \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})$ (for the transitions of $\Delta^{aq}_{k+1} \setminus \Delta^{aq}_{k}$ it holds that $\tilde{q}_\mu = \hat{q}$), and $(p^{\mathcal{C}}_{\mu}, \tilde{q}_\mu, p^{\mathcal{C}}_{\mu+1}) \in \Delta^{\mathcal{C}}_{aq}$.

Together, for the run $\mathcal{D}^{aq}_{k+1} : p^{\mathcal{D}}_{aq00} \xrightarrow{\underline{q}_w q_{b_{m+1}} \cdots q_{b_{m+n}}} p^{\mathcal{D}}$ there is a hedge $h := h' s_{m+1} \cdots s_{m+n}$ with $w b_{m+1} \cdots b_{m+n} \to^*_{\mathcal{R}} v b_{m+1} \cdots b_{m+n} \to^*_{\mathcal{R}} h$, $h \to^*_{\mathcal{B}_0} \underline{q}' \tilde{q}_{m+1} \cdots \tilde{q}_{m+n}$ with

$$\tilde{q}_\mu = \begin{cases} q_\mu & \text{for } (p^{\mathcal{D}}_{\mu}, q_{b_\mu}, p^{\mathcal{D}}_{\mu+1}) \in \Delta^{aq}_{k} \\ \hat{q} & \text{for } (p^{\mathcal{D}}_{\mu}, q_{b_\mu}, p^{\mathcal{D}}_{\mu+1}) \in \Delta^{aq}_{k+1} \setminus \Delta^{aq}_{k} \end{cases}$$

and $\mathcal{C}^*_{aq} : p^{\mathcal{C}}_{aq00} \xrightarrow{\underline{q}'} p^{\mathcal{C}}_{m+1} \xrightarrow{\tilde{q}_{m+1}} p^{\mathcal{C}}_{m+2} \xrightarrow{\tilde{q}_{m+2}} \cdots \xrightarrow{\tilde{q}_{m+n}} p^{\mathcal{C}}$.

For part (a) of Lemma 3.6, both kinds of newly added transitions need to be considered.

For the case that transitions of the form $(p^{\mathcal{D}}_{\mu}, q_{b_\mu}, p^{\mathcal{D}}_{\mu+1}) \in \Delta^{aq}_{k+1} \setminus \Delta^{aq}_{k}$ were used, consider a derivation

$$t \xrightarrow[\mathcal{B}_k]{*} t[x|a(\underline{q}_u)] \xrightarrow[\mathcal{B}_{k+1}]{} t[x|q] \xrightarrow[\mathcal{B}_{k+1}]{*} \tilde{q} \in Q_{\mathcal{B}}. \qquad (\diamond)$$

It will be shown that there is a derivation

$$t \overset{(1)}{\xrightarrow[\mathcal{R}]{*}} t[x|a(u)] \overset{(2)}{\xrightarrow[\mathcal{R}]{*}} t[x|a(h)] \overset{(3)}{\xrightarrow[\mathcal{R}]{*}} s \text{ such that } s \overset{(4)}{\xrightarrow[\mathcal{B}_0]{*}} \tilde{q}.$$

Derivation (1): With $(\diamond)$ it holds that $t_{\downarrow x} \to^*_{\mathcal{B}_k} a(\underline{q}_u)$. For the subtrees $t_{\downarrow x1} \cdots t_{\downarrow x|u|}$ the induction assumption of part *(a)* can be applied. To avoid complicated indices, these are denoted by a hedge: the induction assumption of part *(a)* yields a hedge h' such that $t_{\downarrow x1} \cdots t_{\downarrow x|u|} \to^*_{\mathcal{R}} h'$ with $h' \to^*_{\mathcal{B}_0} \underline{q}_u$. Since $\underline{q}_u \in Q^*_\Sigma$, it follows that $h' = u$.

Derivation (2): In the run $\mathcal{D}^{aq}_{k+1} : p^{\mathcal{D}}_{aq00} \to^{\underline{q}_u} p^{\mathcal{D}} \in F^{\mathcal{D}}_{aq}$ the newly added transitions were used. With part *(b)* shown above, there is a hedge h with $u \to^*_{\mathcal{R}} h$, $h \to^*_{\mathcal{B}_0} \underline{q} \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})^*$, and $\mathcal{C}^*_{aq} : p^{\mathcal{C}}_{aq00} \to^{\underline{q}} p^{\mathcal{C}} \in F^{\mathcal{C}}_{aq}$ (by construction: $p^{\mathcal{D}} \in F^{\mathcal{D}}_{aq}$ iff $p^{\mathcal{C}} \in F^{\mathcal{C}}_{aq}$), and thus $t[x|a(u)] \to^*_{\mathcal{R}} t[x|a(h)]$.

Derivations (3) and (4): Since with $(\diamond)$ it holds that $t[x|a(h)] \to^*_{\mathcal{B}_0} t[x|a(\underline{q})] \to_{\mathcal{B}_0} t[x|q] \to^*_{\mathcal{B}_{k+1}} \tilde{q} \in Q_{\mathcal{B}}$, one less new transition of $\mathcal{B}_{k+1}$ was used in the derivation and by induction on the number κ of newly added transitions used in the derivation there is $s \in T_\Sigma$ with $t[x|a(h)] \xrightarrow[\mathcal{R}]{*} s$ and $s \xrightarrow[\mathcal{B}_0]{*} \tilde{q}$.

For the case that a transition of the form $(q, \hat{q}) \in \Delta^{\mathcal{B}}_{k+1} \setminus \Delta^{\mathcal{B}}_{k}$ with $q \in (Q_{\mathcal{B}} \cap F_\ell)$ for $\ell \in J$ was used, consider a derivation

$$t \xrightarrow[\mathcal{B}_k]{*} t[x|q] \xrightarrow[\mathcal{B}_{k+1}]{\varepsilon} t[x|\hat{q}] \xrightarrow[\mathcal{B}_{k+1}]{*} \tilde{q} \in Q_{\mathcal{B}}. \qquad (\diamond\diamond)$$

It will be shown that there is a derivation

$$t \overset{(1)}{\underset{\mathcal{R}}{\overset{*}{\longrightarrow}}} t[x|s_1] \overset{(2)}{\underset{\mathcal{R}}{\longrightarrow}} t[x|s'] \overset{(3)}{\underset{\mathcal{R}}{\overset{*}{\longrightarrow}}} s \text{ such that } s \overset{(4)}{\underset{\mathcal{B}_0}{\overset{*}{\longrightarrow}}} \tilde{q}.$$

Derivation (1): With $(\diamond\diamond)$ it holds that $t_{\downarrow x} \to^*_{\mathcal{B}_k} q$, and thus with the induction assumption of part *(a)* there is $s_1 \in T_\Sigma$ with $t_{\downarrow x} \to^*_{\mathcal{R}} s_1$ and $s_1 \to^*_{\mathcal{B}_0} q$.

Derivation (2): Since the transition $(q, \hat{q})$ was added to $\mathcal{B}_k$, there is $s' \in T(\mathcal{A}'_\ell)$ with $s' \to^*_{\mathcal{B}_k} \hat{q}$ (cf. Line 5). Since $q \in F_\ell$ it holds that $s_1 \in T(\mathcal{A}_\ell)$ and thus $s_1 \to_{\mathcal{R}} s'$.

Derivations (3) and (4): With $(\diamond\diamond)$ it follows that $t[x|s'] \to^*_{\mathcal{B}_k} t[x|\hat{q}] \to^*_{\mathcal{B}_{k+1}} \tilde{q}$, and thus one less new $\mathcal{B}_{k+1}$-transition was used in the derivation. By induction on κ there exists a tree $s \in T_\Sigma$ with $t[x|s'] \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} s$ and $s \overset{*}{\underset{\mathcal{B}_0}{\longrightarrow}} \tilde{q}$.

Case 2. $(\star\star)$ applied, i.e. $\mathcal{B}_{k+1}$ derived from $\mathcal{B}_k$ because a flat prefix rewrite rule could be applied. This means that transitions of the form $(p^{\mathcal{D}}_{aq\ell m}, \hat{p}^{\mathcal{D}}) \in P^{\mathcal{D}}_{aq} \times P^{\mathcal{D}}_{aq}$ with $p^{\mathcal{D}}_{aq\ell m}$ such that $p_{\ell m} \in F_\ell$ for $\ell \in I$ were added (cf. Line 13 of Algorithm 3.2, and Figure 3.5(c)) to $\mathcal{B}_k$.

For part (c) of Lemma 3.6, transitions of $P^{\mathcal{D}}_{aq} \times P^{\mathcal{D}}_{aq}$ are irrelevant.

For part (b) of Lemma 3.6, consider a run

$$\mathcal{D}^{aq}_{k+1} : p^{\mathcal{D}}_{aq00} \xrightarrow{\underline{q}_u} p^{\mathcal{D}}_2 \xrightarrow{\varepsilon} \hat{p}^{\mathcal{D}} \xrightarrow{q_{b_1} \cdots q_{b_m}} p^{\mathcal{D}}$$

such that $\mathcal{D}^{aq}_k : p^{\mathcal{D}}_{aq00} \to^{\underline{q}_u} p^{\mathcal{D}}_2$ and $(p^{\mathcal{D}}_2, \hat{p}^{\mathcal{D}}) \in \Delta^{aq}_{k+1} \setminus \Delta^{aq}_k$ is the first occurrence of a newly added transition. To facilitate the notation and to make the method more comprehensible, the proof is shown for one newly added transition only, i.e. it is assumed that already $\mathcal{D}^{aq}_k : \hat{p}^{\mathcal{D}} \to^{q_{b_1} \cdots q_{b_m}} p^{\mathcal{D}}$. For all other cases the induction on the number κ of newly added transitions used in the derivation applies and the steps of the proof shown here can be iterated.

Since $(p^{\mathcal{D}}_2, \hat{p}^{\mathcal{D}})$ was added to $\mathcal{D}^{aq}_k$, there is a $u' \in \Sigma^*$ with $\underline{q}_{u'} \in L(\mathcal{A}'_\ell)$ and $\mathcal{D}^{aq}_k : p^{\mathcal{D}}_{aq00} \to^{\underline{q}_{u'}} \hat{p}^{\mathcal{D}}$ (cf. Line 5). With the induction assumption of part *(b)*, there exists a hedge h' with $u' \to^*_{\mathcal{R}} h'$, $h' \to^*_{\mathcal{B}_0} \underline{q}' \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})^*$, and $\mathcal{C}^*_{aq} : p^{\mathcal{C}}_{aq00} \to^{\underline{q}'} \hat{p}^{\mathcal{C}}$.

For the transitions $(\overline{p}^{\mathcal{D}}_\mu, q_{b_\mu}, \overline{p}^{\mathcal{D}}_{\mu+1})$ for $1 \leq \mu \leq m$ with $\overline{p}^{\mathcal{D}}_1 = \hat{p}^{\mathcal{D}}$, $\overline{p}^{\mathcal{D}}_{m+1} = p^{\mathcal{D}}$, the induction assumption of part *(c)* yields trees s_μ with $b_\mu \to^*_{\mathcal{R}} s_\mu$, $s_\mu \to^*_{\mathcal{B}_0} q_\mu \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})$, and $(\overline{p}^{\mathcal{C}}_\mu, q_\mu, \overline{p}^{\mathcal{C}}_{\mu+1}) \in \Delta^{\mathcal{C}}_{aq}$.

Together it follows that $ub_1 \cdots b_m \to_{\mathcal{R}} u'b_1 \cdots b_m \to^*_{\mathcal{R}} h's_1 \cdots s_m =: h$, $h \to^*_{\mathcal{B}_0} \underline{q}'q_1 \cdots q_m \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})^*$, and $\mathcal{C}^*_{aq} : p^{\mathcal{C}}_{aq00} \xrightarrow{\underline{q}'} \hat{p}^{\mathcal{C}} \xrightarrow{q_1 \cdots q_m} p^{\mathcal{C}}$.

For part (a) of Lemma 3.6: Since part *(b)* for *Case* 2 has already been shown, the proof proceeds completely analogous to the proof of added transitions over $P^{\mathcal{D}}_{aq} \times (Q_\Sigma \cap F_\ell) \times P^{\mathcal{D}}_{aq}$ $(\ell \in J)$ in *Case* 1 with the derivation

$$t \xrightarrow[\mathcal{B}_k]{*} t[x|a(\underline{q}_u)] \xrightarrow[\mathcal{B}_{k+1}]{} t[x|q] \xrightarrow[\mathcal{B}_{k+1}]{*} \tilde{q} \in Q_{\mathcal{B}}. \qquad \blacksquare$$

If one instantiates part *(a)* of Lemma 3.6 with $\tilde{q} \in F_{\mathcal{A}}$, i.e. "$t \rightarrow^*_{\mathcal{B}_k} \tilde{q} \in F_{\mathcal{A}} \Rightarrow \exists\, s \in T_\Sigma : t \rightarrow^*_{\mathcal{R}} s$, and $s \rightarrow^*_{\mathcal{B}_0} \tilde{q}$", it follows that each tree t with $t \in T(\mathcal{B}_k)$ can be rewritten by the regular subtree and flat prefix rewriting system $\mathcal{R}$ to a tree s with $s \in T(\mathcal{B}_0) = T(\mathcal{A}) = T$. Thus, the following holds:

Lemma 3.7. *For each ε-N↑TA $\mathcal{B}_k$ computed in Algorithm 3.2, the inclusion*

$$T(\mathcal{B}_k) \subseteq pre^*_{\mathcal{R}}(T)$$

holds.

Thus, for an equality of the languages $T(\mathcal{B}_k)$ and $\text{pre}^*_{\mathcal{R}}(T)$, the other direction remains to be shown. For this, two auxiliary lemmata are needed:

Lemma 3.8. *If there is a run $q_1 \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} q_3$ in $\mathcal{A}_{pre^*_{\mathcal{R}}}$, then there also is a transition $q_1 \xrightarrow{\varepsilon} q_3$ in $\mathcal{A}_{pre^*_{\mathcal{R}}}$.*

PROOF. If there is an ε-transition (q_1, q_2) in $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}}$, then it was inserted by Algorithm 3.2 because there is a tree $t \in T(\mathcal{A}_\ell)$ of the left hand side of a subtree rewrite rule r_ℓ ($\ell \in J$) with $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}} : t \xrightarrow{*} q_1 \in F_\ell$, and a tree $t' \in T(\mathcal{A}'_\ell)$ of the right hand side of rule r_ℓ with $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}} : t' \xrightarrow{*} q_2 \xrightarrow{\varepsilon} q_3$. Thus state q_3 is also reachable from a tree of the right hand side of rule r_ℓ, $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}} : t' \xrightarrow{*} q_3$, and hence the algorithm also added the transition (q_1, q_3) to $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}}$. $\blacksquare$

Lemma 3.9. *For each tree $t \in T(\mathcal{A}_{pre^*_{\mathcal{R}}})$ there is an accepting run of $\mathcal{A}_{pre^*_{\mathcal{R}}}$ on t such that for every subtree $t_{\downarrow x}$ of t with $ht(t_{\downarrow x}) = 1$, $t(x) = a$, and $flatfront(t_{\downarrow x}) = w \in \Sigma^+$, it holds that $\underline{q}_w \in L(\mathcal{C}^*_{aq}) \cup L(\mathcal{D}^{aq}_k)$:*

$$\mathcal{A}_{pre^*_{\mathcal{R}}} : t = t[x|a(w)] \xrightarrow{*} t[x|a(\underline{q}_w)] \rightarrow t[x|q] \xrightarrow{*} \tilde{q} \in F_{\mathcal{A}}.$$

PROOF. Recall that the N↑TA $\mathcal{A}$ was constructed from $\mathcal{A}^\sharp$ such that the only transitions applicable at the leaves of trees of $T(\mathcal{A})$ are transitions of Δ_Σ. As Algorithm 3.2 does not add transitions to $\Delta^{cq'}_k$ of the form $(\{\varepsilon\}, c, q')$ nor transitions that lead back to the initial states of the $\mathcal{D}^{cq'}_k$ for any $c \in \Sigma$ and $q' \in Q_{\mathcal{B}}$, $\mathcal{A}_{\text{pre}^*_{\mathcal{R}}}$ can start runs on trees with transitions of Δ_Σ only.
With $t \in T(\mathcal{A}_{\text{pre}^*_{\mathcal{R}}})$, there exists a word $\underline{q} = \underline{q}(1) \cdots \underline{q}(|w|) \in (Q_\Sigma \dot\cup Q_{\mathcal{A}})^+$, such that there is an accepting run

$$\mathcal{A}_{\text{pre}^*_{\mathcal{R}}} : t = t[x|a(w)] \xrightarrow{*} t[x|a(\underline{q}_w)] \xrightarrow[(\diamond)]{*} t[x|a(\underline{q})] \rightarrow t[x|q] \xrightarrow{*} \tilde{q} \in F_{\mathcal{A}},$$

i.e. $\underline{q} \in L(\mathcal{C}^*_{aq}) \cup L(\mathcal{D}^{aq}_k)$, and the transitions in derivation ($\diamond$) are ε-transitions of $(Q_\mathcal{B} \times Q_\mathcal{B})$. Note that with Lemma 3.8, one transition $(\underline{q}_w(m), \underline{q}(m))$ suffices for each $1 \leq m \leq |w|$.

It will be shown that there also exists an accepting run of the desired form, i.e. that $\underline{q}_w \in L(\mathcal{C}^*_{aq}) \cup L(\mathcal{D}^{aq}_k)$, and no ε-transition needs to be applied at the leaves of $t_{\downarrow x}$. Thereby, two cases are distinguished.

Case $\underline{q} \in L(\mathcal{C}^*_{aq})$. This means that there is a run

$$\mathcal{C}^*_{aq} : \; p^\mathcal{C}_0 \xrightarrow{\underline{q}(1)} p^\mathcal{C}_1 \xrightarrow{\underline{q}(2)} \cdots \xrightarrow{\underline{q}(|w|)} p^\mathcal{C}_{|w|} \in F^\mathcal{C}_{aq},$$

with $p^\mathcal{C}_0 = p^\mathcal{C}_{aq00}$. For all $1 \leq m \leq |w|$, it holds that $p^\mathcal{C}_m$ is of the form $p^\mathcal{C}_{aq0n_m}$ ($n_m \in \mathbb{N}$), i.e. $p^\mathcal{C}_m$ is located in the $\mathcal{C}_{aq}$ copy of $\mathcal{C}^*_{aq}$, since any run in $\mathcal{C}^*_{aq}$ over states of the form $p_{i\lambda}$ (located in the $\mathcal{A}_i$ copy of $\mathcal{C}^*_{aq}$) does not lead to a final state (recall that $\mathcal{C}^*_{aq}$ is not changed by Algorithm 3.2).

For $\underline{q}(m) \in Q_\Sigma \dot\cup Q_\mathcal{A}$, it holds that $\underline{q}_w(m) \xrightarrow{\varepsilon} \underline{q}(m) \in \mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$. Since the ε-transition $(\underline{q}_w(m), \underline{q}(m))$ was added by the algorithm, and $\underline{q}_w(m) \in F_\ell \cap Q_\Sigma$, the transition $p^\mathcal{D}_{m-1} \xrightarrow{\underline{q}_w(m)} p^\mathcal{D}_m$ was also added to $\mathcal{D}^{aq}_k$ for $p^\mathcal{C}_{m-1} \xrightarrow{\underline{q}(m)} p^\mathcal{C}_m$. Consequently, it holds that $\underline{q}_w \in L(\mathcal{D}^{aq}_k)$.

Case $\underline{q} \in L(\mathcal{D}^{aq}_k)$. This means that there is a run

$$\mathcal{D}^{aq}_k : \; p^\mathcal{D}_0 \xrightarrow{\underline{q}(1)} p^\mathcal{D}_1 \xrightarrow{\underline{q}(2)} \cdots \xrightarrow{\underline{q}(|w|)} p^\mathcal{D}_{|w|} \in F^\mathcal{D}_{aq},$$

with $p^\mathcal{D}_0 = p^\mathcal{D}_{aq00}$, and for all $1 \leq m \leq |w|$, it holds that $\underline{q}(m) \in Q_\Sigma$. For $\underline{q}(m) = \underline{q}_w(m)$ nothing needs to be done.
For $\underline{q}(m) \neq \underline{q}_w(m)$, there is an ε-transition $(\underline{q}_w(m), \underline{q}(m))$ in $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$. There are two possibilities: transition $p^\mathcal{C}_{m-1} \xrightarrow{\underline{q}(m)} p^\mathcal{C}_m$ is in $\Delta^\mathcal{C}_{aq}$, or it is not. For the first case, the transition $p^\mathcal{D}_{m-1} \xrightarrow{\underline{q}_w(m)} p^\mathcal{D}_m$ was added to $\mathcal{D}^{aq}_k$.

Otherwise, the transition $p^\mathcal{D}_{m-1} \xrightarrow{\underline{q}(m)} p^\mathcal{D}_m$ was added to $\mathcal{D}^{aq}_k$. According to Line 8 of Algorithm 3.2, this was only possible for $\underline{q}(m) \in F_\ell \cap Q_\Sigma$ and $p^\mathcal{C}_{m-1} \xrightarrow{\hat{q}} p^\mathcal{C}_m \in \Delta^\mathcal{C}_{aq}$. Then, the ε-transition $(\underline{q}(m), \hat{q})$ was also added to $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$ (cf. Line 11), and with Lemma 3.8 there also is an ε-transition $(\underline{q}_w(m), \hat{q})$, and thus $\underline{q}_w(m) \in F_\ell$. Hence, the transition $p^\mathcal{D}_{m-1} \xrightarrow{\underline{q}_w(m)} p^\mathcal{D}_m$ was also added to $\mathcal{D}^{aq}_k$, and consequently $\underline{q}_w \in L(\mathcal{D}^{aq}_k)$ holds. ∎

With these properties, the remaining tree language inclusion can be shown:

Lemma 3.10. *For the ε-N↑TA $\mathcal{A}_{pre^*_\mathcal{R}}$ computed in Algorithm 3.2, the inclusion*

$$pre^*_\mathcal{R}(T) \subseteq T(\mathcal{A}_{pre^*_\mathcal{R}})$$

holds.

PROOF. First note that $T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}})$ is a superset of $T(\mathcal{B}_k)$ for all $\mathcal{B}_k$ computed in the algorithm: $T(\mathcal{B}_0) \subseteq T(\mathcal{B}_1) \subseteq \cdots \subseteq T(\mathcal{B}_k) \subseteq T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}})$. Consider a tree $t \in \mathrm{pre}^*_\mathcal{R}(T)$ and a derivation $t \to^*_\mathcal{R} t' \in T$ with κ many rewriting steps. The claim will be shown by induction on κ:

Case $\kappa = 0$. t was not rewritten by $\mathcal{R}$. Consequently:

$$t \in T(\mathcal{A}) = T(\mathcal{B}_0) \subseteq T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}).$$

Case $\kappa \to \kappa + 1$. It will be distinguished between a subtree rewriting step and a flat prefix rewriting step.

Case 1. A subtree rewriting step was applied to obtain a derivation of length $\kappa + 1$:

$$t = t[x|s_\ell] \underset{\mathcal{R}}{\longrightarrow} t[x|s'_\ell] \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} t' \in T = T(\mathcal{A})$$

with $s_\ell \in T(\mathcal{A}_\ell)$, $s'_\ell \in T(\mathcal{A}'_\ell)$, and $t[x|s'_\ell] \to^*_\mathcal{R} t'$ in κ steps, therefore with the induction assumption $t[x|s'_\ell] \in T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}})$ holds. Consider $\hat{q} \in Q_\mathcal{B}$ with $t[x|s'_\ell] \to^*_{\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}} t[x|\hat{q}] \to^*_{\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}} \tilde{q} \in F_\mathcal{A}$.
Since $s'_\ell \in T(\mathcal{A}'_\ell)$, the transitions $(F_\ell \times \{\hat{q}\})$ have already been added to $\Delta^\mathcal{B}_k$ (cf. Line 11 of Algorithm 3.2, otherwise the algorithm would not have terminated). Since $s_\ell \in T(\mathcal{A}_\ell)$, there is $q \in F_\ell$ with $t[x|s_\ell] \to^*_{\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}} q$. Then t is accepted by $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$ as follows:

$$\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}} : t \overset{*}{\to} t[x|q] \overset{\varepsilon}{\to} t[x|\hat{q}] \overset{*}{\to} \tilde{q} \in F_\mathcal{A}.$$

Case 2. A flat prefix rewriting step was applied to obtain a derivation of length $\kappa + 1$:

$$t = t[x|a(uv)] \underset{\mathcal{R}}{\longrightarrow} t[x|a(u'v)] \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} t' \in T = T(\mathcal{A})$$

with $u, u', v \in \Sigma^*$, $\underline{q}_u \in L(\mathcal{A}_\ell)$, $\underline{q}_{u'} \in L(\mathcal{A}'_\ell)$, and $t[x|a(u'v)] \to^*_\mathcal{R} t'$ in κ steps, therefore with the induction assumption $t[x|a(u'v)] \in T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}})$. With Lemma 3.9 it holds that there is an accepting run of ε-N$\uparrow$TA $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$ on $t[x|a(u'v)]$ of the form

$$\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}} : t[x|a(u'v)] \overset{*}{\to} t[x|a(\underline{q}_{u'}\underline{q}_v)] \to t[x|q] \overset{*}{\to} \tilde{q} \in F_\mathcal{A}.$$

Since $\underline{q}_{u'} \in L(\mathcal{A}'_\ell)$, there is a run $\mathcal{D}^{aq}_k : p^\mathcal{D}_{aq00} \to^{\underline{q}_{u'}} \hat{p}^\mathcal{D} \to^{\underline{q}_v} p^\mathcal{D} \in F^\mathcal{D}_{aq}$, and the algorithm already added the transitions $(\{p^\mathcal{D}_{aq\ell m} \mid p_{\ell m} \in F_\ell\} \times \{\hat{p}^\mathcal{D}\})$ to $\Delta^\mathcal{B}_k$ (cf. Line 13 of Algorithm 3.2, otherwise the algorithm would not have terminated). Since $\underline{q}_u \in L(\mathcal{A}_\ell)$, there also is a run $\mathcal{D}^{aq}_k : p^\mathcal{D}_{aq00} \to^{\underline{q}_u} p^\mathcal{D}_{aq\ell n}$ with $p_{\ell n} \in F_\ell$, and thus there is the run $\mathcal{D}^{aq}_k : p^\mathcal{D}_{aq00} \to^{\underline{q}_u} p^\mathcal{D}_{aq\ell n} \to^\varepsilon \hat{p}^\mathcal{D} \to^{\underline{q}_v} p^\mathcal{D} \in F^\mathcal{D}_{aq}$, and t is accepted by $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$ as follows:

$$\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}} : t \overset{*}{\to} t[x|a(\underline{q}_u\underline{q}_v)] \to t[x|q] \overset{*}{\to} \tilde{q} \in F_\mathcal{A}. \qquad \blacksquare$$

Consequently, with the five lemmata it is shown that for a regular subtree and flat prefix rewriting system $\mathcal{R} \in \mathfrak{R}^{rsfp}$ over unranked trees and a regular set $T \subseteq T_\Sigma$ of unranked trees, one can construct an ε-N$\uparrow$TA $\mathcal{A}_{\text{pre}^*_\mathcal{R}}$ with $T(\mathcal{A}_{\text{pre}^*_\mathcal{R}}) = \text{pre}^*_\mathcal{R}(T)$. With a check whether $t \in \text{pre}^*_\mathcal{R}(T)$, it follows that:

Theorem 3.11. *For every regular subtree and flat prefix rewriting system $\mathcal{R} \in \mathfrak{R}^{rsfp}$ over unranked trees, the reachability problem EF: "Given $\mathcal{R}$, vertex t, and regular set T of vertices, is there a path from t to a vertex in T?" is decidable.*

Since singleton sets are regular, it follows directly that the reachability problem EF is also decidable for subtree and flat prefix rewriting systems.

As for the case of ground tree rewriting systems, the set $\text{post}^*_\mathcal{R}(T) = \{t \in T_\Sigma \mid t' \rightarrow^*_\mathcal{R} t \text{ for } t' \in T\}$ of trees which are reachable from the set T, can be obtained from applying Algorithm 3.2 for the reversed (regular) subtree and flat prefix rewriting system, i.e. the left and right hand sides of the rules are swapped.

Chapter 4

Undecidability Results

In the previously studied reachability problems over transition graphs of rewriting systems, all sets of trees that were employed were bound to be regular, i.e. recognizable by N↓TAs, N↑TAs, or D↑TAs. In this chapter, a variant of the "classical" constrained reachability problem (EU): "Given a rewriting system $\mathcal{R}$, vertex t, and sets T_1, T_2 of trees, is there a path starting in t that only visits vertices of T_1 until eventually reaching a vertex of T_2?" is studied. Thereby, the set T_2 remains regular, but for the set T_1 the tree language recognized by a D↓TA is employed. More precisely, a D↓TA is chosen which can scan the label sequence of the successors of a node before deciding about the states to be assigned to the successors. This model still is less expressive than the tree automata recognizing regular sets of unranked trees.

This problem was introduced in Chapter 1 – Preliminaries, and in order to distinguish it from the classical constrained reachability problem, it is denoted as the reachability problem over the restricted transition graph $G_{\mathcal{R}}|_{T(\mathcal{A})} = (T(\mathcal{A}), \xrightarrow[\mathcal{R}]{})$. For ground tree rewriting systems, this problem remains decidable (as opposed to the undecidability of the classical constrained reachability, cf. [Col02]), but it will be shown that this decidability result fails for the case of unranked trees and (regular) subtree and flat prefix rewriting systems, as well as partial subtree rewriting systems.

Theorem 4.1. *For a D↓TA $\mathcal{A}$ and a subtree and flat prefix rewriting system $\mathcal{R} \in \mathfrak{R}^{sfp}$ the reachability problem in the transition graph $G_{\mathcal{R}}|_{T(\mathcal{A})} = (T(\mathcal{A}), \xrightarrow[\mathcal{R}]{})$ is undecidable.*

This will be proven via a reduction to the halting problem for deterministic Turing machines, which is well known to be undecidable. In order to simulate a deterministic Turing machine M, a subtree and flat prefix rewriting system $\mathcal{R}_M \in \mathfrak{R}^{sfp}$ will be constructed. Since the reachability problem for (regular) subtree and flat prefix rewriting systems was shown to be decidable (cf. Theorem 3.11), the rewriting system alone will not suffice. Thus the transition graph

Figure 4.1 Tree encoding for a Turing machine configuration κ.

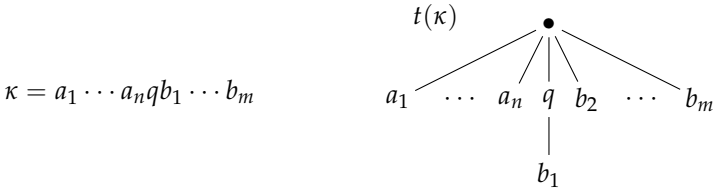
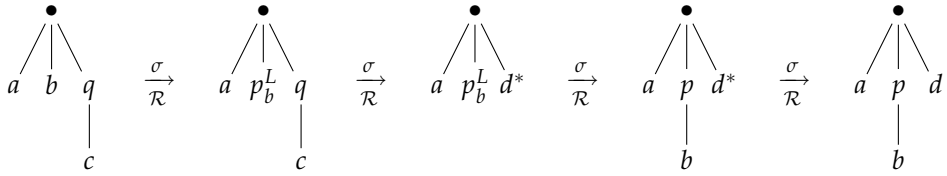


Figure 4.2 Correct simulation of the Turing machine transition $\delta(q, c) = (p, d, L)$.



of the subtree and flat prefix rewriting system is restricted to the language of a D↓TA $\mathcal{A}$, i.e. $G_M|_{T(\mathcal{A})} = (T(\mathcal{A}), \rightarrow_{\mathcal{R}_M})$, with $\mathcal{A}$ detecting and rejecting trees that can be generated by $\mathcal{R}_M$, but do not belong to a simulation of a step of the DTM M. Then, with the initial tree t_{in} of $\mathcal{R}_M$ coding the empty tape, and a set T_h^M of trees coding halting configurations, the DTM M stops on the empty tape iff a tree $t \in T_h^M$ is reachable from t_{in}.

In order to simulate the deterministic Turing machine $M = (Q, \Sigma, \Gamma, q_{in}, q_h, \delta)$, the configurations of M have to be encoded in trees. The tree representation for a configuration $\kappa = a_1 \cdots a_n q b_1 \cdots b_m$ is depicted in Figure 4.1.

It is not possible to simulate a Turing machine transition in just one rewriting step of a subtree and flat prefix rewriting system $\mathcal{R}_M$, as every transition induces a head movement to the left or to the right, and therefore two subtrees of $t(\kappa)$ have to be replaced. A correct simulation of the Turing machine transition $\delta(q, c) = (p, d, L)$ from $\kappa = abqc$ to $\kappa' = apbd$ is depicted in Figure 4.2. The single steps of the rewriting system are the following:

Since the reading head goes to the left into state p, the auxiliary label p_b^L is written left of the current state, remembering the new state to be taken, the letter in the working cell, and, for technical reasons, that this is a left-transition. Then, the old state and old working cell are replaced by the letter that the transition writes, which is still marked. After these two main steps, the rewriting system needs to resume a valid TM-configuration without special marks, which is taken care of in the next two steps. The intermediate steps with markers are necessary to ensure that the correct fields and only those are replaced.

With these steps, one might get the impression that subtree rewrite rules suffice, but in order to simulate a Turing machine with unbounded tape to the left, flat prefix rewriting is required to append another blank symbol, which is realized with auxiliary states q^+ and q^{++}. Note that according to the definition of deterministic Turing machines (cf. Chapter 1 – Preliminaries), a one side infinite tape suffices.

PROOF. Formally, the subtree and flat prefix rewriting system $\mathcal{R}_M \in \mathfrak{R}^{sfp}$ with $\mathcal{R}_M = (\Sigma^M, \Gamma^M, R^M, t_{in}^M)$ for a DTM $M = (Q, \Sigma, \Gamma, q_{in}, q_h, \delta)$ consists of the following components.

- $\Sigma^M := \{\bullet\} \dot\cup Q \dot\cup \underbrace{\Sigma \dot\cup \{\sqcup\}}_{=\Gamma} \dot\cup \{p_b^L \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,L)\big)\}$
  $\dot\cup \{p_b^R \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,R)\big)\}$
  $\dot\cup \{b^* \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,L)$
  $\vee\, \delta(q,a) = (p,b,R)\big)\}$
  $\dot\cup \{q_a^+ \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,L)\big)\}$
  $\dot\cup \{q_a^{++} \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,L)\big)\}$
- Γ^M: a transition alphabet is not needed, consequently, set $\Gamma^M := \{\sigma\}$.
- The rules of R^M are the following:
  1. For a left transition $\delta(q,a) = (p,b,L)$ with $q, p \in Q, a, b \in \Gamma$:

     (1.1) $\forall c \in \Gamma\colon c \overset{\sigma}{\hookrightarrow} p_c^L$ (1.2) $\begin{smallmatrix} q \\ | \\ a \end{smallmatrix} \overset{\sigma}{\hookrightarrow} b^*$

     (1.3) $\forall c \in \Gamma\colon p_c^L \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} p \\ | \\ c \end{smallmatrix}$ (1.4) $b^* \overset{\sigma}{\hookrightarrow} b$
  2. For a right transition $\delta(q,d) = (p,e,R)$ with $q, p \in Q, d, e \in \Gamma$:

     (2.1) $\forall c \in \Gamma\colon c \overset{\sigma}{\hookrightarrow} p_c^R$ (2.2) $\begin{smallmatrix} q \\ | \\ d \end{smallmatrix} \overset{\sigma}{\hookrightarrow} e^*$

     (2.3) $\forall c \in \Gamma\colon p_c^R \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} p \\ | \\ c \end{smallmatrix}$ (2.4) $e^* \overset{\sigma}{\hookrightarrow} e$
  3. In order to handle the left margin, and append a $\sqcup$ when needed (this is only the case for a left transition $\delta(q,a) = (p,b,L)$ with $q, p \in Q, a, b \in \Gamma$), the tree needs to be reduced to height 1 before a flat prefix rewrite rule can be applied:

     (3.1) $\begin{smallmatrix} q \\ | \\ a \end{smallmatrix} \overset{\sigma}{\hookrightarrow} q_a^+$ (3.2) flat prefix rule: $q_a^+ \overset{\sigma}{\hookrightarrow} \sqcup q_a^{++}$

     (3.3) $q_a^{++} \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} q \\ | \\ a \end{smallmatrix}$
- t_{in} codes the initial configuration $\kappa_{in} = q_{in}\sqcup$, so $t_{in} = t(\kappa_{in}) = \begin{smallmatrix} \bullet \\ | \\ q_{in} \\ | \\ \sqcup \end{smallmatrix}$.

The subtree and flat prefix rewriting system $\mathcal{R}_M$ produces trees that may not appear in the simulation of the Turing machine M. Therefore, a D↓TA $\mathcal{A}$ needs to ensure the following restrictions:

1. A tree is of height 1 or 2; there may be at most one state $q \in Q$, and if there is, it occurs in a subtree of the form $\begin{smallmatrix} q \\ | \\ a \end{smallmatrix}$ with $a \in \Gamma$.
2. There may be at most one node with label a^* for $a \in \Gamma$.

3. There has to be at least one of the following:

$$\begin{matrix} q \\ | \\ a \end{matrix} \quad \text{OR} \quad q_a^L \quad \text{OR} \quad q_a^R \quad \text{OR} \quad q_a^+ \quad \text{OR} \quad q_a^{++} \quad \text{for some } q \in Q, a \in \Gamma \ .$$

4. For $p, q, r \in Q$ and $a, b, c \in \Gamma$,

   (a) there may be at most one auxiliary node:
   $q_a^L \quad \text{XOR} \quad p_b^R \quad \text{XOR} \quad r_c^+ \ .$

   (b) there must not be 3 "special" nodes at once:

$$\text{NOT} \left(q_a^{L/R} \quad \text{AND} \quad \begin{matrix} p \\ | \\ b \end{matrix} \quad \text{AND} \quad c^* \right) \ .$$

5. For $p, q \in Q$ and $a, b, c \in \Gamma$,

   (a) p_a^L may only appear left of $\begin{matrix} q \\ | \\ b \end{matrix}$ or c^* with M-transition $\delta(q, b) = (p, c, L)$.

   (b) p_a^R may only appear right of $\begin{matrix} q \\ | \\ b \end{matrix}$ or c^* with M-transition $\delta(q, b) = (p, c, R)$.

6. For $q \in Q$ and $a \in \Gamma$, q_a^+ may only occur if there is no b^* for any $b \in \Gamma$, and q_a^+ can only appear at the leftmost node.

The above mentioned restrictions only require properties of

- the height of the tree and of its subtrees,
- the absence or presence of certain combinations of node labels, and
- the order of the label sequence of the successors of the root.

It is not difficult to construct a D↓TA $\mathcal{A}$ that is able to scan the label sequence of the successors before assigning states to the successors, which ensures these restrictions.

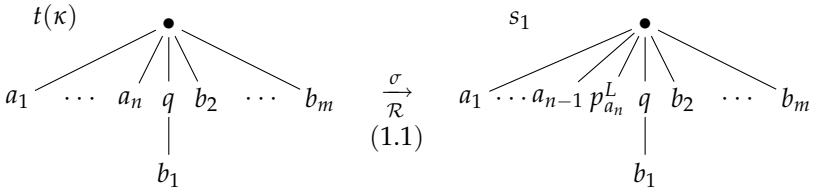
For the correctness of the construction described above, it needs to be shown that for all configurations κ, κ' of M there is a path from $t(\kappa)$ to $t(\kappa')$ in $G_M|_{T(\mathcal{A})}$ iff $\kappa \vdash^* \kappa'$.

For the one direction, a correct derivation from $t(\kappa)$ to $t(\kappa')$ for a left transition is sketched in Figure 4.2. For a right transition, rules (2.1), (2.2), (2.3), and (2.4) are applied for a similar derivation. It is not difficult to see that applying rules (3.1), (3.2), and (3.3) appends a blank symbol to the left of the tape before the Turing machine can go further to the left.

For the other direction, let π be the path from $t(\kappa)$ to $t(\kappa')$ in $G_M|_{T(\mathcal{A})}$. The claim will be shown by induction on the length k of π.
Case $k = 0$. Then it holds that $\kappa = \kappa'$, and consequently $\kappa \vdash^* \kappa'$.
Case $k \to k + 1$. Assume that the claim holds for a path of length k, and let $\kappa = a_1 \cdots a_n q b_1 \cdots b_m$.

It will be shown that the sequence of rewrite rules that are used on π for $\delta(q, b_1) = (p, d, L)$ has to follow the scheme of Figure 4.2 for $n > 0$. If $n = 0$, i.e. q is the leftmost symbol, it will be shown that a blank symbol has to be inserted with rules (3.1), (3.2), and (3.3) before the rewriting system can proceed as in the first case. For $\delta(q, b_1) = (p, d, R)$, the proof proceeds analogously.

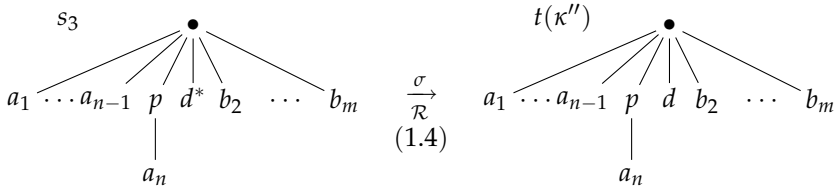
First note that since M is deterministic, there is no right rule $\delta(q, b_1) = (r, e, R)$ for any $r \in Q$ and $e \in \Gamma$, and thus no rule of type (2.2) for $q(b_1)$. For $n > 0$, i.e. state q is not at the left margin, applying rule (3.1) violates restriction 6, and the resulting tree would not be in $T(\mathcal{A})$ and thus not in $G_M|_{T(\mathcal{A})}$. Applying rule (1.2) would result in a tree with no control state and thus violate restriction 3. The only possibilities left are rules (1.1) and (2.1). Since there is no d^* in $t(\kappa)$, the only action allowed is to apply rule (1.1) at a_n:



In the next step, rules (1.1), (2.1), and (3.1), each of which adds another auxiliary state, are prohibited by restriction 4.(a). Adding a second subtree of height 1 by rule (1.3) would violate restriction 1, and thus only rule (1.2) remains, leading to a tree s_2 of height 1 with $\mathit{front}(s_2) = \mathit{flatfront}(s_2) = a_1 \cdots a_{n-1} p^L_{a_n} d^* b_2 \cdots b_m$.

After these two main steps, the tree s_2 needs to be rewritten to a tree that codes a valid Turing machine configuration. Again, restriction 4.(a) prevents adding a second auxiliary state, and rules (1.1), (2.1), and (3.1) cannot be applied. Rules (1.4) respectively (2.4) would result in $p^L_{a_n}$ appearing next to d, which is prohibited by restriction 5.(a). Rule (1.3) remains, which "unfolds" the auxiliary state $p^L_{a_n}$ to $p(a_n)$. Let this tree be s_3.

The only possibility to go on now is to apply rule (1.4) (respectively (2.4)), since there may not be three special nodes (restriction 4.(b) forbids rules (1.1) and (2.1)), both rules (1.2) and (2.2) are excluded because of restrictions 2 and 3, and rule (3.1) cannot be applied because there is d^* (restriction 6). Altogether, $t(\kappa'')$ is reached:



where κ'' is the successor configuration of κ: $\kappa \vdash \kappa''$. By induction hypothesis it holds that $\kappa'' \vdash^* \kappa'$ and therefore $\kappa \vdash^* \kappa'$.

For $n = 0$ state q is at the left margin: $\kappa = q b_1 \cdots b_m$, and rules (1.1) and (2.1) cannot be applied because of restrictions 5.(a) respectively (b). Rule (1.2) would eliminate the only control state and therefore violate restriction 3. Thus,

only rule (3.1) is applicable and results in a tree r_1 of height 1 with $\mathit{flatfront}(r_1) = q^+_{b_1} b_2 \cdots b_m$. With restriction 6 the only rule that can be applied next is flat prefix rewrite rule (3.2), resulting in a tree r_2 with $\mathit{flatfront}(r_2) = \sqcup q^{++}_{b_1} b_2 \cdots b_m$. Again due to restrictions 5.(a) and (b), rules (1.1) and (2.1) are blocked, and the only applicable rule left is (3.3) resulting in $t(\kappa^*) = \bullet(\sqcup, q(b_1), b_2, \ldots, b_m)$ with $\mathit{red}(\kappa^*) = \kappa$. Now, the situation is the same as described above for a left rule $\delta(q, b_1) = (p, d, L)$ and one can proceed accordingly.

In order to show that the reachability problem for $G_M|_{T(\mathcal{A})} = (T(\mathcal{A}), \to_{\mathcal{R}})$ is undecidable, an initial tree and a regular target set are required. The initial tree t_{in} is set to the tree encoding of the initial configuration κ_{in}, so $t_{in} := t(\kappa_{in})$, and the target set T^M_h is fixed to code the halting configurations of M:

$$\begin{aligned} T^M_h &:= \{t(\kappa) \mid \kappa \text{ is a halting configuration of } M\} \\ &= \{t(\kappa) \mid \kappa \text{ contains the unique halting state } q_h\}, \end{aligned}$$

which is easily identified to be regular.

To specify the reduction that M reaches a halting configuration when started on the empty tape iff there is a maximal path in $G_M|_{T(\mathcal{A})}$ starting in t_{in} that visits T^M_h, consider both possibilities:

- First, assume that M eventually reaches a halting configuration when started on the empty tape. This means that there has to be a path through $G_M|_{T(\mathcal{A})}$ starting in $t_{in} = t(\kappa_{in})$ that eventually reaches T^M_h.
- Second, assume that M never reaches a halting configuration when started on the empty tape. Then all infinite paths through $G_M|_{T(\mathcal{A})}$ starting in $t_{in} = t(\kappa_{in})$ never reach T^M_h. Thus, there is no path through $G_M|_{T(\mathcal{A})}$ starting in t_{in} that eventually reaches T^M_h.

Therefore it can be concluded that M reaches a halting configuration when started on the empty tape if and only if every maximal path in $G_M|_{T(\mathcal{A})}$ starting in t_{in} eventually reaches T^M_h. Since the halting problem for deterministic Turing machines is undecidable, the proof is concluded. ■

As every subtree and flat prefix rewriting system is a special case of a regular subtree and flat prefix rewriting system, it follows directly that the reachability problem in $G_{\mathcal{R}}|_{T(\mathcal{A})}$ for a regular subtree and flat prefix rewriting system $\mathcal{R}$ and a D↓TA $\mathcal{A}$ is also undecidable.

Furthermore, this problem remains undecidable for partial subtree rewriting systems:

Theorem 4.2. *For a D↓TA $\mathcal{A}$ and a partial subtree rewriting system $\mathcal{R} \in \mathfrak{R}^{ps}$ the reachability problem in the transition graph $G_{\mathcal{R}}|_{T(\mathcal{A})} = (T(\mathcal{A}), \xrightarrow[\mathcal{R}]{})$ is undecidable.*

PROOF. The proof proceeds completely analogous to the proof for subtree and flat prefix rewriting systems shown above. With the same encoding of configurations, the derivations of left and right rules also proceed completely analogous. To append a $\sqcup$ symbol to the left, one rule suffices.

A partial subtree rewriting system $\mathcal{R} = (\Sigma^M, \Gamma^M, R^M, t_{in}^M)$ is constructed for a DTM $M = (Q, \Sigma, \Gamma, q_{in}, q_h, \delta)$, with

$$\begin{aligned}\Sigma^M := \{\bullet\} \dot\cup Q \dot\cup \underbrace{\Sigma \dot\cup \{\sqcup\}}_{=\Gamma} \dot\cup \quad & \{p_b^L \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,L)\big)\} \\ \dot\cup \quad & \{p_b^R \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,R)\big)\} \\ \dot\cup \quad & \{b^* \mid \exists\, q, p \in Q, \exists\, a, b \in \Gamma\big(\delta(q,a) = (p,b,L) \\ & \qquad\qquad \vee\, \delta(q,a) = (p,b,R)\big)\},\end{aligned}$$

Γ^M, and t_{in}^M as above, and the following set R^M of rules over trees of $T_{\Sigma,\xi}$:

1. For a left transition $\delta(q,a) = (p,b,L)$ with $q, p \in Q, a, b \in \Gamma$:

   (1.1) $\forall c \in \Gamma$: $\begin{array}{c} c \\ | \\ \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{c} p_c^L \\ | \\ \xi \end{array}$ (1.2) $\begin{array}{ccc} & q & \\ / & & \backslash \\ a & & \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{c} b^* \\ | \\ \xi \end{array}$

   (1.3) $\forall c \in \Gamma$: $\begin{array}{c} p_c^L \\ | \\ \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{ccc} & p & \\ / & & \backslash \\ c & & \xi \end{array}$ (1.4) $\begin{array}{c} b^* \\ | \\ \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{c} b \\ | \\ \xi \end{array}$

2. For a right transition $\delta(q,d) = (p,e,R)$ with $q, p \in Q, d, e \in \Gamma$:

   (2.1) $\forall c \in \Gamma$: $\begin{array}{c} c \\ | \\ \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{c} p_c^R \\ | \\ \xi \end{array}$ (2.2) $\begin{array}{ccc} & q & \\ / & & \backslash \\ d & & \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{c} e^* \\ | \\ \xi \end{array}$

   (2.3) $\forall c \in \Gamma$: $\begin{array}{c} p_c^R \\ | \\ \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{ccc} & p & \\ / & & \backslash \\ c & & \xi \end{array}$ (2.4) $\begin{array}{c} e^* \\ | \\ \xi \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{c} e \\ | \\ \xi \end{array}$

3. In order to handle the left margin, and append a $\sqcup$ when needed (this is only the case for a left transition $\delta(q,a) = (p,b,L)$ with $q, p \in Q, a, b \in \Gamma$):

   (3) $\begin{array}{ccc} & \bullet & \\ / & & \backslash \\ q & & \xi \\ | & & \\ a & & \end{array} \overset{\sigma}{\hookrightarrow} \begin{array}{cccc} & & \bullet & \\ & / & | & \backslash \\ \sqcup & & q & \xi \\ & & | & \\ & & a & \end{array}$

The D↓TA $\mathcal{A}$ is inherited from the construction of the subtree and flat prefix rewriting system above. As the auxiliary states q_a^+ and q_a^{++} are not needed, restriction 6 can be omitted, and restrictions 3 and 4.(a) need to be adapted in a straightforward way. Note that instead of rule (3), the rule $\bullet(q(a), \xi) \overset{\sigma}{\hookrightarrow} \bullet(p(\sqcup), b, \xi)$ could be introduced, that realizes the left transition $\delta(q,a) = (p,b,L)$ at the left margin in a single step. In order to reuse the results from the previous proof, the format is kept.

As one can easily see, the trees over T_Σ of the subtree and flat prefix rewriting system $\mathcal{R}$ constructed above are simply extended by variable ξ for rules (1.1) through (2.4). For these rules, ξ can only be substituted by the empty hedge for left or right transitions of the Turing machine due to the structure of the encoded configurations. As for control state q being at the left margin, $\kappa = qb_1 \dots b_m$, again, rules (1.1) and (2.1) are excluded by restrictions 5.(a) respectively (b), as well as rule (1.2) being prevented by restriction 3. Consequently, the only rule that can be applied is rule (3), substituting the rest of the tape inscription (other than the current working cell inscription b_1) as hedge for variable ξ, and appending a $\sqcup$ symbol to the left side of the tape. This results in a tree coding the configuration

$\kappa^* = \sqcup q b_1 \dots b_m$ with $red(\kappa^*) = \kappa$. Thereupon, one can proceed as usual with the left transition of the Turing machine. ■

It is interesting to note that the decidability of the reachability problem over $G_{\mathcal{R}}|_{T(\mathcal{A})}$ for GTRG $G_{\mathcal{R}}$ (cf. [Col02] and Chapter 1 – Preliminaries) fails for partial subtree rewriting systems, even though it was shown that the classes PSRG of transition graphs of partial subtree rewriting systems and GTRG of transition graphs of ground tree rewriting systems coincide. This emphasizes that the concept of a D↓TA which captures a complete successor sequence of a node in an unranked tree cannot be transferred to any kind of encoding as ranked trees, since due to the encoding, hierarchy levels are altered and former siblings of the unranked tree are encoded on different levels.

Chapter 5

Conclusion

In the course of this thesis it was shown that using rewriting systems over unranked trees one can generate a class of infinite graphs that coincides with the class of transition graphs of ground tree rewriting systems over ranked trees. The rewriting principle of these partial subtree rewriting systems consists of substituting unranked trees partially, which corresponds to ground tree rewriting over an encoding of unranked trees as ranked ones. Due to the class equivalence, several decidability results for ground tree rewriting systems over ranked trees can be transferred to partial subtree rewriting systems. On the other hand, it was shown in Chapter 4 – Undecidability Results that when taking the inner structure of the trees into account (by restricting the vertex set of the transition graphs to the language of a deterministic top down tree automaton), an encoding of unranked trees as ranked ones fails to capture the complete structural information.

Furthermore, (regular) subtree and flat prefix rewriting systems over unranked trees were introduced, which add flat prefix rewriting to the known paradigm of subtree substitution. The regular variant of these rewriting systems allows the employment of regular sets of trees on each side of the rewrite rules, similar to regular ground tree rewriting systems or prefix recognizable systems (for pushdown systems). The class of transition graphs of subtree and flat prefix rewriting systems was shown to strictly include the class of transition graphs of partial subtree rewriting systems, which allows to transfer several undecidability results. Additionally, an algorithm was introduced which constructs an automaton that recognizes the set of trees from which a given set of trees can be reached. This algorithm was obtained from a refinement of the well known saturation algorithm, where refining is necessary for capturing the entire set. Thus, the reachability problem for both classes of transition graphs is decidable. Subsequently, it was shown that when restricting transition graphs by a deterministic top down tree automaton, the reachability problem of both rewriting systems can be reduced to the halting problem for deterministic Turing machines, and is thus undecidable.

Further Research

An open problem directly connected to the undecidability of reachability over the restricted transition graph of any of the introduced rewriting systems is to employ the weaker model of deterministic top down tree automata that was introduced in Chapter 1 – Preliminaries. This model is not able to scan the successor sequence before assigning the states to the successors of a node, and thus, cannot verify the demanded restrictions of Chapter 4 – Undecidability Results.

The class of (regular) subtree and flat prefix rewriting systems deserves further studies. Though it was shown that the monadic second order logic is undecidable for (regular) subtree and flat prefix rewriting systems, the first order theory remains to be investigated. Furthermore, the decidability of other reachability problems, such as the recurrence problem, is unsettled.

Generally, other rewriting principles over unranked trees have yet to be investigated. One aspect could be to use other word rewriting techniques than prefix rewriting in combination with subtree substitution. Another interesting point of application is to investigate (semi) monadic rewriting techniques over unranked trees, which were introduced for ranked trees in [GV98].

Bibliography

[BG00] Achim Blumensath and Erich Grädel. Automatic structures. In *Proceedings of the 15th IEEE Symposium on Logic in Computer Science, LICS '00*, pages 51–62. IEEE Computer Science Press, 2000. 2

[BMW01] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1. Unfinished technical report, Hongkong University of Science and Technology, April 2001. `http://citeseer.ist.psu.edu/451005.html`. 4, 21

[Bra69] Walter S. Brainerd. Tree generating regular systems. *Information and Control*, 14(2):217–231, 1969. 2, 17, 24, 33

[BT03] Ahmed Bouajjani and Tayssir Touili. Reachability analysis of process rewrite systems. In *Proceedings of the 23rd International Conference on Foundations in Software Technology and Theoretical Computer Science, FSTTCS 2003*, volume 2914 of *Lecture Notes in Computer Science*. Springer, 2003. 3

[Büc64] Julius R. Büchi. Regular canonical systems. *Archiv für Mathematische Grundlagenforschung*, 6:91–111, 1964. 2

[BVW94] Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. In *Proceedings of the 6th International Conference on Computer Aided Verification, CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 1994. 1

[Cau96] Didier Caucal. On infinite transition graphs having a decidable monadic theory. In *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming, ICALP'96*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1996. 2, 5

[Cau02] Didier Caucal. On infinite terms having a decidable theory. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002. 2

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. Unpublished electronic book, 1997. `http://www.grappa.univ-lille3.fr/tata`. 2, 5, 7, 19, 24, 33

[CDGV94] Jean-Luc Coquidé, Max Dauchet, Rémi Gilleron, and Sándor Vágvölgyi. Bottom-up tree pushdown automata: classification and connection with rewrite systems. *Theoretical Computer Science*, 127(1):69–98, 1994. 24, 33, 40

[CE81] Edmund M. Clarke and E. Allen Emerson. The design and synthesis of synchronization skeletons using temporal logic. In *Workshop on Logics of Programs, Yorktown Heights, New York*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. 1

[CG90] Jean-Luc Coquidé and Rémi Gilleron. Proofs and reachability problem for ground rewrite systems. In *Aspects and Prospects of Theoretical Computer Science*, volume 464 of *Lecture Notes in Computer Science*, pages 120–129. Springer, 1990. 3

[CLT05] Julien Cristau, Christof Löding, and Wolfgang Thomas. Deterministic automata on unranked trees. In *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory, FCT 2005*, volume 3623 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2005. 19, 21, 22

[CNT04] Julien Carme, Joachim Nieren, and Marc Tommasi. Querying unranked trees with stepwise tree automata. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications, RTA 2004*, volume 3091 of *Lecture Notes in Computer Science*, pages 105–118. Springer, 2004. 5, 13, 15, 16

[Col02] Thomas Colcombet. On families of graphs having a decidable first order theory with reachability. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, ICALP 2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2002. 3, 4, 24, 34, 59, 66

[Cou78] Bruno Courcelle. A representation of trees by languages. *Theoretical Computer Science*, 7:25–55, 1978. 5, 10

[Cou89] Bruno Courcelle. On recognizable sets and tree automata. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, pages 93–126. Academic Press, 1989. 5, 10

[DHLT90] Max Dauchet, Thierry Heuillard, Pierre Lescanne, and Sophie Tison. Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. *Information and Computation*, 88(2):187–201, 1990. 3

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier Science Publishers, 1990. 1, 2

[DT90] Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990*, pages 242–248. IEEE Computer Society Press, 1990. 3, 24, 33

[EHRS00] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer*

Aided Verification, CAV 2000, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000. 2, 33

[Eme81] E. Allen Emerson. *Branching Time Temporal Logics and the Design of Correct Concurrent Programs*. PhD thesis, Division of Applied Sciences, Harvard University, 1981. 1

[Eme90] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990. 2, 3, 23

[Eme96] E. Allen Emerson. Automated temporal reasoning about reactive systems. In *Proceedings of the 8th Banff Higher Order Workshop: Logics for Concurrency—Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 41–101, 1996. 1, 23

[Gai82] Haim Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium: Logic Colloquium '81*, pages 105–135, North Holland, 1982. 33

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984. 2, 19

[GT95] Rémi Gilleron and Sophie Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995. 2

[GV98] Pál Gyenizse and Sándor Vágvölgyi. Linear generalized semi-monadic rewrite systems effectively preserve recognizability. *Theoretical Computer Science*, 194(1–2):87–122, 1998. 68

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, 2 edition, 2001. 18, 22, 49

[Löd02a] Christof Löding. Ground tree rewriting graphs of bounded tree width. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science, STACS 2002*, volume 2285 of *Lecture Notes in Computer Science*, pages 559–570. Springer, 2002. 3

[Löd02b] Christof Löding. Model-checking infinite systems generated by ground tree rewriting. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2002. 3, 24, 33

[Löd03] Christof Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, Germany, 2003. 3, 5, 17, 24, 33, 40

[May00] Richard Mayr. Process rewriting systems. *Information and Computation*, 156(1–2):264–286, 2000. 3

[Mor99] Christophe Morvan. On rational graphs. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 1999*, volume 1784 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 1999. 2

[MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985. 1, 2

[Nev02a] Frank Neven. Automata, logic, and XML. In *Computer Science Logic, 16th International Workshop, CSL 2002*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002. 4

[Nev02b] Frank Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002. 4, 5, 13

[PQ68] Claude Pair and Alain Quéré. Définition et étude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968. 4

[Rab77] Michael O. Rabin. Decidable theories. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 595–629. Elsevier Science Publishers, 1977. 2

[See72] Detlef G. Seese. Entscheidbarkeits- und Definierbarkeitsfragen der Theorie „netzartiger" Graphen-I. *Wissenschaftliche Zeitschrift der Humboldt-Universität zu Berlin, Math.-Natur. Reihe*, XXI(5):513–517, 1972. 18, 40

[Suc02] Dan Suciu. Typechecking for semistructured data. In *Database Programming Languages, 8th International Workshop, DBPL 2001*, volume 2397 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2002. 5

[Tak75] Masako Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, 1975. 4

[Tho90] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–192. Elsevier Science Publishers, 1990. 2

[Tho02] Wolfgang Thomas. A short introduction to infinite automata. In *Proceedings of the 5th International Conference on Developments in Language Theory, DLT 2001*, volume 2295 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2002. 1