Infinite Graphs Generated By Tree Rewriting

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Christof Löding
aus Neumünster, Schleswig-Holstein

Berichter: Universitätsprofessor Dr. Wolfgang Thomas
Universitätsprofessor Dr. Erich Grädel

Tag der mündlichen Prüfung: 19. Dezember 2002

Abstract

Finite graphs and algorithms on finite graphs are an important tool for the verification of finite-state systems. To transfer the methods for finite systems, at least partially, to infinite systems a theory of infinite graphs with finite representations is needed. In this thesis the class of the transition graphs of ground tree rewriting systems is studied. To investigate the structure of ground tree rewriting graphs they are analyzed under the aspect of tree-width of graphs and are compared to already well-studied classes of graphs, as the class of pushdown graphs and the class of automatic graphs. Furthermore, the trace languages that are definable by ground tree rewriting graphs are investigated.

The algorithmic properties of ground tree rewriting graphs are studied by means of reachability problems that correspond to the semantics of basic temporal operators. The decidability results from this analysis are used to build up a temporal logic such that the model-checking problem for this logic and ground tree rewriting graphs is decidable.

Zusammenfassung

Endliche Graphen und Graphalgorithmen haben vielfältige Anwendungen in der Informatik, unter anderem in der Verifikation endlicher, zustandsbasierter Systeme. Um die Methoden und Ergebnisse für endliche Systeme zumindest teilweise auf unendliche Systeme zu übertragen, wird eine Theorie unendlicher Graphen mit endlicher Darstellung benötigt. In dieser Arbeit wird die Klasse der Transitionsgraphen von Grundtermersetzungssystemen behandelt. Um die Struktur von Grundtermersetzungsgraphen zu analysieren, werden diese unter dem Aspekt der Baumweite von Graphen untersucht und mit bereits intensiv studierten Graphklassen wie der Klasse der Pushdowngraphen und der Klasse der automatischen Graphen verglichen. Weiterhin werden die durch Grundtermersetzungsgraphen definierbaren Pfadsprachen studiert.

Die algorithmischen Eigenschaften von Grundtermersetzungsgraphen werden anhand von Erreichbarkeitsproblemen, die der Semantik grundlegender temporaler Operatoren entsprechen, untersucht. Die Entscheidbarkeitsergebnisse aus dieser Analyse werden benutzt, um eine temporale Logik aufzubauen, für die das Model-Checking-Problem mit Grundtermersetzungsgraphen entscheidbar ist.

Contents

Chapter 1

Introduction

Finite graphs constitute one of the most basic data structures used in computer science. In the 1960s and 1970s many efficient algorithms for the analysis of finite graphs were developed. Later, finite graphs advanced from a pure data structure to an important tool for modeling and verifying finite state-based systems. In the approach of model-checking [Eme81, CE81] algorithms for the automatic verification of systems modeled by finite graphs with properties written in certain specification logics are developed. In this framework temporal logics like LTL, CTL, and CTL* (cf. [Eme90]) are among the most popular specification logics. In the last 20 years many contributions have been made to this area, and by now the theory of model-checking finite systems with temporal specifications is well understood (cf. [BVW94, Var96, Eme96]).

With the motivation to transfer these methods, at least partially, to infinite systems the development of a new theory of infinite graphs started. Infinite systems arise from the use of potentially unbounded data structures like stacks, counters, or queues. Parametrized systems, which correspond to infinite families of systems consisting of several components, the number of which is determined by the parameter, are another example for infinite systems. To represent systems of this kind infinite graphs are necessary.

For an algorithmic theory of infinite graphs, classes of infinite graphs with finite representations are needed. Given a formalism for finite representations of graphs from a class $\mathcal{K}$, two natural questions arise:

(i) Which decision problems can be solved for the class $\mathcal{K}$ of graphs generated by this formalism?

(ii) How expressive is the given formalism, i.e., what kind of graphs are in $\mathcal{K}$, and how is $\mathcal{K}$ related to other classes of infinite graphs?

An example for a problem of type (i) is the reachability problem:

Given a finite representation of a graph G from $\mathcal{K}$ and two vertices u, v of G, is there a path from u to v?

In automatic verification this problem has to be solved to check whether a system can reach an undesirable state. In contrast to finite graphs, it is not clear a priori whether this problem is decidable for a given class of infinite graphs. It is easy to define classes of infinite graphs allowing to use reachability problems to encode undecidable problems like the halting problem for Turing machines. 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 aim of this thesis is to give a comprehensive analysis of a specific class of infinite graphs, namely the transition graphs of ground tree rewriting systems[1]. The vertices of these graphs are represented by ranked trees (or terms), and the edges are generated by replacements at the front of the trees according to a fixed set of rules. We study the structure of ground tree rewriting graphs (GTR graphs) and their relation to other graph classes, in particular to pushdown graphs. Furthermore, we analyze several variants of the above mentioned reachability problem. As some these problems turn out to be decidable in polynomial time, this analysis shows that GTR graphs constitute, from an algorithmic point of view, a usable class of infinite graphs.

Before we give a more detailed explanation of the topics covered in this thesis we review several classes of graphs that have been considered in the literature.

Classes of Infinite Graphs

Pushdown Graphs. In their seminal paper [MS85] Muller and Schupp analyzed the transition graphs of pushdown automata. The main components of a pushdown automaton are a finite set Q of control states, a finite stack alphabet Γ, and a set Δ of transition rules. The transition rules of the pushdown automaton specify, depending on the current control state and top of the stack, which control state the automaton is in after the next step and how the top of the stack is manipulated in this step. So, a configuration of a pushdown automaton can be described by a word of the form $q\gamma_1 \cdots \gamma_n$, where q is a control state and $\gamma_1, \ldots, \gamma_n$ are letters from the stack alphabet. Words of that kind are the vertices of a pushdown graph. The set of edges

[1] Usually these systems are called ground term rewriting systems.

is described by the transition rules from Δ. If a rule exists that says "from control state q with γ_1 on top of the stack move to control state p and replace γ_1 by γ", then there is an edge from $q\gamma_1 \cdots \gamma_n$ to $p\gamma\gamma_2 \cdots \gamma_n$.

Muller and Schupp [MS85] proved that the monadic second-order (MSO) theory of pushdown graphs is decidable. Since MSO logic subsumes temporal logic their result implies that model-checking for pushdown graphs and temporal logics is decidable. Previous results of Büchi [Büc64] on regular canonical systems already imply that the reachability problem for pushdown graphs is decidable. In [EHRS00] more efficient algorithms for solving reachability problems on pushdown graphs and for model-checking pushdown graphs with temporal logics have been developed. In [Wal96, KV00, Cac02a] algorithms for solving games on pushdown graphs with various winning conditions have been presented. The theory of non-terminating games on graphs is an important aspect for the verification of reactive systems and has been widely studied for finite graphs (cf. [Tho95, Zie98]).

Besides the proof of the decidability of the MSO theory Muller and Schupp provide in the same paper [MS85] an analysis of the structure of pushdown graphs. They give a characterization of pushdown graphs that is independent of the representation by pushdown automata. The result states that a rooted graph of finite degree is a pushdown graph if, and only if, it has only finitely many types of ends. Ends are connected components of the graph obtained by deleting all vertices within a fixed diameter around the designated root vertex. This characterization allows to decide whether a graph is a pushdown graph without working with the explicit representations by pushdown automata.

As noticed in [Cau92b] the operations of a pushdown automaton can be seen as prefix rewriting on words. So, instead of using pushdown automata to generate the graphs one can equivalently use prefix rewriting systems on words. It is shown in [Cau92b] that these prefix rewriting graphs can also be characterized by deterministic graph grammars.

Recapitulating, one can say that pushdown graphs are a well investigated class of infinite graphs with good algorithmic properties. But the various different characterizations of pushdown graphs also show the limited expressiveness of this formalism for the definition of infinite graphs.

Prefix Recognizable Graphs. The equivalence of pushdown graphs to the graphs generated by prefix rewriting systems on words leads to a natural extension of pushdown graphs. Instead of using prefix rewriting rules of the form "a prefix u can be replaced by a prefix v" one can use regular languages in these rules. This defines the class of prefix recognizable graphs [Cau96].

The edge relation of a prefix recognizable graph is of the form $(L_1 \times L_2) \cdot K$, where L_1, L_2, and K are regular word languages.

The algorithmic properties of prefix recognizable graphs are the same as for pushdown graphs: the MSO theory of prefix recognizable graphs is decidable [Cau96], and the methods for solving games on pushdown graphs also work for prefix recognizable graphs [KV00, Cac02b].

From the definition of prefix recognizable graphs it is obvious that they extend the class of pushdown graphs. A class of graphs strictly in between pushdown graphs and prefix recognizable graphs is the class of equational graphs [Cou89]. These graphs are generated by graph grammars with hyperedge replacement. The relation of prefix recognizable graphs and equational graphs was characterized by Barthelmann in [Bar98] using the notion tree-width (cf. [Die00]), and in [CK01] yielding a construction method of hyperedge replacement grammars for prefix recognizable graphs that are equational. A nice overview including several characterizations of prefix recognizable graphs is given in [Blu01].

Automatic and Rational Graphs. A powerful mechanism for defining infinite graphs uses finite two-head automata for the specification of edge relations of graphs. These automata read pairs of words with two heads moving either synchronously or asynchronously over the two words. There is an edge between the two words if the pair of these two words is accepted by the automaton. Graphs whose edge relation can be defined with synchronous automata are called automatic (cf. [BG00]) or synchronized rational (cf. [FS93]). Rational graphs [Mor99] are those whose edge relation can be defined by asynchronous automata. Thus, the class of rational graphs contains the class of automatic graphs.

It is rather easy to see that transition graphs of Turing machines are automatic and therefore the reachability problem cannot be decidable for automatic (and rational) graphs. But the first-order (FO) theory of automatic graphs is decidable (cf. [BG00]), even for FO logic extended by a quantifier "there exist infinitely many". In contrast, the FO theory of rational graphs is undecidable ([Mor99]). There even exists a single rational graph with an undecidable FO theory ([Tho02]), proving that the class of rational graphs strictly contains the class of automatic graphs.

From an algorithmic point of view these two classes of graphs are too strong because they do not admit algorithms for solving basic problems.

Process Rewriting Graphs. The graph classes we presented so far have in common that their vertices are coded by words. In process rewriting graphs [May98, May00, May01, BCMS01] the vertices are represented by

abstract processes, where such an abstract process is a term built up from process constants and operations for sequential and parallel composition. The edges are described by a finite set of rewriting rules. Depending on which operations are allowed on each side of the rewriting rules, one gets different classes of graphs forming the hierarchy of process rewriting graphs. For example, if we allow sequential composition on both sides of the rewriting rules but disallow parallel composition, then we have an ordinary prefix rewriting system over words. Without sequential, but with parallel composition, one obtains the transition graphs of Petri nets. Two other widely studied classes from this hierarchy are basic parallel processes (BPP) and the process algebra PA.

In [May00] it is shown that the hierarchy of process rewriting graphs is strict w.r.t. bisimulation and that reachability is decidable for all graph classes within the process rewriting hierarchy. The model-checking problem for process rewriting graphs and several temporal logics is analyzed in [May01]. There are a lot of articles on the classes BPP and PA. Model-checking for BPP with branching time logics is analyzed in [EK95], the reachability problem for BPP is shown to be solvable in linear time in [EP00], and model-checking for PA with various transition logics is considered in [LS00], just to mention a few.

Ground Tree Rewriting Graphs. We now give more details about ground tree rewriting systems and the graphs generated by these systems, followed by an overview of the results of this thesis.

The vertices in ground tree rewriting graphs are represented by finite ranked trees. A ranked tree is a finite ordered tree, i.e., the successors of a location[2] are ordered, and its locations are labeled with symbols from a finite alphabet A. Each of these symbols comes with an arity determining the number of successors of the vertices labeled with this symbol. A ground tree rewriting system (GTRS) consists of such an alphabet A, an alphabet Σ for the edge labels of the graph, a finite set of rules of the form $s \overset{\sigma}{\hookrightarrow} s'$ for ranked trees s, s', $\sigma \in \Sigma$, and an initial tree t_{in}. To apply a rule to a tree t one has to find the tree from the left hand side of the rule as a subtree in t and then replace it by the tree from the right hand side of the rule. In this way, the rewriting rules define the Σ-labeled edges of the graph. This kind of rewriting is called suffix rewriting in [Cau92a]. The vertices of the graph are all the trees that are reachable from the initial tree by repeated application of the rewriting rules. Thus, a ground tree rewriting system can

[2] We use the notion location instead of vertex to make a clear distinction to the vertices of graphs.

be seen as a finite representation of an infinite graph.

Ground tree rewriting systems have been studied intensively in the past. In [Bra69] the focus is on the set of trees generated by a GTRS, and it is shown that this set is regular. Many problems for GTRS are decidable, among them reachability [Bra69, CG90], fair termination [Tis89], and confluence [DHLT90]. In [DT90] it is shown that the first-order theory of ground tree rewriting systems is decidable.

Overview of this Thesis

In Chapter 2, following the present Introduction, we introduce the basic terminology on graphs, ranked trees, ground tree rewriting, and tree automata. We also introduce an extension of ground tree rewriting, namely regular ground tree rewriting (RGTR), as already studied in [Eng99]. The relation of GTR graphs and RGTR graphs is similar to the relation of pushdown graphs and prefix recognizable graphs. In regular ground tree rewriting the rewriting rules are of the form $T_1 \overset{\sigma}{\hookrightarrow} T_2$ for regular tree languages T_1 and T_2 (cf. [GS84, CDG$^+$97]). In this way graphs of infinite degree can be generated.

The main part of this thesis consists of a structural analysis of GTR graphs and of an algorithmic analysis of RGTR and GTR graphs.

Structural Analysis of GTR graphs. In Chapter 3 we analyze the structure of GTR graphs and relate them to other classes of graphs. In the first part we study GTR graphs under the aspect of tree-width (cf. [Die00]). The tree-width of a graph, usually defined via tree-decompositions, indicates how much a graph resembles a tree. A tree-decomposition structures a graph in a tree-like manner by grouping sets of vertices together and arranging them as a tree according to certain rules that respect the original structure of the graph. The size of these sets of vertices define the width of the tree-decomposition. The tree-width of a graph is the minimal width of all tree-decompositions of the graph. There are examples of infinite graphs that do not admit a tree-decomposition of finite width. Hence, the tree-width of an infinite graph can be finite or infinite. As already mentioned, the notion of tree-width was successfully applied to characterize the relation between prefix recognizable and equational graphs [Bar98]. Our main result from this part of the thesis is of the same nature (see page 62):

Theorem. *A GTR graph is a pushdown graph if, and only if, it has finite tree-width.*

Apart from precisely characterizing the relation between GTR graphs and pushdown graphs, this result allows to relate GTR graphs to prefix recognizable and equational graphs. To complete the comparison with other graph classes we discuss the relation of GTR graphs and automatic graphs.

The second part of the structural analysis deals with traces of GTR graphs. If a graph with edge labels is equipped with initial and final vertices, then it can be interpreted as an infinite automaton. The traces of the graph are the finite words formed by the labels of the paths leading from an initial to a final vertex. The traces of finite graphs define the class of regular languages (cf. [HU79]), and the traces of pushdown graphs correspond to the class of context free languages. This is not surprising since the class of context free languages is exactly the class of languages that can be accepted by pushdown automata (cf. [HU79]). From the characterization of prefix recognizable graphs by prefix rewriting on words using regular languages [Cau96] it follows that, from the perspective of traces, prefix recognizable graphs have the same expressive power as pushdown graphs.

The approach of defining classes of word languages via infinite graphs also yields new characterizations of the context sensitive languages using rational graphs [MS01] or automatic graphs [Ris02]. A nice introduction to the theory of infinite automata is given in [Tho02].

Here we analyze the class of languages defined via traces of GTR graphs. Besides some results on closure properties of this class of languages we determine its relation to classical classes of languages resulting in the following theorem (see page 82):

Theorem. *The class of languages defined as traces of GTR graphs is located strictly between the context free and the context sensitive languages.*

Algorithmic Analysis of RGTR graphs. Our algorithmic analysis, presented in Chapter 4, is carried out for RGTR graphs. We focus on the model-checking problem for RGTR graphs with temporal logics (cf. [Eme90]). In general, model-checking is the task of testing whether a given structure satisfies a given property written in some specification logic. In temporal logics, like CTL, LTL, or CTL*, one can specify reachability properties that talk about paths through the structure. Therefore, temporal formulas are interpreted w.r.t. a given initial vertex. The model-checking problem for RGTR graphs and temporal logics is the following: Given an RGTR graph G, an initial vertex t_{in} of G, and a temporal formula φ, is φ valid in G with t_{in} as initial vertex?

It has been shown for other classes of graphs, e.g. for BPP [EK95], that model-checking these graphs with certain temporal operators is decidable whereas it is undecidable for other temporal operators. Our aim is to build up a logic from predicates for regular sets of trees (cf. [GS84, CDG+97]) as atomic formulas, Boolean combinations of formulas, and temporal operators, such that the model-checking problem for RGTR graphs and this logic is decidable. To identify those temporal operators that can be included in our logic we separately study different reachability problems corresponding to the semantics of basic temporal operators. These reachability problems are listed in the following, where the paths are supposed to start in a given initial vertex t_{in} of the given graph, and T, T' denote regular sets of trees:

One step reachability: Does there exist a successor of t_{in} that is in T?

Reachability: Does there exist a path to a vertex in T?

Constrained reachability: Does there exist a path that remains in T' until it eventually reaches a vertex in T?

Recurrence: Does there exist a path that infinitely often visits T?

Universal reachability: Do all paths eventually reach a vertex in T?

Universal recurrence: Do all infinite paths infinitely often visit T?

One step reachability is easily seen to be decidable. Reachability is known to be decidable for GTR graphs ([Bra69, CG90]) and for RGTR graphs ([Eng99]). Here we adapt an algorithm for the use with RGTR graphs that was presented in [CDGV94] to solve the reachability problem for another generalization of ground tree rewriting. Our main result is that the recurrence problem is decidable and we provide a polynomial time algorithm solving this problem (a version of this algorithm for GTR graphs was published in [Löd02b]). The other problems from the list are shown to be undecidable. The decidability results are summarized in the main theorem of Chapter 4 (see page 123):

Theorem. *The model-checking problem for RGTR graphs and the temporal logic built up from predicates for regular sets of trees, Boolean combinations, and temporal operators for one step reachability, reachability, and recurrence is decidable.*

The undecidability results show that one cannot add other basic temporal operators to the logic without making the model-checking problem undecidable.

Related Work

The theory of infinite graphs and verification of infinite state systems is an active field of research. We would like to mention some recent work of other researchers that emerged during the writing of this thesis and which is related to this work in the sense that ranked trees are used to represent the vertices or states of infinite graphs or systems.

Infinite graphs can be defined using infinite terms over basic graph operations. In [Cou00] operators for adding vertices, inserting edges, and disjoint union of graphs are used to define infinite graphs. Terms built from these operators can also be used to characterize prefix recognizable graphs (cf. [Blu01]). A graph is prefix recognizable if, and only if, it is definable by a regular term over these operations, where an infinite term is regular if it contains only finitely many different subterms. In [Col02] Colcombet uses the same mechanism to define infinite graphs but with an additional operation for the asynchronous product of graphs. It turns out that this operation exactly captures the amount of expressiveness gained when passing from prefix rewriting on words to ground tree rewriting.[3] The article of Colcombet contains three main results:

- In analogy to the result for prefix recognizable graphs it is shown that a graph is an RGTR graph if, and only if, it is definable by a regular term over this extended signature.
- Extending our result on GTR graphs of bounded tree-width that appeared in [Löd02a], Colcombet shows that RGTR graphs of bounded tree-width are equational graphs.
- On the algorithmic side it is shown that a graph defined by an infinite term over the extended signature has a decidable FO theory with reachability predicate if the term defining the graph has a decidable MSO theory.

The article of Colcombet mainly contributes to a structural theory of infinite graphs by providing different characterizations for classes of infinite graphs and analyzing their structure. But the tree rewriting approach is also used more directly for the aspect of modeling and verification.

For the verification of parametrized systems the approach of "regular model-checking" is used in [BJNT00]. In this approach states of the system are represented by finite words. Each position in the word corresponds to a

[3] The approach of typed trees, as used in [Col02], is not directly comparable to the untyped approach used in this thesis.

finite state component in the system. The infiniteness of the system stems from the arbitrary number of components. The transitions of such a system are modeled by regular relations over words. The use of words for the representation of the system makes this approach suitable for systems with a topology that corresponds to the structure of a word, e.g. ring topologies (if we assume that the last letter of a word is connected to the first one).

In [BT02] and [AJNO02] this concept was generalized to trees with the transitions of the system modeled by regular tree transformations. In this way, parametrized systems with more general topologies can be represented. The authors develop techniques to obtain partial algorithms (that do not necessarily terminate) to solve the reachability problem, which is undecidable for these systems in general.

Chapter 2

Tree Rewriting Graphs

In this chapter we formalize the concept of ground tree rewriting that was explained informally in the introduction. In the following sections we give basic notations and definitions for graphs, ranked trees, ground tree rewriting, and tree automata.

2.1 Graphs

A directed edge labeled graph G is a tuple $G = (V, E, \Sigma)$, where V is the set of vertices, Σ is the finite set of edge labels, and $E \subseteq V \times \Sigma \times V$ is the set of edges. We only consider countable graphs, i.e., the set V is countable. Whenever we use the notion of graph without specifying the type of the graph in more detail, then we mean a countable directed edge labeled graph.

If we are not interested in the edge labels or if we consider graphs without edge labels, we omit Σ and assume that $E \subseteq V \times V$.

In Section 3.2 we also consider undirected graphs, i.e., graphs with undirected edges. In directed graphs there are two possible edges between two different vertices u and v, namely (u, v) and (v, u). In undirected graphs there is only one possible edge between u and v, namely $\{u, v\}$. So, in undirected graphs edges are sets of size two instead of ordered pairs. Note that this definition does not allow "self loops", i.e., in undirected graphs there are no edges from a vertex to itself.

The notions tree-width and clique-width used in Section 3.2 do not depend on the direction of edges or on the edge labels. So, for later use we define the undirected unlabeled version G^{und} of a graph $G = (V, E, \Sigma)$ as $G^{\text{und}} = (V, E^{\text{und}})$ with

$$E^{\text{und}} = \{\{u, v\} \mid u \neq v \text{ and } \exists \sigma \in \Sigma \ : \ (u, \sigma, v) \in E \text{ or } (v, \sigma, u) \in E\}.$$

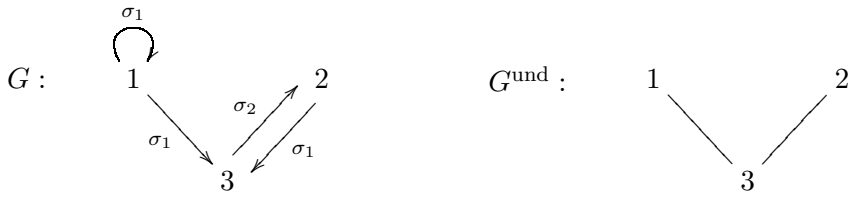


Figure 2.1: A directed labeled graph and its undirected unlabeled version

EXAMPLE 2.1 Figure 2.1 shows the graph

$$G = (\underbrace{\{1,2,3\}}_{V}, \underbrace{\{\sigma_1,\sigma_2\}}_{\Sigma}, \underbrace{\{(1,\sigma_1,1),(1,\sigma_1,3),(2,\sigma_1,3),(3,\sigma_2,2)\}}_{E})$$

and its undirected unlabeled version

$$G^{\text{und}} = (\underbrace{\{1,2,3\}}_{V}, \underbrace{\{\{1,3\},\{2,3\}\}}_{E^{\text{und}}}).$$

□

A graph $G' = (V', E', \Sigma)$ is a subgraph of $G = (V, E, \Sigma)$ iff $V' \subseteq V$ and $E' \subseteq E$. We call G' the subgraph induced by V' iff $E' = E \cap (V' \times \Sigma \times V')$.

A path π in G is a nonempty sequence of vertices $\pi = v_0 \cdots v_n$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, n-1\}$. An undirected path in G is a nonempty sequence of vertices $\pi = v_0 \cdots v_n$ such that $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$ for all $i \in \{0, \ldots, n-1\}$. The length of a (undirected) path $\pi = v_0 \cdots v_n$ is n. For two vertices $u, v \in V$ we say that there is a path (undirected path) from u to v if there is a path (undirected path) $\pi = v_0 \cdots v_n$ in G with $v_0 = u$, and $v_n = v$. Infinite paths are defined in the same way as finite paths but with an infinite sequence of vertices.

The out degree of a vertex v is the number of outgoing edges of v and the in degree is the number of incoming edges of v. The degree of v is the sum of outgoing and incoming edges. A graph G is of finite (in/out) degree if all its vertices have finite (in/out) degree, otherwise G has infinite degree. We say that G has bounded (in/out) degree if there is a number $d \in \mathbb{N}$ such that each vertex has (in/out) degree less than d.

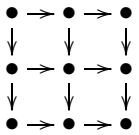
Given a vertex $v \in V$, the connected component of v in G is the subgraph of G induced by the set $\{u \in V \mid \text{ there is an undirected path from } u \text{ to } v\}$.

Two graphs $G_1 = (V_1, E_1, \Sigma)$ and $G_2 = (V_2, E_2, \Sigma)$ are isomorphic iff there is a bijective mapping $\varphi : V_1 \to V_2$ such that $(u, \sigma, v) \in E_1$ iff

$(\varphi(u), \sigma, \varphi(v)) \in E_2$ for all $u, v \in V_1$ and all $\sigma \in \Sigma$. Usually we do not distinguish graphs up to isomorphism. So, two graphs are the same iff they are isomorphic.

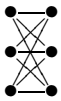
Special graphs that reappear throughout this thesis are listed in the following. In different contexts we use labeled, unlabeled, directed, and undirected versions of these graphs.

- For $m \geq 2$ the $(m \times m)$-grid has m^2 vertices connected in a grid-like manner as indicated in the picture below for the (3×3)-grid:



- The infinite grid is defined similar to the $(m \times m)$-grids but with infinitely many vertices. An example how to obtain the infinite grid as transition graph of a ground tree rewriting system is given in Section 2.3.

- The complete bipartite graph $K_{m,m}$ has $2m$ vertices such that the edge relation is the set $V_1 \times V_2$ for a partition of the vertex set into two sets V_1 and V_2 of size m. The following picture shows the undirected $K_{3,3}$.

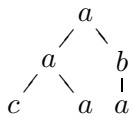


- As usual, trees are graphs such that for each two vertices u, v there is exactly one undirected path from u to v.

In the next section we introduce ranked trees which are different from the above mentioned graph theoretic trees. Throughout this thesis we mainly use ranked trees. So, usually, when we speak of trees we mean ranked trees. If we mean graph theoretic trees, then it should either be clear from the context or we refer to them as unranked or graph theoretic trees.

2.2 Ranked Trees

For a set X we denote by X^* the set of all finite sequences over X and by X^+ all finite nonempty sequences over X. For $w \in X^*$ the length of w is denoted by $|w|$ and the empty word is denoted by ε. By $\mathbb{N}$ we denote the set of natural numbers, i.e., the set of non-negative integers. By $\sqsubseteq$ we denote



$a(a(c,a),b(a))$

graphical notation term notation

Figure 2.2: The tree from Example 2.2

the prefix ordering on $\mathbb{N}^*$ and by $\sqsubset$ the strict prefix ordering. That is, $x \sqsubseteq y$ for $x, y \in \mathbb{N}^*$ iff there is $z \in \mathbb{N}^*$ such that $y = xz$ and $x \sqsubset y$ if $z \in \mathbb{N}^+$.

A ranked alphabet A is a finite family of finite sets $(A_i)_{i\in[k]}$, where $[k] = \{0, \ldots, k\}$. In the following, k always denotes the maximum rank in A. For simplicity we identify A with the set $\bigcup_{i=0}^{k} A_i$. To specify a ranked alphabet one can list all the sets A_i or simply specify the set of all symbols together with their ranks.

A ranked tree t over A is a mapping $t : D_t \to A$ with $D_t \subseteq [k]^*$ such that

- D_t is prefix closed,
- $D_t \neq \emptyset$,
- for each $x \in \mathbb{N}^*$ and $i \in \mathbb{N}$: if $xi \in D_t$, then $xj \in D_t$ for all $j \leq i$, and
- if $x0, \ldots, x(i-1) \in D_t$ and $xi \notin D_t$, then $t(x) \in A_i$.

D_t is called the domain of t and the elements of D_t are called the locations of t (we do not call them vertices to clearly separate them from vertices of a graph). For $x, y \in D_t$ we call x the predecessor of y and y a successor of x iff there is $i \in \mathbb{N}$ such that $y = xi$. The set of all finite (with finite domain) ranked trees over A is denoted by T_A.

We use the term notation and a graphical notation for trees as shown in the following example.

Example 2.2 Let A be the ranked alphabet given by $A_0 = \{a, c\}$, $A_1 = \{b\}$, $A_2 = \{a\}$. Then $t : \{\varepsilon, 0, 1, 00, 01, 10\} \to A$ with $t(\varepsilon) = t(0) = t(01) = t(10) = a$, $t(1) = b$ and $t(00) = c$ is a ranked tree over A. The graphical and the term notation are shown in Figure 2.2. □

The height of a tree $t \in T_A$ is $\text{height}(t) = max\{|x| \mid x \in D_t\}$, i.e., the length of a longest path through t.

Given a tree t and a location x of t, the subtree of t at the location x is the tree obtained by taking all locations of t that have x as prefix, removing the prefix x from all these locations, and keeping the labels. This is formalized as follows. For $x \in \mathbb{N}^*$ we define $xD_t = \{xy \in \mathbb{N}^* \mid y \in D_t\}$ and $x^{-1}D_t = \{y \in \mathbb{N}^* \mid xy \in D_t\}$. For $x \in D_t$ the subtree $t^{\downarrow x}$ of t at x is the tree with domain $D_{t^{\downarrow x}} = x^{-1}D_t$ and $t^{\downarrow x}(y) = t(xy)$.

To formalize the concept of replacing a subtree of a given tree by another tree we introduce the notion of substitution. A substitution is a pair of a location $x \in \mathbb{N}^*$ and a tree $s \in T_A$, written as $[x/s]$. A substitution $[x/s]$ can be applied to a tree t if $x \in D_t$. The result $t[x/s]$ of a substitution applied to t is the tree t' with domain $D_{t'} = xD_s \cup (D_t \setminus xD_{t^{\downarrow x}})$ and

$$t'(y) = \begin{cases} s(z) & \text{if } y = xz \text{ with } z \in D_s, \\ t(y) & \text{if } y \in D_t \setminus xD_{t^{\downarrow x}}. \end{cases}$$

This means we replace the subtree $t^{\downarrow x}$ in t by s.

Example 2.3 Let $t = a(a(c, a), b(a))$ be the tree from Example 2.2 shown in Figure 2.2. Then $t^{\downarrow 0} = a(c, a)$ and $t[0/b(c)] = a(b(c), b(a))$. □

Substitution is the operation that is used in ground tree rewriting systems to transform trees. In the next section these systems are introduced and it is explained how they can be used to generate infinite graphs.

2.3 Ground Tree Rewriting

A ground tree rewriting system (GTRS) is a tuple $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$, where $A = (A_i)_{i \in [k]}$ is a ranked alphabet, Σ is an alphabet, R is a finite set of rules of the form $s \overset{\sigma}{\hookrightarrow} s'$ with $s, s' \in T_A$, $\sigma \in \Sigma$, and $t_{\text{in}} \in T_A$ is the initial tree. The set of rules defines what kind of substitutions are compatible with $\mathcal{R}$ as follows. A substitution $[x/s']$ is $(\mathcal{R}, \sigma)$-applicable to a tree $t \in T_A$ if $x \in D_t$ and if there is a rule $s \overset{\sigma}{\hookrightarrow} s' \in R$ with $s = t^{\downarrow x}$. It is $\mathcal{R}$-applicable to t if it is $(\mathcal{R}, \sigma)$-applicable to t for some $\sigma \in \Sigma$. We write

- $t \xrightarrow[\mathcal{R}]{\sigma} t'$ 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 $\sigma \in \Sigma$ with $t \xrightarrow[\mathcal{R}]{\sigma} t'$, and
- $\xrightarrow[\mathcal{R}]{*}$ for the transitive and reflexive closure of $\xrightarrow[\mathcal{R}]{}$.

The tree language that is generated by $\mathcal{R}$ is $T(\mathcal{R}) = \{t \in T_A \mid t_{\text{in}} \xrightarrow[\mathcal{R}]{*} t\}$.

Rewriting systems of this kind have already been considered in [Bra69] as an extension of regular canonical systems introduced by Büchi [Büc64]. In [Bra69] the set $T(\mathcal{R})$ is the main object under consideration. We are interested in the graph structure that is naturally induced on $T(\mathcal{R})$ by the rewriting relation. The edge labeled graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}}, \Sigma)$ generated by $\mathcal{R}$ is defined by $V_{\mathcal{R}} = T(\mathcal{R})$ and $(t, \sigma, t') \in E_{\mathcal{R}}$ iff $t \xrightarrow[\mathcal{R}]{\sigma} t'$.

Graphs that are isomorphic to $G_{\mathcal{R}}$ for some GTRS $\mathcal{R}$ are called ground tree rewriting graphs or GTR graphs for short. Note that we define the vertex set of $G_{\mathcal{R}}$ to be the set of all trees reachable from the initial tree by repeated application of the rewriting rules. At the end of this chapter we comment on other possibilities for defining the vertex set.

Since every tree in T_A has only finitely many subtrees it is obvious that each vertex of $G_{\mathcal{R}}$ has only finitely many outgoing edges. In the same way one can see that every vertex of $G_{\mathcal{R}}$ can have only finitely many incoming edges. This leads to the first simple remark on GTR graphs.

REMARK 2.4 GTR graphs are of finite degree.

In the following we give some examples of GTR graphs.

EXAMPLE 2.5 Consider the GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ given by the following components.

- $\Sigma = \{0, 1\}$,
- $A = (A_i)_{i \in [2]}$ with $A_0 = \{a, b\}$, $A_1 = \{c\}$, and $A_2 = \{d\}$,
- $R = \{b \overset{0}{\hookrightarrow} c(b), a \overset{1}{\hookrightarrow} c(a)\}$, and
- $t_{\text{in}} = d(a, b)$.

The two rewriting rules can be applied independently to the left and right branch of the tree. This generates the infinite grid as shown in Figure 2.3. □

EXAMPLE 2.6 The GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ given by the following components shows that it is also possible to generate graphs of unbounded (though finite) out degree.

- $\Sigma = \{0\}$,
- $A = (A_i)_{i \in [2]}$ with $A_0 = \{a\}$, $A_1 = \emptyset$, and $A_2 = \{b\}$,

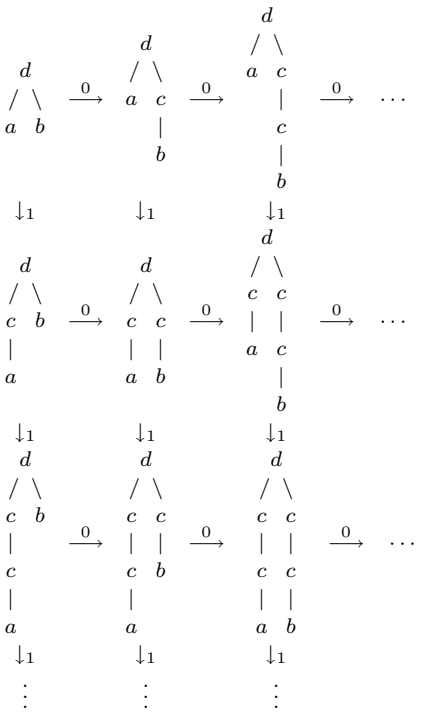


Figure 2.3: The infinite grid generated by the GTRS from Example 2.5.

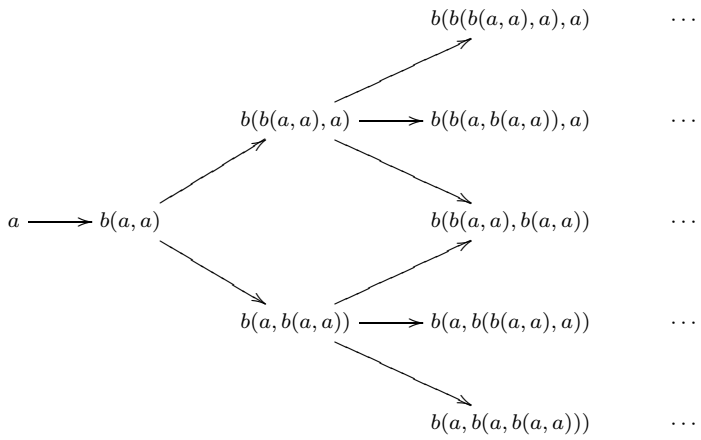


Figure 2.4: The graph of unbounded out degree from Example 2.6.

- $R = \{a \overset{0}{\hookrightarrow} b(a,a)\}$, and
- $t_{\text{in}} = a$.

Each application of the rewriting rule produces one more a. So, the more rewriting steps are needed to obtain t from t_{in}, the more locations can be used for substitutions in t. The initial part of the graph $G_{\mathcal{R}}$ is shown in Figure 2.4. The edge labels are omitted because 0 is the only edge label. □

As a last example we give a rewriting system generating a graph of unbounded in degree. Note that it is not possible to just reverse the rules of the GTRS from the previous example and choosing a different initial tree. This would give a GTRS reducing the size of a tree with each rewriting step. Since the vertices are all the trees that are reachable from the initial tree, such a system would generate a finite graph.

EXAMPLE 2.7 The GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ given by the following components generates a graph of unbounded in degree.

- $\Sigma = \{0,1,2,3\}$,
- $A = (A_i)_{i\in[2]}$ with $A_0 = \{b,c,d,e\}$, $A_1 = \emptyset$, and $A_2 = \{a\}$,
- $R = \{c \overset{0}{\hookrightarrow} a(b,c), a(b,c) \overset{1}{\hookrightarrow} a(d,e), e \overset{2}{\hookrightarrow} a(b,e), d \overset{3}{\hookrightarrow} b\}$, and
- $t_{\text{in}} = a(b,c)$.

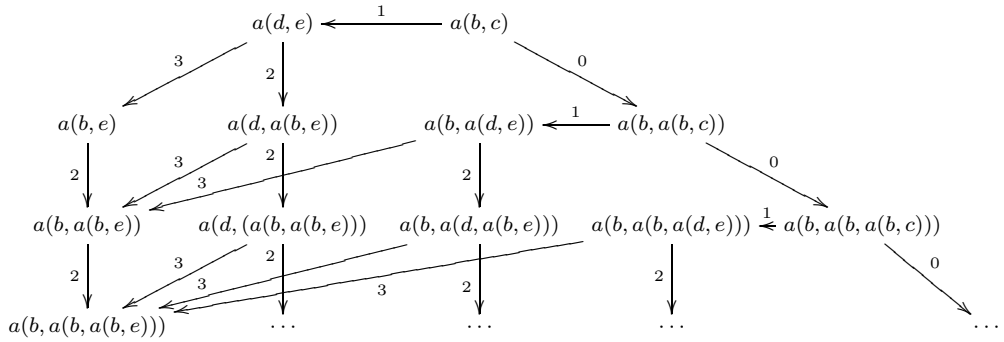


Figure 2.5: The graph of unbounded in degree from Example 2.7.

The graph is shown in Figure 2.5 with the trees displayed in term notation. □

In Section 2.1 we defined the notion of path through a graph. A GTR graph only consists of the trees reachable from the initial tree, but substitutions can also be applied to trees outside of $T(\mathcal{R})$. This leads to the notion of $\mathcal{R}$-path, as defined below. Furthermore, we have to introduce some terminology to refer to the substitutions generating the edges of paths through GTR graphs.

An $\mathcal{R}$-path (or just path if $\mathcal{R}$ is clear from the context) is a finite or infinite sequence $t_0, t_1, t_2, \ldots$ of trees such that $t_i \underset{\mathcal{R}}{\longrightarrow} t_{i+1}$ for all t_i, t_{i+1} in this sequence. For each such path there is a sequence of substitutions $[x_0/s_0], [x_1/s_1], [x_2/s_2], \ldots$ such that $x_i \in D_{t_i}$, $t_i^{\downarrow x_i} \overset{\sigma}{\hookrightarrow} s_i$ is a rewriting rule of $\mathcal{R}$, and $t_{i+1} = t_i[x_i/s_i]$. Such a sequence of substitutions is called an $\mathcal{R}$-derivation of the path $t_0, t_1, t_2, \ldots$ or just derivation if the rewriting system $\mathcal{R}$ is clear from the context. It might happen that there are two different substitutions $[x_i/s_i]$ and $[x_i'/s_i']$ such that $t_{i+1} = t_i[x_i/s_i]$ and $t_{i+1} = t_i[x_i'/s_i']$. Therefore, there may be more than one derivation for a sequence of trees. Sometimes, it is desirable to speak of the derivation of a sequence of trees. In this case we take the unique derivation $[x_0/s_0], [x_1/s_1], [x_2/s_2], \ldots$ with maximal x_i, i.e., if $t_{i+1} = t_i[x/s]$, then either $x \sqsubseteq x_i$ or there is no rule $t_i^{\downarrow x} \overset{\sigma}{\hookrightarrow} s$. Note that this is just a convention to make things precise in some proofs. It would also be possible to take the minimal x_i but the idea behind this convention is that as few locations of the tree as possible should be involved in the rewritings.

If π is a finite $\mathcal{R}$-path, then $\pi : t \xrightarrow[\mathcal{R}]{*} t'$ means that π is an $\mathcal{R}$-path from t to t'. For trees t, t' with $t \xrightarrow[\mathcal{R}]{*} t'$, a derivation of $t \xrightarrow[\mathcal{R}]{*} t'$ is a derivation of a sequence of trees leading from t to t' via rewriting rules from $\mathcal{R}$. This derivation is not unique since there may be different sequences of trees leading from t to t'.

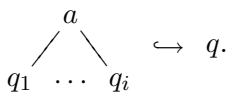
2.4 Tree Automata

Finite automata over finite words are a formalism to represent certain kinds of infinite sets of words, the so called regular sets or regular languages. The theory of finite automata and regular word languages can be transferred to finite ranked trees by means of tree automata. Tree automata are devices with finite memory that read input trees and accept or reject them. We use these automata in the next section to define more general rewriting rules and in Chapter 4 to specify infinite sets of vertices of GTR graphs.

In the following we introduce the models of nondeterministic bottom-up automata with and without ε-transitions. For a more comprehensive introduction to tree automata see e.g. [GS84] or [CDG+97].

A nondeterministic tree automaton (NTA) is a tuple $\mathcal{A} = (Q, A, \Delta, F)$, where Q is a finite set of states, $A = (A_i)_{i \in [k]}$ is a ranked alphabet, $F \subseteq Q$ is a set of final states and $\Delta \subseteq (\bigcup_{i=0}^{k} Q^i \times A_i) \times Q$ is the transition relation.

Tree automata can be viewed as special ground rewriting systems over the alphabet A with A_0 augmented by the elements of Q. The elements of the transition relation are interpreted as rewriting rules as follows. Let $q_1, \ldots, q_i, q \in Q$ and $a \in A_i$. The transition $(q_1, \ldots, q_i, a, q)$ corresponds to the (unlabeled) rewriting rule



This allows us to use the terminology of ground rewriting systems from the previous section to define the set $T(\mathcal{A})$ of trees accepted by the NTA $\mathcal{A}$:

$$T(\mathcal{A}) = \{t \in T_A \mid \exists q \in F \ : \ t \xrightarrow[\mathcal{A}]{*} q\}.$$

So $T(\mathcal{A})$ is the set of all trees that can be transformed into a final state using the transitions as rewriting rules. A set of trees that can be accepted by an NTA is called regular.

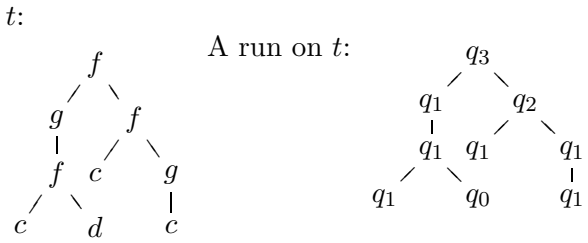


Figure 2.6: A run of the NTA $\mathcal{A}_1$ from Example 2.8

EXAMPLE 2.8 We construct an NTA $\mathcal{A}_1$ over the ranked alphabet A consisting of $A_0 = \{c, d\}$, $A_1 = \{g\}$, and $A_2 = \{f\}$ that accepts all trees with exactly three occurrences of c. The automaton has states $q_0, \ldots, q_3$ counting the number of c's seen so far. If it reaches a location where the sum of the c's in the subtrees is larger than 3, it cannot make a transition. It accepts if it can reduce the given tree to state q_3. Formally, let $\mathcal{A}_1 = (\{q_0, q_1, q_2, q_3\}, A, \Delta_1, \{q_3\})$ with

$$\begin{aligned} \Delta_1 \quad = \quad & \{(c, q_1), (d, q_0)\} \\ \cup \quad & \{(q_i, g, q_i) \mid i \in \{0, 1, 2, 3\}\} \\ \cup \quad & \{(q_i, q_j, f, q_{i+j}) \mid i, j \in \{0, 1, 2, 3\} \text{ and } i + j \leq 3\}. \end{aligned}$$

□

Another way of defining acceptance of an NTA is to use the notion of run. A run ρ of $\mathcal{A}$ on a tree t is a mapping $\rho : D_t \to Q$ such that for each $x \in D_t$, if $t(x) \in A_i$, then $(\rho(x0), \ldots, \rho(x(i-1)), t(x), \rho(x)) \in \Delta$. A run ρ is accepting if $\rho(\varepsilon) \in F$. One can easily verify that $t \xrightarrow[\mathcal{A}]{*} q$ iff there is a run of $\mathcal{A}$ on t with $\rho(\varepsilon) = q$. Figure 2.6 shows an accepting run of $\mathcal{A}_1$ from Example 2.8.

An NTA as defined above reduces a tree t over A to a single state by removing from t one occurrence of a symbol from A in each rewriting step. To simplify constructions of automata we introduce an extension of this model by allowing the automaton to change a state without removing one of the input symbols. This leads to the model ε-NTA.

An ε-NTA is a tuple $\mathcal{A} = (Q, A, \Delta, F)$, where Q, A, F are as for NTA and $\Delta \subseteq ((\bigcup_{i=0}^{k} Q^i \times A_i) \times Q) \cup (Q \times Q)$ is the transition function. The only difference to NTA is that transitions of the form (p, q) for $p, q \in Q$ are allowed. Transitions of this kind are called ε-transitions. An ε-transition (p, q) for $p, q \in Q$ corresponds to a rewriting rule $p \hookrightarrow q$.

EXAMPLE 2.9 We change the NTA from Example 2.8 such that it accepts all

trees where the number of occurrences of c is divisible by three instead of being equal to three. For this purpose, we add the transition (q_2, q_2, f, q_1) and the ε-transition (q_3, q_0) to $\mathcal{A}_1$: we define $\mathcal{A}_2 = (\{q_0, q_1, q_2, q_3\}, A, \Delta_2, \{q_3\})$ with $\Delta_2 = \Delta_1 \cup \{(q_3, q_0)\}$.

If $\mathcal{A}_2$ reaches a location labeled f such that the two subtrees contain two c's each, then it can use the new transition (q_2, q_2, f, q_1) to count modulo 3. If $\mathcal{A}_2$ reaches q_3 after processing a subtree of the input t, it can guess whether there are more c's in the remaining part of t and then reset the counter to 0, or it can guess that it already has seen all c's and stay in state q_3. □

There is a direct connection between the sets $T(\mathcal{R})$ of trees generated by GTRS and regular sets of trees. This connection was already studied in [Bra69] with the following result.

PROPOSITION 2.10 ([Bra69]) *For each GTRS $\mathcal{R}$ the set $T(\mathcal{R})$ is regular.*

This result can also be seen as a special case of the algorithm from Chapter 4 for solving the reachability problem.

In many proofs, if a tree t can be reduced to a state p of an automaton $\mathcal{A}$, i.e., $t \xrightarrow[\mathcal{A}]{*} p$, we are interested in the state that is used at a specific location $x \in D_t$ in this reduction. If this state is q, then we write

$$t \xrightarrow[\mathcal{A}]{*} t[x/q] \xrightarrow[\mathcal{A}]{*} p.$$

This means that the automaton reduces the subtree of t at location x to the state q and then reduces the remaining part of t to p. In the following remark we give two simple arguments that are used repeatedly in connection with this notation (especially in Section 4.2).

REMARK 2.11 Let $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA, $p, q \in Q$, $t, s \in T_A$, and $x \in D_t$.

(i) $t \xrightarrow[\mathcal{A}]{*} t[x/q]$ iff $t^{\downarrow x} \xrightarrow[\mathcal{A}]{*} q$.

(ii) If $t[x/q] \xrightarrow[\mathcal{A}]{*} p$ and $s \xrightarrow[\mathcal{A}]{*} q$, then $t[x/s] \xrightarrow[\mathcal{A}]{*} p$.

The proof of this remark is trivial and therefore omitted, but we think that isolating these two facts might ease the reading of several proofs.

Basic Constructions and Algorithms

An important property of the class of regular tree languages is the closure under Boolean operations. Here, we give constructions for union and intersection. For other closure properties and correctness proofs for the subsequent constructions we refer the reader to [CDG+97].

For the complexity analysis of automaton constructions we need the size $|\mathcal{A}|$ of a tree automaton. Similar to [CDG+97] we define the size $|\alpha|$ of a transition $\alpha = (q_1, \ldots, q_i, a, q) \in \Delta$ as $|\alpha| = i+2$. The size of an ε-transition α is 2. The size of $\mathcal{A}$ is the number of states plus the sum of the size of all transitions in Δ, i.e.,

$$|\mathcal{A}| = |Q| + \sum_{\alpha \in \Delta} |\alpha|.$$

To refer to the language of an automaton with a certain state as final state we define for $q \in Q$ the automaton $\mathcal{A}(q) = (Q, A, \Delta, \{q\})$. With this definition we get $t \in T(\mathcal{A}(q))$ iff $t \xrightarrow[\mathcal{A}]{*} q$.

Let $\mathcal{A}_1 = (Q_1, A, \Delta_1, F_1)$, $\mathcal{A}_2 = (Q_2, A, \Delta_2, F_2)$ be two ε-NTAs. The automaton $\mathcal{A}_1 \cup \mathcal{A}_2$ recognizing the union of $T(\mathcal{A}_1)$ and $T(\mathcal{A}_2)$ is defined as

$$\mathcal{A}_1 \cup \mathcal{A}_2 = (Q_1 \dot\cup Q_2, A, \Delta_1 \cup \Delta_2, F_1 \cup F_2).$$

The automaton $\mathcal{A}_1 \times \mathcal{A}_2$ for the intersection of $T(\mathcal{A}_1)$ and $T(\mathcal{A}_2)$ is defined as

$$\mathcal{A}_1 \times \mathcal{A}_2 = (Q_1 \times Q_2, A, \Delta_\times, F_1 \times F_2),$$

where $\Delta_\times$ contains the transitions of the form $((p_1, q_1), \ldots, (p_i, q_i), a, (p, q))$ for $(p_1, \ldots, p_i, a, p) \in \Delta_1$ and $(q_1, \ldots, q_i, a, q) \in \Delta_2$.

PROPOSITION 2.12 *Let $\mathcal{A}_1, \mathcal{A}_2$ be ε-NTA. Then*

(i) $T(\mathcal{A}_1 \cup \mathcal{A}_2) = T(\mathcal{A}_1) \cup T(\mathcal{A}_2)$ *with* $|\mathcal{A}_1 \cup \mathcal{A}_2| \in \mathcal{O}(|\mathcal{A}_1| + |\mathcal{A}_2|)$, *and*
(ii) $T(\mathcal{A}_1 \times \mathcal{A}_2) = T(\mathcal{A}_1) \cap T(\mathcal{A}_2)$ *with* $|\mathcal{A}_1 \times \mathcal{A}_2| \in \mathcal{O}(|\mathcal{A}_1| \cdot |\mathcal{A}_2|)$.

As for automata on finite words, complementation of nondeterministic tree automata uses the subset construction for determinization.

PROPOSITION 2.13 *For each ε-NTA $\mathcal{A}$ there is an ε-NTA $\overline{\mathcal{A}}$ with $T(\overline{\mathcal{A}}) = T_A \setminus T(\mathcal{A})$ and $|\overline{\mathcal{A}}| \in \mathcal{O}(2^{|\mathcal{A}|})$.*

An important operation on tree automata is to compute the set of all reachable states, where a state q is reachable in $\mathcal{A}$ iff there is a tree t such that $t \xrightarrow[\mathcal{A}]{*} q$. This can be done efficiently (see e.g. [CDG+97]).

PROPOSITION 2.14 *The set of all reachable states of an ε-NTA $\mathcal{A}$ can be computed in time $\mathcal{O}(|\mathcal{A}|)$.*

A linear time algorithm can be found in Appendix A.1.

2.5 Regular Ground Tree Rewriting

Regular ground tree rewriting systems are defined in the same way as GTRS but with regular sets instead of single trees in the rewriting rules (similar to the relation between pushdown graphs and prefix recognizable graphs, see Subsections 3.1.1 and 3.3.1). A regular ground tree rewriting system (RGTRS) is a tuple $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$, where A, Σ, and t_{in} are the same as for GTRS and R is a finite set of rewriting rules of the form $T \overset{\sigma}{\hookrightarrow} T'$ for regular sets $T, T' \subseteq T_A$ of trees. For algorithmic applications we assume that these regular sets are given by nondeterministic tree automata.

A substitution $[x/s']$ is $(\mathcal{R}, \sigma)$-applicable to a tree t if $x \in D_t$ and if there is a rule $T \overset{\sigma}{\hookrightarrow} T' \in R$ with $t^{\downarrow x} \in T$ and $s' \in T'$. With this adapted definition of $(\mathcal{R}, \sigma)$-applicable substitution the definitions of $\mathcal{R}$-applicable, $\xrightarrow[\mathcal{R}]{\sigma}$, $\xrightarrow[\mathcal{R}]{}$, $\xrightarrow[\mathcal{R}]{*}$, $T(\mathcal{R})$, and $G_{\mathcal{R}}$ are the same as for GTRS. We also carry over the definition of $\mathcal{R}$-path and derivation in the obvious way.

In the same way as before, graphs that are isomorphic to $G_{\mathcal{R}}$ for some RGTRS $\mathcal{R}$ are called regular ground tree rewriting graphs or RGTR graphs for short.

EXAMPLE 2.15 The RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ we define resembles the GTRS from Example 2.5. The ranked alphabet is $A = (A_i)_{i \in [2]}$ with $A_0 = \{a, b\}$, $A_1 = \{c\}$, and $A_2 = \{d\}$, the alphabet for the edge labels is $\Sigma = \{0, 1\}$, and the initial tree is $t_{\text{in}} = d(a, b)$. One rule contains an infinite set of trees on the right hand side. For this purpose we define the NTA $\mathcal{A} = (\{q_0, q_1\}, A, \Delta, \{q_1\})$ with $\Delta = \{(b, q_0), (q_0, c, q_1), (q_1, c, q_1)\}$. The set $T(\mathcal{A})$ accepted by $\mathcal{A}$ contains all the unary trees of the form $c(\cdots c(b) \cdots)$. The set R of rewriting rules is defined as $R = \{\{a\} \overset{1}{\hookrightarrow} \{c(a)\}, \{b\} \overset{0}{\hookrightarrow} T(\mathcal{A})\}$.

The graph $G_{\mathcal{R}}$ is shown in Figure 2.7 without edge labels. Basically, it is the infinite grid, but in each row each vertex has infinitely many edges to the right. □

In Chapter 4 on model-checking for RGTR graphs we need the following relations and notations.

- For trees t, t' we write $t \xrightarrow[\mathcal{R}]{+} t'$ if t' is reachable from t with at least one substitution, i.e., if there is a tree t'' such that $t \xrightarrow[\mathcal{R}]{} t'' \xrightarrow[\mathcal{R}]{*} t'$.

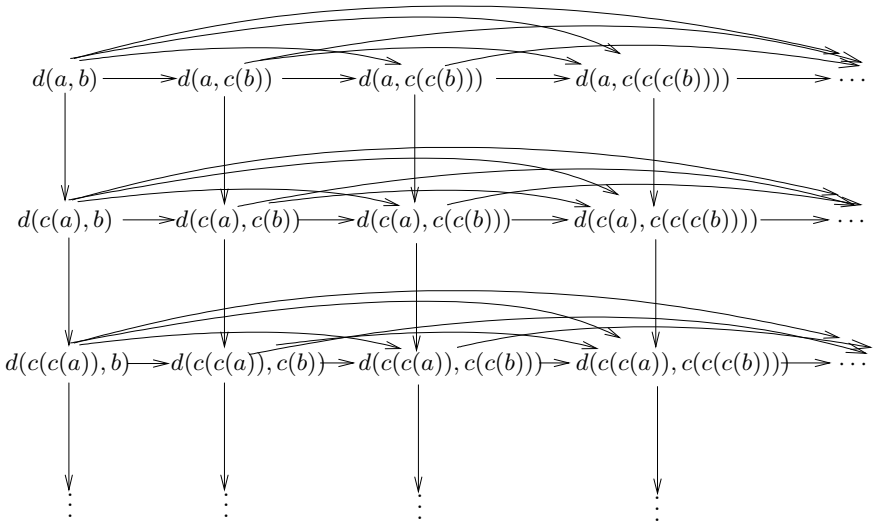


Figure 2.7: The graph of the RGTRS from Example 2.15

- For sets $T_1, \ldots, T_n$ of trees we write

$$T_1 \xrightarrow[\mathcal{R}]{*} T_2 \xrightarrow[\mathcal{R}]{*} \cdots \xrightarrow[\mathcal{R}]{*} T_n$$

  if there are $t_i \in T_i$ for each $i \in \{1, \ldots, n\}$ such that $t_1 \xrightarrow[\mathcal{R}]{*} t_2 \xrightarrow[\mathcal{R}]{*} \cdots \xrightarrow[\mathcal{R}]{*} t_n$. We also use this notation for $\xrightarrow[\mathcal{R}]{+}$ and combinations of $\xrightarrow[\mathcal{R}]{*}$ and $\xrightarrow[\mathcal{R}]{+}$.

- For a tree t and a set T of trees we write $t \xrightarrow[\mathcal{R}]{\omega} T$ if there is an infinite sequence $t_0 \xrightarrow[\mathcal{R}]{} t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \cdots$ with $t = t_0$ such that infinitely many of the t_i are in T. Furthermore, for sets T_1, T_2 of trees, we write $T_1 \xrightarrow[\mathcal{R}]{\omega} T_2$ if there exists $t \in T_1$ with $t \xrightarrow[\mathcal{R}]{\omega} T_2$.

- If π is an infinite $\mathcal{R}$-path starting in a tree t and visiting a set T of trees infinitely often, then we denote this by $\pi : t \xrightarrow[\mathcal{R}]{\omega} T$.

We also use these notations for single trees instead of sets of vertices and write, e.g., $t \xrightarrow[\mathcal{R}]{\omega} t'$ instead of $t \xrightarrow[\mathcal{R}]{\omega} \{t'\}$. Please note that in the above definitions we always ask for the existence of a tree. So $T_1 \xrightarrow[\mathcal{R}]{*} T_2$ is true if there exists a tree in T_1 from where we can reach T_2. We do not require

that from all trees in T_1 we can reach T_2 as the notation might suggest at first glance.

Another relation derived from the rewriting relation is the inverse of the rewriting relation. This relation is used in Chapter 4 where we analyze forward and backward reachability problems. The inverse $\mathcal{R}^{-1}$ of an RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ is obtained from $\mathcal{R}$ by reversing the rules in R, i.e., $\mathcal{R}^{-1} = (A, \Sigma, R^{-1}, t_{\text{in}})$ with $T' \overset{\sigma}{\hookrightarrow} T \in R^{-1}$ iff $T \overset{\sigma}{\hookrightarrow} T' \in R$.

As for automata, we need to define the size $|\mathcal{R}|$ of an RGTRS $\mathcal{R}$ for the complexity analysis of the algorithms in Chapter 4. Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be an RGTRS with $R = \{T_1 \overset{\sigma_1}{\hookrightarrow} T_1', \ldots, T_m \overset{\sigma_m}{\hookrightarrow} T_m'\}$ such that $T_i = T(\mathcal{A}_i)$ and $T_i' = T(\mathcal{A}_i')$ for NTAs $\mathcal{A}_i$ and $\mathcal{A}_i'$. For the algorithms that we are going to analyze the initial tree does not play any role. For this reason we define the size $|\mathcal{R}|$ of $\mathcal{R}$ simply as

$$|\mathcal{R}| = \sum_{i=1}^{m} (|\mathcal{A}_i| + |\mathcal{A}_i'|).$$

As the definition of RGTR graphs is similar to the definition of GTR graphs, the first step is to compare these two classes with each other. Proposition 2.10 can be extended to RGTRS. This follows from the results in Chapter 4. Another proof can be found in [Eng99].

PROPOSITION 2.16 *For each RGTRS $\mathcal{R}$ the set $T(\mathcal{R})$ is regular.*

Since finite sets of trees, and singleton sets in particular, are regular, it is obvious that every GTR graph is an RGTR graph. As Example 2.6 shows, it is possible to generate GTR graphs with unbounded but still finite out degree. By using infinite regular sets of trees in the rewriting rules of RGTRS the graphs generated by these systems may have infinite degree, as in Example 2.15.

This rises the question whether the difference of GTR graphs and RGTR graphs only consists of graphs of infinite degree. The following theorem gives an affirmative answer to this question.

THEOREM 2.17 *If G is an RGTR graph of finite degree, then G is a GTR graph.*

Proof. Let G be an RGTR graph of finite degree and let $\mathcal{R}$ be an RGTRS such that $G_{\mathcal{R}}$ is isomorphic to G. Let $T \overset{\sigma}{\hookrightarrow} T'$ be a rewriting rule of $\mathcal{R}$. We show that we can replace this rule by a finite set of GTR rules.

If T or T' is empty, then we can safely omit the rule. If T and T' are finite, then we replace $T \overset{\sigma}{\hookrightarrow} T'$ by the set of rules $\{t \overset{\sigma}{\hookrightarrow} t' \mid t \in T \text{ and } t' \in T'\}$.

If T' is infinite, then there is no $t \in T(\mathcal{R})$ such that $t^{\downarrow x} \in T$ for some $x \in D_t$. Otherwise, there would be an edge from t to all $t[x/t']$ with $t' \in T'$, contradicting the assumption that G has finite degree. Thus, the rule $T \overset{\sigma}{\hookrightarrow} T'$ is not used for generating $G_{\mathcal{R}}$ and can be omitted.

If T is infinite, then we show that there is $h \in \mathbb{N}$ such that for all $t \in T(\mathcal{R})$ and $x \in D_t$ with $t^{\downarrow x} \in T$ we have $\text{height}(t^{\downarrow x}) \leq h$. Then we can replace $T \overset{\sigma}{\hookrightarrow} T'$ by the set of rules $\{t \overset{\sigma}{\hookrightarrow} t' \mid t \in T \text{ with } \text{height}(t) \leq h, \text{ and } t' \in T'\}$. This set is finite, since T' is finite and there are only finitely many trees of height less than or equal to h.

To define h we use a consequence of the pumping lemma for regular tree languages (see e.g. in [CDG+97]):

Let $\mathcal{A} = (Q, A, \Delta, F)$ be an NTA. If there is a tree $t \in T(\mathcal{A})$ with $\text{height}(t) > |Q|$ and $t \xrightarrow[\mathcal{A}]{*} q$ for some $q \in Q$, then there are infinitely many $s \in T(\mathcal{A})$ with $s \xrightarrow[\mathcal{A}]{*} q$.

We use this fact as follows. The set T is regular and therefore there exists an NTA $\mathcal{A} = (Q_{\mathcal{A}}, A, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$ accepting T. By Proposition 2.16 there exists an automaton $\mathcal{B} = (Q_{\mathcal{B}}, A, \Delta_{\mathcal{B}}, F_{\mathcal{B}})$ accepting the set $T(\mathcal{R})$. Let $h = |Q_{\mathcal{A}}| \cdot |Q_{\mathcal{B}}|$ and assume that there exist $t \in T(\mathcal{R})$ and $x \in D_t$ with $\text{height}(t^{\downarrow x}) > h$ and $t^{\downarrow x} \in T$. Then there is $p \in F_{\mathcal{A}}$ with $t^{\downarrow x} \xrightarrow[\mathcal{A}]{*} p$. Furthermore, since $t \in T(\mathcal{R}) = T(\mathcal{B})$, there are $q \in Q_{\mathcal{B}}$ and $q' \in F_{\mathcal{B}}$ with $t \xrightarrow[\mathcal{B}]{*} t[x/q] \xrightarrow[\mathcal{B}]{*} q'$. In particular we get $t^{\downarrow x} \xrightarrow[\mathcal{B}]{*} q$. Let $\mathcal{C} = \mathcal{A} \times \mathcal{B}$. From the definition of the product automaton we get $t^{\downarrow x} \xrightarrow[\mathcal{C}]{*} (p, q)$. Using the consequence of the pumping lemma stated above we obtain infinitely many trees s with $s \xrightarrow[\mathcal{C}]{*} (p, q)$ because $\text{height}(t^{\downarrow x}) > h = |Q_{\mathcal{A}}| \cdot |Q_{\mathcal{B}}|$. Again by the definition of the product automaton we get for all s with $s \xrightarrow[\mathcal{C}]{*} (p, q)$:

- $s \xrightarrow[\mathcal{A}]{*} p$,
- $s \xrightarrow[\mathcal{B}]{*} q$, and therefore
- $t[x/s] \xrightarrow[\mathcal{B}]{*} t[x/q] \xrightarrow[\mathcal{B}]{*} q'$.

Thus, there are infinitely many trees of the form $t[x/s]$ with $s \in T$ (by the first item) and $t[x/s] \in T(\mathcal{R})$ (by the third item). From all these trees there is an edge to $t[x/t']$ for $t' \in T'$. Hence, $t[x/t']$ has infinite in degree, a contradiction. □

The preceding theorem characterizes the class of GTR graphs as those graphs that have finite degree and are RGTR graphs. In Chapter 3 we

present results of the same kind, relating GTR graphs and pushdown graphs using the structural properties of bounded tree-width and bounded clique-width (instead of finite degree).

As already mentioned in Section 2.3 we finish this chapter with a brief discussion on other possiblities of defining the vertex set of graphs generated by (regular) ground tree rewriting.

- One option is to consider the whole set T_A of trees over A, i.e., to define the graph generated by a rewriting system $\mathcal{R}$ as $(T_A, \xrightarrow[\mathcal{R}]{})$. For the results on model-checking from Chapter 4 this does not make any difference. But if A includes a symbol of rank at least two, then the graph $(T_A, \xrightarrow[\mathcal{R}]{})$ has unbounded tree-width for every $\mathcal{R}$ that contains at least one rewriting rule $s \xrightarrow[\mathcal{R}]{\sigma} s'$ with $s \neq s'$. Therefore, we have chosen the definition with the vertex set generated from the initial tree to make the graphs rooted, i.e., having a designated root vertex. This allows a precise comparison to the class of pushdown graphs from [MS85] that have the initial configuration of the pushdown automaton as root vertex (see Section 3.2) using the notion of tree-width.

- It is possible to subsume our definition with the initial tree and the definition from the previous item by allowing to restrict the set of vertices to an arbitrary regular set of trees. In this case, the undecidability proofs from Section 4.3 can easily be adapted to show the undecidablity of reachability for this class of graphs. Hence, this way of defining the graphs is too strong if one is interested in model-checking.

- In [Col02] Colcombet uses a stronger typing policy by not only defining the rank of a symbol but also fixing the type of its successors. The set of vertices is then defined as all trees of a certain type. This way of restricting the vertex set is more flexible as it allows to define graphs that are not connected but it also makes the underlying objects more complex.

Chapter 3

The Structure of GTR Graphs

Ground tree rewriting graphs are defined via a formalism that allows to generate these graphs. This definition gives no direct information on the kind of graphs that belong to the class of GTR graphs because the definition depends on the names of the vertices (i.e., the trees that represent the vertices). Given a graph, there is no direct way to see whether it is a GTR graph. For example, consider the graph shown in Figure 3.1. A formal definition of G is $G = (V, E, \Sigma)$ with $\Sigma = \{0, 1, 2\}$, $V = \{X\}^* \times \{Y\}^*$, and

$$\begin{aligned} E \;=\; & \{((X^i, \varepsilon), 0, (X^{i+1}, \varepsilon)) \mid i \geq 0\} \\ \cup\; & \{((X^i, Y^j), 1, (X^{i-1}, Y^{j+1})) \mid i > 0, j \geq 0\} \\ \cup\; & \{((\varepsilon, Y^j), 2, (\varepsilon, Y^{j-1})) \mid j > 0\}. \end{aligned}$$

If this graph is a GTR graph, then after some time we will probably find an appropriate GTRS generating this graph. Otherwise, we have to prove that no GTRS generates this graph. For this purpose, it is desirable to have general theorems saying that graphs with certain properties cannot be GTR graphs. A simple example of such a property is given by Remark 2.4, stating that GTR graphs are of finite degree. So, given a graph of infinite degree, it is clear that it is no GTR graph.

The most desirable result is a characterization of GTR graphs that does not depend on their representation. An example for such a result is the characterization of pushdown graphs by Muller and Schupp [MS85] (see Subsection 3.1.1). We do not have a complete result of that kind, but building on the result of Muller and Schupp we obtain such a characterization for a subclass of the class of GTR graphs. More precisely, we show in Section 3.2 that GTR graphs of bounded tree-width or of bounded clique-width are

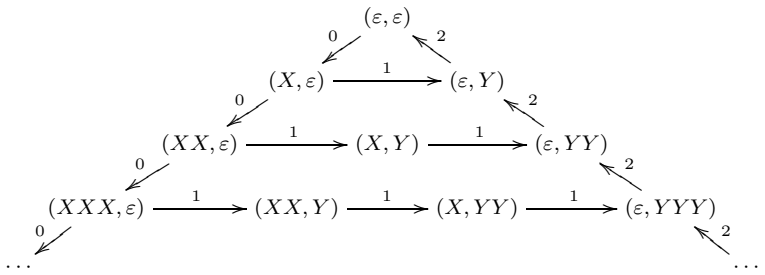


Figure 3.1: Is this graph a GTR graph?

pushdown graphs. Together with the result of Muller and Schupp this gives a characterization of these subclasses of GTR graphs that is independent of their representation. Furthermore, these results yield a precise description of the relation between pushdown graphs and GTR graphs.

In Section 3.3 we continue our analysis with a comparison of GTR graphs to other classes of graphs, namely prefix recognizable graphs [Cau96], equational graphs [Cou89], and automatic graphs [BG00]. These investigations allow us to place the class of GTR graphs into the known hierarchy of these graph classes as shown in Figure 3.12 on page 64.

The last section in this chapter deals with traces of GTR graphs. From this perspective one interprets infinite graphs as automata with an infinite set of states. In this way, one can use infinite graphs to define languages of finite words (see [Tho02] for an overview). Since pushdown automata characterize the class of context free languages it is not astonishing that this is exactly the class of languages that can be defined as traces of pushdown graphs. A recent result by Morvan and Stirling states that the context sensitive languages can be characterized as the traces of rational graphs [MS01]. This result is sharpened in [Ris02], where it is shown that even the synchronized rational or automatic graphs suffice to trace all context sensitive languages.

Our results show that the traces of GTR graphs define a class of languages strictly in between the context free and the context sensitive languages. Furthermore, we analyze the closure properties of this class of languages. Besides the interest in traces of infinite graphs from the perspective of formal language theory, these investigations also give more insight into the structure of GTR graphs and provide methods for showing that certain graphs cannot be represented by ground tree rewriting systems. An example

is the graph from Figure 3.1. This graph can be used to trace a context sensitive language that cannot be traced by GTR graphs (see Subsection 3.4.1).

3.1 Preliminaries

In this section we give definitions and results that are needed throughout this chapter. In particular, we introduce pushdown graphs and state a seminal result of Muller and Schupp on the structure of these graphs [MS85].

3.1.1 Pushdown Automata

In formal language theory pushdown automata are used as an automaton model for the class of context free languages (see e.g. [HU79]). This aspect of pushdown automata is used in Section 3.4, but first of all, we are interested in the configuration graphs of pushdown automata.

A pushdown automaton (PDA) M is a tuple $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$, where

- Q is a finite set of states,
- Σ is the input alphabet,
- Γ is the stack alphabet (disjoint from Q),
- $\Delta \subseteq Q \times (\Gamma \cup \{\varepsilon\}) \times \Sigma \times Q \times \Gamma^*$ is the transition relation,
- $q_{\text{in}} \in Q$ is the initial state, and
- $\gamma_{\text{in}} \in \Gamma$ is the initial stack symbol.

Sometimes, we write the transitions of a pushdown automaton in a similar way as the rewriting rules of a GTRS. Then a transition $(p, \gamma, \sigma, q, w)$ with $p, q \in Q$, $\sigma \in \Sigma$, $\gamma \in \Gamma$, and $w \in \Gamma^*$ is written as $p\gamma \overset{\sigma}{\hookrightarrow} qw$ and $(p, \varepsilon, \sigma, q, w)$ is written as $p \overset{\sigma}{\hookrightarrow} qw$.

A configuration of M is a word qw with a state $q \in Q$ and a stack content $w \in \Gamma^*$. Given configurations q_1w_1, q_2w_2 and an input $\sigma \in \Sigma$ we write $q_1w_1 \vdash_M^\sigma q_2w_2$ if there is a transition $(q_1, \gamma, \sigma, q_2, v) \in \Delta$ (with $\gamma \in \Gamma \cup \{\varepsilon\}$) such that $w_1 = \gamma w$ and $w_2 = vw$. We write $q_1w_1 \vdash_M q_2w_2$ if there is an input $\sigma \in \Sigma$ with $q_1w_1 \vdash_M^\sigma q_2w_2$, and by $\vdash_M^*$ we denote the transitive and reflexive closure of $\vdash_M$. The set of configurations $C(M)$ generated by M is the set of configurations that are reachable from the initial configuration of M:

$$C(M) = \{qw \mid q_0\gamma_0 \vdash_M^* qw\}.$$

The directed edge labeled graph $G_M = (V_M, E_M, \Sigma)$ generated by M has the set of vertices $V_M = C(M)$, and for two words $q_1w_1, q_2w_2 \in V_M$ there is an edge labeled with σ from q_1w_1 to q_2w_2 iff $q_1w_1 \vdash_M^\sigma q_2w_2$.

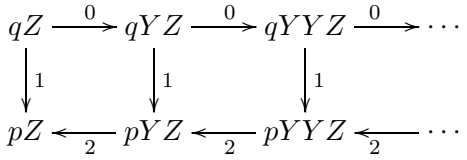


Figure 3.2: The pushdown graph from Example 3.1

EXAMPLE 3.1 Consider the pushdown automaton

$$M = (\{p, q\}, \{0, 1, 2\}, \{Y, Z\}, \Delta, q, Z) \text{ with } \Delta = \{q \overset{0}{\hookrightarrow} qY, q \overset{1}{\hookrightarrow} p, pY \overset{2}{\hookrightarrow} p\}.$$

The graph G_M is depicted in Figure 3.2. □

As usual, a graph G is a pushdown graph iff G is isomorphic to the graph G_M for some PDA M.

A first simple observation is that a pushdown automaton can be simulated by a GTRS with unary trees. The stack symbols are used as symbols of rank one, and the control states as symbols of rank zero. The rewriting rules correspond to the transitions of the pushdown automaton, and the initial tree is the tree corresponding to the initial configuration of the pushdown automaton.

REMARK 3.2 Every pushdown graph is a GTR graph.

The examples from Section 2.3 show that the class of GTR graphs strictly contains the class of pushdown graphs.

As for GTR graphs the above definition of pushdown graphs has the disadvantage that it depends on the names of the vertices and does not tell anything about the structure of the graphs. In [MS85] Muller and Schupp gave a structural characterization of pushdown graphs that does not depend on the defining formalism. In the following we informally describe their result.

The initial configuration of a pushdown automaton can be viewed as the root of the corresponding pushdown graph. All vertices of the graph are reachable from this root. Graphs with such a designated vertex are called rooted. The distance of a vertex to the root is the minimal number of edges that have to be traversed to reach the vertex from the root. In this definition the direction of the edges does not play any role, so we are allowed to traverse edges in both directions.

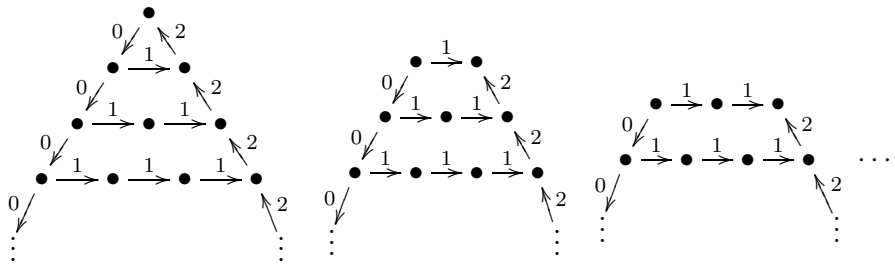


Figure 3.3: The ends of the graph from Figure 3.1

If we choose a number $n \in \mathbb{N}$ and delete all vertices within diameter n around the root, then the remaining graph consists of several connected components. These components are called ends of the graph. Such an end has a front, namely the vertices with distance exactly $n+1$ to the root if the end was obtained by deleting the vertices within diameter n around the root. Two ends are isomorphic if there is a graph isomorphism between these two ends with the additional property that it maps front vertices to front vertices.

The result of Muller and Schupp states that a rooted graph of bounded degree is a pushdown graph iff it does not have infinitely many pairwise non-isomorphic ends.

Proposition 3.3 ([MS85]) *Let G be a rooted graph of finite degree. Then G is a pushdown graph iff G has only finitely many isomorphism classes of ends.*

Example 3.4 The ends of the graph from Figure 3.1 are shown in Figure 3.3. All these ends are pairwise non-isomorphic because the size of the fronts increases and therefore no isomorphism between two different ends can map all front vertices of one end to the front vertices of the other end.

3.1.2 Factorizations of Trees

In Sections 3.2 and 3.4 we transform certain classes of GTRS into pushdown automata. For this purpose we need the operations of concatenation and factorization on trees to be able to represent a restricted class of trees as words.

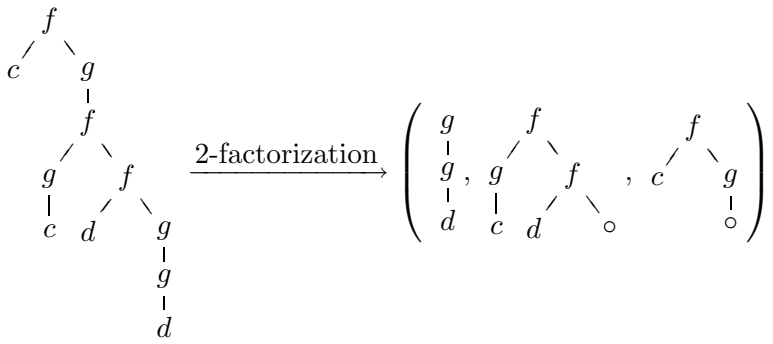


Figure 3.4: An example for a 2-factorization

Given a ranked alphabet A, we introduce a new symbol $\circ$ of rank 0, the concatenation symbol, and define $A_\circ = A \cup \{\circ\}$. The set $S_{A_\circ}$ of special trees over $A_\circ$ consists of all trees $s \in T_{A_\circ}$ with exactly one concatenation symbol, i.e., $s(x) = \circ$ for exactly one $x \in D_s$. For trees $s \in S_{A_\circ}$ and $t \in T_A \cup S_{A_\circ}$ we define the concatenation $s \cdot t$ of s and t in the obvious way:

$$s \cdot t = s[x/t] \text{ for the } x \in D_s \text{ with } s(x) = \circ.$$

We define the operation of factorization of a tree with an integer parameter h determining in some sense the size of the factors. An h-factorization of a tree $t \in T_A$ and $h \in \mathbb{N}$ is a list $(t', s_1, \ldots, s_n)$ with $t' \in T_A$, $s_i \in S_{A_\circ}$, and the following properties:

- If $\text{height}(t) < 2 \cdot h$, then $t' = t$ and $n = 0$.
- If $\text{height}(t) \geq 2 \cdot h$, then $n \geq 1$ and there is a location $x \in D_t$ with $|x| = h$, $\text{height}(t^{\downarrow x}) \geq h$, $s_n = t[x/\circ]$, and $(t', s_1, \ldots, s_{n-1})$ is an h-factorization of $t^{\downarrow x}$.

Figure 3.4 shows an example of an h-factorization for $h = 2$. In general, there may be different h-factorizations of a tree t if there are several locations $x \in D_t$ with $|x| = h$ and $\text{height}(t^{\downarrow x}) \geq h$.

When we transform GTRS into pushdown automata the stack contents together with the control states represent h-factorizations of trees. To realize this we are interested in trees with a unique h-factorization for a given h. Furthermore, since the stack alphabet of a pushdown automaton is finite, the number of possible special trees used in the factorizations should be finite. To identify a class of trees with these properties we introduce the notion

of independence degree. Let $x, y \in \mathbb{N}^*$ be two locations and $z \in \mathbb{N}^*$ the maximal common prefix of x and y. The independence degree $\text{indep}(x, y)$ is defined as

$$\text{indep}(x, y) = \min(|x| - |z|, |y| - |z|).$$

Note that the independence degree of two locations is zero iff they are comparable with respect to $\sqsubseteq$. The independence degree of a tree t is

$$\text{indep}(t) = \max\{\text{indep}(x, y) \mid x, y \in D_t\}.$$

If the independence degree of a tree is smaller than h, then there are no independent subtrees both of height larger than h, where two subtrees are independent if they are rooted at locations incomparable w.r.t. $\sqsubseteq$. Therefore, an independence degree smaller than h implies a unique h-factorization:

LEMMA 3.5 *Let $h \in \mathbb{N}$ and $t \in T_A$ with $\text{indep}(t) < h$. Then t has a unique h-factorization $(t', s_1, \ldots, s_n)$ with $\text{height}(s_i) < 2 \cdot h$ for each $i \in \{1, \ldots, n\}$.*

Proof. We prove this lemma by induction on $h' = \text{height}(t)$. If $h' < 2 \cdot h$, then the unique h-factorization of t is t itself and therefore the claim holds.

If $h' \geq 2 \cdot h$, then there is only one location x with $|x| = h$ and $\text{height}(t^{\downarrow x}) \geq h$ because the existence of two such locations would contradict $\text{indep}(t) < h$. Hence, we get $\text{height}(t[x/\circ]) < 2 \cdot h$, and for each h-factorization $(t', s_1, \ldots, s_n)$ of t we have $s_n = t[x/\circ]$. Then $(t', s_1, \ldots, s_{n-1})$ is an h-factorization of $t^{\downarrow x}$, and by induction it is the only h-factorization of $t^{\downarrow x}$ and $\text{height}(s_i) < 2 \cdot h$ for each $i \in \{1, \ldots, n-1\}$. □

According to the previous lemma all the special trees in h-factorizations of trees with independence degree less than h come from the finite set

$$S^h_{A_\circ} = \{s \in S_{A_\circ} \mid \text{height}(s) < 2h \text{ and } |x| = h \text{ for the } x \in D_s \text{ with } s(x) = \circ\}.$$

Still motivated by the intention of transforming certain classes of GTRS into pushdown automata, we want to ensure that all the trees generated by these GTRS have height at least h for a given integer h. This is just to avoid technical complications in the transformations.

LEMMA 3.6 *For every GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and every $h \in \mathbb{N}$ with $h > \text{height}(t_{\text{in}})$ there is a GTRS $\mathcal{R}' = (A', \Sigma, R, t'_{\text{in}})$ such that $G_{\mathcal{R}}$ and $G_{\mathcal{R}'}$ are isomorphic, $\text{height}(t) \geq h$ for all $t \in T(\mathcal{R}')$, and $\text{height}(t'_{\text{in}}) < 2 \cdot h$.*

Proof. Let $A' = A \cup \{\bot\}$ where $\bot$ is a new symbol of rank 1 that does not appear in A. The new initial tree t'_{in} is obtained by adding a $\bot$-sequence of length h to t_{in} above the root. Formally, t'_{in} is defined as follows. Let x be the location 0^h. Then

$$D_{t'_{\text{in}}} = \{y \in \mathbb{N}^* \mid y \sqsubset x\} \cup xD_{t_{\text{in}}} \text{ and}$$
$$t'_{\text{in}}(y) = \begin{cases} \bot & \text{if } y \sqsubset x, \\ t_{\text{in}}(z) & \text{if } y = xz \text{ with } z \in D_{t_{\text{in}}}. \end{cases}$$

Since $\bot$ did not appear in the old rewriting system, this $\bot$-sequence of length h will never be changed and therefore $\text{height}(t) \geq h$ for all $t \in T(\mathcal{R}')$. From the assumption $h > \text{height}(t_{\text{in}})$ we get $\text{height}(t'_{\text{in}}) < 2 \cdot h$. The mapping $\varphi : T(\mathcal{R}') \to T(\mathcal{R})$ with $\varphi(t) = t^{\downarrow x}$ (for $x = 0^h$) is an isomorphism between $G_{\mathcal{R}'}$ and $G_{\mathcal{R}}$. □

Finally, we define the constant $h_{\mathcal{R}}$ for a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ as the maximal height of the initial tree of $\mathcal{R}$ and the trees occurring in the rules of $\mathcal{R}$:

$$h_{\mathcal{R}} = \max\{\text{height}(t) \mid t = t_{\text{in}} \text{ or } \exists t' \in T_A, \sigma \in \Sigma : t \overset{\sigma}{\hookrightarrow} t' \in R \text{ or } t' \overset{\sigma}{\hookrightarrow} t \in R\}.$$

From the definition follows that $|\text{height}(t') - \text{height}(t)| < h_{\mathcal{R}}$ if $t \underset{\mathcal{R}}{\longrightarrow} t'$. We will use $h_{\mathcal{R}}$ to obtain lower bounds for the number of steps needed to rewrite a tree t into a tree t' if we know the height difference between these two trees.

Remark 3.7 Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS and let $t, t' \in T_A$ with $|\text{height}(t) - \text{height}(t')| > h_{\mathcal{R}} \cdot d$ for some $d > 1$. If π is an $\mathcal{R}$-path with $\pi : t \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} t'$, then the length of π is greater than d.

3.2 GTR Graphs of Bounded Width

In graph theory the class of (unranked) trees constitutes one of the most basic classes of graphs. Their simple structure allows efficient algorithms and many nontrivial theorems in graph theory are much simpler when restricted to the class of trees. For example, the famous graph minor theorem of Robertsen and Seymour was first proved in the special case of trees by Kruskal in 1960 (for details see e.g. [Die00] and references therein).

The tree-width of a graph G is a measure of how much G resembles a tree. A lot of good properties of trees carry over to classes of graphs with bounded tree-width. In this section we study GTR graphs of bounded tree-width with the result that this class equals the class of pushdown graphs.

Another measure for the complexity of infinite graphs is clique-width introduced in [CO00]. It is studied in connection with monadic second-order logic on graphs. We prove that GTR graphs of bounded clique-width are also of bounded tree-width and therefore are pushdown graphs. In combination with the result of Muller and Schupp [MS85] that pushdown graphs have a decidable monadic second-order theory, this also yields that GTR graphs of bounded clique-width have a decidable monadic second-order theory.

A short version of the results from this section was published in [Löd02a].

3.2.1 Tree-Width

There are different equivalent ways on how to define tree-width of graphs. The most widely used definition goes via tree-decompositions. A tree-decomposition of a graph $G = (V, E)$ is a pair $(T, (W_x)_{x \in V_T})$ where $T = (V_T, E_T)$ is a tree (unranked), $W_x \subseteq V$ for all $x \in V_T$, and

- $V = \bigcup_{x \in V_T} W_x$ and $\forall (u, v) \in E \;\; \exists x \in V_T \;:\; u, v \in W_x$,
- $\forall v \in V$ the subgraph of T induced by $\{x \in V_T \mid v \in W_x\}$ is connected.

The width of a tree-decomposition $(T, (W_x)_{x \in V_T})$ is

$$\text{width}(T, (W_x)_{x \in V_T}) = \begin{cases} \max\{|W_x| \mid x \in V_T\} - 1 \text{ if } \{|W_x| \mid x \in V_T\} \text{ is finite} \\ \infty \text{ otherwise.} \end{cases}$$

The "-1" is used in the definition because trees should have tree-width 1. The tree-width $tw(G)$ of G is

$$tw(G) = \min\{\text{width}(T, (W_x)_{x \in V_T}) \mid (T, (W_x)_{x \in V_T}) \text{ tree-decomposition of } G\}.$$

Note that these definitions do not depend on edge labels or direction of edges. Hence, the tree-width of a graph and the tree-width of its undirected version are the same.

Example 3.8 We take the graph G from Figure 3.1 as example. In Figure 3.5 the same graph is drawn in a different way. The boxes and curves around the vertices indicate the tree-decomposition. A formal definition of the tree-decomposition $(T, (W_x)_{x \in V_T})$ with $T = (V_T, E_T)$ is given by

- $V_T = \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid j \in \{0, \ldots, i \text{ div } 2\}\}$, where div denotes integer division.
- An edge $((i_1, j_1), (i_2, j_2))$ is in E_T iff
  - $i_2 = i_1 + 1$ and $j_1 = j_2 = 0$, or

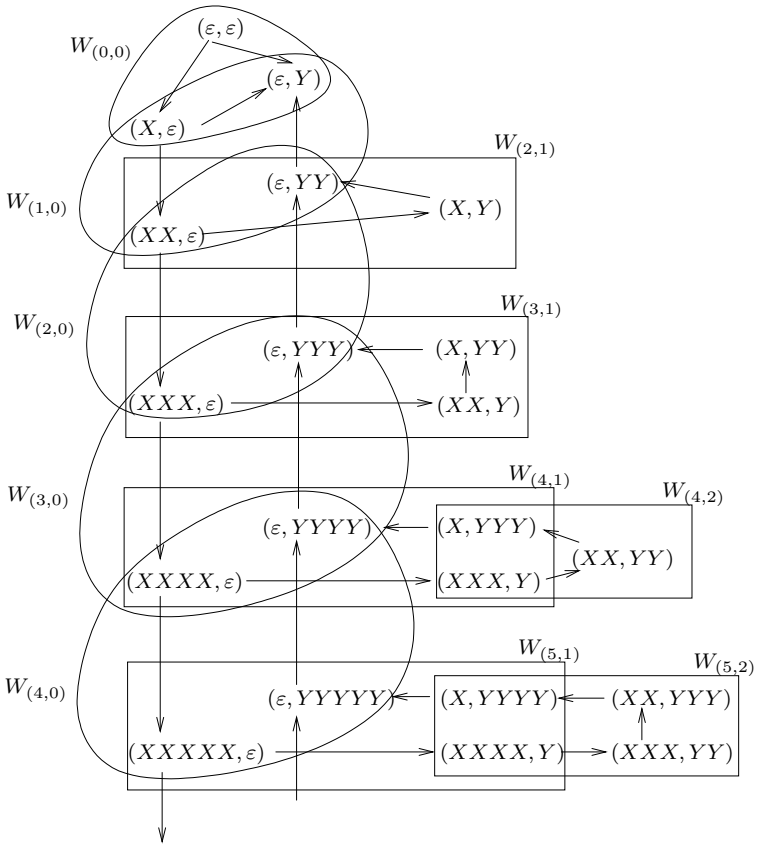


Figure 3.5: A tree-decomposition of the graph from Figure 3.1

- $i_1 = i_2$ and $j_2 = j_1 + 1$.

- $W_{(i,0)} = \{(X^i, \varepsilon), (\varepsilon, Y^i), (X^{i+1}, \varepsilon), (\varepsilon, Y^{i+1})\}$.
- $W_{(i,j)} = \{(X^k, Y^l) \mid k + l = i \text{ and } k \in \{j-1, j\} \text{ or } l \in \{j-1, j\}\}$ for $j \geq 1$.

The width of this tree-decomposition is 3, implying that the tree-width of G is at most 3. In fact it is possible to find a tree-decomposition of width 2 by refining the given one. Since only trees (and forests) have tree-width one we can conclude that the tree-width of G is 2. □

A tree-decomposition of a graph G can easily be transformed into a tree-decomposition for a subgraph H of G by deleting all vertices in the sets W_x that are in G but not in H. Obviously, this operation does not increase the

width of the decomposition. Hence, the tree-width of a subgraph H of G is at most the tree-width of G.

Conversely, the tree-width of an infinite graph is already determined by the tree-width of its finite subgraphs.

This leads to the following proposition.

PROPOSITION 3.9 ([Tho88, Tho89]) *Let G be a graph and $k \in \mathbb{N}$. Then $tw(G) \leq k$ iff $tw(H) \leq k$ for each finite subgraph of G.*

Tree-decompositions are useful for showing upper bounds on the tree-width of a graph G. If we find a tree-decomposition of G of width m, then the tree-width of G is at most m. In our proofs, we mainly need to show lower bounds on the tree-width of graphs. For this purpose we introduce "brambles". A bramble of G is a finite family of finite sets of vertices $\mathcal{B} = (B_i)_{i \in I}$ such that

(1) each B_i induces a connected subgraph in G, and

(2) for each $i, j \in I$: $B_i \cap B_j \neq \emptyset$ or there are $u \in B_i$, $v \in B_j$ with $(u, v) \in E$.

A set $S \subseteq V$ covers $\mathcal{B}$ iff $S \cap B_i \neq \emptyset$ for each $i \in I$. The width of a bramble is $\text{width}(\mathcal{B}) = \min\{|S| \mid S \text{ covers } \mathcal{B}\}$.

EXAMPLE 3.10 Figure 3.6 shows a graph with vertices $\{1, \ldots, 8\}$. The curves around the vertices on the right hand side of the figure indicate the following sets of a bramble:

$$\{5\}, \{7\}, \{8\}, \{2, 3, 4\}, \{2, 3, 6\}, \{1, 4, 6\}.$$

All these sets are connected in the graph and are pairwise neighbored. A cover S for this bramble must contain at least the vertices 5,7, and 8. Furthermore, since $\{2, 3, 4\} \cap \{2, 3, 6\} \cap \{1, 4, 6\} = \emptyset$, S must contain at least two more vertices to cover all the sets of the bramble. A possible cover would be $S = \{2, 4, 5, 7, 8\}$. Thus, the width of the bramble is 5. □

The following result relates brambles and tree-width. A proof for finite graphs can be found in [Die00]. For infinite graphs this result can easily be deduced from the finite case in combination with Proposition 3.9.

PROPOSITION 3.11 *A graph G contains a bramble of width at least m iff $tw(G) \geq m - 1$.*

As an application one can use this proposition to determine the tree-width of grids. A proof of the following proposition can be found in [Die00].

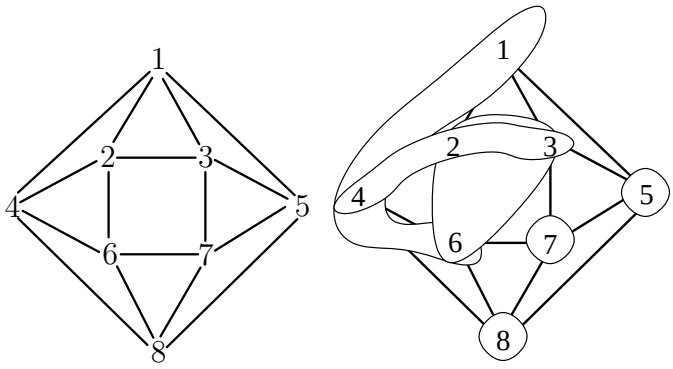


Figure 3.6: A bramble of width 5

PROPOSITION 3.12 *The $(m \times m)$-grid has tree-width m.*

As we have seen in Example 2.5 the infinite grid is a GTR graph. Since the infinite grid contains all $(m \times m)$-grids as subgraphs it follows from Proposition 3.9 that the infinite grid has unbounded tree-width. In contrast to that, all pushdown graphs have bounded tree-width.

PROPOSITION 3.13 *For each pushdown automaton $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ the graph G_M has bounded tree-width.*

Proof. Let $k = (\max\{|w| \mid \exists(p, \gamma, \sigma, q, w) \in \Delta\}) + 1$. We define the tree-decomposition $(T, (W_x)_{x \in V_T})$ with $T = (V_T, E_T)$ by

- $V_T = \Gamma^*$,
- $E_T = \{(v, w) \mid w = \gamma v \text{ with } \gamma \in \Gamma\}$,
- $W_x = \{qw \in V_M \mid w = vx \text{ with } v \in \Gamma^*, |v| \leq k\}$ for $x \in \Gamma^*$.

Note that possibly $W_x = \emptyset$ for some $x \in \Gamma^*$ but this is not in conflict to the definition of tree-decomposition.

We have to check if the two conditions from the definition of tree-decomposition are satisfied. For the first condition, let $(p_1 v_1, p_2 v_2) \in E_M$. Then there exists a transition $(q_1, \gamma, \sigma, q_2, v) \in \Delta$ (with $\gamma \in \Gamma \cup \{\varepsilon\}$) such that $v_1 = \gamma x$ and $v_2 = vx$. Since $|\gamma| \leq 1 \leq k$ and $|v| \leq k$ by definition of k, we know that $p_1 v_1, p_2 v_2 \in W_x$.

Now let $qw \in V_M$. We have to show that the subgraph of T that is induced by $\{x \in V_T \mid qw \in W_x\}$ is connected. If $w = \gamma_1 \cdots \gamma_n$ with $\gamma_i \in \Gamma$

for all $i \in \{1, \ldots, n\}$, then $qw \in W_x$ iff $x = \gamma_i \cdots \gamma_n$ for some $i \in \{1, \ldots, k\}$. By the definition of E_T we have $(\gamma_i \cdots \gamma_n, \gamma_{i+1} \cdots \gamma_n) \in E_T$ for all $i \in \{1, \ldots, n-1\}$. Therefore, the subgraph of T induced by $\{x \in V_T \mid qw \in W_x\}$ is connected.

This proves that we indeed defined a tree-decomposition. It remains to show that the size of the set W_x is bounded. Each element in W_x is of the form qw with $q \in Q$ and $w = vx$ with $|v| \leq k$. There are at most $|\Gamma|^{k+1}$ such v's. Thus, the size of W_x is bounded by $|Q| \cdot |\Gamma|^{k+1}$. □

In the following we will show that GTR graphs of bounded tree-width are pushdown graphs. For this purpose we introduce in the next subsection an extended model of pushdown automata, namely infix pushdown automata. Then the proof is divided into two main parts. First we show that GTR graphs of bounded tree-width are infix pushdown graphs (Subsection 3.2.3). After this step we are already in the domain of infinite graphs with vertices coded by words. In a second step we show that infix pushdown graphs of bounded tree-width are pushdown graphs. The rough structure of the proof looks as follows:

$$\begin{array}{c} \text{GTR graph} \\ \text{of} \\ \text{bounded tree-width} \end{array} \rightsquigarrow \begin{array}{c} \text{infix pushdown graph} \\ \text{of} \\ \text{bounded tree-width} \end{array} \rightsquigarrow \text{pushdown graph}$$

3.2.2 Infix Pushdown Automata

We introduce an extended type of pushdown automata, where in addition to the transition rules manipulating the top of the stack a restricted type of infix rewriting in the stack is allowed.

An infix PDA is a tuple $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$, where $Q, \Sigma, \Gamma, q_{\text{in}}, \gamma_{\text{in}}$ are as in usual pushdown automata and in Δ there are transitions of the usual form but also transitions from $\Gamma \times \Sigma \times \Gamma$. These transitions are called infix rules whereas the other rules are called prefix rules.

For configurations $q_1 w_1$ and $q_2 w_2$ we write

- $q_1 w_1 \vdash^{\sigma}_{M_{\text{pre}}} q_2 w_2$ if there is a transition $(q_1, \gamma, \sigma, q_2, v) \in \Delta$ such that $w_1 = \gamma w$ and $w_2 = vw$ for some $w \in \Gamma^*$,
- $q_1 w_1 \vdash^{\sigma}_{M_{\text{in}}} q_2 w_2$ if $q_1 = q_2$ and if there is an infix rule $(\gamma_1, \sigma, \gamma_2) \in \Delta$ such that $w_1 = w\gamma_1 w'$ and $w_2 = w\gamma_2 w'$ with $w, w' \in \Gamma^*$, and
- $q_1 w_1 \vdash^{\sigma}_{M} q_2 w_2$ if $q_1 w_1 \vdash^{\sigma}_{M_{\text{pre}}} q_2 w_2$ or $q_1 w_1 \vdash^{\sigma}_{M_{\text{in}}} q_2 w_2$.

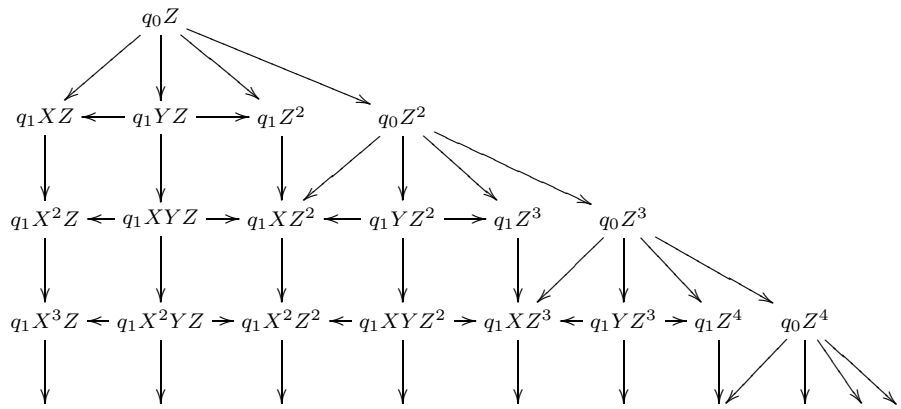


Figure 3.7: The graph generated by M_1 from Example 3.14.

So the infix rules change single letters inside the stack.

We adopt the notations $\vdash_M$, $\vdash_{M_{\text{pre}}}$, $\vdash_{M_{\text{in}}}$, $\vdash^*_M$, $\vdash^*_{M_{\text{pre}}}$, $\vdash^*_{M_{\text{in}}}$, $C(M)$, and G_M for infix pushdown automata in the obvious way.

EXAMPLE 3.14 Let $M_1 = (\{q_0, q_1\}, \{\sigma\}, \{X, Y, Z\}, \Delta, q_0, Z)$ be an infix PDA with

$$\Delta = \{q_0 \overset{\sigma}{\hookrightarrow} q_0Z, q_0 \overset{\sigma}{\hookrightarrow} q_1X, q_0 \overset{\sigma}{\hookrightarrow} q_1Y, q_0 \overset{\sigma}{\hookrightarrow} q_1Z, q_1 \overset{\sigma}{\hookrightarrow} q_1X, Y \overset{\sigma}{\hookrightarrow} X, Y \overset{\sigma}{\hookrightarrow} Z\}.$$

The only stack symbol appearing on the left hand side of an infix rule is Y. The generated graph G_{M_1} is shown in Figure 3.7 (without the edge label σ). Note that all the configurations from $C(M_1)$ contain at most one Y. Nevertheless, the graph G_{M_1} is not of bounded tree-width. □

EXAMPLE 3.15 Let $M_2 = (\{q_0, q_1\}, \{\sigma\}, \{X, Y\}, \Delta, q_0, X)$ be an infix PDA with

$$\Delta = \{q_0 \overset{0}{\hookrightarrow} q_0X, q_0 \overset{1}{\hookrightarrow} q_1Y, q_1 \overset{2}{\hookrightarrow} q_1X, Y \overset{3}{\hookrightarrow} X\}.$$

The generated graph is shown in Figure 3.8. It is isomorphic to the GTR graph from Example 2.7. Note that there is only one infix rule, and as in the previous example all the configurations from $C(M_2)$ contain at most one symbol Y, to which this infix rule can be applied. □

In the next subsection we transform ground tree rewriting systems generating graphs of bounded tree-width into infix pushdown automata. The obtained automata will have certain properties described in the following normal form.

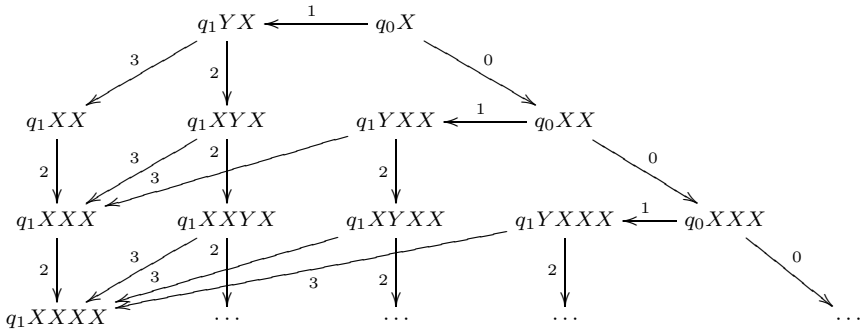


Figure 3.8: The graph generated by M_2 from Example 3.15.

DEFINITION 3.16 Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA. Then M is in normal form iff

1. all elements in Δ are of the form

$$(q, \varepsilon, \sigma, q', \varepsilon), (q, \gamma, \sigma, q', \varepsilon), (q, \varepsilon, \sigma, q', \gamma'), \text{ or } (\gamma, \sigma, \gamma')$$

   with $q, q' \in Q$, $\gamma, \gamma' \in \Gamma \setminus \{\gamma_{\text{in}}\}$, and $\sigma \in \Sigma$, and

2. whenever $qw \vdash_{M_{\text{pre}}} q'\gamma w \vdash_{M_{\text{in}}} q'\gamma' w$, then there is a $q'' \in Q$ such that $qw \vdash_{M_{\text{pre}}} q''w \vdash_{M_{\text{pre}}} q'\gamma' w$. □

The essential property of infix pushdown automata in normal form is that every reachable configuration can also be reached only by applying prefix rules. For later use, we formulate the more general claim that, given a reachable configuration quw, there is a reachable configuration $q'w$ such that one can reach quw from $q'w$ only by applying prefix rules and without "touching" the w.

LEMMA 3.17 *Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form, $q \in Q$, and $u, w \in \Gamma^*$. If $quw \in C(M)$, then there exists $q' \in Q$ such that $q' \vdash^*_{M_{\text{pre}}} qu$, and $q'w \in C(M)$.*

Proof. Since $quw \in C(M)$, there is a directed path from $q_{\text{in}}\gamma_{\text{in}}$ to quw. We choose this path such that every infix rule is applied when the corresponding symbol is on top of the stack (this is possible because by point 1 of Definition 3.16 the prefix rules increasing the stack length do not depend on the stack content). Then, by point 2 of Definition 3.16, we can replace all the infix rules on this path and get $q_{\text{in}}\gamma_{\text{in}} \vdash^*_{M_{\text{pre}}} quw$. On this path π from $q_{\text{in}}\gamma_{\text{in}}$

to quw, there must be a vertex $q'w$ such that every successor of $q'w$ on π contains the suffix w. But this means $q' \vdash^*_{M_{\text{pre}}} qu$. □

3.2.3 From GTRS to Infix Pushdown Automata

A GTR graph of bounded tree-width cannot contain large grids as subgraphs. We will use this fact to show that if a GTRS $\mathcal{R}$ generates a graph of that kind, then the trees in $T(\mathcal{R})$ are of bounded independence degree (see Subsection 3.1.2). This enables us to code these trees by words using h-factorizations and to simulate the rewriting rules of $\mathcal{R}$ by an infix pushdown automaton.

Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS. We extend the notion of $\mathcal{R}$-applicable substitution defined in Section 2.3 to sequences of substitutions in the obvious way. A sequence $[x_1/s_1], \ldots, [x_m/s_m]$ of substitutions is $\mathcal{R}$-applicable to $t \in T_A$ if $[x_1/s_1]$ is $\mathcal{R}$-applicable to t and $[x_2/s_2], \ldots, [x_m/s_m]$ is $\mathcal{R}$-applicable to $t[x_1/s_1]$.

Two sequences $[x_1/s_1], \ldots, [x_m/s_m]$ and $[y_1/t_1], \ldots, [y_n/t_n]$ of substitutions are called independent iff x_i and y_j are not comparable w.r.t. $\sqsubseteq$ for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

A sequence $[x_1/s_1], \ldots, [x_m/s_m]$ of substitutions that is $\mathcal{R}$-applicable to $t \in T_A$ is called repetition free if all the trees that it produces are different, i.e., if $t[x_1/s_1] \cdots [x_i/s_i] \neq t[x_1/s_1] \cdots [x_j/s_j]$ for all $i, j \in \{1, \ldots, m\}$ with $i \neq j$. Note that this property does not depend on t.

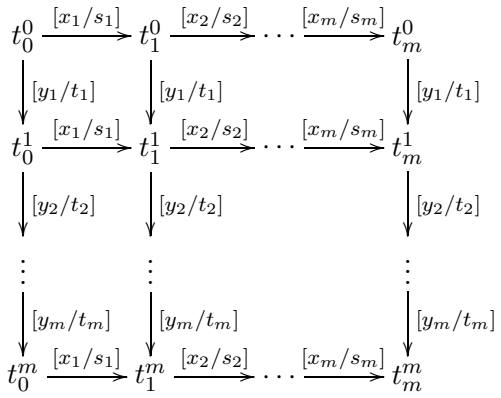
Since independent sequences of substitutions can be interleaved arbitrarily, it is not difficult to see that they generate a grid with size corresponding to the length of the sequences as subgraph[1] if they are repetition free.

LEMMA 3.18 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS. If there are two independent and repetition free sequences of substitutions of length m that are $\mathcal{R}$-applicable to $t \in T(\mathcal{R})$, then $G_{\mathcal{R}}$ has a $(m+1) \times (m+1)$-grid as subgraph.*

Proof. Let $[x_1/s_1], \ldots, [x_m/s_m]$ and $[y_1/t_1], \ldots, [y_m/t_m]$ be two such sequences and for $i, j \in \{0, \ldots, m\}$ let $t^j_i = t[x_1/s_1] \cdots [x_i/s_i][y_1/t_1] \cdots [y_j/t_j]$.

[1] In general, this subgraph is not an induced subgraph.

Then $G_{\mathcal{R}}$ contains the following subgraph.



□

A consequence of this lemma for the trees generated by a GTRS $\mathcal{R}$ where $G_{\mathcal{R}}$ has bounded tree-width is that these trees are of bounded independence degree.

LEMMA 3.19 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS such that $G_{\mathcal{R}}$ has bounded tree-width. Then there is an h such that* $\text{indep}(t) < h$ *for each $t \in T(\mathcal{R})$.*

Proof. Let $m = tw(G_{\mathcal{R}})$, and define $h = h_{\mathcal{R}} \cdot m + 1$. Suppose there is $t \in T(\mathcal{R})$ with $\text{indep}(t) \geq h$. Then there are locations $x_1, x_2 \in D_t$ with maximal common prefix x such that $|x_1| - |x| \geq h$ and $|x_2| - |x| \geq h$ or equivalently two successors y, z of x with $\text{height}(t^{\downarrow y}) \geq h - 1$ and $\text{height}(t^{\downarrow z}) \geq h - 1$. Since $t \in T(\mathcal{R})$ there must be a sequence $[x_1/s_1], \ldots, [x_n/s_n]$ of substitutions leading from t_{in} to t. Let

$$\pi : t_0 \underset{\mathcal{R}}{\longrightarrow} t_1 \underset{\mathcal{R}}{\longrightarrow} \cdots \underset{\mathcal{R}}{\longrightarrow} t_{n-1} \underset{\mathcal{R}}{\longrightarrow} t_n$$

with $t_0 = t_{\text{in}}$ and $t_n = t$ be the path corresponding to this sequence of substitutions. Since $\text{height}(t_n^{\downarrow x}) \geq h$ we know that $x_n \not\sqsubseteq x$. Otherwise, $\text{height}(t_n^{\downarrow x}) \leq h_{\mathcal{R}}$, by the definition of $h_{\mathcal{R}}$, and therefore $\text{height}(t_n^{\downarrow x}) < h$ because $h_{\mathcal{R}} < h$. Hence, we can choose $i \in \{1, \ldots, n\}$ minimal such that $x_j \not\sqsubseteq x$ for all $j \in \{i, \ldots, n\}$. By the choice of i we get $y, z \in D_{t_{i-1}}$, $\text{height}(t_{i-1}^{\downarrow y}) < h_{\mathcal{R}}$, and $\text{height}(t_{i-1}^{\downarrow z}) < h_{\mathcal{R}}$. Thus, if we extract from $[x_i/s_i], \ldots, [x_n/s_n]$ all the substitutions with y as prefix and all the substitutions with z as prefix, then we get two independent sequences that are $\mathcal{R}$-applicable to t_{i-1}. These two sequences can easily be made repetition free by leaving out parts that produce repetitions. Still, they must be of length at least m because they transform the subtrees $t_{i-1}^{\downarrow y}$ and $t_{i-1}^{\downarrow z}$, which are of height less than $h_{\mathcal{R}}$, into

$t^{\downarrow y}$ and $t^{\downarrow z}$, which are of height greater than or equal to $h-1 = h_{\mathcal{R}} \cdot m$. Now Lemma 3.18 can be applied, i.e., $G_{\mathcal{R}}$ has the $(m+1) \times (m+1)$-grid as subgraph and therefore by Propositions 3.12 and 3.9 tree-width $\geq m+1$, a contradiction. □

Now we can use h-factorizations to construct an infix PDA generating a graph isomorphic to the graph generated by the GTRS.

THEOREM 3.20 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS such that $G_{\mathcal{R}}$ has bounded tree-width. Then we can construct an infix PDA M in normal form such that $G_{\mathcal{R}}$ is isomorphic to G_M.*

Proof. Choose h according to Lemma 3.19. By Lemma 3.6 we can assume that $\text{height}(t) \geq h$ for all $t \in T(\mathcal{R})$, and $\text{height}(t_{\text{in}}) < 2 \cdot h$. We define the infix PDA $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ as follows.

- $Q = \{q_t \mid t \in T_A \text{ with } h \leq \text{height}(t) < 2 \cdot h\}$.
- $\Gamma = \{\gamma_{\text{in}}\} \dot\cup \{\gamma_s \mid s \in S^h_{A_\circ}\}$ (see Page 35 for the definition of $S^h_{A_\circ}$).
- $q_{\text{in}} = q_{t_{\text{in}}}$.
- Δ contains the transitions

$$\begin{aligned}
(q_t, \varepsilon, \sigma, q_{t'}, \varepsilon) \quad &\text{if} \quad t \xrightarrow[\mathcal{R}]{\sigma} t', \\
(q_t, \gamma_s, \sigma, q_{t'}, \varepsilon) \quad &\text{if} \quad s \cdot t \xrightarrow[\mathcal{R}]{\sigma} t', \\
(q_t, \varepsilon, \sigma, q_{t'}, \gamma_{s'}) \quad &\text{if} \quad t \xrightarrow[\mathcal{R}]{\sigma} s' \cdot t', \\
(\gamma_s, \sigma, \gamma_{s'}) \quad &\text{if} \quad s \xrightarrow[\mathcal{R}]{\sigma} s'.
\end{aligned}$$

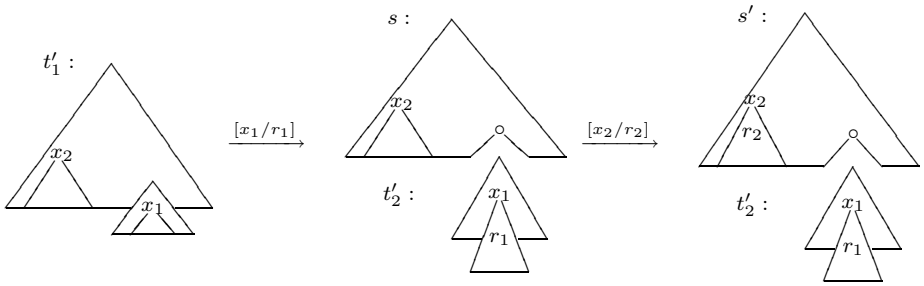
According to Lemma 3.19, all trees from $T(\mathcal{R})$ have independence degree less than h and therefore a unique h-factorization by Lemma 3.5. The mapping $\varphi : T(\mathcal{R}) \to C(M)$ with $\varphi(t) = q_{t'}\gamma_{s_1} \cdots \gamma_{s_m}\gamma_{\text{in}}$ for the unique h-factorization $(t', s_1, \ldots, s_m)$ of t is an isomorphism between $G_{\mathcal{R}}$ and G_M.

The transitions of M obviously satisfy the required format for the normal form of infix PDA. The second requirement of Definition 3.16 is satisfied because a sequence

$$q_{t'_1}\gamma_{s_1} \cdots \gamma_{s_m}\gamma_{\text{in}} \vdash_{M_{\text{pre}}} q_{t'_2}\gamma_s\gamma_{s_1} \cdots \gamma_{s_m} \vdash_{M_{\text{in}}} q_{t'_2}\gamma_{s'}\gamma_{s_1} \cdots \gamma_{s_m}$$

in M corresponds to an $\mathcal{R}$-path $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} t$ with derivation $[x_1/r_1], [x_2/r_2]$, where the above configurations of M correspond to the h-factorizations of t_1, t_2, and t. The locations x_1 and x_2 are not comparable w.r.t $\sqsubseteq$ because x_2 must be located in the s-part of the h-factorization of t_2. The situation

(a):



(b):

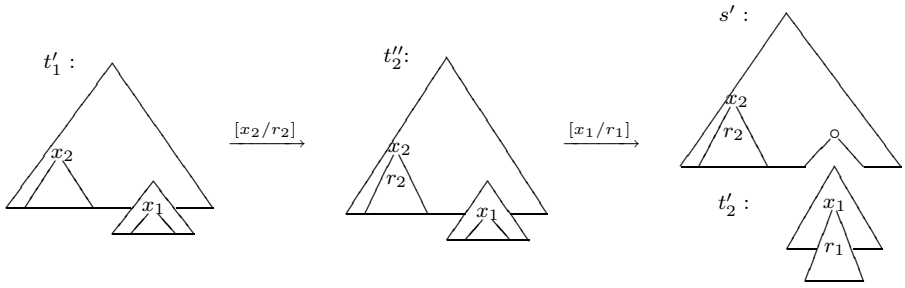


Figure 3.9: The infix PDA defined in Theorem 3.20 is in normal form.

looks as depicted in the Figure 3.9 (a). By exchanging the substitutions, we get a sequence as indicated in Figure 3.9 (b). In M this corresponds to a sequence as required in the definition of normal form:

$$q_{t'_1}\gamma_{s_1}\cdots\gamma_{s_m}\gamma_{\text{in}} \vdash_{M_{\text{pre}}} q_{t''_2}\gamma_{s_1}\cdots\gamma_{s_m} \vdash_{M_{\text{pre}}} q_{t'_2}\gamma_{s'}\gamma_{s_1}\cdots\gamma_{s_m}.$$

□

The graph of the infix PDA M constructed in the previous theorem has bounded tree-width since it is isomorphic to the graph of the given GTRS, which is assumed to be of bounded tree-width. From this property one can also deduce that M will never reach a configuration with more than $2m$ infix symbols on the stack, where m is the tree-width of the graph and infix symbols are the symbols of the stack alphabet that occur on the left hand side of an infix rule. If there are more than $2m$ infix symbols on the stack, then the corresponding infix rules can be applied in any order, generating an $(m \times m)$-grid as subgraph.

At first glance, one might think that infix pushdown automata with this additional property cannot generate more graphs than usual pushdown automata. The first approach would be to remove the infix rules by adding a component to the control states remembering the infix symbols that are on the stack. Then one could simulate the infix rules by changing this additional component in the control states. Since the possible number of infix symbols on the stack is bounded all this can be done with finite memory.

As the Examples 3.14 and 3.15 show, this approach must fail. The infix pushdown automata in these examples never reach a configuration with more than one infix symbol on the stack. Nevertheless, they generate graphs of unbounded tree-width and of unbounded degree, in particular, graphs that are not pushdown graphs. This implies that we still have to use that the generated graph is of bounded tree-width.

3.2.4 From Infix Pushdown Graphs to Pushdown Graphs

With Theorem 3.20 it remains to show that every infix pushdown graph of bounded tree-width is a pushdown graph. The idea is to find a structural characterization of infix pushdown graphs of bounded tree-width similar to the characterization of pushdown graphs stated in Proposition 3.3. For each vertex qw in the infix pushdown graph we define the connected component of qw in the graph restricted to vertices of length greater than or equal to the length of qw. The goal is to show that if the graph has bounded tree-width, then there are only finitely many isomorphism classes of these connected

components. The difference to the property of pushdown graphs given by Muller and Schupp is that we refer to the length of the vertices and not to the distance from a designated root vertex.

Once we have this structural characterization, this finite number of isomorphism classes can be used to obtain a usual PDA for generating the graph.

Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form. For $w, v \in \Gamma^*$ we define

$$w \preceq_M v \text{ iff } \forall q \in Q \ : \ (qw \in C(M) \Rightarrow qv \in C(M)),$$
$$w \approx_M v \text{ iff } w \preceq_M v \text{ and } v \preceq_M w.$$

The following lemma is immediate from the definition of $\approx_M$.

LEMMA 3.21 *The relation $\approx_M$ is an equivalence relation of finite index.*

For $n \in \mathbb{N}$ we define the graph $G_M|_n$ as the subgraph of G_M induced by the vertices $\{qw \in C(M) \mid |qw| \geq n\}$. For $qw \in C(M)$ the graph $G_M(qw)$ is the connected component of qw in $G_M|_{|qw|}$. The front $\Omega(qw)$ of $G_M(qw)$ is the set of all vertices in $G_M(qw)$ with the same length as qw, i.e.,

$$\Omega(qw) = \{pv \in Q\Gamma^* \mid pv \text{ is a vertex of } G_M(qw) \text{ and } |pv| = |qw|\}.$$

Let $pv \in C(M)$, qw be a vertex in $G_M(pv)$, and $\pi = (p_1x_1u_1, \ldots, p_nx_nu_n)$ be an undirected path in $G_M(pv)$ from pv to qw (i.e., $pv = p_1x_1u_1$ and $qw = p_nx_nu_n$) with $|u_i| = |v|$ for all $i \in \{1, \ldots, n\}$. The depth of π is $d(\pi) = max\{|x_i| \mid i \in \{1, \ldots, n\}\}$ and the number of suffix changes on π is $sc(\pi) = |\{i \in \{1, \ldots, n-1\} \mid u_i \neq u_{i+1}\}|$.

Let $\Pi(pv, qw)$ be the set of undirected paths in $G_M(pv)$ from pv to qw. For each such pair pv, qw we fix an arbitrary path $\pi(pv, qw) \in \Pi(pv, qw)$ with the following properties:

1. $\forall \pi \in \Pi(pv, qw) : sc(\pi(pv, qw)) \leq sc(\pi)$ and
2. $\forall \pi \in \Pi(pv, qw) : sc(\pi(pv, qw)) = sc(\pi) \Rightarrow d(\pi(pv, qw)) \leq d(\pi)$.

In Lemma 3.23 we show that vertices that belong to the same front have a connection of small depth. As a preparation we need the following lemma.

LEMMA 3.22 *Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form, and let $w, v, x \in \Gamma^*$ with $w \preceq_M v$. If $pw, qxw \in C(M)$ such that there is an undirected path π without suffix changes in $G_M(pw)$ from pw to qxw, then $pv, qxv \in C(M)$, and there is a path π' without suffix changes in $G_M(pv)$ from pv to qxv that has the same depth as π.*

Proof. We show by induction on the length of π that for each vertex $p'yw$ in π the vertex $p'yv$ is in $C(M)$. Then we obtain the path π' by replacing the suffix w in the π vertices by v.

Let k be the length of π. If $k = 0$, then pw is the only vertex on π. From $pw \in C(M)$, $w \preceq_M v$, and the definition of $\preceq_M$ we get $pv \in C(M)$.

So assume $k \geq 1$. It is sufficient to show that $qxv \in C(M)$. For the other vertices on π the claim holds by induction. From Lemma 3.17 we know that there exists a $q' \in Q$ such that $q'w \in C(M)$ and $q' \vdash^*_{M_{\text{pre}}} qx$. Then $w \preceq_M v$ implies $q'v \in C(M)$, and from $q' \vdash^*_{M_{\text{pre}}} qx$ we can conclude $qxv \in C(M)$. □

LEMMA 3.23 *Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form. There exists $d_M \in \mathbb{N}$ such that $d(\pi(pv, qw)) < d_M$ for each $pv \in C(M)$ and $qw \in \Omega(pv)$.*

Proof. Let

$$D = \{d(\pi(pv, qv)) \mid pv \in C(M), qv \in \Omega(pv), \text{ and } sc(\pi(pv, qv)) = 0\}.$$

Because $sc(\pi(pv, pv)) = 0$ for each $pv \in C(M)$, the set D is not empty. We show that D is finite. If $v \approx_M w$ for $v, w \in \Gamma^*$, then, by Lemma 3.22, $d(\pi(pv, qv)) = d(\pi(pw, qw))$ for all $p, q \in Q$. Since Q is finite and $\approx_M$ has finite index, the set D is finite. Thus, we can define $d_M = (max\ D) + 1$.

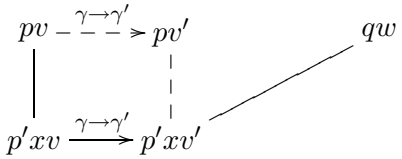
Now let $pv \in C(M)$, $qw \in \Omega(pv)$, and $k = sc(\pi(pv, qw))$. We show the claim by induction on k.

If $k = 0$, then $v = w$, and therefore $d(\pi(pv, qw)) \in D$. But then clearly $d(\pi(pv, qw)) < d_M$ by definition of d_M.

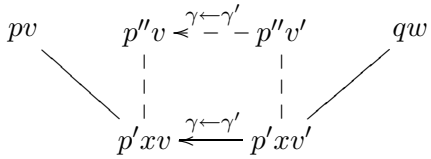
So consider the case $k \geq 1$. Let $p'xv'$ (with $|v'| = |v|$) be the first vertex on the path $\pi(pv, qw)$ without the suffix v. Then v' is of the form $y\gamma'z$ with $y\gamma z = v$ and $(\gamma, \sigma, \gamma') \in \Delta$ or $(\gamma', \sigma, \gamma) \in \Delta$ for some $\sigma \in \Sigma$ (remember that v can only be changed by infix rules since M is in normal form).

Case 1: $(\gamma, \sigma, \gamma') \in \Delta$. Then $v \preceq_M v'$ because from each configuration with v as suffix the corresponding configuration with v' as suffix can directly be reached using the infix transition. Hence, $pv' \in C(M)$. The direct predecessor of $p'xv'$ on $\pi(pv, qw)$ is $p'xv$ and there is a path from pv to $p'xv$ without suffix changes. This implies that there is a path from pv' to $p'xv'$ without suffix changes (by Lemma 3.22). This means that $sc(\pi(pv', qw)) < sc(\pi(pv, qw))$ and therefore $d(\pi(pv', qw)) < d_M$ by induction. Combining the path $\pi(pv', qw)$ and the transition $pv \vdash_{M_{\text{in}}} pv'$ we get a path from pv to qw with the same number of suffix changes as in $\pi(pv, qw)$. Furthermore, the depth of this path is less than d_M

and therefore $d(\pi(pv, qw)) < d_M$. This construction is illustrated in the picture below. The solid lines indicate the path $\pi(pv, qw)$ and the dashed lines the constructed part.



Case 2: $(\gamma', \sigma, \gamma) \in \Delta$. Then $v' \preceq_M v$ similar to case 1. Since $p'xv' \in C(M)$, there is $p'' \in Q$ such that $p''v' \in C(M)$ and there is a path without suffix changes from $p''v'$ to $p'xv'$ (by Lemma 3.17). This means $sc(\pi(p''v', qw)) < sc(\pi(pv, qw))$ and therefore $d(\pi(p''v', qw)) < d_M$ by induction. From $v' \preceq_M v$ and $p''v' \in C(M)$ we get $p''v \in C(M)$ and a path without suffix changes from $p''v$ to $p'xv$ (Lemma 3.22). This implies $sc(\pi(pv, p''v)) = 0$ and therefore $d(\pi(pv, p''v)) < d_M$. Now we combine the path $\pi(pv, p''v)$, the transition $p''v' \vdash_{M_{\text{in}}} p''v$, and the path $\pi(p''v', qw)$ and obtain a path from pv to qw with the same number of suffix changes as $\pi(pv, qw)$ and depth less than d_M. Hence, $d(\pi(pv, qw)) < d_M$. This construction is illustrated in the picture below, where again the solid lines indicate the path $\pi(pv, qw)$ and the dashed lines the constructed part.



□

After this general observation on the structure of infix pushdown graphs we turn to infix pushdown graphs of bounded tree-width. Using Lemmas 3.17 and 3.23 one can construct large brambles from large fronts $\Omega(qw)$. Therefore, these fronts have to be small in graphs of bounded tree-width.

LEMMA 3.24 *Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form such that G_M is of bounded tree-width. Then there exists $c \in \mathbb{N}$ such that $|\Omega(qw)| \leq c$ for each $qw \in C(M)$.*

Proof. Let m be the tree-width of G_M and let $l \in \mathbb{N}$. Assume that there is $qw \in C(M)$ such that $|\Omega(qw)| > |Q| \cdot |\Gamma|^{l \cdot d_M} \cdot l$ (with d_M from Lemma 3.23).

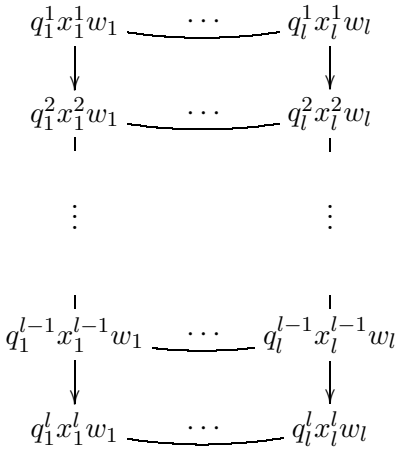
Under this assumption we construct a bramble of width l. This is sufficient because for $l \geq m+2$ we get a contradiction by Proposition 3.11 and therefore we know that the size of the fronts $\Omega(qw)$ is bounded by $|Q| \cdot |\Gamma|^{(m+2)\cdot d_M} \cdot (m+2)$.

Since $\Omega(qw)$ contains at least $|Q| \cdot |\Gamma|^{l \cdot d_M} \cdot l$ elements it is clear that $|w| > l \cdot d_M$. Let n be the size of the set

$$\{v \in \Gamma^{|w|-l\cdot d_M} \mid \exists y \in \Gamma^{l\cdot d_M}, p \in Q \ : \ pyv \in \Omega(qw)\}.$$

By a simple combinatorial argument $\Omega(qw)$ can contain at most $|Q|\cdot|\Gamma|^{l\cdot d_M}\cdot n$ elements and therefore $n \geq l$ because $|Q|\cdot|\Gamma|^{l\cdot d_M}\cdot n \geq |\Omega(qw)| > |Q|\cdot|\Gamma|^{l\cdot d_M}\cdot l$. Thus, we can find $q_1x_1w_1, \ldots, q_lx_lw_l \in \Omega(qw)$ with $|x_i| = l \cdot d_M$ and $w_i \neq w_j$ for $i, j \in \{1, \ldots, l\}$ with $i \neq j$. From Lemma 3.17 follows that there are $q_1^1, \ldots q_l^1 \in Q$ with $q_i^1 w_i \in C(M)$ and $q_i^1 \vdash^*_{M_{\text{pre}}} q_i x_i$. Let π_i be the path from $q_i^1 w_i$ to $q_i x_i w_i$ that we obtain from $q_i^1 \vdash^*_{M_{\text{pre}}} q_i x_i$.

For all $i \in \{2, \ldots, l-1\}$ and $j \in \{1, \ldots, l\}$ we pick $q_j^i \in Q$ and $x_j^i \in \Gamma^{i\cdot d_M}$ such that $q_j^i x_j^i w_j$ is on the path π_j and after $q_j^i x_j^i w_j$ there is no vertex of smaller length on π_j. Furthermore, we define $q_j^l = q_j$, $x_j^l = x_j$, and $x_j^1 = \varepsilon$. We get the following picture:



Note that for each $i, j, k \in \{1, \ldots, l\}$ the vertices $q_j^i x_j^i w_j$ and $q_k^i x_k^i w_k$ belong to the same front because they are of the same length, $q_j^i x_j^i w_j$ is connected to $q_j^l x_j^l w_j$ without visiting vertices of length smaller than $|q_j^i x_j^i w_j|$, $q_k^i x_k^i w_k$ is connected to $q_k^l x_k^l w_k$ without visiting vertices of length smaller than $|q_k^i x_k^i w_k|$, and $q_j^l x_j^l w_j$ and $q_k^l x_k^l w_k$ belong to the same front. Thus, the path $\pi(q_j^i x_j^i w_j, q_k^i x_k^i w_k)$ exists and has depth less than d_M (Lemma 3.23).

In this grid-like structure we define the ith row R_i and the ith column C_i as

$$R_i = \{pv \mid pv \in \pi(q_j^i x_j^i w_j, q_k^i x_k^i w_k) \text{ for some } j, k \in \{1, \ldots, l\}\},$$
$$C_i = \{pv \mid pv \in \pi_i\}.$$

Note that $C_i \cap C_j = \emptyset$ and $R_i \cap R_j = \emptyset$ for $i \neq j$. For the columns this follows because all the vertices in C_i have the suffix w_i, all the vertices in C_j have the suffix w_j, $|w_i| = |w_j|$, and $w_i \neq w_j$. For the rows this follows because the length of a vertex in R_i is $\geq |w_i| + i \cdot d_M + 1$ (by definition of R_i) and $< |w_i| + (i+1) \cdot d_M + 1$ (by Lemma 3.23).

Now we define the family $\mathcal{B} = (B_{i,j})_{i,j \in \{1,\ldots,l\}}$ of sets as

$$B_{i,j} = R_i \cup C_j.$$

From the definition of R_i and C_j it is clear that the sets $B_{i,j}$ are connected in G_M. Furthermore, $B_{i_1,j_1} \cap B_{i_2,j_2} \neq \emptyset$, for $i_1, j_1, i_2, j_2 \in \{1, \ldots, l\}$ because the vertex $q_{j_2}^{i_1} x_{j_2}^{i_1} w_{j_2}$ is in R_{i_1} and in C_{j_2} and therefore in $B_{i_1,j_1} \cap B_{i_2,j_2}$. Hence, $\mathcal{B}$ is a bramble. Suppose the set S of vertices covers $\mathcal{B}$. If $|S| < l$, then there is at least one i with $R_i \cap S = \emptyset$, and at least one j with $C_j \cap S = \emptyset$. But then also $B_{i,j} \cap S = \emptyset$. Thus, $\mathcal{B}$ has width at least l. □

This enables us to prove the desired structural characterization. We want to show that there cannot be infinitely many pairwise non-isomorphic connected components $G_M(qw)$ if the size of the fronts $\Omega(qw)$ is globally bounded. Two components $G_M(pv)$ and $G_M(qw)$ are isomorphic if there is an isomorphism mapping front vertices to front vertices. By abuse of notation we call this end-isomorphic.

Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form and let $pv, qw \in C(M)$. The graphs $G_M(pv)$ and $G_M(qw)$ are end-isomorphic iff there is a graph isomorphism between $G_M(pv)$ and $G_M(qw)$ such that vertices from $\Omega(pv)$ are mapped to vertices from $\Omega(qw)$. If $G_M(pv)$ and $G_M(qw)$ are end-isomorphic, then we write $G_M(pv) \sim_M G_M(qw)$.

LEMMA 3.25 *Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form. If G_M is of bounded tree-width, then $\sim_M$ has finite index.*

Proof. We show that $G_M(pv) \sim_M G_M(qw)$ if there is a bijection φ between $\Omega(pv)$ and $\Omega(qw)$ such that

(1) $p_1 v_1 \vdash_M^\sigma p_2 v_2$ iff $\varphi(p_1 v_1) \vdash_M^\sigma \varphi(p_2 v_2)$ for each $p_1 v_1, p_2 v_2 \in \Omega(pv)$, $\sigma \in \Sigma$,

(2) $\varphi(p_1 v_1) \in p_1 \Gamma^*$ for each $p_1 v_1 \in \Omega(pv)$, and

(3) if $\varphi(p_1 v_1) = p_1 w_1$ and $\varphi(p_2 v_1) = p_2 w_2$, then $w_1 = w_2$.

We extend this bijection to a graph isomorphism between $G_M(pv)$ and $G_M(qw)$. Let $p_1 x v_1$ be a vertex of $G_M(pv)$ with $|v_1| = |v|$. Then there

is $p' \in Q$ such that $p'v_1 \in C(M)$ and $p' \vdash_M^* p_1 x$ (by Lemma 3.17). Furthermore, $p'v_1 \in \Omega(pv)$ because $p'v_1$ and pv are both connected to $p_1 x v_1$ in $G_M(pv)$.

Let $p'w_1 = \varphi(p'v_1)$ and define $\varphi(p_1 x v_1) = p_1 x w_1$. It is easy to verify that φ is a graph isomorphism. Since the size of the sets $\Omega(qw)$ is bounded (Lemma 3.24) we can conclude that $\sim_M$ has finite index. □

Using the previous lemma, the proof of the following theorem is similar to the proof of Muller and Schupp [MS85] that finitely generated context free graphs are pushdown graphs.

THEOREM 3.26 *Let $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ be an infix PDA in normal form. If G_M is of bounded tree-width, then G_M is a pushdown graph.*

Proof. By Lemma 3.25 we know that $\sim_M$ has finite index. Let $\mathcal{G} = \{G_0, \ldots, G_n\}$ be a (minimal) set of representatives of the $\sim_M$-classes with $G_i = (V_i, E_i, \Sigma)$. For each $i \in \{1, \ldots, n\}$ there is a configuration $q_i w_i \in C(M)$ such that $G_i = G_M(q_i w_i)$. Let Ω_i be the front of G_i, i.e., $\Omega_i = \Omega(q_i w_i)$.

The rough idea of the transformation is the following: The PDA stores on the stack through which of the graphs G_i it went to reach a certain node. So, if the infix PDA is in a vertex pv, then the top stack symbol indicates to which end-isomorphism class the graph $G_M(pv)$ belongs, i.e., in which of the G_i the PDA currently is. The state of the new PDA represents the vertex on the front that pv corresponds to (under some fixed end-isomorphism).

There is one problem when we just use the G_i as stack symbols. In some G_i there might be a vertex on the front that has two successors u_1, u_2. These successors themselves are on the front of graphs that are end-isomorphic to G_{i_1} and G_{i_2}. Now, if $i_1 = i_2 = j$, then the PDA would only remember that it went to some graph that is end-isomorphic to G_j but it cannot distinguish the two different graphs. For this reason we have to work with a larger set of stack symbols that arises from the "second level subgraphs" (see [MS85]).

For $i \in \{0, \ldots, n\}$ let G_i' be the subgraph of G_i that is induced by the set $V_i \setminus \Omega_i$. The graph G_i' has several connected components (or at least one). Let m_i denote the number of the connected components of G_i' and call them $H_{i,0}, \ldots, H_{i,m_i}$. In [MS85] these $H_{i,j}$ are called second level subgraphs. Since we deleted the front of G_i to obtain the graphs $H_{i,j}$ each of the $H_{i,j}$ is end-isomorphic to one of the G_k. To avoid the problem sketched above, we introduce for each $i \in \{0, \ldots, n\}$, $j \in \{0, \ldots, m_i\}$ a stack symbol $z_{i,j}^k$ if $H_{i,j} \sim_M G_k$. Furthermore, we fix an end-isomorphism $\phi_{i,j}^k : H_{i,j} \to G_k$. The

initial configuration is not on the front of a second level subgraph. Therefore, we also introduce an extra initial stack symbol z_0.

As mentioned above, the states of the PDA represent the position on the front. Thus, for each $i \in \{0, \ldots, n\}$ we fix a bijection $\psi_i : \{x_0, \ldots, x_{|\Omega_i|-1}\} \to \Omega_i$ and we let $l = (\max\{|\Omega_i| \mid i \in \{0, \ldots, n\}\}) - 1$.

To define the initial state, we assume that $G_M(q_{\text{in}}\gamma_{\text{in}}) \sim_M G_0$ under an end-isomorphism ϕ and that $\psi_0(x_0) = \phi(q_{\text{in}}\gamma_{\text{in}})$, i.e., the initial vertex is mapped to the vertex of G_0 that corresponds to x_0.

Now we are ready to define a PDA $M' = (Q', \Sigma, \Gamma', \Delta', x_0, z_0)$ such that G_M and $G_{M'}$ are isomorphic.

- $Q' = \{x_0, \ldots, x_l\}$.
- $\Gamma' = \{z^k_{i,j} \mid i \in \{0, \ldots, n\}, j \in \{0, \ldots, m_i\}, H_{i,j} \sim_M G_k\} \cup \{z_0\}$.
- Δ' is defined as follows. The first two items treat the cases when the top stack symbol is z_0. For the general case there are three items because the stack length can remain the same, increase, or decrease.
  - $(x_r, \sigma, z_0, x_s, z_0) \in \Delta'$ iff $(\psi_0(x_r), \sigma, \psi_0(x_s)) \in E_0$.
  - $(x_r, \sigma, z_0, x_s, z^k_{0,j} z_0) \in \Delta'$ iff $((\psi_0(x_r), \sigma, u) \in E_0$ such that u is a vertex of $H_{0,j} \sim_M G_k$ and $\psi_k(x_s) = \phi^k_{0,j}(u)$.
  - $(x_r, \sigma, z^k_{i,j}, x_s, z^k_{i,j}) \in \Delta'$ iff $(\psi_k(x_r), \sigma, \psi_k(x_s)) \in E_k$.
  - $(x_r, \sigma, z^k_{i,j}, x_s, z^{k'}_{i',j'} z^k_{i,j}) \in \Delta'$ iff $i' = k$, $(\psi_k(x_r), \sigma, u) \in E_k$ such that u is a vertex of $H_{i',j'} \sim_M G_{k'}$ and $\psi_q k'(x_s) = \phi^{k'}_{i',j'}(u)$.
  - $(x_r, \sigma, z^k_{i,j}, x_s, \varepsilon) \in \Delta'$ iff $(u, \sigma, \psi_i(x_s)) \in E_i$ such that u is a vertex of $H_{i,j} \sim_M G_k$ and $\psi_k(x_r) = \phi^k_{i,j}(u)$. □

3.2.5 Clique-Width

The notion clique-width of graphs was introduced in [CO00] for finite graphs and in [Cou00] for infinite graphs. Although GTR graphs are directed, we consider undirected clique-width. This means that we refer to the undirected graph obtained from a directed graph by replacing every directed edge with an undirected one and removing the self loops.

To define clique-width of graphs one considers (infinite) graph expressions over colored graphs. In a colored graph each vertex is assigned a color, where the colors are natural numbers. These expressions are built up from the constant graph with one vertex, colored 1, and operators for disjoint union of graphs, recoloring vertices, and inserting edges between vertices of

different colors. The clique-width of a graph G is the minimal number of colors that is needed in such an expression defining G. Thus, the clique-width of a graph gives information on how many classes of vertices we have to distinguish at most in the process of building up the graph with the above mentioned operations.

For the definition of clique-width we proceed as [Cou00] and use infinite terms over the ranked alphabet consisting of the operators mentioned above, including $\perp$ denoting the empty graph. These terms are ranked trees as defined in Chapter 2, but since these trees are used for a different purpose we call them terms to clearly distinguish them from the trees we are using in ground tree rewriting systems.

For $m \geq 1$ we define the ranked alphabet

$$F_m = \{\oplus, \rho_{i\to j}, \eta_{i,j}, \underline{1}, \perp \mid 1 \leq i, j \leq m \text{ and } i \neq j\},$$

where $\oplus$ is of rank 2, $\rho_{i\to j}$ and $\eta_{i,j}$ are of rank 1, and $\underline{1}$ and $\perp$ are of rank 0. By $T^{\infty}_{F_m}$ we denote the set of finite and infinite terms (ranked trees) over F_m.

For a term $\tau \in T^{\infty}_{F_m}$ we are interested in the graph G_τ defined by τ. For this purpose, we first define G_τ for finite terms $\tau \in T_{F_m}$. We use an inductive definition that produces on each stage a graph, and a coloring function mapping vertices of the graph to natural numbers. The colored graph defined by τ is called $\mathrm{val}(\tau)$.

So, for $\tau \in T_{F_m}$ we define $\mathrm{val}(\tau)$ inductively as follows:

- $\mathrm{val}(\perp) = (\emptyset, \emptyset, c)$ is the empty graph with $c : \emptyset \to \mathbb{N}$.
- $\mathrm{val}(\underline{1}) = (\{1\}, \emptyset, c)$ with $c : \{\underline{1}\} \to \mathbb{N}$ and $c(1) = 1$.
- If $\tau = \tau_1 \oplus \tau_2$, then $\mathrm{val}(\tau)$ is the disjoint union of $\mathrm{val}(\tau_1)$ and $\mathrm{val}(\tau_2)$. That is, if $\mathrm{val}(\tau_1) = (V_1, E_1, c_1)$ and $\mathrm{val}(\tau_1) = (V_2, E_2, c_2)$, then $\mathrm{val}(\tau) = (V, E, c)$ with
  - $V = (V_1 \times \{1\}) \cup (V_2 \times \{2\})$,
  - $E = \{((u, i), (v, i)) \mid i \in \{1, 2\} \text{ and } (u, v) \in E_i\}$, and
  - $c(v, i) = c_i(v)$ for $i \in \{1, 2\}$ and $v \in V_i$.
- If $\tau = \rho_{i\to j}(\tau_1)$ and $\mathrm{val}(\tau_1) = (V, E, c)$, then $\mathrm{val}(\tau) = (V, E, c')$ with $c'(v) = c(v)$ if $c(v) \neq i$ and $c'(v) = j$ if $c(v) = i$. So, $\mathrm{val}(\tau)$ is obtained from $\mathrm{val}(\tau_1)$ by changing the color i into the color j for each vertex colored i.

- If $\tau = \eta_{i,j}(\tau_1)$ and $\mathrm{val}(\tau_1) = (V, E, c)$, then $\mathrm{val}(\tau) = (V, E', c)$ with $E' = E \cup \{\{u, v\} \mid c(u) = i \text{ and } c(v) = j\}$. This operation inserts edges between the vertices of color i and the vertices of color j.

The graph G_τ is defined from $\mathrm{val}(\tau)$ by omitting the coloring function, i.e., $G_\tau = (V, E)$ if $\mathrm{val}(\tau) = (V, E, c)$ for some coloring function c.

To define G_τ for an infinite term τ we approximate τ by an increasing sequence of finite terms. For $\tau_1, \tau_2 \in T^\infty_{F_m}$ the relation $\tau_1 \prec \tau_2$ holds iff

- $D_{\tau_1} \subseteq D_{\tau_2}$ and
- for all $x \in D_{\tau_1}$ either $\tau_1(x) = \tau_2(x)$ or $\tau_1(x) = \bot$.

There is a direct connection between this partial order on the set of finite terms over F_m and the subgraph relation.

REMARK 3.27 If $\tau_1, \tau_2 \in T_{F_m}$ with $t_1 \prec t_2$, then G_{τ_1} is a subgraph of G_{τ_2}.

For $n \in \mathbb{N}$ the n-truncation $\tau^{(n)}$ of $\tau \in T^\infty_{F_m}$ is the unique finite term from T_{F_m} with domain $D_{\tau^{(n)}} = \{x \in D_\tau \mid |x| \leq n\}$ such that for all $x \in D_{\tau^{(n)}}$

$$\tau^{(n)}(x) = \begin{cases} \tau(x) \text{ if } |x| < n, \\ \bot \text{ if } |x| = n. \end{cases}$$

With this definition an infinite term $\tau \in T^\infty_{F_m}$ is the unique upper bound w.r.t. $\prec$ of the increasing sequence

$$\tau^{(0)} \prec \tau^{(1)} \prec \tau^{(2)} \prec \cdots$$

Now we can define G_τ for an infinite term $\tau \in T^\infty_{F_m}$ by

$$G_\tau = \bigcup_{n \in \mathbb{N}} G_{\tau^{(n)}}.$$

Here the union does not mean disjoint union but true union of the vertex and edge sets. This graph is the least upper bound of the sequence $G_{\tau^{(0)}}, G_{\tau^{(1)}}, G_{\tau^{(2)}}, \ldots$ w.r.t. the subgraph relation.

The clique-width of a graph G is the minimal m such that G^{und} is isomorphic to G_τ for some $\tau \in T^\infty_{F_m}$. If no such m exists, then the clique-width of G is ∞.

EXAMPLE 3.28 We define an infinite term $\tau \in T^\infty_{F_5}$ such that G_τ is isomorphic to the undirected version of the "triangle graph" from Figure 3.1. For

this purpose we first define finite terms τ_n for $n \geq 0$ such that $\mathrm{val}(\tau_n)$ is the colored graph

$$1 - \underbrace{3 - \cdots - 3}_{n \text{ times}} - 2$$

where the numbers correspond to the colors of the vertices in $\mathrm{val}(\tau_n)$. These terms are used to add the horizontal layers to the graph. The terms τ_n and τ are shown in Figure 3.10.

For formal reasons the graph G_τ is defined in a top-down manner by approximating it with growing n-truncations. To understand how the graph is built up from τ one should read the term bottom-up. To illustrate this idea Figure 3.10 also shows the graph defined by the subterm $\tau^{\downarrow x_n}$ where x_n is the location where the finite term τ_n is merged into τ. This graph is shown without the names of the vertices but with the intended colors although, formally, this is not correct because we defined the graph of an infinite term without coloring function. □

Similar to Proposition 3.9 there is a relation between the clique-width of an infinite graph and the clique-width of its finite subgraphs, where in this case only induced subgraphs are considered.

PROPOSITION 3.29 ([Cou00]) *A graph G has bounded clique-width iff there exists $m \in \mathbb{N}$ such that $cw(H) \leq m$ for each induced finite subgraph of G.*

The relation of tree-width and clique-width is well studied for finite graphs. Using Propositions 3.9 and 3.29 it is rather easy to transfer these results to infinite graphs.

PROPOSITION 3.30 ([CO00]) *Let G be a graph. If G has bounded tree-width, then G has bounded clique-width.*

PROPOSITION 3.31 ([GW00]) *Let G be a graph. If G is of bounded clique-width and there is an $n \in \mathbb{N}$ such that $K_{n,n}$ is not a subgraph of G, then G is of bounded tree-width.*

These propositions guarantee that in the absence of large bipartite subgraphs $K_{n,n}$ the notions of bounded tree-width and of bounded clique-width are the same. The following lemma shows that GTR graphs do not contain large bipartite subgraphs. Note that this is not clear in the first place because the examples from Section 2.3 show that GTR graphs may have unbounded degree.

LEMMA 3.32 *Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be a GTRS. Then $K_{m,m}$ is not a subgraph of $G_\mathcal{R}$ for all $m > 2 \cdot |R| + 1$.*

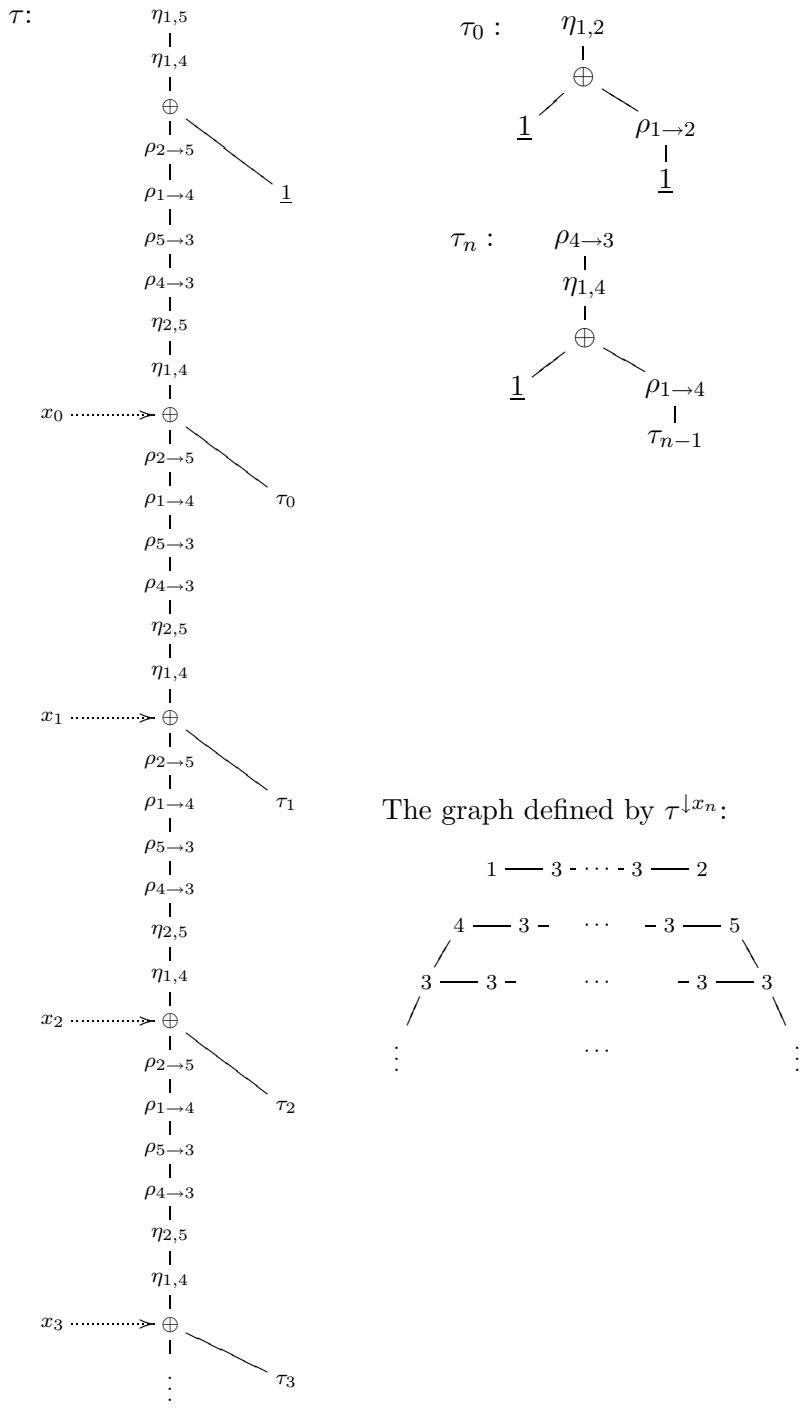


Figure 3.10: An infinite term $\tau \in T^{\infty}_{F_m}$ to define the graph from Figure 3.1

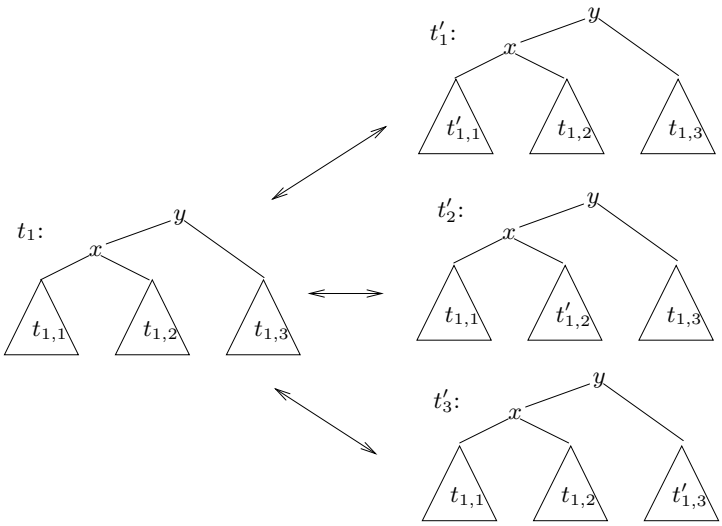


Figure 3.11: Illustration for the proof of Lemma 3.32

Proof. Let $m > 2 \cdot |R| + 1$ and suppose that $K_{m,m}$ is a subgraph of $G_{\mathcal{R}}$. Then there are different trees $t_1, \ldots t_m, t'_1, \ldots, t'_m \in V_{\mathcal{R}}$ such that $(t_i, t'_j) \in E$ or $(t'_j, t_i) \in E$ for all $i, j \in \{1, \ldots, m\}$. For all $i, j \in \{1, \ldots, m\}$ let $t_{i,j}, t'_{i,j} \in T_A$ and $x_{i,j} \in D_{t_i}$ such that

- $(t_i)^{\downarrow x_{i,j}} = t_{i,j}$,
- $t'_j = [x_{i,j}/t'_{i,j}]$,
- $t_{i,j} \overset{\sigma}{\hookrightarrow} t'_{i,j} \in R$ or $t'_{i,j} \overset{\sigma}{\hookrightarrow} t_{i,j} \in R$.

There are only $|R|$ possible pairs of $t_{i,j}$ and $t'_{i,j}$ because these pairs must form a rewriting rule from R. Hence, by the choice of m, for each $i \in \{1, \ldots, m\}$ there must be $j_1, j_2, j_3 \in \{1, \ldots, m\}$ such that x_{i,j_1}, x_{i,j_2}, and x_{i,j_3} are pairwise incomparable. W.l.o.g. we assume that $i = 1$, $j_1 = 1$, $j_2 = 2$, and $j_3 = 3$. Let $x, y, z \in \mathbb{N}^*$ be such that x is the maximal common prefix of $x_{1,1}$ and $x_{1,2}$, y is the maximal common prefix of $x_{1,2}$ and $x_{1,3}$, and z is the maximal common prefix of $x_{1,1}$ and $x_{1,3}$. Again w.l.o.g. we can assume that $y = z$ and $y \sqsubseteq x$. This can be obtained by reordering t'_1, t'_2 and t'_3. Then the situation is as indicated in Figure 3.11, where the double-sided arrows mean that there is an edge in $G_{\mathcal{R}}$ between the trees in either direction.

Now we want to show that it is not possible to connect another $2|R|$ trees to t'_1, t'_2, and t'_3. This is done by an analysis of which of the subtrees

at $x_{1,1}$, $x_{1,2}$, and $x_{1,3}$ are affected by the rewritings that generate the edges between t_l and t'_1, t'_2, and t'_3 for $l \in \{2, \ldots, m\}$.

First suppose that for each $l \in \{2, \ldots, m\}$ there is a $j \in \{1, 2, 3\}$ such that the rewriting that connects t_l and t'_j affects all the subtrees at $x_{1,1}$, $x_{1,2}$, and $x_{1,3}$, i.e., the substitution takes place at a prefix of y. Since $m > 2|R|+1$ there must be at least two of these rewritings that use the same rule. This is a contradiction because the trees t'_1, t'_2, and t'_3 only differ in the subtrees at $x_{1,1}$, $x_{1,2}$, and $x_{1,3}$ (so if the same rule is used for say t_{l_1}, t_{l_2} at a prefix of y, then $t_{l_1} = t_{l_2}$).

Therefore, there is an $l \in \{2, \ldots, m\}$ such that for each $j \in \{1, 2, 3\}$ one of the subtrees at $x_{1,1}$, $x_{1,2}$, or $x_{1,3}$ is not affected by the rewriting generating an edge between t_l and t'_j. This means that for all $j \in \{1, 2, 3\}$ either $y \sqsubset x_{l,j}$ or y and $x_{l,j}$ are incomparable.

Consider the case that y and $x_{l,1}$ are incomparable. Then, since t_l is rewritten to t'_1 using a substitution at a location incomparable to y, we have $t_l^{\downarrow x_{1,1}} = t'_{1,1}$ and $t_l^{\downarrow x_{1,3}} = t_{1,3}$. Hence, the substitution transforming t_l into t'_3 must take place at a prefix of y, contradicting the choice of l. Analogously, we get a contradiction if y and $x_{l,2}$ or y and $x_{l,3}$ are incomparable.

Up to now we know that $y \sqsubset x_{l,j}$ for all $j \in \{1, 2, 3\}$, i.e., the substitutions transforming t_l into t'_1, t'_2, and t'_3, respectively, take place strictly below y. We show that this also leads to a contradiction.

Case 1: $x_{l,1}$ is not comparable to x. In the picture, this corresponds to the case that $x_{l,1}$ is located somewhere below y but not in the subtree containing x. Then $t_l^{\downarrow x} = t_1'^{\downarrow x}$ and hence $x_{l,2}$ and $x_{l,3}$ must be comparable to x because the subtree of t_l at x has to be changed to obtain t'_2 and t'_3. This, in turn, implies that $t_l^{\downarrow x_{1,3}} = t_2'^{\downarrow x_{1,3}} = t_{1,3}$ and $t_l^{\downarrow x_{1,3}} = t_3'^{\downarrow x_{1,3}} = t'_{1,3}$, which is a contradiction.

Case 2: $x_{l,1}$ is comparable to x. In the picture, this corresponds to the case that $x_{l,1}$ is located somewhere below y in the subtree containing x. Then $t_l^{\downarrow x_{1,3}} = t_2'^{\downarrow x_{1,3}} = t_{1,3}$ and therefore the substitution at $x_{l,3}$ rewriting t_l into t'_3 must change $t_l^{\downarrow x_{1,3}}$. Thus, $x_{l,3}$ cannot be comparable to x. We can conclude that $t_l^{\downarrow x} = t_3'^{\downarrow x} = t_1^{\downarrow x}$. In combination with $t_l^{\downarrow x_{1,3}} = t_{1,3}$ this yields $t_l^{\downarrow y} = t_1^{\downarrow y}$. But this is the only subtree where t_l and t_1 could be different and therefore $t_l = t_1$, contradicting the assumption that $l \in \{2, \ldots, m\}$. □

Now, using Propositions 3.30 and 3.31, we can conclude the following.

THEOREM 3.33 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS. Then $G_{\mathcal{R}}$ has bounded clique-width iff $G_{\mathcal{R}}$ has bounded tree-width.*

Proof. Proposition 3.30 states that bounded tree-width implies bounded clique width. According to Lemma 3.32 GTR graphs do not have the complete bipartite graph $K_{n,n}$ as subgraph if n is large enough. Therefore, if $G_{\mathcal{R}}$ has bounded clique-width, then $G_{\mathcal{R}}$ has bounded tree-width by Proposition 3.31. □

3.2.6 Characterization of GTR Graphs of Bounded Width

The following theorem summarizes the main results we have obtained in this section.

THEOREM 3.34 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS. Then the following statements are equivalent.*

(i) $G_{\mathcal{R}}$ *is of bounded clique-width.*

(ii) $G_{\mathcal{R}}$ *is of bounded tree-width.*

(iii) $G_{\mathcal{R}}$ *is a pushdown graph.*

Proof. The equivalence of (i) and (ii) is stated in Theorem 3.33. From Proposition 3.13 we know that (iii) implies (ii) and, by Theorems 3.20 and 3.26, (ii) implies (iii). □

With this theorem it is easy to show that the graph from Figure 3.1 is not a GTR graph. In the subsection on pushdown automata we have seen that this graph has infinitely many non-isomorphic ends and therefore is no pushdown graph (Example 3.4 and Proposition 3.3). In the subsection on tree-width we have seen that the graph has bounded tree-width (Example 3.8). Hence, it cannot be a GTR graph according to the above theorem.

A result similar to Theorem 3.34 was obtained for RGTR graphs in [Col02]. This result is stated in Subsection 3.3.1 on the comparison of GTR graphs to prefix recognizable and equational graphs.

3.3 Comparison to Other Classes of Graphs

The results from the previous section allow a precise comparison of the classes of GTR graphs and pushdown graphs. The aim of this section is to classify the position of the class of GTR graphs in the known hierarchy of other natural classes of infinite graphs.

Figure 3.12 shows a synopsis of the results from this and the previous section. The edge annotations in italic face describe structural properties to pass from the graph class at the upper end of the edge to the graph class at the lower end of the edge. For example, the edge between GTR graphs and pushdown graphs is annotated with *bounded tree-width*, meaning that restricting the class of GTR graphs to GTR graphs of bounded tree-width yields the class of pushdown graphs.

As can be seen from the diagram we compare the class of GTR graphs with the classes of pushdown graphs, equational graphs, prefix recognizable graphs, and automatic graphs. In this picture we completely leave out the hierarchy of process rewriting graphs [May00], which has attracted attention for the formal description of parallel processes. The structure of the graphs from this hierarchy has mainly been analyzed w.r.t. bisimulation whereas our comparison is up to isomorphism. Therefore, a comparison with these graphs would lead too far away from the central topic of this thesis.

3.3.1 Prefix Recognizable and Equational Graphs

Prefix recognizable graphs (PR graphs) [Cau96] extend pushdown graphs in a similar way as RGTR graphs extend GTR graphs. The vertices of a prefix recognizable graph are words over some finite alphabet Γ and the edge relation is generated by rewriting rules of the form

$$L \cdot K \overset{\sigma}{\hookrightarrow} L' \cdot K \text{ with regular languages } L, L', K \subseteq \Gamma^*.$$

There is a σ-labeled edge between $u, v \in \Gamma^*$ if there is a rewriting rule $L \cdot K \overset{\sigma}{\hookrightarrow} L' \cdot K$ and u, v are of the form $u = u_1 w$, $v = v_1 w$ with $u_1 \in L$, $v_1 \in L'$, and $w \in K$.

There are a lot of different characterizations of the class of PR graphs (see [Blu01]). From the definition it is obvious that the class of PR graphs is a proper extension of the class of pushdown graphs. So we have two different extensions of pushdown graphs, namely GTR graphs and PR graphs. The question is how these two extensions are related to each other. There are several results on PR graphs that allow to answer this question. We use the following proposition.

PROPOSITION 3.35 ([Blu01]) *If a PR graph G is of finite degree, then it has bounded tree-width.*

With this proposition it is easy to see that the intersection of the classes of GTR graphs and PR graphs is exactly the class of pushdown graphs.

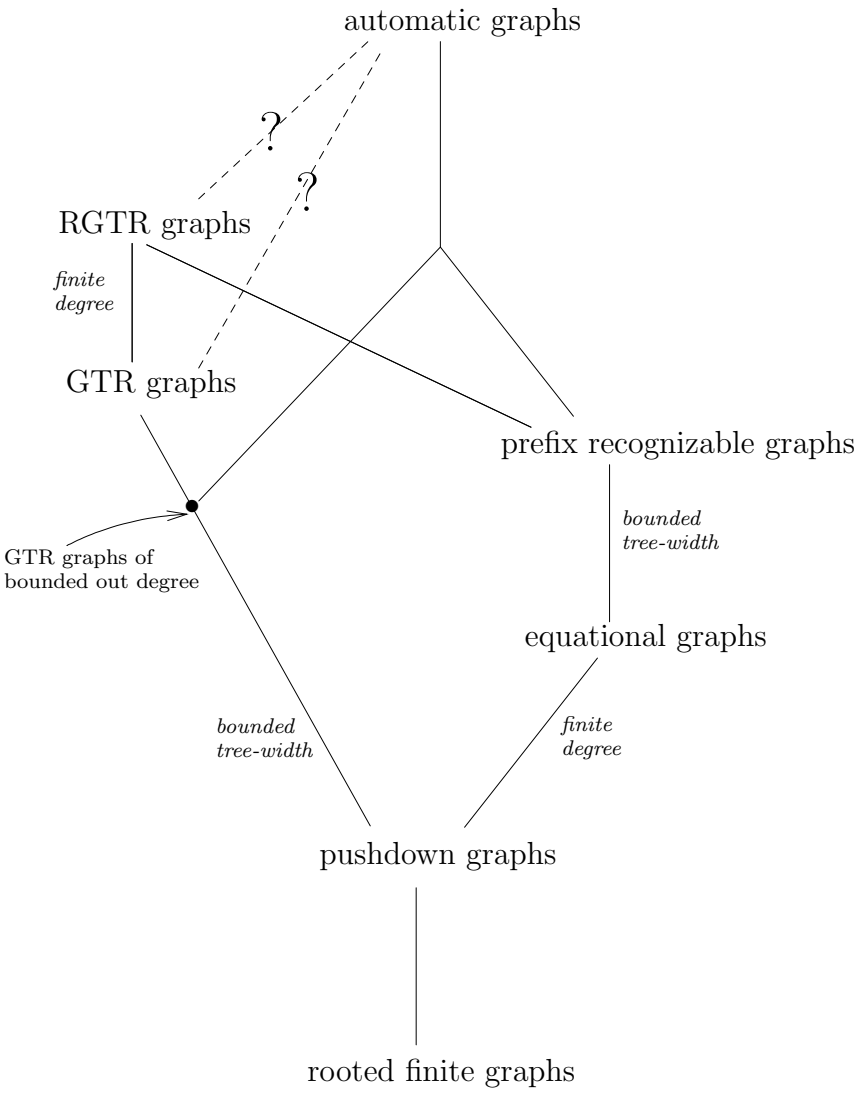


Figure 3.12: Hierarchy of graph classes

THEOREM 3.36 *A graph G is a pushdown graph iff it is a GTR graph and a PR graph.*

Proof. The direction from the left to the right is obvious. So, assume that G is a GTR graph and a PR graph. Since G is a GTR graph it is of finite degree. Then, by Proposition 3.35 G has bounded tree-width. This implies that G is a pushdown graph (Theorem 3.34). □

Equational graphs [Cou89] are graphs defined by equations of hyperedge-replacement grammars. The following characterization of equational graphs was given by Barthelmann.

PROPOSITION 3.37 ([Bar98]) *A graph G is equational iff it is a PR graph of bounded tree-width.*

In particular equational graphs are PR graphs and therefore we can conclude the following.

COROLLARY 3.38 *A graph G is a pushdown graph iff it is a GTR graph and an equational graph.*

In [Col02] a result for RGTR graphs similar to Theorem 3.34 is obtained.

THEOREM 3.39 ([Col02]) *If G is an RGTR graph of bounded tree-width, then G is an equational graph.*

On the other hand, the class of RGTR graphs obviously contains the class of PR graphs. An open question is how PR graphs may be characterized inside the class of RGTR graphs similar to the above characterization for equational graphs by bounded tree-width.

We think that bounded clique-width should give such a characterization, i.e., every RGTR graph of bounded clique width is a PR graph. But there is no evidence for this claim to be true, except that it would "fit into the picture".

3.3.2 Automatic Graphs

A graph is called automatic ([BG00]) or synchronized rational ([FS93]) if its edge relation can be recognized by a finite automaton. In the sequel we formalize this concept.

Let Γ be a finite alphabet and let $\diamond$ be a symbol that is not in Γ. For two words $u, v \in \Gamma^*$ with $u = a_1 \cdots a_m$, $v = b_1 \cdots b_n$ with $a_i, b_j \in \Gamma$ let

$$u \wedge v = \begin{cases} \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \cdots \begin{bmatrix} a_m \\ b_m \end{bmatrix} \begin{bmatrix} \diamond \\ b_{m+1} \end{bmatrix} \cdots \begin{bmatrix} \diamond \\ b_n \end{bmatrix} & \text{if } m < n, \\ \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ b_n \end{bmatrix} \begin{bmatrix} a_{n+1} \\ \diamond \end{bmatrix} \cdots \begin{bmatrix} a_m \\ \diamond \end{bmatrix} & \text{if } m \geq n. \end{cases}$$

Note that $u \wedge v$ is a word over the alphabet $(\Gamma \cup \{\diamond\})^2$. A relation $R \subseteq \Gamma^* \times \Gamma^*$ is called automatic if the language $L_R \subseteq ((\Gamma \cup \{\diamond\})^2)^*$ defined as

$$L_R = \{u \wedge v \mid (u, v) \in R\}$$

is regular. For an alphabet Σ, a family $(E_\sigma)_{\sigma \in \Sigma}$ of automatic relations $E_\sigma \subseteq \Gamma^* \times \Gamma^*$ defines a graph $G = (V, E, \Sigma)$ with

- $V = \{v \in \Gamma^* \mid \exists u \in \Gamma^*, \sigma \in \Sigma \ : \ (u, v) \in E_\sigma \text{ or } (v, u) \in E_\sigma\}$
- $E = \{(u, \sigma, v) \mid \sigma \in \Sigma \text{ and } (u, v) \in E_\sigma\}$.

A graph is automatic if it is isomorphic to such a graph with automatic edge relations.

Example 3.40 The "triangle graph" from Figure 3.1 is an automatic graph. In Figure 3.1 the vertices are coded by pairs of words. Note that there is no connection between these pairs and the automatic relations we are looking for. We merge such a pair into one word by concatenating the two words, i.e., the pair (X^i, Y^j) becomes the single word $X^i Y^j$. Now we define the edge relations for the different edge labels as follows:

- $E_0 = \{(X^i, X^{i+1}) \mid i \geq 0\}$,
- $E_1 = \{(X^i Y^j, X^{i-1} Y^{j+1}) \mid i \geq 1, j \geq 0\}$,
- $E_2 = \{(Y^j, Y^{j-1}) \mid j \geq 1\}$.

It is not difficult to verify that these three relations are automatic. □

This example shows that there are automatic graphs that are no GTR graphs. In the following we deal with the other inclusion, i.e., with the question whether the class of GTR graphs is contained in the class of automatic graphs. As indicated by the dashed lines in the diagram from Figure 3.12 we do not know the answer to this question. But we do have a partial result, namely that GTR graphs of bounded out degree are automatic graphs.

In the sequel we present this result and end with a brief discussion on the difficulties connected with the general question.

In Subsection 3.1.2 we have seen how to code trees with a small independence degree by words. This coding enabled us to construct an infix pushdown automaton for simulating the tree rewriting on these codings (see Subsection 3.2.3). The edge relations defined by infix pushdown automata are quite simple automatic relations. To use the power of automatic relations we extend the idea of h-factorizations and obtain a larger class of trees that can be coded as words with these generalized h-factorizations such that the tree rewriting relation is an automatic relation on these codings. The extension of h-factorizations is obtained by allowing more than one connection point labeled with $\circ$ in the factors of the tree.

Recall that, given a ranked alphabet A, we defined $A_\circ$ to be the alphabet A augmented by the symbol $\circ$ of rank 0. To formalize generalized h-factorizations we define for $h \in \mathbb{N}$ the sets

$$T^h_{A_\circ} = \{t \in T_{A_\circ} \mid \text{height}(t) < 2h \text{ and } |x| = h \text{ for each } x \in D_t \text{ with } t(x) = \circ\}$$

and

$$\Gamma^h_A = T^h_{A_\circ} \cup \{t \in T_A \mid \text{height}(t) < 2h\}.$$

The generalized h-factorization $\text{gf}_h(t)$ of $t \in T_A$ is a word over Γ^h_A defined as follows.

- If $\text{height}(t) < 2h$, then $\text{gf}_h(t) = t$.
- If $\text{height}(t) \geq 2h$, then let $x_1, \ldots, x_n \in D_t$ be those locations with $|x_i| = h$, $\text{height}(t^{\downarrow x_i}) \geq h$, and $x_1 <_{\text{lex}} \cdots <_{\text{lex}} x_n$, where $<_{\text{lex}}$ denotes the usual lexicographical order on $\mathbb{N}^*$. Define

$$\text{gf}_h(t) = t[x_1/\circ, \ldots, x_n/\circ] \cdot \text{gf}_h(t^{\downarrow x_1}) \cdots \text{gf}_h(t^{\downarrow x_n}).$$

To pass from generalized h-factorizations back to trees we define a mapping assigning a tree from $T_{A_\circ}$ to each prefix of a generalized h-factorization. Let $w = \text{gf}_h(t)$ be a generalized h-factorization and let v be a nonempty prefix of w. Then $\text{tree}(v) = v$ if $|v| = 1$. If $|v| > 1$ with $v = v_1 \cdot s$, then let $s_1 = \text{tree}(v_1)$ and define $\text{tree}(v) = s_1[x/s]$, where $x \in D_{s_1}$ is the smallest location in the lexicographical ordering with $s_1(x) = \circ$. With this definition we get $\text{tree}(\text{gf}_h(t)) = t$.

The width of generalized h-factorizations, defined below, is a measure for the amount of memory needed by a finite automaton that checks whether the tree corresponding to a factorization is accepted by a tree automaton.

Intuitively, the width of a generalized h-factorization $\mathrm{gf}_h(t)$ is the maximal number of "pending $\circ$-symbols" when reconstructing t from $\mathrm{gf}_h(t)$ step by step. Formally, this looks as follows. For $t \in T_{A_\circ}$ let $|t|_\circ$ be the number of occurrences of $\circ$ in t:

$$|t|_\circ = |\{x \in D_t \mid t(x) = \circ\}|.$$

The width of a generalized h-factorization w is

$$\mathrm{width}(w) = \max\{|\mathrm{tree}(v)|_\circ \mid v \text{ prefix of } w\}.$$

If v is a prefix of a generalized h-factorization $\mathrm{gf}_h(t)$, then $\mathrm{tree}(v)$ agrees with t on all locations $x \in D_{\mathrm{tree}(v)}$ with $\mathrm{tree}(v)(x) \neq \circ$ and at each location x with $\mathrm{tree}(v)(x) = \circ$ there is a subtree $t^{\downarrow x}$ of height at least h in t. Thus, if $\mathrm{width}(\mathrm{gf}_h(t)) = c$, then there exist locations $x_1, \ldots, x_c \in D_t$ that are pairwise incomparable w.r.t. $\sqsubseteq$ such that $\mathrm{height}(t^{\downarrow x_i}) \geq h$ for all $i \in \{1, \ldots, c\}$. This observation leads to the following lemma.

LEMMA 3.41 *Let $\mathcal{R}$ be a GTRS. If the out degree of $G_\mathcal{R}$ is less than c, then $\mathrm{width}(\mathrm{gf}_h(t)) < c$ for each $h > h_\mathcal{R}$ and $t \in T(\mathcal{R})$.*

Proof. As explained above, t contains at least c independent subtrees of height at least h if $\mathrm{width}(\mathrm{gf}_h(t)) \geq c$. If $h > h_\mathcal{R}$, then these independent subtrees have to be generated by independent applications of rewriting rules. So, there must be a tree in $T(\mathcal{R})$ to which rewriting rules can be applied at c different locations. This tree has out degree at least c, contradicting the assumption on $G_\mathcal{R}$. □

Assume that we are given a regular set $T \subseteq T_A$ of trees such that for all the trees $t \in T$ the generalized h-factorization of t has width less than c. We show that in this situation the coding of T by generalized h-factorizations is a regular set of words, i.e., we show that the language

$$L_T = \{w \in (\Gamma_A^h)^* \mid w \text{ is a generalized } h\text{-factorization of } t \in T\}$$

is a regular language of words. This is a necessary property if we want to use generalized h-factorizations to transform GTR graphs to automatic graphs since the definition of automatic relations via finite automata implies that the vertex set of an automatic graph is a regular language.

LEMMA 3.42 *Let $\mathcal{R}$ be a GTRS. If there are $c \in \mathbb{N}$ and $h > h_\mathcal{R}$ such that for each $t \in T(\mathcal{R})$ the generalized h-factorization of t has width less than c, then $L_{T(\mathcal{R})}$ is a regular language.*

Proof. Let $\mathcal{A} = (Q, A, \Delta, F)$ be an NTA with $T(\mathcal{A}) = T(\mathcal{R})$. We extend $\mathcal{A}$ by adding the transitions $(\circ, q)$ for $q \in Q$ to Δ. Now, for $t \in \Gamma_A^h$, we can speak of a run of $\mathcal{A}$ on t.

The automaton for $L_{T(\mathcal{A})}$ guesses a run of $\mathcal{A}$ and, at each prefix v of the input, maintains a list of states of $\mathcal{A}$ for the locations x with $\mathrm{tree}(v)(x) = \circ$. Since the width of the generalized h-factorizations of trees from $T(\mathcal{A}) = T(\mathcal{R})$ is bounded by c the length of this list is also bounded by c. To guess a correct run the automaton for $L_{T(\mathcal{A})}$ guesses a run on each input symbol from Γ_A^h with state q at the root if q is the state at the end of the list. Then it removes q from the end of the list and adds the states from the guessed run that are at the locations labeled with $\circ$. So, the automaton guesses a generalized h-factorization of an accepting run of $\mathcal{A}$ on the input word.

Formally, we define an automaton $\mathcal{B} = (P, \Gamma_A^h, P_0, \Delta_\mathcal{B}, P_F)$ over finite words with state set P, input alphabet Γ_A^h, initial states $P_0 \subseteq P$, transition relation $\Delta_\mathcal{B} \subseteq P \times \Gamma_A^h \times P$, and final states $P_F \subseteq P$. The components of $\mathcal{B}$ are defined by

- $P = \{(q_1, \ldots, q_m) \mid q_i \in Q \text{ and } 1 \leq m \leq c\} \dot\cup \{p_f\}$,
- $P_0 = \{(q) \in P \mid q \in F\}$, and
- $P_F = \{p_f\}$.
- $\Delta_\mathcal{B}$ contains the following transitions. Let $(q_1, \ldots, q_m) \in P$, $t \in \Gamma_A^h$ and let ρ be a run of $\mathcal{A}$ on t with $\rho(\varepsilon) = q_m$. Let $x_1, \ldots, x_n \in D_t$ be the locations in lexicographical order with $t(x_i) = \circ$.
  - If $n > 0$, then $((q_1, \ldots, q_m), t, (q_1, \ldots, q_{m-1}, \rho(x_1), \ldots, \rho(x_n))) \in \Delta_\mathcal{B}$.
  - If $n = 0$ and $m > 1$, then $((q_1, \ldots, q_m), t, (q_1, \ldots, q_{m-1})) \in \Delta_\mathcal{B}$.
  - If $n = 0$ and $m = 1$, then $((q_1), t, p_f) \in \Delta_\mathcal{B}$.

□

The idea from the preceding proof can be extended to show that the relation on generalized h-factorizations induced by a ground tree rewriting relation is automatic.

LEMMA 3.43 *Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be a GTRS. If there are $c \in \mathbb{N}$ and $h > h_\mathcal{R}$ such that for each $t \in T(\mathcal{R})$ the generalized h-factorization of t has width less than c, then $G_\mathcal{R}$ is an automatic graph.*

Proof. For $\sigma \in \Sigma$ let $E_\sigma \subseteq (\Gamma_A^h)^* \times (\Gamma_A^h)^*$ be the relation on generalized h-factorizations induced by $\xrightarrow[\mathcal{R}]{\sigma}$, i.e., $(u, v) \in E_\sigma$ iff u and v are generalized h-factorizations of trees from $T(\mathcal{R})$ with $\mathrm{tree}(u) \xrightarrow[\mathcal{R}]{\sigma} \mathrm{tree}(v)$. Let $(u, v) \in E_\sigma$ and $u_1, u_2, v_2 \in (\Gamma_A^h)^*$, $t, t' \in \Gamma_A^h$ with

$$u = u_1 \cdot t \cdot u_2 \text{ and } v = u_1 \cdot t' \cdot v_2.$$

So t and t' are the first letters that differ in u and v. Let $x_1, \ldots, x_n$ be the locations from D_t in lexicographical order with $t(x_i) = \circ$ and let $w_i = \mathrm{gf}_h(t^{\downarrow x_i})$. Then $u_2 = w_1 \cdots w_n w$ for some $w \in (\Gamma_A^h)^*$. Since E_σ is derived from the rewriting relation $\xrightarrow[\mathcal{R}]{\sigma}$ there are three possibilities on how t' and v_2 relate to t and u_2.

1. The locations in t' labeled with $\circ$ are the same as in t. Then $v_2 = u_2$ and $t \xrightarrow[\mathcal{R}]{\sigma} t'$.
2. The locations in t' labeled with $\circ$ are $x_1, \ldots, x_j, x, x_{j+1}, \ldots, x_n$. Then $v_2 = w_1 \cdots w_j \cdot s \cdot w_{j+1} \cdots w_n w$ and $t^{\downarrow x} \xrightarrow[\mathcal{R}]{\sigma} s$.
3. The locations in t' labeled with $\circ$ are $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. Then $v_2 = w_1 \cdots w_{j-1} \cdot w_{j+1} \cdots w_n w$ and $\mathrm{tree}(w_j) \xrightarrow[\mathcal{R}]{\sigma} (t')^{\downarrow x_j}$. Note that $|w_j| = 1$ in this situation because $h > h_\mathcal{R}$.

The automaton from Lemma 3.42 can easily be modified to recognize this relation. We describe the idea for this modification. The technical details are straightforward.

Let $\mathcal{A}$ be an NTA accepting $T(\mathcal{R})$. The automaton for E_σ, instead of maintaining a list of states from $\mathcal{A}$, maintains a list of pairs of states from $\mathcal{A}$ to check whether both input words are accepted by $\mathcal{A}$. So, as long as the input words do not differ, in each step it adds pairs $(p_1, p_1'), \ldots, (p_m, p_m')$ of $\mathcal{A}$-states to the list if $y_1, \ldots, y_m$ are the locations labeled with $\circ$, in the same way as the automaton for $L_T(\mathcal{A})$ from Lemma 3.42.

The first time a difference in the two input words occurs, the automaton has to verify if one of the above cases holds.

In the first case it can directly check whether $t \xrightarrow[\mathcal{R}]{\sigma} t'$ and then proceed as before, remembering that it has seen a difference in the two input words.

In the second case the location x has to be treated in a different way since it is labeled with $\circ$ only in t'. In this situation the automaton adds $(q_1, q_1'), \ldots, (q_j, q_j'), (t^{\downarrow x}, q), (q_{j+1}, q_{j+1}'), \ldots, (q_n, q_n')$ to the list. Then the automaton reads $w_1 \cdots w_j$ as usual. If it reaches the letter s in the second word v, then the pair $(t^{\downarrow x}, q)$ is at the end of the list and it can be checked if

$t^{\downarrow x} \xrightarrow[\mathcal{R}]{\sigma} s$. Now the automaton can continue to check the $w_{j+1} \cdots w_n w$ part with a delay of one letter between the two input words.

The third case can be treated in a similar way as the second one. The automaton adds $(q_1, q'_1), \ldots, (q_{j-1}, q'_{j-1}), (q_j, (t')^{\downarrow x}), (q_{j+1}, q'_{j+1}), \ldots, (q_n, q'_n)$ to the list. The w_j part of the input is reached when $(q_j, (t')^{\downarrow x})$ is at the end of the list. The automaton can check if $\mathrm{tree}(w_j) \xrightarrow[\mathcal{R}]{\sigma} (t')^{\downarrow x}$ (recall that $|w_j| = 1$) and then continue with the $w_{j+1} \cdots w_n w$ part of the input with one letter delay between the two input words.

In this way we can build an automaton for the relation E_σ. Thus, the relations E_σ for all $\sigma \in \Sigma$ are automatic and therefore $G_\mathcal{R}$ is automatic. □

Combining Lemma 3.41 and Lemma 3.43 yields the following theorem.

THEOREM 3.44 *Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be a GTRS. If $G_\mathcal{R}$ is of bounded out degree, then $G_\mathcal{R}$ is an automatic graph.*

This theorem justifies the solid line from the automatic graphs to the GTR graphs of bounded degree in Figure 3.12.

We end this section with a brief discussion on the difficulties one encounters when trying to show that a GTR graph is not automatic. One possibility to show that a graph is not automatic is to use decidability results. Since the logics that are decidable for automatic graphs are also decidable for GTR graphs this method cannot be used here.

The other method that is known for showing that graphs are not automatic uses the following lemma (cf. [BG00]).

LEMMA 3.45 *Let G be an automatic graph of finite degree. Then there is $c \in \mathbb{N}$ such that for each vertex v of G and each $n \in \mathbb{N}$ the number of vertices at distance n from v is at most $2^{c \cdot n}$.*

This method also fails for GTR graphs as can be seen from the following explanation.

There is a generic method of coding trees from T_A as words by interpreting the term notation of trees as words over the alphabet A augmented by symbols for the parentheses and the comma. Ground tree rewriting just causes local changes in these codings. These changes can be checked by an automaton. So, given a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$, the automaton for the edge relations simply accepts all inputs $u \wedge v$ if $u = w_1 t w_2$ and $v = w_1 t' w_2$ with $t \hookrightarrow t' \in R$. The reason why we cannot use this coding for automatic representations of GTR graphs is that a finite automaton cannot check if a given word is a correct coding of a tree. But this straightforward approach

yields an automatic graph that contains the given GTR graph as induced subgraph.

REMARK 3.46 For each GTR graph $G_{\mathcal{R}}$ there is an automatic graph G of finite degree such that $G_{\mathcal{R}}$ is isomorphic to an induced subgraph of G.

This remark implies that Lemma 3.45 can also be formulated for GTR graphs instead of automatic graphs. Therefore, we cannot use this lemma to show that a GTR graph is not automatic. Anyhow, we think that there are GTR graphs that are not automatic, e.g., we were not able to find an automatic representation for the graph from Example 2.6 and Figure 2.4. The vertices of this graph are all trees over A consisting of the symbols b and a, where b has rank 2 and a has rank 0. The edges are generated by the only rewriting rule $a \hookrightarrow b(a)$.

For this graph to be automatic one needs a coding $\varphi : T_A \to \Gamma^*$ for some alphabet Γ such that the relation induced by the rewriting relation is automatic. In particular, this means that the image of φ is a regular language. It seems to be quite unlikely that such a regular coding exists, which still allows to recognize the rewriting relation by a finite automaton. For this reason, we end this section with the following conjecture.

Conjecture. The GTR graph from Example 2.6 is not automatic.

3.4 Traces of GTR Graphs

The trace of a path through an edge-labeled transition graph is the finite word corresponding to the sequence of the edge labels of this path. If an edge-labeled transition graph is equipped with an initial state and a set of final states, then the traces of this graph are all the traces of paths starting in the initial state and ending in a final state. The traces of finite graphs, for example, form the class of regular languages.

GTR graphs are infinite edge-labeled transition graphs with an initial state t_{in}. So, if we add a set of final states to a GTRS, then we can view it as an infinite automaton accepting languages of finite words. In this section we analyze the class of languages of finite words that can be defined in this way.

For a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$, $w \in \Sigma^*$, and $t, t' \in T_A$ we define

$$t \xrightarrow[\mathcal{R}]{w} t' \text{ iff } \begin{cases} w = \varepsilon \text{ and } t = t' \text{ or} \\ w = \sigma v \text{ with } \sigma \in \Sigma, v \in \Sigma^* \text{ and } t \xrightarrow[\mathcal{R}]{\sigma} t'' \xrightarrow[\mathcal{R}]{v} t' \text{ for some } t'' \in T_A \end{cases}$$

and for a set $T \subseteq T_A$

$$t \xrightarrow[\mathcal{R}]{w} T \text{ iff } \exists t' \in T \ : \ t \xrightarrow[\mathcal{R}]{w} t'.$$

The language accepted by $(\mathcal{R}, T)$ is

$$L(\mathcal{R}, T) = \{w \in \Sigma^* \mid t_{\text{in}} \xrightarrow[\mathcal{R}]{w} T\}.$$

The set T is called the set of final states or final vertices. The class $\mathcal{L}_{\text{GTR}}$ consists of all languages that can be accepted by a GTRS with a regular set of final states. So, if $L \subseteq \Sigma^*$ for some alphabet Σ, then L is in $\mathcal{L}_{\text{GTR}}$ iff there is a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and an NTA $\mathcal{A}$ such that $L = L(\mathcal{R}, T(\mathcal{A}))$.

EXAMPLE 3.47 We want to find a rewriting system $\mathcal{R}$ and an NTA $\mathcal{A}$ such that

$$L(\mathcal{R}, T(\mathcal{A})) = \{w \in \{0,1,2\}^* \mid |w|_0 = |w|_1 = |w|_2\},$$

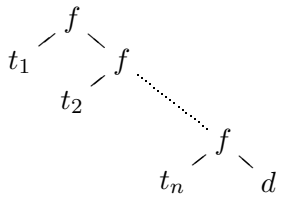
where $|w|_i$ denotes the number of occurrences of i in w for each $i \in \{0,1,2\}$. The GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ is defined by $A_0 = \{c, d\}$, $A_1 = \{0,1,2\}$, $A_2 = \{f\}$, $\Sigma = \{0,1,2\}$, $t_{\text{in}} = d$, and

$$R = \{d \overset{i}{\hookrightarrow} f(i(c), d) \mid i \in \Sigma\} \cup \{c \overset{i}{\hookrightarrow} i(c) \mid i \in \Sigma\}.$$

The NTA $\mathcal{A} = (Q, A, \Delta, F)$ has the state set $Q = \{q_\Lambda \mid \Lambda \subseteq \Sigma\}$, the final states $F = \{q_\emptyset\}$, and Δ contains the transitions

- $(c, q_{\{0,1,2\}}), (d, q_\emptyset)$,
- $(q_\Lambda, i, q_{\Lambda \setminus \{i\}})$ for all $i \in \{0,1,2\}$ and $\Lambda \subseteq \Sigma$ with $i \in \Lambda$, and
- $(q_\emptyset, q_\emptyset, f, q_\emptyset)$.

$\mathcal{A}$ accepts the trees of the following form, where for each $i \in \{1, \ldots, n\}$ the subtree t_i equals d or has c as only leaf and contains each $\sigma \in \{0,1,2\}$ exactly once:



Hence, if a tree is accepted by $\mathcal{A}$, then it contains the same number of 0, 1, and 2. Each rewriting rule of $\mathcal{R}$ labeled with $i \in \Sigma$ inserts exactly one i into the tree. It follows that $L(\mathcal{R}, T(\mathcal{A})) \subseteq \{w \in \Sigma^* \mid |w|_0 = |w|_1 = |w|_2\}$.

How a word w with $|w|_0 = |w|_1 = |w|_2$ can be accepted is illustrated in Figure 3.13 for $w = 011202$. □

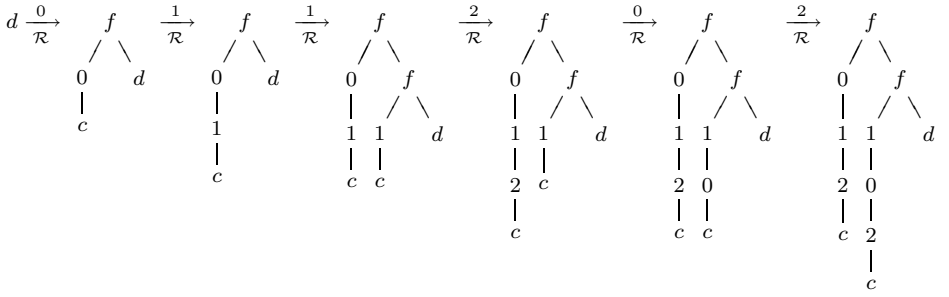


Figure 3.13: The word 011202 is accepted by $(\mathcal{R}, T(\mathcal{A}))$ from Example 3.47

In the same way we can use pushdown graphs to define languages of finite words. For a pushdown automaton $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ and a set $C \subseteq C(M)$ of configurations we define $L(M, C)$ to be the set of all words $w \in \Sigma^*$ such that a path from $q_0 Z_0$ to a configuration from C is labeled with w. The class $\mathcal{L}_{\text{PDA}}$ consists of all languages of the form $L(M, C)$ for a regular set C of configurations.

3.4.1 Classification in the Chomsky Hierarchy

The Chomsky Hierarchy (see e.g. [HU79]) has four levels: the regular languages $\mathcal{L}_{\text{REG}}$, the context free languages $\mathcal{L}_{\text{CF}}$, the context sensitive languages $\mathcal{L}_{\text{CS}}$ and the recursively enumerable languages $\mathcal{L}_{\text{RE}}$ with the following strict inclusions between these classes:

$$\mathcal{L}_{\text{REG}} \subsetneq \mathcal{L}_{\text{CF}} \subsetneq \mathcal{L}_{\text{CS}} \subsetneq \mathcal{L}_{\text{RE}}.$$

Since pushdown graphs are defined by pushdown automata the following result is not surprising.

PROPOSITION 3.48 *The classes $\mathcal{L}_{\text{CF}}$ and $\mathcal{L}_{\text{PDA}}$ coincide.*

Here we show that the class $\mathcal{L}_{\text{GTR}}$ is strictly between the classes $\mathcal{L}_{\text{CF}}$ and $\mathcal{L}_{\text{CS}}$. The main part is the separation of the classes $\mathcal{L}_{\text{GTR}}$ and $\mathcal{L}_{\text{CS}}$ by showing that the language

$$L_{012} = \{w \in \{0,1,2\}^* \mid w = 0^i 1^i 2^i \text{ for some } i \in \mathbb{N}\}$$

is not in $\mathcal{L}_{\text{GTR}}$. The intuition is that, although the rewriting system can keep track of the number of occurrences of the different letters as in Example 3.47, it cannot control the order of the letters. To store the occurrences of the

letters it may be necessary to use rewritings at locations "far from each other" and then the order of these rewritings can be exchanged. To formalize this we proceed as follows.

(1) By the notion location distance of a path π we formalize what it means that on π rewritings at locations far from each other are used.
(2) We show that if there is a uniform bound on the location distance of accepting paths, then the accepted language is context free.
(3) Finally, we prove that a GTRS accepting L_{012} must have the property from (2). This would imply that L_{012} is context free, which is a contradiction (see [HU79]).

Let $x, y, z \in \mathbb{N}^*$ be such that z is the maximal common prefix of x and y. The distance $\mathrm{dist}(x, y)$ of x and y is

$$\mathrm{dist}(x, y) = (|x| - |z|) + (|y| - |z|).$$

Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be a GTRS, $t, s \in T_A$, and let π be an $\mathcal{R}$-path with $\pi : t \xrightarrow[\mathcal{R}]{*} s$ and derivation $[x_0/s_0], \ldots, [x_n/s_n]$. The location distance $\mathrm{locdist}(\pi)$ of π is the maximal distance of two "unsynchronized" locations x_i and x_j in the derivation, where x_i and x_j are called synchronized if there is a location x_k between x_i and x_j that is a prefix of both, x_i and x_j. Formally, we have

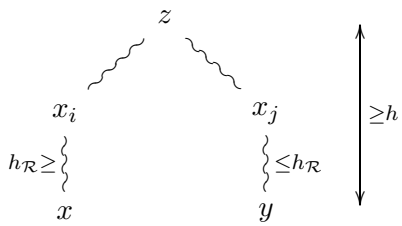
$$\mathrm{locdist}(\pi) = \max \left\{ \mathrm{dist}(x_i, x_j) \;\middle|\; \begin{array}{l} i, j \in \{0, \ldots, n\} \text{ with } i < j \text{ and} \\ \forall k \in \{i, \ldots, j\} \;:\; x_k \not\sqsubseteq x_i \text{ or } x_k \not\sqsubseteq x_j) \end{array} \right\}.$$

We want to simulate a GTRS with the property from item (2) from above by a pushdown automaton to show that the accepted language is context free. As in the conversion from GTRS to infix PDA (see Subsection 3.2.3) the stack content together with the control state of the pushdown automaton represent factorizations of the trees. For this purpose, we need to show that the trees on accepting paths are of bounded independence degree to get unique h-factorizations of these trees representable with a finite number of symbols (see Lemma 3.5).

LEMMA 3.49 *Let $\mathcal{R}$ be a GTRS, $d \in \mathbb{N}$, $h = h_{\mathcal{R}} + d$, and let π be an $\mathcal{R}$-path starting in t_{in}. If $\mathrm{locdist}(\pi) < d$, then $\mathrm{indep}(t) < h$ for each tree t on π.*

Proof. Let π be the path $t_0 \xrightarrow[\mathcal{R}]{} t_1 \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_n$ with $t_0 = t_{\mathrm{in}}$ and derivation $[x_1/s_1], \ldots, [x_n/s_n]$. Assume that there exists $m \in \{1, \ldots, n\}$ with $\mathrm{indep}(t_m) \geq h$. Then there are two locations $x, y \in D_{t_m}$ with maximal

common prefix z such that $|x| - |z| \geq h$ and $|y| - |z| \geq h$. Since $h > h_{\mathcal{R}}$ and $h_{\mathcal{R}} > \text{height}(t_{\text{in}})$ the locations x and y are not in $D_{t_{\text{in}}}$. Let $i < m$ be maximal with $x \notin D_{t_i}$ and let $j < m$ be maximal with $y \notin D_{t_j}$. W.l.o.g. we can assume that $i \leq j$ (otherwise exchange x and y). The choice of i and j implies that $x_i \sqsubseteq x$ and $x_j \sqsubseteq y$ with $|x| - |x_i| \leq h_{\mathcal{R}}$ and $|y| - |x_j| \leq h_{\mathcal{R}}$. In particular z is the maximal common prefix of x_i and x_j. This situation is illustrated in the following picture:



We can conclude that

$$|x_i| - |z| = (|x| - |z|) - (|x| - |x_i|) \geq (h_{\mathcal{R}} + d) - h_{\mathcal{R}} = d \text{ and}$$
$$|x_j| - |z| = (|x| - |z|) - (|x| - |x_j|) \geq (h_{\mathcal{R}} + d) - h_{\mathcal{R}} = d,$$

and therefore $\text{dist}(x_i, x_j) \geq d$. It remains to show that $x_k \not\sqsubseteq z$ for all $i < k < j$. By the choice of i and j we know that $x \in D_{t_k}$ for all k with $i < k < j$, implying $\text{height}(t_k^{\downarrow z}) \geq |x| - |z| \geq h > h_{\mathcal{R}}$. Therefore, by definition of $h_{\mathcal{R}}$, no rule can be applied to $t_k^{\downarrow z}$. Thus, we have shown $\text{locdist}(\pi) \geq d$, contradicting the assumption. □

The construction in the next lemma is similar to the conversion from GTRS to infix PDA from Subsection 3.2.3, except that we just simulate accepting paths because we are interested in the accepted language. Furthermore, since only paths with bounded location distance are simulated, we do not need any infix rules and thus obtain a "pure" PDA.

LEMMA 3.50 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be a GTRS, $T \subseteq T_A$ be a regular set of trees, and $L = L(\mathcal{R}, T)$. If there is $d \in \mathbb{N}$ such that $\text{locdist}(\pi) < d$ for each path $\pi : t_{\text{in}} \xrightarrow[\mathcal{R}]{*} T$, then L is context free.*

Proof. We show the claim by constructing a pushdown automaton that can simulate the accepting paths of $\mathcal{R}$. If we choose $h = h_{\mathcal{R}} + d$ according to Lemma 3.49, then the independence degree of the trees on accepting paths is less than h. Since the transformation from Lemma 3.6 preserves regular sets of trees and the independence degree of trees, we can assume that $\text{height}(t_{\text{in}}) < 2 \cdot h$ and $\text{height}(t) \geq h$ for all $t \in T(\mathcal{R})$.

We define the pushdown automaton $M = (Q, \Sigma, \Gamma, \Delta, q_{\text{in}}, \gamma_{\text{in}})$ as follows.

- $Q = \{q_t \mid t \in T_A \text{ with } h \leq \text{height}(t) < 2 \cdot h\}$.
- $\Gamma = \{\gamma_{\text{in}}\} \dot\cup \{\gamma_s \mid s \in S^h_{A_\circ}\}$ (see Page 35 for the definition of $S^h_{A_\circ}$).
- $q_{\text{in}} = q_{t_{\text{in}}}$.
- Δ contains the transitions

$$\begin{aligned}
(q_t, \gamma_{\text{in}}, \sigma, q_{t'}, \gamma_{\text{in}}) \quad &\text{if} \quad t \xrightarrow[\mathcal{R}]{\sigma} t', \\
(q_t, \gamma_s, \sigma, q_{t'}, \varepsilon) \quad &\text{if} \quad s \cdot t \xrightarrow[\mathcal{R}]{\sigma} t', \\
(q_t, \varepsilon, \sigma, q_{t'}, \gamma_{s'}) \quad &\text{if} \quad t \xrightarrow[\mathcal{R}]{\sigma} s' \cdot t', \\
(q_t, \gamma_s, \sigma, q_{t'}, \gamma_{s'}) \quad &\text{if} \quad s \cdot t \xrightarrow[\mathcal{R}]{\sigma} s' \cdot t'.
\end{aligned}$$

The set of accepting configurations is defined as

$$C_T = \{q_t \gamma_{s_1} \cdots \gamma_{s_n} \gamma_{\text{in}} \mid s_n \cdots s_1 \cdot t \in T\}.$$

We claim that $L(M, C_T) = L(\mathcal{R}, T)$. For the inclusion $L(M, C_T) \subseteq L(\mathcal{R}, T)$ it suffices to note that $q_t \gamma_{s_1} \cdots \gamma_{s_n} \gamma_{\text{in}} \vdash^\sigma_M q_{t'} \gamma_{s'_1} \cdots \gamma_{s'_m} \gamma_{\text{in}}$ implies $s_n \cdots s_1 \cdot t \xrightarrow[\mathcal{R}]{\sigma} s'_m \cdots s'_1 \cdot t'$. Then an easy induction shows that every word w that leads from $q_{\text{in}} \gamma_{\text{in}}$ to a configuration from C_T also leads from t_{in} to a tree from T.

For the inclusion $L(\mathcal{R}, T) \subseteq L(M, C_T)$ we have to show that every accepting path in $\mathcal{R}$ can be simulated in M. Since all the trees on accepting paths have independence degree less than h it is clear that they can be represented as configurations of M (Lemma 3.5). But the PDA M can only simulate substitutions in the $(s_1 \cdot t)$-part of an h-factorization $(t, s_1, \ldots, s_n)$. We have to show that this is no restriction for accepting paths.

For $t_0 = t_{\text{in}}$ and $t_n \in T$ consider an accepting path $t_0 \xrightarrow[\mathcal{R}]{} t_1 \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_n$ with derivation $[x_0/r_0], \ldots, [x_{n-1}/r_{n-1}]$. Pick $i \in \{0, \ldots, n-1\}$ and let $(t'_i, s_1, \ldots, s_m)$ be the unique h-factorization of t_i. Let $x^\circ_m = \varepsilon$ and for $j \in \{0, \ldots, m-1\}$ define

$$x^\circ_j = x^\circ_{j+1} \cdot x$$

for the $x \in D_{s_{j+1}}$ with $s_{j+1}(x) = \circ$. So x°_j is the location in D_{t_i} that corresponds to the root of s_j for $j \in \{1, \ldots, m\}$ and to the root of t'_i for $j = 0$. To verify that the substitution leading from t_i to t_{i+1} takes place in the $(s_1 \cdot t'_i)$-part of t_i we have to show that $x^\circ_1 \sqsubseteq x_i$. For this purpose we prove the following claim by induction on i.

Claim: There exists $j \leq i$ such that $x^\circ_0 \sqsubseteq x_j$ and $x_k \not\sqsubseteq x_j$ for all k with $j < k < i$.

If the claim is true and $x_1^\circ \not\sqsubseteq x_i$, then the maximal common prefix of x_j and x_i is a proper prefix of x_1°. Therefore, $\mathrm{dist}(x_i, x_j) \geq |x_j| - |x_1^\circ| \geq |x_0^\circ| - |x_1^\circ| = h > d$. Furthermore, x_i cannot be a prefix of x_j because otherwise $\mathrm{height}(t_i^{\downarrow x_i}) > h_\mathcal{R}$ and by definition of $h_\mathcal{R}$ there is no rule with a tree of that height on the left hand side. Thus, x_j and x_i are witnesses for $\mathrm{locdist}(\pi) \geq d$.

It remains to show the claim. If $i = 0$, then the h-factorization of t_i is t_i itself and the claim holds with $x_j = x_i = x_0$.

For $i \geq 1$ let $(t'_{i-1}, \hat{s}_1, \ldots, \hat{s}_{m'})$ be the h-factorization of t_{i-1} and define the locations $\hat{x}_j^\circ$ for t_{i-1} according to x_j° for t_i. We distinguish three cases for the possible values of m'.

If $m' = m - 1$, then the h-factorization of t_{i-1} contains one component more than the h-factorization of t_i. So, the substitution $[x_{i-1}/r_{i-1}]$ must increase the height of t'_{i-1} such that it exceeds $2 \cdot h$. Therefore, we have $\hat{s}_l = s_{l+1}$ for $l \in \{1, \ldots, m'\}$ and (t'_i, s_1) is an h-factorization of $t'_{i-1}[x_{i-1}/r_{i-1}]$. Since $\mathrm{height}(t'_i) \geq h > h_\mathcal{R}$, the substitution $[x_{i-1}/r_{i-1}]$ was in the t'_i part. Therefore, $x_0^\circ \sqsubseteq x_{i-1}$ and the claim holds for $j = i - 1$.

If $m' = m$, then $\hat{x}_0^\circ = x_0^\circ$. By induction we know that there is $j' \leq i - 1$ with $x_0^\circ \sqsubseteq x_{j'}$ and $x_k \not\sqsubseteq x_{j'}$ for all k with $j' < k < i - 1$. If $x_{i-1} \not\sqsubseteq x_{j'}$, then we choose $j = j'$. If $x_{i-1} \sqsubseteq x_{j'}$, then $x_0^\circ \sqsubseteq x_{i-1}$ because otherwise $x_{i-1} \sqsubseteq x_0^\circ$ and then $\mathrm{height}(t_{i-1}^{\downarrow x_{i-1}})$ would exceed $h_\mathcal{R}$ and no rewriting rule can be applied. Thus, we can choose $j = i - 1$.

If $m' = m + 1$, then $\hat{x}_1^\circ = x_0^\circ$ and since the h-factorization of t_i contains one component less than the h-factorization of t_{i-1}, the height of t'_{i-1} must decrease. This is only possible if $\hat{x}_0^\circ \sqsubseteq x_{i-1}$ and we get $x_0^\circ = \hat{x}_1^\circ \sqsubseteq \hat{x}_0^\circ \sqsubseteq x_{i-1}$ and the claim holds with $j = i - 1$. □

As mentioned before we show in the proof of Lemma 3.52 that a GTRS accepting L_{012} must have a uniform bound on the location distance of accepting paths, leading to a contradiction to the previous lemma. For this proof we need the following technical lemma providing us with a pumping argument for paths with certain properties.

LEMMA 3.51 *For each NTA $\mathcal{A} = (Q, A, \Delta, F)$ and GTRS $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ there is $h \in \mathbb{N}$ such that for all $\sigma \in \Sigma$ the following holds.*

Let $t_0 \xrightarrow[\mathcal{R}]{\sigma} t_1 \xrightarrow[\mathcal{R}]{\sigma} \cdots \xrightarrow[\mathcal{R}]{\sigma} t_n$ and let $x \in \mathbb{N}^*$ be such that $\mathrm{height}(t_0^{\downarrow x}) \leq h_\mathcal{R}$ and $\mathrm{height}(t_i^{\downarrow x}) > h$ for some $i \in \{1, \ldots, n-1\}$.

(i) If $\mathrm{height}(t_n^{\downarrow x}) \leq h_\mathcal{R}$ or $x \notin D_{t_n}$, then there is $m < n$ with $t_0 \xrightarrow[\mathcal{R}]{\sigma^m} t_n$.

(ii) If $t_n \in T(\mathcal{A})$, then there exists $m < n$ with $t_0 \xrightarrow[\mathcal{R}]{\sigma^m} T(\mathcal{A})$.

Proof. The intuition for (i) is that the subtree $t_0^{\downarrow x}$ of small height is transformed into a tree of large height and back to a tree $t_n^{\downarrow x}$ of small height. This goal of transforming $t_0^{\downarrow x}$ into $t_n^{\downarrow x}$ can also be achieved with a shorter derivation not leading through the tree of large height. This is captured by the number h_1 below.

For (ii) we use that there must be an upper bound on the minimal length of paths leading from trees of height bounded by $h_{\mathcal{R}}$ to a regular set of trees. This is captured by h_2. Then h_1 and h_2 are merged into one number h satisfying (i) and (ii).

For $\sigma \in \Sigma$, $s \in T_A$ and $T \subseteq T_A$ we define

$$
\begin{aligned}
d_\sigma(s,T) &= \begin{cases} \min\{n \in \mathbb{N} \mid s \xrightarrow[\mathcal{R}]{\sigma^n} T\} \text{ if such an } n \text{ exists,} \\ 0 \text{ otherwise,} \end{cases} \\
h_1 &= \max\{d_\sigma(s,\{t\}) \mid \sigma \in \Sigma, \text{height}(s) \le h_{\mathcal{R}}, \text{ and height}(t) \le h_{\mathcal{R}}\}, \\
h_2 &= \max\{d_\sigma(s, T(\mathcal{A}(q))) \mid \text{height}(s) \le h_{\mathcal{R}}, q \in Q, \text{ and } \sigma \in \Sigma\}, \text{ and} \\
h &= (h_1 + h_2) \cdot h_{\mathcal{R}}.
\end{aligned}
$$

(i): Let t_j be the last tree before t_i with $\text{height}(t_j^{\downarrow x}) \le h_{\mathcal{R}}$ and let t_k be the first tree after t_i with $\text{height}(t_k^{\downarrow x}) \le h_{\mathcal{R}}$. One portion of the derivation must transform $t_j^{\downarrow x}$ into $t_i^{\downarrow x}$ and back to $t_k^{\downarrow x}$. By the choice of h and the definition of $h_{\mathcal{R}}$ this takes more than h_1 steps and therefore this portion can be replaced by a shorter path from $t_j^{\downarrow x}$ to $t_k^{\downarrow x}$.
(ii): As in (i) let t_j be the last tree before t_i with $\text{height}(t_j^{\downarrow x}) \le h_{\mathcal{R}}$. If $x \notin D_{t_k}$ or $\text{height}(t_k^{\downarrow x}) \le h_{\mathcal{R}}$ for some $k \in \{i, \ldots, n\}$, then we are in case (i). Otherwise, there is a portion of the derivation that transforms $t_j^{\downarrow x}$ into $t_n^{\downarrow x}$. By the choice of h this takes more than h_2 steps because first $t_j^{\downarrow x}$ has to be transformed into $t_i^{\downarrow x}$ and $\text{height}(t_i^{\downarrow x}) > h$.

Since $t_n \in T(\mathcal{A})$ there is $q \in Q$ with $t_n \xrightarrow[\mathcal{A}]{*} t_n[x/q] \xrightarrow[\mathcal{A}]{*} F$. Then the part of the derivation transforming $t_j^{\downarrow x}$ into $t_n^{\downarrow x}$ can be replaced by a shorter transformation, of length less than or equal to h_2, rewriting $t_j^{\downarrow x}$ into a tree $t' \in T(\mathcal{A}(q))$. In total we get $t_0 \xrightarrow[\mathcal{R}]{\sigma^m} t_n[x/t']$ with $m < n$ and $t_n[x/t'] \xrightarrow[\mathcal{A}]{*} t_n[x/q] \xrightarrow[\mathcal{A}]{*} F$. □

Now we have the tools for separating $\mathcal{L}_{\text{GTR}}$ and $\mathcal{L}_{\text{CS}}$.

LEMMA 3.52 *The language L_{012} is not in $\mathcal{L}_{\text{GTR}}$.*

Proof. Assume that $L_{012} = L(\mathcal{R}, T(\mathcal{A}))$ for a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and an NTA $\mathcal{A}$. We want to apply Lemma 3.50 with $d = 2 \cdot h + 1$ where h is chosen according to Lemma 3.51. Let π be an accepting path of the form

$$t_0 \xrightarrow[\mathcal{R}]{0} t_1 \xrightarrow[\mathcal{R}]{0} \cdots \xrightarrow[\mathcal{R}]{0} t_n \xrightarrow[\mathcal{R}]{1} t_{n+1} \xrightarrow[\mathcal{R}]{1} \cdots \xrightarrow[\mathcal{R}]{1} t_{2n} \xrightarrow[\mathcal{R}]{2} t_{2n+1} \xrightarrow[\mathcal{R}]{2} \cdots \xrightarrow[\mathcal{R}]{2} t_{3n}$$

with $t_0 = t_{\text{in}}$, $t_{3n} \in T(\mathcal{A})$, and derivation $[x_0/s_0], [x_1/s_1], \ldots, [x_{3n-1}/s_{3n-1}]$. If we can show that $\text{locdist}(\pi) < d$, then L_{012} is context free by Lemma 3.50. Since L_{012} is not context free we get a contradiction.

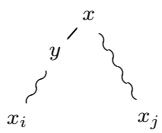
Let $i, j \in \{0, \ldots, 3n-1\}$ with $i < j$ and let x be the maximal common prefix of x_i and x_j. Assume that $\text{dist}(x_i, x_j) \geq d$ and $x_k \not\sqsubseteq x$ for all $k \in \{i, \ldots, j\}$.

We first show (in case 1) that i, j must be from the same interval, i.e., both are from the 0-, the 1-, or the 2-part of π. Otherwise we can exchange substitutions from different intervals and obtain an accepting path not obeying the strict order on the letters 0,1, and 2.

Then we show (cases 2,3,4) that i, j cannot be from the same interval because otherwise we could apply Lemma 3.51 and shorten one of the intervals obtaining an accepting path where the numbers of occurrences of 0,1, and 2 are not the same.

Case 1. Assume that $i < n$ and $j \geq n$ or $i < 2n$ and $j \geq 2n$. We only consider the case $i \in \{0, \ldots, n-1\}$ and $j \in \{n, \ldots, 2n-1\}$. The other cases are similar.

Let y be the minimal prefix of x_i with $x \sqsubset y$. So we have the following situation.



Now we collect the substitutions between the positions i and j that are below y and all the other substitutions and exchange their order. Let $i_1 < \ldots < i_l \in \{i, \ldots, j\}$ be the indices with $y \sqsubseteq x_{i_k}$ for all $k \in \{1, \ldots, l\}$ and let $j_1 < \ldots < j_m \in \{i, \ldots, j\}$ be the indices with $y \not\sqsubseteq x_{i_k}$ for all $k \in \{1, \ldots, m\}$. Since x_i and x_j are unsynchronized the locations from $\{x_{i_1}, \ldots, x_{i_l}\}$ and from $\{x_{j_1}, \ldots, x_{j_m}\}$ are pairwise not comparable w.r.t $\sqsubseteq$. Therefore, we can replace the part $[x_i/s_i], \ldots, [x_j/s_j]$ of the derivation by

$$[x_{j_1}/s_{j_1}], \ldots, [x_{j_m}/s_{j_m}], [x_{i_1}/s_{i_1}], \ldots, [x_{i_l}/s_{i_l}],$$

obtaining the derivation of another accepting path from t_0 to t_{3n}. But since $i_1 = i$ and $j_m = j$ there is a 1 before a 0 on this accepting path.

Note that we obtained this contradiction solely from x_i and x_j not being synchronized by an intermediate location x_k, without any assumption on the distance of x_i and x_j.

Case 2 Assume that $i, j \in \{0, \ldots, n-1\}$. We want to apply Lemma 3.51 (i). From the definition of $h_{\mathcal{R}}$ and from $t_0 = t_{\text{in}}$ we know that $\text{height}(t_0^{\downarrow x}) \leq h_{\mathcal{R}}$. From $\text{dist}(x_i, x_j) \geq d = 2 \cdot h + 1$ we know that either $\text{height}(t_i^{\downarrow x}) > h$ or $\text{height}(t_j^{\downarrow x}) > h$. It remains to show that there is $l \in \{j+1, \ldots, n-1\}$ such that $\text{height}(t_l^{\downarrow x}) \leq h_{\mathcal{R}}$. For this purpose let z_1 and z_2 be the maximal prefixes of x_n, x_i and x_n, x_j, respectively. Then either $z_1 \sqsubseteq x$ or $z_2 \sqsubseteq x$. Since in case 1 the distance between the locations did not play any role, x_i and x_n, and x_j and x_n must be synchronized. That is, there must be $l_1 \in \{i, \ldots, n\}$ with $x_{l_1} \sqsubseteq z_1$ and $l_2 \in \{j, \ldots, n\}$ with $x_{l_2} \sqsubseteq z_2$. Since $z_1 \sqsubseteq x$ or $z_2 \sqsubseteq x$, we can find $l \in \{i, \ldots, n\}$ with $x_l \sqsubseteq x$. Thus, $\text{height}(t_l^{\downarrow x}) \leq h_{\mathcal{R}}$. Since x_i and x_j are not synchronized by the assumption, this l cannot be between i and j and therefore $l \in \{j+1, \ldots, n\}$. Now we can apply Lemma 3.51 (i) and shorten the path segment between t_0 and t_l. This yields an accepting path with fewer occurrences of 0 than occurrences of 1 and 2.

Case 3 Assume that $i, j \in \{n, \ldots, 2n-1\}$. As in case 2 $\text{height}(t_i^{\downarrow x}) > h$ or $\text{height}(t_j^{\downarrow x}) > h$. In the same way as in case 2 we can find $l \in \{j+1, \ldots, 2n-1\}$ such that $\text{height}(t_l^{\downarrow x}) \leq h_{\mathcal{R}}$. To apply Lemma 3.51 (i) it remains to show that there is $m \in \{n, \ldots, i\}$ such that $\text{height}(t_m^{\downarrow x}) \leq h_{\mathcal{R}}$. This m is found in a similar way as the l was found in case 1. Let z_1 and z_2 be the maximal prefixes of x_{n-1}, x_i and x_{n-1}, x_j, respectively. Then either $z_1 \sqsubseteq x$ or $z_2 \sqsubseteq x$. By case 1 there must be $m_1 \in \{n, \ldots, i+1\}$ with $x_{m_1-1} \sqsubseteq z_1$ and $m_2 \in \{n, \ldots, j+1\}$ with $x_{m_2-1} \sqsubseteq z_2$. Since $z_1 \sqsubseteq x$ or $z_2 \sqsubseteq x$ there is $m \in \{n, \ldots, j+1\}$ with $x_{m-1} \sqsubseteq x$. Thus, $\text{height}(t_m^{\downarrow x}) \leq h_{\mathcal{R}}$. Again, x_i and x_j are not synchronized by assumption. Hence, $m-1$ cannot be between i and j and therefore $m \in \{n, \ldots, i\}$. We can use Lemma 3.51 (i) and shorten the path segment between t_m and t_l. This yields an accepting path with fewer occurrences of 1 than occurrences of 0 and 2.

Case 4 Assume that $i, j \in \{2n, \ldots, 3n-1\}$. As in case 2 $\text{height}(t_i^{\downarrow x}) > h$ or $\text{height}(t_j^{\downarrow x}) > h$. In the same way as in case 3 we can find $m \in \{2n, \ldots, i\}$ such that $\text{height}(t_m^{\downarrow x}) \leq h_{\mathcal{R}}$. Since $t_{3n} \in T(\mathcal{A})$ we can

apply Lemma 3.51 (ii) to the path segment between t_m and t_{3n}. This yields an accepting path with fewer occurrences of 2 than occurrences of 0 and 1.

We have shown that for each $i, j \in \{0, \ldots, 3n-1\}$ with $i < j$ either $\mathrm{dist}(x_i, x_j) < d$ or there is $k \in \{i, \ldots, j\}$ with $x_k \sqsubseteq x_i$ and $x_k \sqsubseteq x_j$. Hence, $\mathrm{locdist}(\pi) < d$. □

Besides separating the classes $\mathcal{L}_{\mathrm{GTR}}$ and $\mathcal{L}_{\mathrm{CS}}$ this lemma provides a method for showing that the triangle graph from Figure 3.1 is not a GTR graph. This graph with the root vertex as the only accepting vertex recognizes the language L_{012}^*. The proof of the above lemma can be easily adapted to show that this language is not in $\mathcal{L}_{\mathrm{GTR}}$.

THEOREM 3.53 $\mathcal{L}_{\mathrm{CF}} \subsetneq \mathcal{L}_{\mathrm{GTR}} \subsetneq \mathcal{L}_{\mathrm{CS}}$.

Proof. The class of GTR graphs contains the class of pushdown graphs. The traces of pushdown graphs are exactly the context free languages [MS85]. Therefore, $\mathcal{L}_{\mathrm{CF}} \subseteq \mathcal{L}_{\mathrm{GTR}}$.

The language $\{w \in \{0,1,2\}^* \mid |w|_0 = |w|_1 = |w|_2\}$ is not in $\mathcal{L}_{\mathrm{CF}}$ but in $\mathcal{L}_{\mathrm{GTR}}$ as shown in Example 3.47.

If a language L is in $\mathcal{L}_{\mathrm{GTR}}$, then it can be accepted by a nondeterministic linear space bounded Turing machine that simulates the rewriting system and then checks if the resulting tree is accepting. Since the context sensitive languages are exactly the languages accepted by nondeterministic linear space bounded Turing machines, we get $\mathcal{L}_{\mathrm{GTR}} \subseteq \mathcal{L}_{\mathrm{CS}}$.

The language L_{012} is context sensitive but not in $\mathcal{L}_{\mathrm{GTR}}$ (Lemma 3.52). □

3.4.2 Closure Properties

We can use the results from the previous section to show that $\mathcal{L}_{\mathrm{GTR}}$ is not closed under all of the Boolean operations. In fact, this class of languages is only closed under union.

THEOREM 3.54 *The class $\mathcal{L}_{\mathrm{GTR}}$ of languages is closed under union.*

Proof. Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ and $\mathcal{R}' = (A', \Sigma, R', t'_{\mathrm{in}})$ be ground tree rewriting systems and $T \subseteq T_A$, $T' \subset T_{A'}$. We construct a GTRS $\hat{\mathcal{R}} = (\hat{A}, \Sigma, \hat{R}, \hat{t}_{\mathrm{in}})$ and a set $\hat{T} \subseteq T_{\hat{A}}$ such that $L(\hat{\mathcal{R}}, \hat{T}) = L(\mathcal{R}, T) \cup L(\mathcal{R}', T')$. By renaming the symbols from A' and adapting $\mathcal{R}'$ we can assume that A and A' are

disjoint. Let $\perp$ be a symbol of rank 0 that is neither in A nor in A'. The components of $\hat{\mathcal{R}}$ are $\hat{A} = A \cup A' \cup \{\perp\}$, $\hat{t}_{\text{in}} = \perp$, and

$$\hat{R} = R \cup R' \cup \{\perp \overset{\sigma}{\hookrightarrow} t \mid t \in T_A, t_{\text{in}} \xrightarrow[\mathcal{R}]{\sigma} t\} \cup \{\perp \overset{\sigma}{\hookrightarrow} t' \mid t' \in T_{A'}, t'_{\text{in}} \xrightarrow[\mathcal{R}']{\sigma} t'\}.$$

The set $\hat{T}$ is defined as

$$\hat{T} = \begin{cases} T \cup T' \cup \{\perp\} & \text{if } t_{\text{in}} \in T \text{ or } t'_{\text{in}} \in T' \\ T \cup T' & \text{otherwise.} \end{cases}$$

This construction corresponds to the usual construction for the union of two nondeterministic finite automata and it is easy to verify that

$$\hat{t}_{\text{in}} \xrightarrow[\hat{\mathcal{R}}]{w} \hat{T} \text{ iff } t_{\text{in}} \xrightarrow[\mathcal{R}]{w} T \text{ or } t'_{\text{in}} \xrightarrow[\mathcal{R}']{w} T'$$

for each $w \in \Sigma^*$. □

In Example 3.47 we have seen a language belonging to $\mathcal{L}_{\text{GTR}}$, and from Lemma 3.52 we know a language not belonging to $\mathcal{L}_{\text{GTR}}$. We can use this to show that $\mathcal{L}_{\text{GTR}}$ is not closed under intersection. In fact, we can show the stronger result that $\mathcal{L}_{\text{GTR}}$ is not even closed under intersection with regular languages.

THEOREM 3.55 *The class $\mathcal{L}_{\text{GTR}}$ of languages is*

(i) not closed under intersection with regular languages and

(ii) not closed under complement.

Proof. For (i) note that

$$L_{012} = \{w \in \{0,1,2\}^* \mid |w|_0 = |w|_1 = |w|_2\} \cap 0^*1^*2^*.$$

So L_{012} is not in $\mathcal{L}_{\text{GTR}}$ (Lemma 3.52) but can be written as an intersection of a language from $\mathcal{L}_{\text{GTR}}$ (see Example 3.47) and a regular language.

Since $\mathcal{L}_{\text{GTR}}$ is closed under union and not under intersection it cannot be closed under complement because

$$L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$$

for languages $L_1, L_2 \subseteq \Sigma^*$. □

Chapter 4

Model-Checking for RGTR Graphs

In general, model-checking is the task of testing whether a given structure satisfies a given property (cf. [Eme96]). The structure is taken from a fixed class $\mathcal{K}$ of structures and the property is usually written in a specification logic $\mathcal{L}$. For a structure K from $\mathcal{K}$ and a formula φ from $\mathcal{L}$ we say that K is a model of φ, written as $K \models \varphi$, if φ is true in K. The model-checking problem for a class $\mathcal{K}$ of structures and a logic $\mathcal{L}$ is: "Given a structure K from $\mathcal{K}$ and a formula φ from $\mathcal{L}$, is K a model of φ?"

In temporal logics, like CTL, LTL, or CTL*, (cf. [Eme90]) one can specify reachability properties that talk about paths through the structure. In this case an initial state s_{in} of the structure is marked as the starting point of these paths and the question is, does K with initial state s_{in} satisfy the property specified by φ, written as $(K, s_{\text{in}}) \models \varphi$.

The goal of this chapter is to analyze the model-checking problem for RGTR graphs and temporal logics, i.e., we fix the class of structures as the class of RGTR graphs. The question is, what kind of temporal specification logic $\mathcal{L}$ can we use such that the problem

$$(G_{\mathcal{R}}, t_{\text{in}}) \models \varphi?$$

can be solved by an algorithm for each RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and for each formula φ from $\mathcal{L}$.

To answer this question the basic reachability problems that can be expressed in temporal logic are introduced in Section 4.1 and are studied in more detail in Sections 4.2 and 4.3: the decidable ones are discussed in Section 4.2 whereas the other ones are shown to be undecidable in Section 4.3. In Section 4.4 a logic with decidable model-checking problem is derived

from the decidability results. This logic is a fragment of the computation tree logic CTL* (cf. [Eme90]) and the undecidability results imply that this fragment is in some sense a maximal fragment of CTL* with a decidable model-checking problem for RGTR graphs.

In the verification of reactive systems two player games on graphs play an important role to formally model the behavior and the interaction of the system and the environment (cf. [Tho95]). In Section 4.5 we study this aspect for GTR graphs by analyzing reachability games on GTR graphs. We propose two different reachability games. The first one cover one of the undecidable reachability problems from Section 4.3 as a special case. The second one also turns out to be undecidable, a result that does not follow from the results in Section 4.3 but the method of proof is similar.

The proofs in this chapter also show that for the problems under consideration there is no difference between RGTR graphs and GTR graphs because the decidability results are given for RGTR graphs and the undecidability results are obtained for GTR graphs. So, from an algorithmic point of view the two classes of GTR graphs and RGTR graphs have a relation similar to the one of pushdown graphs and prefix recognizable graphs.

4.1 Basic Reachability Problems

The elementary properties we analyze in this chapter and that can be expressed in a temporal logic like CTL* are the following:

One step reachability: Given an RGTRS $\mathcal{R}$, a vertex t, and a set T of vertices of $G_{\mathcal{R}}$, does there exist a successor of t that is in T?

Reachability: Given an RGTRS $\mathcal{R}$, a vertex t, and a set T of vertices of $G_{\mathcal{R}}$, does there exist a path from t to a vertex in T?

Constrained reachability: Given an RGTRS $\mathcal{R}$, a vertex t, and sets T_1, T_2 of vertices of $G_{\mathcal{R}}$, does there exist a path from t that remains in T_2 until it eventually reaches a vertex in T_1?

Recurrence: Given an RGTRS $\mathcal{R}$, a vertex t, and a set T of vertices of $G_{\mathcal{R}}$, does there exist a path from t that infinitely often visits T?

All of these questions can also be posed in their "universal versions". These versions do not ask for the existence of a path but whether all paths have the respective property:

Universal one step reachability: Given an RGTRS $\mathcal{R}$, a vertex t, and a set T of vertices of $G_{\mathcal{R}}$, are all successors of t in T?

Problem	Temporal Operator	
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

Table 4.1: Overview of decidability results.

Universal reachability: Given an RGTRS $\mathcal{R}$, a vertex t, and a set T of vertices of $G_{\mathcal{R}}$, do all paths from t eventually reach a vertex in T?

Universal constrained reachability: Given an RGTRS $\mathcal{R}$, a vertex t, and sets T_1, T_2 of vertices of $G_{\mathcal{R}}$, do all paths from t remain in T_2 until they eventually reach a vertex from T_1?

Universal recurrence: Given an RGTRS $\mathcal{R}$, a vertex t, and a set T of vertices of $G_{\mathcal{R}}$, do all infinite path from t infinitely often visit T?

For the sets of vertices used in the specification of the problems we use regular sets of trees. This allows us to use finitely represented infinite sets, which can serve as input for an algorithm.

One should note that the universal one step reachability can be expressed as the negation of the one step reachability: all successors of t are in T if no successor of t is in the complement of T. This relation is not valid for the other problems.

Furthermore, it should be mentioned that (universal) reachability is a special case of (universal) constrained reachability by setting the set T_2 to the set of all trees. The problem of universal reachability is shown to be undecidable in Section 4.3. Therefore, it is clear that the universal constrained reachability is also undecidable.

Table 4.1 gives an overview of the results from the next two sections and also lists for each problem the corresponding temporal operator in CTL* notation.

4.2 Decidable Properties

This section discusses the decidable problems that are listed in Table 4.1. The decidability results for one step reachability and reachability are not new, and proofs similar to the ones here can be found in [CDG+97], Section 3.2.5, in the context of closure properties of ground tree transducers. One step reachability is a special case of concatenation of ground tree transducers and reachability corresponds to iteration of ground tree transducers.

The algorithm for solving the reachability problem is akin to the one from [CDGV94] for solving the reachability problem of semi-monadic linear rewrite systems.

The decidability proof of the recurrence problem appeared in [Löd02b] for GTR graphs and is adapted here for RGTR graphs.

Before we go into details we explain the basic idea underlying the constructions for one step reachability and reachability. The crucial point in both constructions is that an NTA can "simulate" rewriting rules by ε-transitions as illustrated in Figure 4.1. Suppose we are given an RGTRS $\mathcal{R}$ and an NTA $\mathcal{A}$ for the goal set, i.e., the set of vertices we want to reach. The basic task in the reachability problems is to enable the automaton to simulate the rewriting rules. For one step reachability exactly one rewriting step and for reachability arbitrarily many. Consider the following situation:

- There is a rewriting rule $T_i \hookrightarrow T_i'$ in $\mathcal{R}$.
- There is a state q of the automaton that is reachable only via trees from T_i.
- There are $t \in T_A$ and $t' \in T(\mathcal{A})$ with the following properties.
  - $t^{\downarrow x} \xrightarrow[\mathcal{A}]{*} q$, in particular $t^{\downarrow x} \in T_i$,
  - $(t')^{\downarrow x} \in T_i'$, and $(t')^{\downarrow x} \xrightarrow[\mathcal{A}]{*} p$ for some state p of $\mathcal{A}$.

Since $t^{\downarrow x} \in T_i$ and $(t')^{\downarrow x} \in T_i'$, we know that $t \xrightarrow[\mathcal{R}]{} t'$. This situation is depicted in Figure 4.1. If we want $\mathcal{A}$ to also accept t, we can add an ε-transition from q to p. Then the automaton can reduce $t^{\downarrow x}$ to q as before, use the ε-transition to jump to p, and thus "pretend" that he read $(t')^{\downarrow x}$ instead of $t^{\downarrow x}$. This insertion of ε-transitions for simulating rewriting rules is the basic operation that is used in two different variations to construct, given an NTA $\mathcal{A}$, automata for the sets $\{t \in T_A \mid t \xrightarrow[\mathcal{R}]{} T(\mathcal{A})\}$ and $\{t \in T_A \mid t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A})\}$.

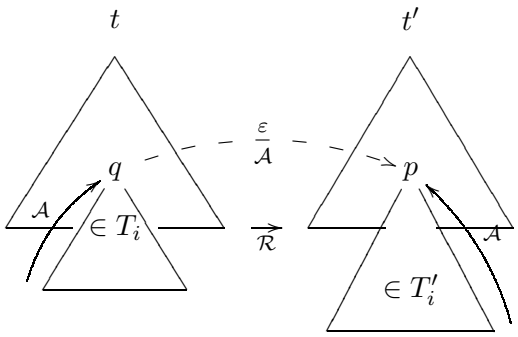


Figure 4.1: The basic idea for the reachability constructions. The ε-transition is added to $\mathcal{A}$ to simulate the rewriting rule $T_i \hookrightarrow T_i'$.

4.2.1 One Step Reachability (EX)

In this subsection we prove that the one step rewriting relation of RGTRS preserves regularity. Given an RGTRS $\mathcal{R}$ and an ε-NTA $\mathcal{A}$ we show how to construct an ε-NTA $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ accepting the set

$$\mathrm{pre}_\mathcal{R}(T(\mathcal{A})) := \{t \in T_A \mid t \underset{\mathcal{R}}{\longrightarrow} T(\mathcal{A})\}.$$

Informally, the construction works as follows. The ε-NTA $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ guesses a rule $T_i \overset{\sigma}{\hookrightarrow} T_i'$ and a subtree $t^{\downarrow x} \in T_i$ of the input tree such that $t^{\downarrow x}$ can be replaced by a tree from T_i' yielding a tree from $T(\mathcal{A})$. For this purpose, $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ contains for each left hand side T_i of a rule a copy of an automaton accepting T_i. To simulate the rewriting step there are ε-transitions from the final states of the $\mathcal{A}_i$ into a copy of $\mathcal{A}$ to the states that can be reached via the trees from T_i'. Once such an ε-transitions is used, $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ proceeds in the same way as $\mathcal{A}$.

To ensure that $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ really simulates one rewriting step but not more than one, we introduce two copies of $\mathcal{A}$. The 1-copy signals the simulation of a rewriting step in the subtree, and the 0-copy signals that there was no simulation.

THEOREM 4.1 *Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be an RGTRS and $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA. One can construct an ε-NTA $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ of size $\mathcal{O}(|\mathcal{R}|+|\mathcal{A}|)$ accepting the set* $\mathrm{pre}_\mathcal{R}(T(\mathcal{A}))$.

Proof. Let $A = (A_i)_{i\in[k]}$ and $R = \{T_1 \overset{\sigma_1}{\hookrightarrow} T_1', \ldots, T_m \overset{\sigma_m}{\hookrightarrow} T_m'\}$, where the sets T_i and T_i' are accepted by NTAs $\mathcal{A}_i = (Q_i, A, \Delta_i, F_i)$ and $\mathcal{A}_i' =$

$(Q'_i, A, \Delta'_i, F'_i)$ respectively. Let $I = \{1, \ldots, m\}$ and for each $i \in I$ define $P'_i = \{q \in Q \mid \exists s \in T'_i \,:\, s \xrightarrow[\mathcal{A}]{*} q\}$ the set of all states of $\mathcal{A}$ that are reachable via a tree from T'_i. Then $\mathcal{A}_{\mathrm{pre}_\mathcal{R}} = (Q', A, \Delta', F')$ is defined as follows:

- $Q' = (Q \times \{0,1\}) \cup (\dot{\bigcup}_{i\in I} Q_i)$.
- $F' = F \times \{1\}$.
- $\Delta' = \left(\dot{\bigcup}_{i\in I} \Delta_i\right) \cup \{(p,(q,1)) \mid p \in F_i, q \in P'_i\} \cup \Delta''$ where Δ'' contains all the transitions of the form $((q_1,l_1),\ldots,(q_n,l_n),a,(q,l))$ with $(q_1,\ldots,q_n,a,q) \in \Delta$ and $l_1,\ldots,l_n,l \in \{0,1\}$ such that $\sum_{j=1}^n l_j \leq 1$ and $l = \sum_{j=1}^n l_j$.

For the correctness first note that by the construction of $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ we have

$$t[x/q] \xrightarrow[\mathcal{A}]{*} q' \Leftrightarrow t[x/(q,1)] \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} (q',1) \qquad (\star)$$

for all $t \in T_A$, $x \in D_t$, and $q, q' \in Q$.

$\mathrm{pre}_\mathcal{R}(T(\mathcal{A})) \subseteq T(\mathcal{A}_{\mathrm{pre}_\mathcal{R}})$**:** Assume that $t \xrightarrow[\mathcal{R}]{\sigma_i} T(\mathcal{A})$. Then there is $x \in D_t$ and $s \in T'_i$ such that $t^{\downarrow x} \in T_i$ and $t[x/s] \in T(\mathcal{A})$. Since $s \in T'_i$ and $t[x/s] \in T(\mathcal{A})$ there are $q \in P'_i$ and $q' \in F$ with $s \xrightarrow[\mathcal{A}]{*} q$ and $t[x/q] \xrightarrow[\mathcal{A}]{*} q'$. Since $t^{\downarrow x} \in T_i$ there exists $p \in F_i$ with $t^{\downarrow x} \xrightarrow[\mathcal{A}_i]{*} p$. Then we get $t \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} t[x/p] \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{} t[x/(q,1)]$ by the definition of Δ'. With $(\star)$ we obtain $t[x/(q,1)] \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} (q',1)$ and therefore $t \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} (q',1) \in F'$.

$T(\mathcal{A}_{\mathrm{pre}_\mathcal{R}}) \subseteq \mathrm{pre}_\mathcal{R}(T(\mathcal{A}))$**:** Let $t \in T(\mathcal{A}_{\mathrm{pre}_\mathcal{R}})$ and $q' \in F$ with $t \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} (q',1)$. By the construction of $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ the states from $Q \times \{1\}$ can only be reached via an ε-transition of the form $(p,(q,1))$ with $p \in F_i$ and $q \in P'_i$. So there are $x \in D_t$, $i \in I$, $p \in F_i$, and $q \in P'_i$ such that $t \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} t[x/p] \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{} t[x/(q,1)] \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} (q',1)$. By the definition of P'_i there exists $s \in T'_i$ with $s \xrightarrow[\mathcal{A}]{*} q$. From $(\star)$ and $t[x/(q,1)] \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} (q',1)$ we can conclude that $t[x/q] \xrightarrow[\mathcal{A}]{*} q'$. Hence, we get $t[x/s] \in T(\mathcal{A})$ because of $t[x/s] \xrightarrow[\mathcal{A}]{*} t[x/q] \xrightarrow[\mathcal{A}]{*} q' \in F$. From $t \xrightarrow[\mathcal{A}_{\mathrm{pre}_\mathcal{R}}]{*} t[x/p]$, $p \in F_i$, and the definition of $\mathcal{A}_{\mathrm{pre}_\mathcal{R}}$ we can conclude that $t \xrightarrow[\mathcal{A}_i]{*} t[x/p]$. Thus, $t^{\downarrow x} \in T_i$ because $\mathcal{A}_i$ accepts $t^{\downarrow x}$. Therefore, $t \xrightarrow[\mathcal{R}]{\sigma_i} t[x/s] \in T(\mathcal{A})$, i.e., $t \in \mathrm{pre}_\mathcal{R}(T(\mathcal{A}))$. □

It is easy to see that the set

$$\mathrm{post}_{\mathcal{R}}(T(\mathcal{A})) := \{t \in T_A \mid T(\mathcal{A}) \xrightarrow[\mathcal{R}]{} t\}$$

is regular, too. This directly follows from Theorem 4.1 and the equality

$$\mathrm{post}_{\mathcal{R}}(T(\mathcal{A})) = \mathrm{pre}_{\mathcal{R}^{-1}}(T(\mathcal{A})).$$

COROLLARY 4.2 *Let $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ be an RGTRS and $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA. One can construct an ε-NTA $\mathcal{A}_{\mathrm{post}_{\mathcal{R}}}$ of size $\mathcal{O}(|\mathcal{R}| + |\mathcal{A}|)$ accepting the set $\mathrm{post}_{\mathcal{R}}(T(\mathcal{A}))$.*

4.2.2 Reachability (EF)

In this subsection we prove that the transitive closure of the rewriting relation of RGTRS preserves regularity. Given an RGTRS $\mathcal{R}$ and an ε-NTA $\mathcal{A}$, we show how to construct an ε-NTA $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ accepting the set

$$\mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A})) := \{t \in T_A \mid t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A})\}.$$

The algorithm for computing $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ is shown in Figure 4.2. The idea is similar to the one from the previous section. The automaton $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ contains a copy of the automata $\mathcal{A}_i$, which are accepting the sets from the left hand side of the rewriting rules, and a copy of the automaton $\mathcal{A}$. In this case we do not need two copies of $\mathcal{A}$, as in the case of one step reachability, because we do not have to control the number of rewriting steps that are simulated.

If q is a final state of one of the $\mathcal{A}_i$ accepting the left hand side of the rule $T_i \overset{\sigma}{\hookrightarrow} T_i'$ and p is a state of $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ that can be reached via a tree from T_i', then an ε-transition is added between q and p to enable $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ to simulate this rewriting rule.

Since the state set of the automaton stays fixed during the construction only finitely many transitions can be added. Therefore, the algorithm always terminates. The correctness of the construction follows from the next lemmas.

LEMMA 4.3 *For each automaton $\mathcal{B}_j$ computed in the algorithm from Figure 4.2, each $t \in T_A$, and each $q \in F_i$ for $i \in \{1, \ldots, m\}$ the following holds: If $t \xrightarrow[\mathcal{B}_j]{*} q$, then there exists $s \in T_i$ with $t \xrightarrow[\mathcal{R}]{*} s$.*

Proof. The proof goes by induction on j. The NTA $\mathcal{B}_0$ is the union of $\mathcal{A}$ and all the $\mathcal{A}_i$, where the state sets are assumed to be pairwise disjoint. Therefore, if $t \xrightarrow[\mathcal{B}_0]{*} q$ for some $q \in F_i$, then $t \xrightarrow[\mathcal{A}_i]{*} q$. Hence, $t \in T(\mathcal{A}_i) = T_i$

Algorithm: REACH

INPUT: RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ with $R = \{T_1 \overset{\sigma_1}{\hookrightarrow} T_1', \ldots, T_m \overset{\sigma_m}{\hookrightarrow} T_m'\}$,
NTAs $\mathcal{A}_i = (Q_i, A, \Delta_i, F_i)$ with $T(\mathcal{A}_i) = T_i$ for $i = 1, \ldots, m$
NTAs $\mathcal{A}_i'$ with $T(\mathcal{A}_i') = T_i'$ for $i = 1, \ldots, m$
ε-NTA $\mathcal{A} = (Q, A, \Delta, F)$

1. $\mathcal{B}_0 = (P, A, \Delta_0^{\mathcal{B}}, F)$ with $P := Q \dot\cup (\dot\bigcup_{i=1}^m Q_i)$ and $\Delta_0^{\mathcal{B}} := \Delta \dot\cup (\dot\bigcup_{i=1}^m \Delta_i)$
2. $j := 0$
3. **while** $\exists T_i \hookrightarrow T_i' \in R, p \in P$ with $T_i' \xrightarrow[\mathcal{B}_j]{*} p$ and $(F_i \times \{p\}) \not\subseteq \Delta_j^{\mathcal{B}}$ **do**
4. $\quad \Delta_{j+1}^{\mathcal{B}} := \Delta_j^{\mathcal{B}} \cup (F_i \times \{p\})$
5. $\quad \mathcal{B}_{j+1} := (P, A, \Delta_{j+1}^{\mathcal{B}}, F)$
6. $\quad j := j + 1$
7. **end**

OUTPUT: ε-NTA $\mathcal{A}_{\text{pre}_{\mathcal{R}}^*} := \mathcal{B}_j$

Figure 4.2: An algorithm to compute an automaton for $\text{pre}_{\mathcal{R}}^*(T(\mathcal{A}))$

and the claim holds for $j = 0$ with $s = t$. In the induction step we use a second induction on the number l of transitions from $\Delta_j^{\mathcal{B}} \setminus \Delta_{j-1}^{\mathcal{B}}$ used in a shortest derivation of $t \xrightarrow[\mathcal{B}_j]{*} q$. If $l = 0$, then $t \xrightarrow[\mathcal{B}_{j-1}]{*} q$ and we are done by induction on j. Otherwise let $(q_n, p) \in \Delta_j^{\mathcal{B}} \setminus \Delta_{j-1}^{\mathcal{B}}$ with $q_n \in F_n$ and $x \in D_t$ such that

$$t \xrightarrow[\mathcal{B}_{j-1}]{*} t[x/q_n] \xrightarrow[\mathcal{B}_j]{} t[x/p] \xrightarrow[\mathcal{B}_j]{*} q. \qquad (\star)$$

Then $t^{\downarrow x} \xrightarrow[\mathcal{B}_{j-1}]{*} q_n$ and therefore $t^{\downarrow x} \xrightarrow[\mathcal{R}]{*} s_n$ with $s_n \in T_n$ by induction on j. Since the transition (q_n, p) was added to $\mathcal{B}_{j-1}$ there is $s_n' \in T_n'$ with $s_n' \xrightarrow[\mathcal{B}_{j-1}]{*} p$. Together with $(\star)$ we obtain

$$t[x/s_n'] \xrightarrow[\mathcal{B}_{j-1}]{*} t[x/p] \xrightarrow[\mathcal{B}_j]{*} q.$$

By induction on l there exists $s \in T_i$ with $t[x/s_n'] \xrightarrow[\mathcal{R}]{*} s$. Combining this with $t^{\downarrow x} \xrightarrow[\mathcal{R}]{*} s_n$, $s_n \in T_n$, and $s_n' \in T_n'$ we get

$$t \xrightarrow[\mathcal{R}]{*} t[x/s_n] \xrightarrow[\mathcal{R}]{} t[x/s_n'] \xrightarrow[\mathcal{R}]{*} s. \qquad \square$$

LEMMA 4.4 *For each $\mathcal{B}_j$ computed in the algorithm from Figure 4.2, the inclusion $T(\mathcal{B}_j) \subseteq \mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A}))$ holds.*

Proof. The proof goes by induction on j, similar to the one of Lemma 4.3. For $j = 0$ the claim follows from $T(\mathcal{B}_0) = T(\mathcal{A}) \subseteq \mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A}))$.

For $j \geq 1$ let $t \in T_A$ and $q' \in F$ with $t \xrightarrow[\mathcal{B}_j]{*} q'$. As in the previous lemma we use a second induction on the number l of transitions from $\Delta^{\mathcal{B}}_j \setminus \Delta^{\mathcal{B}}_{j-1}$ used in a shortest derivation of $t \xrightarrow[\mathcal{B}_j]{*} q'$. If $l = 0$, then $t \xrightarrow[\mathcal{B}_{j-1}]{*} q'$ and we are done by induction on j. Otherwise, let $(q, p) \in \Delta^{\mathcal{B}}_j \setminus \Delta^{\mathcal{B}}_{j-1}$ with $q \in F_i$ and $x \in D_t$ such that

$$t \xrightarrow[\mathcal{B}_{j-1}]{*} t[x/q] \xrightarrow[\mathcal{B}_j]{} t[x/p] \xrightarrow[\mathcal{B}_j]{*} q'. \qquad (\star)$$

First of all, since $q \in F_i$ and $t^{\downarrow x} \xrightarrow[\mathcal{B}_{j-1}]{*} q$, we know from Lemma 4.3 that there is $s \in T_i$ such that $t^{\downarrow x} \xrightarrow[\mathcal{R}]{*} s$. Furthermore, since the transition (q, p) was added to $\mathcal{B}_{j-1}$, there is $s' \in T'_i$ with $s' \xrightarrow[\mathcal{B}_{j-1}]{*} p$. Together with $(\star)$ we obtain

$$t[x/s'] \xrightarrow[\mathcal{B}_{j-1}]{*} t[x/p] \xrightarrow[\mathcal{B}_j]{*} q'$$

and therefore $t[x/s'] \in \mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A}))$ by induction on l. From $t^{\downarrow x} \xrightarrow[\mathcal{R}]{*} s$, $s \in T_i$, and $s' \in T'_i$ we get $t \xrightarrow[\mathcal{R}]{*} t[x/s] \xrightarrow[\mathcal{R}]{} t[x/s']$ and hence $t \in \mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A}))$. □

LEMMA 4.5 *For the ε-NTA $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$, computed in the algorithm from Figure 4.2, the inclusion $\mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A})) \subseteq T(\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}})$ holds.*

Proof. First note that $T(\mathcal{B}_{j-1}) \subseteq T(\mathcal{B}_j)$. Let $t \in \mathrm{pre}^*_{\mathcal{R}}(T(\mathcal{A}))$ and let l be the number of rewritings used in a shortest derivation of $t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A})$. If $l = 0$, then $t \in T(\mathcal{A}) = T(\mathcal{B}_0) \subseteq T(\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}})$. If $l \geq 1$, then there exist $x \in D_t$, $i \in \{1, \ldots, m\}$, and $s \in T'_i$ such that $t^{\downarrow x} \in T_i$ and $t[x/s] \xrightarrow[\mathcal{R}]{*} T(\mathcal{A})$ with $l - 1$ steps. By induction $t[x/s] \in T(\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}})$. Let $p \in P$ with

$$t[x/s] \xrightarrow[\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}]{*} t[x/p] \xrightarrow[\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}]{*} F.$$

Since $s \in T'_i$ and $s \xrightarrow[\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}]{*} p$ we know that $F_i \times \{p\} \subseteq \Delta'$ where Δ' is the transition relation of $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$. Otherwise the algorithm would have made one

more step to insert $F_i \times \{p\}$ into the transition relation. Since $t^{\downarrow x} \in T_i$ there is $q \in F_i$ with $t^{\downarrow x} \xrightarrow[\mathcal{A}_i]{*} q$. Then t is accepted by $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$ as follows:

$$t \xrightarrow[\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}]{*} t[x/q] \xrightarrow[\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}]{} t[x/p] \xrightarrow[\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}]{*} F.$$

□

The results from the previous lemmas are summarized in the following theorem.

THEOREM 4.6 *Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be an RGTRS and $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA. Then $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}$ is an ε-NTA of size $\mathcal{O}(|\mathcal{R}| \cdot (|\mathcal{A}| + |\mathcal{R}|))$ with $T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}) = \mathrm{pre}^*_\mathcal{R}(T(\mathcal{A}))$.*

Proof. The two inclusions of the claim $T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}) = \mathrm{pre}^*_\mathcal{R}(T(\mathcal{A}))$ follow from Lemma 4.4 and Lemma 4.5. For the size of the automaton note that $|\mathcal{B}_0| = |\mathcal{A}| + \sum_{i=1}^m |\mathcal{A}_i| \in \mathcal{O}(|\mathcal{A}| + |\mathcal{R}|)$. The algorithm adds ε-transitions of the form (q, p) with $q \in \bigcup_{i=1}^m F_i$ and $p \in P$ to the automaton. There are $(\sum_{i=1}^m |F_i|) \cdot |P|$ transitions of this form. Clearly, $(\sum_{i=1}^m |F_i|) \leq |\mathcal{R}|$ and $|P| \leq |\mathcal{A}| + |\mathcal{R}|$. In total, we get $|\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}| \in \mathcal{O}(|\mathcal{R}| \cdot (|\mathcal{A}| + |\mathcal{R}|))$. □

As for one step reachability one can also apply the algorithm REACH to the inverse rewriting system to obtain an automaton $\mathcal{A}_{\mathrm{post}^*_\mathcal{R}}$ for the set

$$\mathrm{post}^*_\mathcal{R}(T(\mathcal{A})) = \{t \in T_A \mid T(\mathcal{A}) \xrightarrow[\mathcal{R}]{*} t\}.$$

COROLLARY 4.7 *Let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be an RGTRS and $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA. Then $\mathcal{A}_{\mathrm{post}^*_\mathcal{R}} := \mathrm{REACH}(\mathcal{R}^{-1}, \mathcal{A})$ is an ε-NTA of size $\mathcal{O}(|\mathcal{R}| \cdot (|\mathcal{A}| + |\mathcal{R}|))$ with $T(\mathcal{A}_{\mathrm{post}^*_\mathcal{R}}) = \mathrm{post}^*_\mathcal{R}(T(\mathcal{A}))$.*

The main problem for the complexity of REACH is the test in the **while**-loop. Before each iteration one has to check whether there exist i and p with $T'_i \xrightarrow[\mathcal{B}_j]{*} p$. In a naive implementation all such pairs i, p would be computed in each iteration resulting in a rather bad time complexity. In Appendix A.2 we show how to avoid this problem and propose an implementation with the following property.

THEOREM 4.8 *Algorithm $\mathrm{REACH}(\mathcal{R}, \mathcal{A})$ runs in time $O(|\mathcal{R}|^2 \cdot (|\mathcal{A}| + |\mathcal{R}|))$.*

In the next subsection we study the recurrence problem. For the algorithm to solve that problem we have to deal with iterated reachability problems of the form $T(\mathcal{A}) \xrightarrow[\mathcal{R}]{*} T(\mathcal{B}) \xrightarrow[\mathcal{R}]{*} T(\mathcal{C})$. More precisely, we have to compute

the set of all tuples (p, q, r), where p, q, r are states of $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, such that $T(\mathcal{A}(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{B}(q)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{C}(r))$. It is possible to compute for each such tuple the automaton $\mathcal{A}(p) \times (\mathcal{B}(q) \times \mathcal{C}(r)_{\mathrm{pre}^*_\mathcal{R}})_{\mathrm{pre}^*_\mathcal{R}}$ and test it for emptiness. As this method is inefficient, we develop a better one in the sequel.

The following lemma can be used to avoid computations for all tuples (p, q, r) of states by capturing all the tuples with the computation of one automaton.

LEMMA 4.9 *$t \in T(\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}(q))$ iff $t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q))$.*

Proof. Since the algorithm REACH is independent of the set F of final states of $\mathcal{A}$ we get $\mathcal{A}_{\mathrm{pre}^*_\mathcal{R}}(q) = (\mathcal{A}(q))_{\mathrm{pre}^*_\mathcal{R}}$. □

Using the observation $T(\mathcal{A}(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{B}(q)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{C}(r))$ iff there is a tree $t \in T(\mathcal{B}(q))$ such that $t \in \mathrm{post}^*_\mathcal{R}(T(\mathcal{A}(p)))$ and $t \in \mathrm{pre}^*_\mathcal{R}(T(\mathcal{C}(r)))$, nested computations of $\mathrm{pre}^*_\mathcal{R}$ can be avoided. This can be combined with the previous lemma as follows.

LEMMA 4.10 *Let p, q, r be states of ε-NTAs $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, respectively. Then (p, q, r) is reachable in $\mathcal{A}_{\mathrm{post}^*_\mathcal{R}} \times \mathcal{B} \times \mathcal{C}_{\mathrm{pre}^*_\mathcal{R}}$ iff $T(\mathcal{A}(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{B}(q)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{C}(r))$.*

Proof. The state (p, q, r) is reachable in $\mathcal{A}_{\mathrm{post}^*_\mathcal{R}} \times \mathcal{B} \times \mathcal{C}_{\mathrm{pre}^*_\mathcal{R}}$ iff there is $t_2 \in T_A$ such that $t_2 \in T(\mathcal{A}_{\mathrm{post}^*_\mathcal{R}}(p))$, $t_2 \in T(\mathcal{B}(q))$, and $t_2 \in T(\mathcal{C}_{\mathrm{pre}^*_\mathcal{R}}(r))$. According to Lemma 4.9, this holds iff there are $t_1 \in T(\mathcal{A}(q))$ and $t_3 \in T(\mathcal{C}(r))$ such that $t_1 \xrightarrow[\mathcal{R}]{*} t_2 \xrightarrow[\mathcal{R}]{*} t_3$. □

LEMMA 4.11 *Given ε-NTAs $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ one can compute the set of all (p, q, r) with $T(\mathcal{A}(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{B}(q)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{C}(r))$ in time $\mathcal{O}(|\mathcal{R}|^2|\mathcal{B}|(|\mathcal{A}| + |\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|))$.*

Proof. By Lemma 4.10 it is sufficient to compute the set of reachable states in $\mathcal{D} = \mathcal{A}_{\mathrm{post}^*_\mathcal{R}} \times \mathcal{B} \times \mathcal{C}_{\mathrm{pre}^*_\mathcal{R}}$, which can be done in linear time in the size of $\mathcal{D}$ (Proposition 2.14). The size of $\mathcal{D}$ is

$$|\mathcal{D}| = |\mathcal{A}_{\mathrm{post}^*_\mathcal{R}}| \cdot |\mathcal{B}| \cdot |\mathcal{C}_{\mathrm{pre}^*_\mathcal{R}}| \in \mathcal{O}(|\mathcal{R}| \cdot (|\mathcal{A}| + |\mathcal{R}|) \cdot |\mathcal{B}| \cdot |\mathcal{R}| \cdot (|\mathcal{C}| + |\mathcal{R}|)|).$$

Constructing $\mathcal{A}_{\mathrm{post}^*_\mathcal{R}}$ takes time $\mathcal{O}(|\mathcal{R}|^2(|\mathcal{A}| + |\mathcal{R}|))$ and constructing $\mathcal{C}_{\mathrm{pre}^*_\mathcal{R}}$ takes time $\mathcal{O}(|\mathcal{R}|^2(|\mathcal{C}|+|\mathcal{R}|))$. Therefore, the total time is in $\mathcal{O}(|\mathcal{R}|^2|\mathcal{B}|(|\mathcal{A}|+ |\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|))$. □

4.2.3 Recurrence (EGF)

In this subsection we develop an algorithm to compute for an RGTRS $\mathcal{R}$ and an ε-NTA $\mathcal{A}$ the set of all trees t with $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$. Since RGTR graphs are infinite, in general, a path that visits a set $T(\mathcal{A})$ of vertices infinitely often either visits a single vertex from this set infinitely often, or it visits infinitely many different vertices from $T(\mathcal{A})$. To distinguish these two cases we make the following definition for an RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$, $t \in T_A$, and an ε-NTA $\mathcal{A} = (Q, A, \Delta, F)$:

- $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop if there is $t' \in T(\mathcal{A})$ such that $t \xrightarrow[\mathcal{R}]{\omega} t'$.
- $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop if $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ and not $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop.

For the remainder of this section we fix an ε-NTA $\mathcal{A} = (Q, A, \Delta, F)$ and an RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ with $R = \{T_1 \overset{\sigma_1}{\hookrightarrow} T_1', \ldots, T_m \overset{\sigma_m}{\hookrightarrow} T_m'\}$, where the sets T_i, T_i' are accepted by NTAs $\mathcal{A}_i = (Q_i, A, \Delta_i, F_i)$ and $\mathcal{A}_i' = (Q_i', A, \Delta_i', F_i')$, respectively.

To obtain the algorithm we proceed in two steps. In the first step both cases (recurrence with and without loop) are reduced to instances of the reachability problem. This works as follows: Assume that for some $i \in \{1, \ldots, m\}$ and $q \in Q$ we have $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$. Let $t \in T_A$ and assume that a tree s is reachable from t in $\mathcal{R}$ such that there is a location $x \in D_s$ with $s^{\downarrow x} \in T_i$ and $s[x/q] \xrightarrow[\mathcal{A}]{*} F$. Then we can replace this subtree of s at x by an appropriate tree $t' \in T_i'$ with $t' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$. Since $s[x/q] \xrightarrow[\mathcal{A}]{*} F$ we get $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$.

On the other hand, we also show the converse, i.e., if $t \in T_A$ with $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$, then there are $i \in \{1, \ldots, m\}$ and $q \in Q$ with $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$, and from t a tree s with the above properties is reachable.

Hence, we have reduced the problem to a reachability problem. The difficulty is that we used the condition $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ in the definition of the set of trees that has to be reached. This problem is shown to be decidable in the second step.

Step 1: Reduction to Reachability

We start with a technical lemma using that substitutions that are not "synchronized" via a substitution at a common ancestor are independent.

LEMMA 4.12 *Let $t, t_1, t_2, \ldots, t_{n+1} \in T_A$ such that $t \xrightarrow[\mathcal{R}]{*} t_1$, and $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_{n+1}$ with derivation $[x_1/s_1], \ldots, [x_n/s_n]$. If $j \in \{1, \ldots, n\}$ is such that $x_l \not\sqsubseteq x_j$ for all $l \in \{j, \ldots, n\}$, then*

(i) $t \xrightarrow[\mathcal{R}]{*} t_{n+1}[x_j/t_j^{\downarrow x_j}]$ and

(ii) $t_j^{\downarrow x_j} \xrightarrow[\mathcal{R}]{} t_{j+1}^{\downarrow x_j} \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_{n+1}^{\downarrow x_j}$.

Proof. The idea for (i) is to omit all substitutions $[x_l/s_l]$ with $x_j \sqsubseteq x_l$ and $l \geq j$, whereas for (ii) we use exactly these substitutions.

The formal proof is an induction on n. For $n = 0$ the claim obviously holds. For $n \geq 1$ let $j \in \{1, \ldots, n\}$ be such that $x_l \not\sqsubseteq x_j$ for all $l \in \{j, \ldots, n\}$.

In the case $j = n$ we have $t_{n+1}[x_n/t_n^{\downarrow x_n}] = t_n$ because only the subtree $t_n^{\downarrow x_n}$ is modified in the substitution leading from t_n to t_{n+1}, and therefore (i) holds. For (ii) in the case $j = n$ note that $t_{n+1}^{\downarrow x_n} = s_n$ and $t_n^{\downarrow x_n} \xrightarrow[\mathcal{R}]{} s_n$.

If $j \leq n-1$, then by induction $t \xrightarrow[\mathcal{R}]{*} t_n[x_j/t_j^{\downarrow x_j}]$. If $x_j \sqsubseteq x_n$, then $t_{n+1}[x_j/t_j^{\downarrow x_j}] = t_n[x_j/t_j^{\downarrow x_j}]$. Otherwise, x_j and x_n are incomparable, and therefore $t_{n+1}[x_j/t_j^{\downarrow x_j}] = (t_n[x_j/t_j^{\downarrow x_j}])[x_n/s_n]$. Thus, $t \xrightarrow[\mathcal{R}]{*} t_n[x_j/t_j^{\downarrow x_j}] \xrightarrow[\mathcal{R}]{} (t_n[x_j/t_j^{\downarrow x_j}])[x_n/s_n]$.

For (ii) the induction hypothesis is $t_j^{\downarrow x_j} \xrightarrow[\mathcal{R}]{*} t_n^{\downarrow x_j}$. If $x_j \sqsubseteq x_n$, then we can apply the substitution $[x_j^{-1}x_n/s_n]$ to $t_n^{\downarrow x_j}$ to reach $t_{n+1}^{\downarrow x_j}$. Otherwise, we have $t_{n+1}^{\downarrow x_j} = t_n^{\downarrow x_j}$. □

For the reduction to the reachability problem we define the following two sets:

$$\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}) = \left\{ t \in T_A \;\middle|\; \begin{array}{r} \exists x \in D_t, q \in Q, i \in \{1, \ldots, m\} \;:\; t^{\downarrow x} \in T_i, \\ T_i' \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q)) \xrightarrow[\mathcal{R}]{*} T_i, t[x/q] \xrightarrow[\mathcal{A}]{*} F \end{array} \right\},$$

$$\mathrm{Rec}_2(\mathcal{R}, \mathcal{A}) = \left\{ t \in T_A \;\middle|\; \begin{array}{c} \exists x \in D_t, q \in Q, i \in \{1, \ldots, m\} \;:\; t^{\downarrow x} \in T_i, \\ T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q)) \text{ without loop }, t[x/q] \xrightarrow[\mathcal{A}]{*} F \end{array} \right\}.$$

The following lemma shows that it is sufficient to construct an ε-NTA for the set $\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}) \cup \mathrm{Rec}_2(\mathcal{R}, \mathcal{A})$ because the reachability algorithm (Figure 4.2) applied to this ε-NTA yields an automaton for the set $\{t \in T_A \mid t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})\}$.

The conditions for $\mathrm{Rec}_1(\mathcal{R}, \mathcal{A})$ can be checked by an ε-NTA since the problem $T_i' \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q)) \xrightarrow[\mathcal{R}]{*} T_i$ is decidable for each pair of $i \in \{1, \ldots, m\}$ and $q \in Q$ (by Lemma 4.11). Similarly, we can construct an automaton for $\mathrm{Rec}_2(\mathcal{R}, \mathcal{A})$ if we can decide the problem $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop for each pair of $i \in \{1, \ldots, m\}$ and $q \in Q$.

LEMMA 4.13 *Let $t \in T_A$.*

(i) $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ *with loop iff* $t \in \mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}))$.

(ii) $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ *without loop iff* $t \in \mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_2(\mathcal{R}, \mathcal{A})) \setminus \mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}))$.

Proof.

(i) "⇒": If $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop, then there are $t_0, \ldots, t_n \in T_A$ with $n \in \mathbb{N}$ such that $t_k \in T(\mathcal{A})$ for some $k \in \{0, \ldots, n\}$, $t \xrightarrow[\mathcal{R}]{*} t_0$, and

$$t_0 \xrightarrow[\mathcal{R}]{} t_1 \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_n \xrightarrow[\mathcal{R}]{} t_0$$

with derivation $[x_0/s_0], \ldots, [x_n/s_n]$. We choose $j \in \{0, \ldots, n\}$ such that x_j is a minimal location from $\{x_0, \ldots, x_n\}$. This minimality implies that $x_l \not\sqsubset x_j$ for all $l \in \{0, \ldots, n\}$, and $x_j \in D_{t_l}$ for all $l \in \{0, \ldots, n\}$ because $x_j \in D_{t_j}$ and no substitutions are made above x_j. Let $i \in \{1, \ldots, m\}$ be such that $t_j^{\downarrow x_j} \in T_i$ and $s_j \in T'_i$, and let $q \in Q$ with $t_k \xrightarrow[\mathcal{A}]{*} t_k[x_j/q] \xrightarrow[\mathcal{A}]{*} F$. Define the tree $t' = t_k[x_j/t_j^{\downarrow x_j}]$. By applying Lemma 4.12 (i) to the sequence $t_0 \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_j \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_k$ we know $t \xrightarrow[\mathcal{R}]{*} t'$.

To show that $t' \in \mathrm{Rec}_1(\mathcal{R}, \mathcal{A})$, and hence t in $\mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}))$, first note that, by the definition of t' and the properties of t_k, we have already established $(t')^{\downarrow x_j} \in T_i$ and $t'[x_j/q] \xrightarrow[\mathcal{A}]{*} F$. Furthermore, since $x_l \not\sqsubset x_j$ for all $l \in \{0, \ldots, n\}$, we know that $t_j^{\downarrow x_j} \xrightarrow[\mathcal{R}]{} t_{j+1}^{\downarrow x_j} \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_k^{\downarrow x_j} \xrightarrow[\mathcal{R}]{} \cdots \xrightarrow[\mathcal{R}]{} t_j^{\downarrow x_j}$ by Lemma 4.12 (ii). Hence, we get $T'_i \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q)) \xrightarrow[\mathcal{R}]{*} T_i$ because $t_{j+1}^{\downarrow x_j} = s_j \in T'_i$, $t_k^{\downarrow x_j} \in T(\mathcal{A}(q))$, and $t_j^{\downarrow x_j} \in T_i$. Therefore, all conditions are satisfied and $t' \in \mathrm{Rec}_1(\mathcal{R}, \mathcal{A})$.

"⇐": Assume that $t \xrightarrow[\mathcal{R}]{*} t' \in \mathrm{Rec}_1(\mathcal{R}, \mathcal{A})$ and let $x \in D_{t'}$, $q \in Q$, $i \in \{1, \ldots, m\}$ be such that $(t')^{\downarrow x} \in T_i$, $T'_i \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q)) \xrightarrow[\mathcal{R}]{*} T_i$, and $t'[x/q] \xrightarrow[\mathcal{A}]{*} F$. Furthermore, let $s \in T_i$, $s' \in T'_i$, and $s'' \in T(\mathcal{A}(q))$ be such that $s' \xrightarrow[\mathcal{R}]{*} s'' \xrightarrow[\mathcal{R}]{*} s$. From $t'[x/q] \xrightarrow[\mathcal{A}]{*} F$ and $s'' \xrightarrow[\mathcal{A}]{*} q$ we get $t[x/s''] \xrightarrow[\mathcal{A}]{*} F$. Then $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ as follows:

$$t \xrightarrow[\mathcal{R}]{*} t' \xrightarrow[\mathcal{R}]{} t'[x/s'] \xrightarrow[\mathcal{R}]{*} \underbrace{t'[x/s'']}_{\in T(\mathcal{A})} \xrightarrow[\mathcal{R}]{*} t'[x/s] \xrightarrow[\mathcal{R}]{} t'[x/s'].$$

(ii) "⇒": For $t_0 \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop let $t_1, t_2, \ldots$ be an infinite sequence of trees such that

$$t_0 \xrightarrow[\mathcal{R}]{} t_1 \xrightarrow[\mathcal{R}]{} t_2 \cdots$$

with derivation $[x_0/s_0], [x_1/s_1], \ldots$ and $t_l \in T(\mathcal{A})$ for infinitely many $l \in \mathbb{N}$. Among the minimal locations from the derivation there must be one location x such that infinitely many substitutions are made below x. Therefore, we can choose $j \in \mathbb{N}$ such that $x_l \not\sqsubset x_j$ for all $l > j$ and $x_j \sqsubseteq x_l$ for infinitely many l. Note that $x_l \not\sqsubset x_j$ for all $l > j$ implies that $x_j \in D_{t_l}$ for all $l > j$. Since infinitely many trees along the sequence are accepted by $\mathcal{A}$ and since $\mathcal{A}$ has finitely many states, there is $q \in Q$ such that $t_l \xrightarrow[\mathcal{A}]{*} t_l[x_j/q] \xrightarrow[\mathcal{A}]{*} F$ for infinitely many l.

Let $i \in \{1, \ldots, m\}$ with $t_j^{\downarrow x_j} \in T_i$ and $s_j \in T_i'$, and let $k > j$ be such that $t_k \xrightarrow[\mathcal{A}]{*} t_k[x_j/q] \xrightarrow[\mathcal{A}]{*} F$. As in the proof of (i) we define $t' = t_k[x_j/t_j^{\downarrow x_j}]$ and get $t_0 \xrightarrow[\mathcal{R}]{*} t'$ (Lemma 4.12 (i)). Furthermore, we have $(t')^{\downarrow x_j} = t_j^{\downarrow x_j} \in T_i$ and $t'[x_j/q] = t_k[x_j/q] \xrightarrow[\mathcal{A}]{*} F$.

It remains to show that $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop. Since $s_j \in T_i'$, $x_j \sqsubseteq x_l$ for infinitely many l, and $t_l \xrightarrow[\mathcal{A}]{*} t_l[x_j/q] \xrightarrow[\mathcal{A}]{*} F$ for infinitely many l, we know that $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ because $s_j = t_{j+1}^{\downarrow x_j} \xrightarrow[\mathcal{R}]{*} t_{l_1}^{\downarrow x_j} \xrightarrow[\mathcal{R}]{*} t_{l_2}^{\downarrow x_j} \xrightarrow[\mathcal{R}]{*} \cdots$ by Lemma 4.12 (ii) for $t_{l_1}, t_{l_2}, \ldots \in T(\mathcal{A}(q))$. Assume that $T_i' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ with loop. Then there is $s' \in T_i'$ and $s'' \in T(\mathcal{A}(q))$ such that $s' \xrightarrow[\mathcal{R}]{\omega} s''$. Hence, $t'[x_j/s'] \xrightarrow[\mathcal{R}]{\omega} t'[x_j/s'']$ with loop. From $s'' \in T(\mathcal{A}(q))$ and $t'[x_j/q] \xrightarrow[\mathcal{A}]{*} F$ we get $t'[x_j/s''] \in T(\mathcal{A})$ and thus $t'[x_j/s'] \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop. From $t_0 \xrightarrow[\mathcal{R}]{*} t'$, $(t')^{\downarrow x_j} \in T_i$, and $s' \in T_i'$ we get $t_0 \xrightarrow[\mathcal{R}]{*} t'[x_j/s']$ and therefore $t_0 \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop, contradicting the assumption that $t_0 \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop.

"⇐": If $t \in \mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_2(\mathcal{R}, \mathcal{A})) \setminus \mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}))$, then we know from (i) that not $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop. Let $t' \in (\mathrm{Rec}_2(\mathcal{R}, \mathcal{A}) \setminus \mathrm{Rec}_1(\mathcal{R}, \mathcal{A}))$ with $t \xrightarrow[\mathcal{R}]{*} t'$. Let $x \in D_{t'}$, $q \in Q$, $i \in \{1, \ldots, m\}$, and $s' \in T_i'$ be such that $(t')^{\downarrow x} \in T_i$, $s' \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop, and $t'[x/q] \xrightarrow[\mathcal{A}]{*} F$. Then obviously $t'[x/s'] \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ and therefore $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop.

□

A simple consequence of the previous lemma is

LEMMA 4.14 $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}) \Leftrightarrow t \in \mathrm{pre}^*_{\mathcal{R}}(\mathrm{Rec}_1(\mathcal{R},\mathcal{A}) \cup \mathrm{Rec}_2(\mathcal{R},\mathcal{A}))$.

In the sequel we develop a procedure to decide for $i \in \{1,\ldots,m\}$ and $q \in Q$ if $T'_i \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop. This, in turn, enables us to construct an ε-NTA for $\mathrm{Rec}_1(\mathcal{R},\mathcal{A}) \cup \mathrm{Rec}_2(\mathcal{R},\mathcal{A})$.

Step 2: Recurrence without loop

We start by a general analysis of sequences of trees that infinitely often visit a regular set of trees. So, in the beginning we detach from the more specific question "$T'_i \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop" and study general sequences of trees starting from some tree t and visiting some regular set $T(\mathcal{A})$ infinitely often without loop. The goal is to find a normal form for such sequences. Later, we use this normal form to solve our original problem "$T'_i \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop". The normal form is obtained from a given sequence of trees by removing unnecessary rewritings, where unnecessary means that these rewritings are not essential for visiting the set $T(\mathcal{A})$ infinitely often. A first tool is the limit of a sequence of trees. It contains all the locations of the trees in this sequence that eventually stay fixed.

To simplify notation, a path π always means an $\mathcal{R}$-path π for the RGTRS fixed at the beginning of this subsection. Let $[x_0/s_0],[x_1/s_1],\ldots$ be the derivation of an infinite path π starting in a tree $t \in T_A$. A location $x \in \mathbb{N}^*$ is called stable on π if it is present in all the trees from π, and if it is never involved in any of the substitutions. More formally, $x \in \mathbb{N}^*$ is stable on π if $x \in D_{\pi(i)}$ and $x_i \not\sqsupseteq x$ for all $i \in \mathbb{N}$. Recall that $\pi(i)$ denotes the ith vertex on the path π. Since the vertices of an $\mathcal{R}$-path are trees, $\pi(i)$ denotes the ith tree on π.

The limit $\lim(\pi)$ of π is defined as

$$\lim(\pi) = \{x \in \mathbb{N}^* \mid \exists i \in \mathbb{N} \ : \ x \text{ stable on } \pi[i,\infty)\}.$$

Note that $\lim(\pi)$ is prefix closed since x stable on $\pi[i,\infty)$ for some $i \in \mathbb{N}$ implies, by the definition of stable, that each prefix y of x is stable on $\pi[i,\infty)$. Furthermore, note that x stable on $\pi[i,\infty)$ for some $i \in \mathbb{N}$ also implies that x is stable on $\pi[j,\infty)$ for all $j > i$. We will use this fact implicitly in several proofs.

For a location $x \notin \lim(\pi)$ there are two possibilities: either x is involved in infinitely many substitutions or x only occurs in finitely many trees on π. We are interested in paths where the latter case holds for all locations that

are not in the limit of the paths. An infinite path π is called stable if for all $x \notin \lim(\pi)$ there exists an $i \in \mathbb{N}$ such that $x \notin D_{\pi(j)}$ for all $j \geq i$. Informally speaking, π is stable if there is no location that is involved in infinitely many substitutions on π. A first observation is that $\pi : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop implies that π is stable.

LEMMA 4.15 *If $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop and if π is an infinite path with $\pi : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$, then π is stable.*

Proof. Let $\pi = t_0, t_1, t_2, \ldots$ be an infinite path with $\pi : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$, and let $[x_0/s_0], [x_1/s_1], \ldots$ be the derivation of π. Let X be the set of locations that are minimal among those locations that occur infinitely often in the sequence $x_0, x_1, \ldots$ of locations from the derivation. If X is empty, then π is stable because no location is involved in infinitely many substitutions. In the case that X is not empty we show that $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop. By the assumption that $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop this implies that X must be empty and therefore π is stable.

We start with the case $|X| = 1$ and then show how reduce the general case to X being a singleton set. So assume that $X = \{x\}$. By the minimality of x there exists an index $k \in \mathbb{N}$ such that $x_j \not\sqsupset x$ for all $j \geq k$. Since x equals infinitely many of the x_j there must be $i \in \{1, \ldots, m\}$ such that $x = x_j$, $t_j^{\downarrow x} \in T_i$ and $s_j \in T_i'$ for infinitely many j. Then we can find j, j_1, j_2 with $k \leq j_1 < j < j_2$ such that $x = x_{j_1} = x_{j_2}$, $t_{j_1}^{\downarrow x}, t_{j_2}^{\downarrow x} \in T_i$, $s_{j_1}, s_{j_2} \in T_i'$, and $t_j \in T(\mathcal{A})$.

From Lemma 4.12 (i) we know that $t = t_0 \xrightarrow[\mathcal{R}]{*} t_j[x/s_{j_1}]$, and from Lemma 4.12 (ii) we know that $s_{j_1} \xrightarrow[\mathcal{R}]{*} t_j^{\downarrow x} \xrightarrow[\mathcal{R}]{*} t_{j_2}^{\downarrow x}$. Furthermore, $t_{j_2}^{\downarrow x} \xrightarrow[\mathcal{R}]{} s_{j_1}$ because $t_{j_2}^{\downarrow x} \in T_i$ and $s_{j_1} \in T_i'$. Therefore, we get

$$t \xrightarrow[\mathcal{R}]{*} t_j[x/s_{j_1}] \xrightarrow[\mathcal{R}]{*} t_j[x/t_j^{\downarrow x}] \xrightarrow[\mathcal{R}]{*} t_j[x/t_{j_2}^{\downarrow x}] \xrightarrow[\mathcal{R}]{} t_j[x/s_{j_1}].$$

Since $t_j[x/t_j^{\downarrow x}] = t_j \in T(\mathcal{A})$ we get $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop.

If X contains more than one location, we choose some $x \in X$. Since x is minimal among the locations occurring infinitely often in the derivation, there must be a suffix of π such that x is in the domain of all trees on this suffix of π. Therefore, there exists $q \in Q$ such that $t_j \xrightarrow[\mathcal{A}]{*} t_j[x/q] \xrightarrow[\mathcal{A}]{*} F$ for infinitely many $j \in \mathbb{N}$. Again by the minimality of x, there is $k \in \mathbb{N}$ such that $x_j \not\sqsupset x$ for all $j \geq k$. We choose k such that $t_k \xrightarrow[\mathcal{A}]{*} t_k[x/q] \xrightarrow[\mathcal{A}]{*} F$. Then we remove from the derivation of π all substitutions $[x_j/s_j]$ with $j \geq k$

such that x and x_j are not comparable w.r.t. $\sqsubseteq$. Since there is no x_j with $j \geq k$ such that $x_j \sqsubseteq x$ (by the choice of k) the sequence of substitutions obtained like this is a derivation of a path π'. This path π' has the following properties:

- It is still infinite because there are infinitely many x_j with $x = x_j$.
- By the construction of π', the choice of k, and the choice of q we have $t[x/q] \xrightarrow[\mathcal{A}]{*} F$ for each tree t on the suffix $\pi'[k, \infty)$. Since all substitutions $[x_j/s_j]$ with $j \geq k$ and $x \sqsubseteq x_j$ remain in the derivation of π' we know that $(t')^{\downarrow x} \xrightarrow[\mathcal{A}]{*} q$ for infinitely many trees t' on $\pi'[k, \infty)$. All these trees t' are accepted by $\mathcal{A}$ and thus $\pi' : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$.
- Since all substitutions $[x_j/s_j]$ with $j \geq k$ are removed if x and x_j are incomparable, the only minimal location that appears infinitely often in the derivation of π' is x.

Hence, π' is an infinite path with $\pi' : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$. The set of minimal locations that appear infinitely often in the derivation of π' has cardinality one. Therefore, $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ with loop as we have seen above. □

If an infinite path π is stable, then there is no location involved in infinitely many substitutions. Hence, the domain of the trees on π must grow. Therefore, from the previous lemma, we can conclude that the trees on paths π with $\pi : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ without loop grow indefinitely. To develop an algorithm that can check whether such a path exists, we show that if such a path exists, then there also exists a path π with the following "nice" properties.

- The trees on π only "grow in one direction", i.e., there is no indefinite growth in independent parts of the trees.
- For infinitely many trees on π there are accepting runs of $\mathcal{A}$. These runs do not differ too much, i.e., there are infinitely many trees on π such that there are accepting runs on these trees that agree on growing initial segments of the trees.

These properties are formalized with the next definitions and the subsequent lemma.

A branch β of a prefix closed set $X \subseteq \mathbb{N}^*$ is a maximal subset of X that is linearly ordered by $\sqsubseteq$. The initial segment $\beta^{\uparrow x}$ of β up to a location $x \in \beta$ is the set

$$\beta^{\uparrow x} = \{y \in \beta \mid y \sqsubset x\}.$$

For $x \in \beta$ we denote by $\mathrm{succ}_\beta(x)$ the successor of x on β. With this definition we get the equality

$$\beta^{\uparrow \mathrm{succ}_\beta(x)} = \beta^{\uparrow x} \cup \{x\}.$$

The following lemma shows that the desired properties of π, which are described above, can always be guaranteed.

LEMMA 4.16 *If* $t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ *without loop, then there is an infinite path* π *with* $\pi : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ *without loop and the following properties:*

(i) $\lim(\pi)$ *has exactly one infinite branch* β*.*

(ii) For the infinite branch β *from (i) there is a mapping* $\rho_\beta : \beta \to Q$ *with the following property. For each* $x \in \beta$ *there are infinitely many* $i \in \mathbb{N}$ *such that there is an accepting run* ρ_i *of* $\mathcal{A}$ *on* $\pi(i)$ *that agrees with* ρ_β *on the initial segment* $\beta^{\uparrow x}$ *of* β*.*

Proof. Let π' be an infinite path with $\pi' : t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})$ and derivation $[x_0/s_0], [x_1/s_1], \ldots$

From Lemma 4.15 we can conclude that π' is stable and that $\lim(\pi')$ is infinite. Since $\lim(\pi')$ is finitely branching we can choose an infinite branch β of $\lim(\pi')$ by König's Lemma. We first define the mapping ρ_β and then remove all the rewritings from π' that might generate other infinite branches.

To define ρ_β we again make use of König's Lemma by defining a graph with vertices of the form (τ, x) with location $x \in \beta$ and partial mapping τ defined on $\beta^{\uparrow x}$. Formally, (τ, x) is a vertex of this graph if $\tau : \beta \to Q$ is defined on all $y \in \beta^{\uparrow x}$ and undefined on all $y \in \beta \setminus \beta^{\uparrow x}$, and if there are infinitely many trees on π' that are accepted by $\mathcal{A}$ with an accepting run that agrees with τ on $\beta^{\uparrow x}$. Note that (τ, ε), where τ is the mapping that is undefined everywhere, is a vertex of the graph because $\beta^{\uparrow \varepsilon}$ is empty and therefore every accepting run agrees with τ on $\beta^{\uparrow \varepsilon}$. Furthermore, this graph has infinitely many vertices because for each x the initial segment $\beta^{\uparrow x}$ is finite. Thus, among the infinitely many accepting runs on trees on π', there must be infinitely many that agree on $\beta^{\uparrow x}$, defining a mapping τ such that (τ, x) is a vertex of the graph.

We continue by defining the edge relation of the graph. Between two vertices (τ_1, x_1) and (τ_2, x_2) there is an edge if $x_2 = \mathrm{succ}_\beta(x_1)$, and $\tau_1(y) = \tau_2(y)$ for all $y \in \beta^{\uparrow x_1}$.

This graph is an infinite tree (in the graph theoretic sense) with root (τ, ε). The degree of this graph is bounded by $|Q|$. Therefore, by König's Lemma, there is an infinite path through this graph. The mappings on this

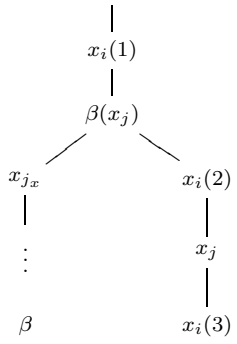
path agree on growing initial segments of β. In the limit this path defines the mapping ρ_β.

To obtain π from π', we stop rewriting subtrees that "leave" β above a location $x \in \beta$ as soon as x becomes stable and a tree is reached that is accepted by $\mathcal{A}$ with a run that agrees with ρ_β on an initial segment of β including x.

More formally, this looks as follows. For $y \in \mathbb{N}^*$ let $\beta(y)$ be the maximal prefix of y that is in β. For $x \in \beta$ let $j_x \in \mathbb{N}$ be minimal such that x is stable on $\pi'[j_x, \infty)$ and $\pi'(j_x)$ is accepted by $\mathcal{A}$ with a run that agrees with ρ_β on $\beta^{\uparrow \mathrm{succ}_\beta(x)}$. To see that such j_x exists first note that there exists $j \in \mathbb{N}$ such that x is stable on $\pi[i, \infty)$ for all $i \geq j$. Furthermore, by the definition of ρ_β, there are infinitely many $\pi(j)$ that are accepted by $\mathcal{A}$ with a run that agrees with ρ_β on $\beta^{\uparrow \mathrm{succ}_\beta(x)}$. Hence, we can choose j_x with the desired properties.

To obtain the path π we remove from the derivation of π' all substitutions $[x_j/s_j]$ if there is $x \in \beta$ such that $j \geq j_x$ and $\beta(x_j) \sqsubset x_{j_x}$.

We show that the sequence of substitutions obtained in this way is a derivation of a path π. Assume that a substitution $[x_j/s_j]$ is removed. We have to show that all substitutions $[x_i/s_i]$ with $i > j$ and $x_i \sqsubseteq x_j$ or $x_j \sqsubseteq x_i$ are also removed because these are the only substitutions depending on $[x_j/s_j]$. We distinguish three cases for x_i indicated by the three occurrences of x_i in the following picture:



In case 2, i.e., $x_i \notin \beta$ and $x_i \sqsubseteq x_j$, and in case 3, i.e., $x_j \sqsubseteq x_i$, it is clear that $[x_i/s_i]$ is removed from π' because $\beta(x_i) = \beta(x_j)$ and $i > j \geq j_x$.

Case 1, i.e., $x_i \in \beta$ and $x_i \sqsubseteq x_j$, cannot occur because $i > j \geq j_x$ and x_{j_x} is stable on $\pi'[j_x, \infty)$. Therefore, x_{j_x} is also stable on $\pi'[i, \infty)$ and x_i cannot be a prefix of x_{j_x} by definition of stable. Thus, the new sequence of substitutions is indeed the derivation of a path π.

We show that π is infinite and that β is the only infinite branch of π: Since j_x was chosen to be minimal with the required properties it is clear

that $j_y \leq j_x$ for $y \sqsubseteq x$. This implies that no substitutions $[x_j/s_j]$ with $x_j \in \beta$ are removed because if $y \in \beta$ is a proper prefix of $x \in \beta$, then y is stable on $\pi'[j_x, \infty)$. There are infinitely many substitutions $[x_j/s_j]$ with $x_j \in \beta$ since β is an infinite branch of $\lim(\pi')$. Hence, the new derivation is infinite. It also follows that β is an infinite branch of $\lim(\pi)$.

Let β' be a branch of $\lim(\pi)$ different from β. Let $y \in \beta' \setminus \beta$ and let $x \in \beta$ be such that $\beta(y) \sqsubset x$. Then all substitutions $[x_i/s_i]$ with $i \geq j_x$ and $x_i \in \beta' \setminus \beta$ were removed. Therefore, β' must be finite.

It remains to show that ρ_β satisfies (ii). For this purpose we define the mapping $\varphi : \mathbb{N} \to \mathbb{N}$, relating the trees on π with trees on π', inductively by $\varphi(0) = 0$ and

$$\varphi(j+1) = \begin{cases} \varphi(j) & \text{if the substitution } [x_j/s_j] \text{ was removed,} \\ \varphi(j)+1 & \text{otherwise.} \end{cases}$$

By the definition of π, the tree $\pi(\varphi(j_x))$ is accepted for each $x \in \beta$ by $\mathcal{A}$ with a run that agrees with ρ_β on $\beta^{\uparrow x}$. Since there are infinitely many substitutions that are not removed from π', the mapping φ is not ultimately constant. Furthermore, the sequence $(j_x)_{x \in \beta}$ is not bounded. Therefore, for each $x \in \beta$, there are infinitely many accepting runs that agree with ρ_β on $\beta^{\uparrow x}$. □

Having established this normal form, we are ready to develop the decision procedure. The idea is to simulate the substitutions on π along β with a finite amount of information. We define a finite graph $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ with edges labeled from the set $\{0, \ldots, k-1, /, !\}$, where k was the maximal rank of symbols used in the RGTRS $\mathcal{R}$. Passing an edge in $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ corresponds to different actions in the simulation of the substitutions along β. The symbols $0, \ldots, k-1$ mean that we go down the branch β, the symbol / means that we simulate a sequence of substitutions, and the symbol ! means that we simulate a sequence of substitutions such that in between a tree from the set that should be visited infinitely often occurs. An infinite path with infinitely many !-edges through $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ corresponds to an $\mathcal{R}$-path with infinitely many trees from the set that should be visited infinitely often.

We first define the graph $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ and then give a more detailed description of the idea and the correctness proofs.

For $i, j \in \{1, \ldots, m\}$ we define

$$T_i \rhd_{\mathcal{R}} T'_j :\Leftrightarrow i = j \text{ or } T'_i \xrightarrow[\mathcal{R}]{*} T_j.$$

We need the following property of this relation.

REMARK 4.17 If $T_i \rhd_{\mathcal{R}} T'_j$, then $t \xrightarrow[\mathcal{R}]{*} t'$ for all $t \in T_i$, $t' \in T'_j$.

Recall that $\mathcal{A}'_i$ denotes an NTA for T'_i, the right hand side of the ith rewriting rule of $\mathcal{R}$. Define the automaton $\mathcal{B}' = \bigcup_{i=1}^{m} \mathcal{A}'_i$ and call its state set P', i.e., $P' = \dot{\bigcup}_{i=1}^{m} Q'_i$.

DEFINITION 4.18 The finite graph $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ has the vertex set $P' \times Q \times Q$. The edges of $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ are labeled from the set $\{0, \ldots, k-1, /, !\}$ and are defined as follows.

(1) $(p, q, r) \xrightarrow{i} (\hat{p}, \hat{q}, \hat{r})$ for $i \in \{0, \ldots, k-1\}$ iff there is $l \in \{i+1, \ldots, k\}$, $a \in A_l$, a transition $(p_0, \ldots, p_{l-1}, a, p)$ in $\mathcal{B}'$, and transitions $(q_0, \ldots, q_{l-1}, a, q)$, $(r_0, \ldots, r_{l-1}, a, r)$ in $\mathcal{A}$ with the following properties.

- $\hat{p} = p_i$, $\hat{q} = q_i$, $\hat{r} = r_i$.
- For each $j \in \{0, \ldots, l-1\} \setminus \{i\}$ there is a tree t_j with $t_j \xrightarrow[\mathcal{B}']{*} p_j$ and $t_j \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r_j)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q_j))$.

(2) $(p, q, r) \xrightarrow{/} (\hat{p}, q, r)$ iff there are $i, j \in \{1, \ldots, m\}$ such that $\hat{p} \in F'_j$, $T_i \rhd_{\mathcal{R}} T'_j$, and $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T_i$,

(3) $(p, q, r) \xrightarrow{!} (\hat{p}, q, q)$ iff there are $i, j \in \{1, \ldots, m\}$ such that $\hat{p} \in F'_j$, $T_i \rhd_{\mathcal{R}} T'_j$, $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T_i$, and

- $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T_i$ or
- $T'_i \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T_j$.

Note that the conditions in (3) extend the conditions from (2) by the last two items. □

Assume that there is a path π according to Lemma 4.16 that visits $T(\mathcal{A}(q))$ infinitely often and starts with a tree t from T'_i. Let $p \in F'_i$ be such that $t \in T(\mathcal{B}'(p))$.

The idea of the graph $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ is to simulate the substitutions that are made on π along the branch β. During this simulation we go down the branch β and jump to increasing positions on the path π. So, we are always at a location y in β and at a position n on π such that y is in $D_{\pi(n)}$.

The first component of the $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ vertices holds information about the trees on π, namely what state of $\mathcal{B}'$ may be reached when reading the subtree $\pi(n)^{\downarrow y}$.

The second component of the $G_{\mathrm{Rec}}(\mathcal{R},\mathcal{A})$ vertices keeps track of the value $\rho_\beta(y)$. We know that there are infinitely many trees on π that are accepted with runs of $\mathcal{A}$ that agree with ρ_β on growing initial segments of β. Nevertheless, these runs may differ from ρ_β from a certain location onwards. The third component of the $G_{\mathrm{Rec}}(\mathcal{R},\mathcal{A})$ vertices contains this information. As soon as we pass an accepted tree, the third component is reset to the value of the second component, and we wait for the next tree that is accepted with a run that agrees with ρ_β on the initial segment of β up to the current location y.

We start at position $n = 0$ on π at the location $y = \varepsilon$ at β with the vertex (p, q, q). An edge labeled with $i \in \{0, \ldots, k-1\}$ means that we go down one step along β to $\mathrm{succ}_\beta(y) = yi$, so the new y is the successor of the old y on β. The edges labeled with / mean that we jump to a new position n on π, namely to the position just after the last substitution at the current location y on β. Finally, the edges labeled with ! mean the same as the /-edges with the difference that between the current position n and the new position n on π there is a tree that is accepted by $\mathcal{A}(q)$ with a run that agrees with ρ_β on the initial segment of β up the location that was the current location the last time an !-edge was taken. Condition (ii) of Lemma 4.16 ensures that we can infinitely often use such an !-edge.

LEMMA 4.19 *Let $i \in \{1, \ldots, m\}$ and $q \in Q$. If $T'_i \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$ without loop, then there is $p \in F'_i$ and an infinite path with infinitely many !-edges through $G_{\mathrm{Rec}}(\mathcal{R},\mathcal{A})$ starting from (p, q, q).*

Proof. Let π be an infinite path according to Lemma 4.16 and let β be the unique infinite branch in $\lim(\pi)$. We know that the first tree $\pi(0)$ on π is in T'_i. Let $p \in F'_i$ such that there is an accepting run ρ'_ε of $\mathcal{A}'_i$ on $\pi(0)$ with $\rho'_\varepsilon(\varepsilon) = p$.

To formalize the idea described above we inductively define a sequence of tuples $(p_j, q_j, r_j, y_j, z_j, z'_j, n_j)$ such that the sequence (p_j, q_j, r_j) is an infinite path with infinitely many !-edges through $G_{\mathrm{Rec}}(\mathcal{R},\mathcal{A})$ starting from (p, q, q). The other auxiliary components keep track of the current location y_j on β, the location z_j that was the current location the last time an !-edge was used, the location z'_j that was the current location the last time a /-edge was used, and the current position n_j on π. For this inductive definition we need some notation.

For $x \in \beta$ let $\mathrm{stable}(x)$ denote the minimal position on π such that x is stable on $\pi[\mathrm{stable}(x), \infty)$. There are two possibilities for the last substitution before the position $\mathrm{stable}(x)$. Either a subtree rooted at a proper prefix of

x was rewritten, or the subtree at x was rewritten itself. For those locations where the latter holds we know that $(\pi(\text{stable}(x)))^{\downarrow x} \in T'_l$ for some $l \in \{1, \dots, m\}$. Hence, we can fix an accepting run ρ'_x of $\mathcal{A}'_l$ on $(\pi(\text{stable}(x)))^{\downarrow x}$. We will need these runs to update the first component of our tuples, which are states from $\mathcal{B}'$.

The initial tuple is $(p_0, q_0, r_0, y_0, z_0, z'_0, n_0) = (p, q, q, \varepsilon, \varepsilon, \varepsilon, 0)$. Assume that for $j \in \mathbb{N}$ the tuple $(p_j, q_j, r_j, y_j, z_j, z'_j, n_j)$ is already defined. By Lemma 4.16 (ii) we can choose a minimal $l_j \geq n_j$ such that $\pi(l_j)$ is accepted by a run ρ_j that agrees with ρ_β on $\beta^{\uparrow \text{succ}_\beta(z_j)}$. To define the next tuple we have to distinguish three cases (corresponding to the three types (1),(2),(3) of edges in $G_{\text{Rec}}(\mathcal{R}, \mathcal{A})$).

(a) If y_j is stable on $\pi[n_j, \infty)$, then

$$\begin{aligned} &y_{j+1} = \text{succ}_\beta(y_j), \quad && z_{j+1} = z_j && z'_{j+1}, = z'_j \\ &n_{j+1} = n_j, \\ &p_{j+1} = \rho'_{z'_{j+1}}(y_{j+1}), && q_{j+1} = \rho_\beta(y_{j+1}), && r_{j+1} = \rho_j(y_{j+1}). \end{aligned}$$

If y_j is not stable on $\pi[n_j, \infty)$, then we distinguish two sub cases.

(b) If $l_j \geq \text{stable}(y_j)$, then

$$\begin{aligned} &y_{j+1} = y_j, && z_{j+1} = z_j && z'_{j+1}, = y_j \\ &n_{j+1} = \text{stable}(y_j), \\ &p_{j+1} = \rho'_{y_j}(y_j), && q_{j+1} = q_j, && r_{j+1} = r_j. \end{aligned}$$

(c) If $l_j < \text{stable}(y_j)$, then

$$\begin{aligned} &y_{j+1} = y_j, && z_{j+1} = y_j && z'_{j+1}, = y_j \\ &n_{j+1} = \text{stable}(y_j), \\ &p_{j+1} = \rho'_{y_j}(y_j), && q_{j+1} = q_j, && r_{j+1} = q_j. \end{aligned}$$

This sequence corresponds to an infinite path through $G_{\text{Rec}}(\mathcal{R}, \mathcal{A})$ in the following sense. If tuple $j+1$ was defined from tuple j using rule (a), (b), or (c), then there is an edge in $G_{\text{Rec}}(\mathcal{R}, \mathcal{A})$ between (p_j, q_j, r_j) and $(p_{j+1}, q_{j+1}, r_{j+1})$ of type (1), (2), or (3), respectively.

First note that $\pi(n_j)^{\downarrow y_j} \xrightarrow[\mathcal{B}']{*} p_j$ for each j. This easily follows by induction on j with the definition of the runs $\rho'_{z'_{j+1}}$ and ρ'_{y_j}.

If rule (a) is used, then the conditions for an edge of type (1) are satisfied because y_j is stable on $\pi[n_j, \infty)$ and therefore we can reach from $\pi(n_j)^{\downarrow y_j}$ a tree in $T(\mathcal{A}(r_j))$, namely $\pi(l_j)$, and then a tree that is accepted with a run

that agrees with ρ_β on $\beta^{\uparrow \mathrm{succ}_\beta(y_j)}$. Hence, the transitions as required in (1) must exist.

If rule (b) is used, then y_j is the minimal location of β that is not stable on $\pi[n_j, \infty)$. This can be shown by induction since we only pass to the next location on β (by using rule (a)) if the current location is stable. Thus, there must be a first substitution at y_j on $\pi[n_j, \infty)$, using $T_{i_1} \hookrightarrow T'_{i_1}$ for some i_1. Since $\pi(n_j)^{\downarrow y_j} \xrightarrow[\mathcal{B}']{*} p_j$ (see above) we get $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T_{i_1}$. By definition of n_{j+1} the last substitution at y_j occurs at position $n_{j+1}-1$, using $T_{i_2} \hookrightarrow T'_{i_2}$ for some i_2. Therefore, $T_{i_1} \rhd_{\mathcal{R}} T'_{i_2}$. Furthermore, recall that ρ'_{y_j} is an accepting run of $\mathcal{A}'_{i_2}$ on $\pi(\mathrm{stable}(y_j))^{\downarrow y_j}$ and hence $p_{j+1} \in T'_{i_2}$. Thus, the conditions for an edge of type (2) between (p_j, q_j, r_j) and $(p_{j+1}, q_{j+1}, r_{j+1})$ are satisfied for $i_1, i_2 \in \{1, \ldots, m\}$. Similarly, the conditions for an edge of type (3) between (p_j, q_j, r_j) and $(p_{j+1}, q_{j+1}, r_{j+1})$ are satisfied if rule (c) is used because the tree $\pi(l_j)$ is lying between $\pi(n_j)$ and $\pi(n_{j+1})$.

It remains to show that (c) is used infinitely often to obtain a path with infinitely many !-edges. A first observation is that if rule (b) was used, then rule (a) is used in the next step because in (b) we have the definitions $y_{j+1} = y_j$ and $n_{j+1} = \mathrm{stable}(y_j)$.

As long as (c) is not used, (b) must be used eventually because only finitely many locations from β can be stable on a fixed suffix of π. But (b) cannot be used twice in series, therefore (a) and (b) are used in turn (where there may be more than one application of (a) in series).

If (a) or (b) is used, then $l_{j+1} = l_j$. But every time (b) is used the n_j value increases. By definition of l_j we have $l_j \geq n_j$. Therefore, rule (c) has to be applied eventually. □

LEMMA 4.20 *If there is an infinite path containing infinitely many* !*-edges through* $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ *starting from* (p, q, q)*, for some* $p \in F'_i$ *and* $q \in Q$*, then* $T'_i \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q))$.

Proof. We cut the infinite path through $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ into segments ending with an !-edge and show what kind of $\mathcal{R}$-path we can construct from such a finite path segment in $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$. The desired result is then obtained by concatenating these segments, as we will see later.

So let $p, p' \in P'$, $q, r, q' \in Q$, and $v_1, \ldots, v_n$ be vertices of $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ such that

$$(p, q, r) \xrightarrow{\lambda_1} v_1 \xrightarrow{\lambda_2} v_2 \xrightarrow{\lambda_3} \cdots \xrightarrow{\lambda_n} v_n \xrightarrow{!} (p', q', q')$$

in $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ with $\lambda_1, \ldots, \lambda_n \in \{0, \ldots, k-1, /\}$. Let x be the location that is obtained from the sequence $\lambda_1 \cdots \lambda_n$ by omitting all /.

We prove the following claim by induction on n.

Claim: There is $t \in T(\mathcal{B}'(p))$ such that for all $t' \in T(\mathcal{B}'(p'))$ there is $t'' \in T_A$ with

$$t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{+} t'' \text{ with } (t'')^{\downarrow x} = t' \text{ and } t''[x/q'] \xrightarrow[\mathcal{A}]{*} q. \qquad (\star)$$

If $n = 0$, then $(p, q, r) \xrightarrow{!} (p', q', q')$ with $q' = q$ and $x = \varepsilon$. By definition of the !-edges there are $i, j \in \{1, \dots, m\}$ such that $p' \in F'_j$, $T_i \rhd_{\mathcal{R}} T'_j$, and there is $t \in T(\mathcal{B}'(p))$ with $t \xrightarrow[\mathcal{R}]{*} T_i$. Let $t' \in T(\mathcal{B}'(p'))$ and set $t'' = t'$. Then $t'' \in T'_j$ because $t'' = t' \in T(\mathcal{B}'(p'))$ and $p' \in F'_j$. We have to show $(\star)$ for this t and t''. By definition of t'' and because $x = \varepsilon$ and $q' = q$ we obviously have $(t'')^{\downarrow x} = t'$ and $t''[x/q'] \xrightarrow[\mathcal{A}]{*} q$.

From the definition of !-edges it follows that either $t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T_i$ or $T'_i \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T_j$. In both cases we get $t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{+} t''$ because $t'' \in T'_j$. In the first case we use $T_i \rhd_{\mathcal{R}} T'_j$ and Remark 4.17. In the second case we use that $t \xrightarrow[\mathcal{R}]{*} T_i$ and $t'' \in T'_j$.

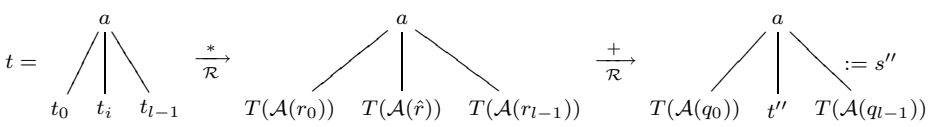
If $n \geq 1$, then

$$(p, q, r) \xrightarrow{\lambda_1} (\hat{p}, \hat{q}, \hat{r}) \xrightarrow{\lambda_2} v_2 \xrightarrow{\lambda_3} \cdots \xrightarrow{\lambda_n} v_n \xrightarrow{!} (p', q', q').$$

Let $\hat{x}$ be the location obtained from $\lambda_2 \cdots \lambda_n$ by omitting /. By the induction hypothesis there is $\hat{t} \in T(\mathcal{B}'(\hat{p}))$ such that for all $t' \in T(\mathcal{B}'(p'))$ there is a t'' such that $(\star)$ is valid for $\hat{t}, \hat{x}, \hat{q}, \hat{r}, t'$, and t''.

If $\lambda_1 = /$, then $x = \hat{x}$, $q = \hat{q}$, $r = \hat{r}$. Furthermore, there are $i, j \in \{1, \dots, m\}$ such that $\hat{p} \in F'_j$, $T_i \rhd_{\mathcal{R}} T'_j$, and there is $t \in T(\mathcal{B}'(p))$ with $t \xrightarrow[\mathcal{R}]{*} T_i$. But $\hat{p} \in F'_j$ implies $\hat{t} \in T'_j$ and then $T_i \rhd_{\mathcal{R}} T'_j$ implies $t \xrightarrow[\mathcal{R}]{*} \hat{t}$, by Remark 4.17. Since $(\star)$ holds for $\hat{t}$ it also holds for t because $x = \hat{x}$, $q = \hat{q}$, and $r = \hat{r}$.

Now consider the case $\lambda_1 = i \in \{0, \dots, k-1\}$. Let $l \in \{i+1, \dots, k\}$, $a \in A_l$, p_j, q_j, r_j for $j \in \{0, \dots, l-1\}$, and t_j for $j \in \{0, \dots, l-1\} \setminus \{i\}$ be as required in item (1) from the definition of $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$, and define $t_i = \hat{t}$. Let $t' \in T(\mathcal{B}'(p'))$. For $t = a(t_1, \dots, t_l)$ the part $t \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{+} s''$ of the claim follows as indicated in the picture below. In this picture sets of trees as, e.g., $T(\mathcal{A}(r_0))$ mean that there is a tree from that set that can be substituted at this position. The rightmost tree in the picture is defined to be s''.



Note that the tree in the middle is in $T(\mathcal{A}(r))$ because $\hat{r} = r_i$ and the transition $(r_0, \ldots, r_{l-1}, a, r)$ is in $\mathcal{A}$.

We know that $t''[x/q'] \xrightarrow[\mathcal{A}]{*} \hat{q}$. From $\hat{q} = q_i$ follows $s''[ix/q'] \xrightarrow[\mathcal{A}]{*} q$ because the transition $(q_0, \ldots, q_{l-1}, a, q)$ is in $\mathcal{A}$. Furthermore, we have $(s'')^{\downarrow ix} = (t'')^{\downarrow x} = t'$. This ends the proof of the claim. Now we show how to iterate this result.

On a path with infinitely man !-edges starting in (p_1, q_1, q_1) with $p_1 \in F'_i$ let $(p_2, q_2, q_2), (p_3, q_3, q_3), \ldots$ be the vertices reached after the !-edges. For all $j \geq 1$ there is $t_j \in T(\mathcal{B}'(p_j))$ such that $(\star)$ holds for location x_j (so x_j is obtained from the edge labels of the corresponding path segment by omitting /) and for all $t' \in T(\mathcal{B}'(p_{j+1}))$. In particular, we can use t_{j+1} in place of t'. So, for each j, let t''_j be a tree with

$$t_j \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q_j)) \xrightarrow[\mathcal{R}]{+} t''_j \text{ with } (t''_j)^{\downarrow x_j} = t_{j+1} \text{ and } t''_j[x_j/q_{j+1}] \xrightarrow[\mathcal{A}]{*} q_j.$$

From the condition $(t''_j)^{\downarrow x_j} = t_{j+1}$ we get an infinite path of the form

$$t_1 \xrightarrow[\mathcal{R}]{*} t''_1 \xrightarrow[\mathcal{R}]{*} t''_1[x_1/t''_2] \xrightarrow[\mathcal{R}]{*} t''_1[x_1/t''_2[x_2/t''_3]] \xrightarrow[\mathcal{R}]{*} \cdots,$$

and with the condition $t''_j[x_j/q_{j+1}] \xrightarrow[\mathcal{A}]{*} q_j$ we can conclude that there are infinitely many trees from $T(\mathcal{A}(q_1))$ on this path. Furthermore, t_1 is in T'_i because $p_1 \in F'_i$. Thus, $T'_i \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A}(q_1))$. □

Since we want to use $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ in a decision procedure we need the following lemma.

LEMMA 4.21 *The graph $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ can be constructed effectively in time $\mathcal{O}(|\mathcal{R}|^4 \cdot |\mathcal{A}|^4)$.*

Proof. To construct $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ we have to check the conditions for the edges in (1), (2), and (3) of Definition 4.18. These are mainly instances of the reachability problem. It is clear that instances of the reachability problem with three sets involved are more difficult than the ones with only two sets involved. So we have to estimate the complexity of

(i) $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q))$ in (1) and

(ii) $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T_i$ and $T'_i \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(r)) \xrightarrow[\mathcal{R}]{*} T_j$ in (3).

According to Lemma 4.11 the set of all (p, q, r) with property (i) can be computed in time $\mathcal{O}(|\mathcal{R}|^3 \cdot |\mathcal{A}| \cdot (|\mathcal{A}| + |\mathcal{R}|))$ because $|\mathcal{B}'| \leq |\mathcal{R}|$ by definition of $|\mathcal{R}|$.

For (ii) let $\mathcal{B}$ denote the automaton $\bigcup_{i=1}^{m} \mathcal{A}_i$. We can compute the set of all tuples (q_1, q_2, q_3) of states from $\mathcal{B}'$, $\mathcal{A}$, and $\mathcal{B}$ with $T(\mathcal{B}'(q_1)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q_2)) \xrightarrow[\mathcal{R}]{*} T(\mathcal{B}(q_3))$ in time $\mathcal{O}(|\mathcal{R}|^4 \cdot |\mathcal{A}|)$, again by Lemma 4.11. The first part of (ii) corresponds to q_3 belonging to one of the sets F_i. The second part of (ii) corresponds to q_1 belonging to one of the sets F_i' and q_3 belonging to one of the F_j.

The time needed to compute the pairs with $T(\mathcal{B}'(p)) \xrightarrow[\mathcal{R}]{*} T_i$ and $T_i \rhd_{\mathcal{R}} T_j'$ can be be bounded as for (ii) because these are special instances of the problems from (ii).

Then we can check for each pair (p, q, r), $(\hat{p}, \hat{q}, \hat{r})$ of vertices of $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ whether the conditions of (1), (2), or (3) are satisfied. After the previous computations these tests can be made in constant time and have to be carried out for $\mathcal{O}(|\mathcal{R}|^2 \cdot |\mathcal{A}|^4)$ pairs. Therefore, the overall time complexity can clearly be bounded by $\mathcal{O}(|\mathcal{R}|^4 \cdot |\mathcal{A}|^4)$. □

The Algorithm

Now we are ready to summarize the results in an algorithm to solve the recurrence problem. The algorithm is shown in Figure 4.3. In line 6 by REACH($\mathcal{R}$,$\mathcal{B}$) we mean the output of the reachability algorithm (Figure 4.2) applied to $\mathcal{B}$.

THEOREM 4.22 *The algorithm* RECUR($\mathcal{R}$,$\mathcal{A}$) *from Figure 4.3 computes in time* $\mathcal{O}(|\mathcal{R}|^4 \cdot |\mathcal{A}|^4)$ *an* ε*-NTA* $\mathcal{A}_{\mathcal{R},\omega}$ *of size* $\mathcal{O}(|\mathcal{R}| \cdot (|\mathcal{A}| + |\mathcal{R}|))$ *with* $T(\mathcal{A}_{\mathcal{R},\omega}) = \{t \in T_A \mid t \xrightarrow[\mathcal{R}]{\omega} T(\mathcal{A})\}$.

Proof. For the complexity the operations on $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ are the dominating factor. $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$ itself can be constructed in time $\mathcal{O}(|\mathcal{R}|^4 \cdot |\mathcal{A}|^4)$ and the set of all (p, q, q) such that there is an infinite path with infinitely many !-edges starting in (p, q, q) can be determined by an algorithm computing the strongly connected components of $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$. This can be done in linear time in the size of $G_{\mathrm{Rec}}(\mathcal{R}, \mathcal{A})$, which can clearly be bounded by $\mathcal{O}(|\mathcal{R}|^4 \cdot |\mathcal{A}|^4)$.

For the size of $\mathcal{A}_{\mathcal{R},\omega}$ note that $\mathcal{B}$ already contains a copy of each $\mathcal{A}_i$ and still has the property $t \xrightarrow[\mathcal{B}]{*} F_i$ iff $t \in T_i$. Therefore, the first line of REACH can be skipped and the size of the automaton $\mathcal{A}_{\mathcal{R},\omega}$ is the same as the size of $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$.

For the correctness it is sufficient to show that $\mathcal{B}$ after line 5 recognizes the set $\mathrm{Rec}_1(\mathcal{R}, \mathcal{A}) \cup \mathrm{Rec}_2(\mathcal{R}, \mathcal{A})$ (by Lemma 4.14). This is ensured by line 2 for $\mathrm{Rec}_1(\mathcal{R}, \mathcal{A})$ and by line 3 for $\mathrm{Rec}_2(\mathcal{R}, \mathcal{A})$. □

Algorithm: RECUR

INPUT: RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ with $R = \{T_1 \overset{\sigma_1}{\hookrightarrow} T'_1, \ldots, T_m \overset{\sigma_m}{\hookrightarrow} T'_m\}$,
NTAs $\mathcal{A}_i = (Q_i, A, \Delta_i, F_i)$ with $T(\mathcal{A}_i) = T_i$ for $i = 1, \ldots, m$
NTAs $\mathcal{A}'_i$ with $T(\mathcal{A}'_i) = T'_i$ for $i = 1, \ldots, m$
NTA $\mathcal{A} = (Q, A, \Delta, F)$

1. Construct $G_{\text{Rec}}(\mathcal{R}, \mathcal{A})$ (Definition 4.18).
2. Mark each pair (i, q) with $T'_i \xrightarrow[\mathcal{R}]{*} T(\mathcal{A}(q)) \xrightarrow[\mathcal{R}]{*} T_i$.
3. Mark each pair (i, q) with $q \in Q$, $i \in \{1, \ldots, m\}$ such that there is $p \in F'_i$ and an infinite path with infinitely many !-edges through $G_{\text{Rec}}(\mathcal{R}, \mathcal{A})$ starting in (p, q, q).
4. Let $\mathcal{B} = \mathcal{A} \cup \bigcup_{i=1}^{m} \mathcal{A}_i$.
5. Add ε-transitions (p, q) to $\mathcal{B}$ if $p \in F_i$ and (i, q) is marked.
6. Let $\mathcal{A}_{\mathcal{R},\omega} = \text{REACH}(\mathcal{R},\mathcal{B})$.

OUTPUT: ε-NTA $\mathcal{A}_{\mathcal{R},\omega}$

Figure 4.3: Algorithm to solve the recurrence problem

4.3 Undecidable Properties

In this section we prove that model-checking for GTR graphs with the temporal operators AF, EU, and AGF is undecidable. All the proofs use reductions from undecidable properties of deterministic Turing machines. We construct, given a Turing machine M, a GTRS $\mathcal{R}(M)$ that can simulate computations of M. Of course, since reachability is decidable for GTR graphs, it is not possible to exactly simulate a Turing machine with a GTRS. Therefore, in $G_{\mathcal{R}(M)}$ there will be paths that correspond to correct computations of M and there will also be paths that correspond to computations of M with some errors. But the way $\mathcal{R}(M)$ is constructed allows to detect these errors. Every time an error occurs in the computation, the path in $G_{\mathcal{R}(M)}$ contains a tree from a regular set T^M_{err}. If T^M_{stop} contains all trees coding a halting configuration of M, then $G_{\mathcal{R}(M)}$ is a model of the formula $AF(T^M_{\text{err}} \vee T^M_{\text{stop}})$ iff every path in $G_{\mathcal{R}(M)}$ that starts in the initial tree eventually reaches a tree modeling an error or a halting configuration. If the initial tree codes the initial configuration of M on the empty tape, then this means that M stops on the empty tape and therefore we have encoded the halting problem.

The construction of $\mathcal{R}(M)$ is given in the next subsection and in the

following subsections it is shown how to use the idea sketched above to show the undecidability results.

The general idea underlying the simulation of Turing machines is similar to the idea used in [EK95]. In [EK95] it is shown via a reduction from the halting problem for counter machines that the problem of universal reachability for basic parallel processes is undecidable.

4.3.1 Simulation of Turing Machines

We only give a brief description of the Turing machine model we use. For an introduction to Turing machines see e.g. [HU79].

A deterministic Turing machine (DTM) is a tuple $M = (Q, \Gamma, B, q_{\text{in}}, q_s, \delta)$, where Q is a finite set of states, Γ is the tape alphabet (disjoint from Q), $B \subset \Gamma$ is the input alphabet, q_{in} is the initial state, q_s is the halting state, and $\delta : (Q \setminus \{q_s\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. The transition function is completely specified and q_s is the unique halting state. Furthermore, we assume that the tape is infinite to the left and to the right. The tape alphabet Γ contains a blank symbol $\sqcup$ that is not an element of the input alphabet.

A configuration κ of M is a word $\kappa = a_1 \cdots a_k q b_l \cdots b_1$ with $q \in Q$, $k, l \geq 0$ and $a_i, b_j \in \Gamma$ for $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, l\}$. The reduced version $\text{red}(\kappa)$ is obtained from κ by removing the blank symbols from the left and right hand side of the configuration:

$$\text{red}(\kappa) = a_i \cdots a_k q b_l \cdots b_j$$

with i, j minimal such that $a_i \neq \sqcup \neq b_j$. Two configurations are equivalent iff their reduced versions are equal. In the following we identify equivalent configurations.

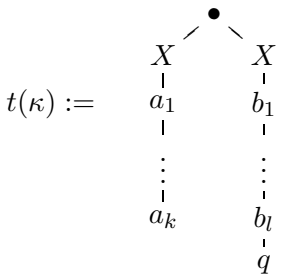
The successor configuration of κ is

- $a_1 \cdots a_{k-1} p a_k c b_{l-1} \cdots b_1$ if $\delta(q, b_l) = (p, c, L)$ and
- $a_1 \cdots a_k c p b_{l-1} \cdots b_1$ if $\delta(q, b_l) = (p, c, R)$.

In this definition we assume that $k, l > 0$. If k or l equals 0, then we use a configuration equivalent to κ by adding a blank symbol to the appropriate side of κ.

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

To simulate Turing machines by ground rewriting we define for each configuration κ a corresponding tree $t(\kappa)$ coding this configuration. If $\kappa = a_1 \cdots a_k q b_l \cdots b_1$, then let



With this coding of configurations it is not possible to exactly simulate the transitions of M by a GTRS because in a transition a symbol from Γ has to be moved either from the left branch to the right branch of the tree or the other way round. To realize this with ground tree rewriting the whole tree has to be rewritten. Since the length of Turing machine configurations is not bounded this would require an infinite set of rewriting rules. Therefore, the symbol that has to be moved from one branch to the other has to be guessed. To be able to detect wrong guesses, a protocol for the simulation of Turing machine transitions is introduced.

Figure 4.4 shows the correct simulation of the transition $\delta(q, a) = (p, b, L)$ when M is in the configuration cqa. The initial tree (on the left hand side of the figure) codes the configuration cqa and the final tree (on the right hand side of the figure) codes the configuration pcb.

The single steps in the simulation are the following. The symbols a and q are replaced by b and p. Since M moves the head to the left, the symbol c, which is at the end of the left branch, has to be guessed. The symbol ad_1 indicates that c, the symbol directly above ad_1, was added to the right branch of the tree. Now, the left branch has to confirm with re_1, where re stands for "remove" because the c has to be removed from the left branch. Then the two branches alternately increase their ad_i and re_i symbols until they reach ad_3 and re_2. Finally, the c can be removed from the left branch and then it is inserted at the appropriate place in the right branch.

The idea behind this is the following: If the guess is wrong, e.g., if d was guessed instead of c, then after the second step a tree containing the subtrees $c(re_1)$ and $d(ad_1)$ with $c \neq d$ is reached. This enables us to detect on a path through the graph generated by the rewriting system if an error occurred in the simulation of M. The process of going through all the symbols ad_2, re_2, and ad_3 has technical reasons. It ensures that other errors apart from the wrong guessing, e.g., repeated deletion of symbols without simulation of a transition, can be detected.

Formally, we define the GTRS $\mathcal{R}(M) = (A^M, \Sigma, R^M, t^M_{\text{in}})$ for a given

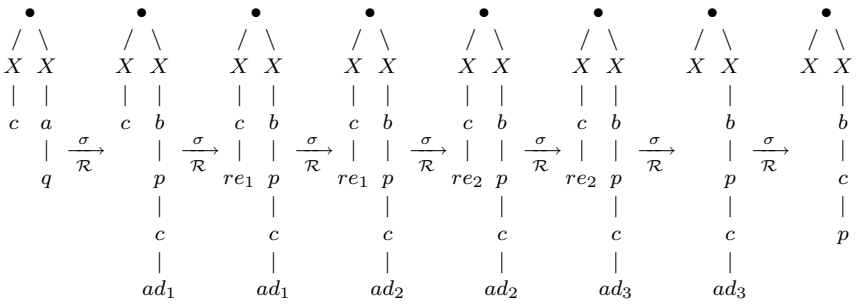
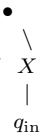
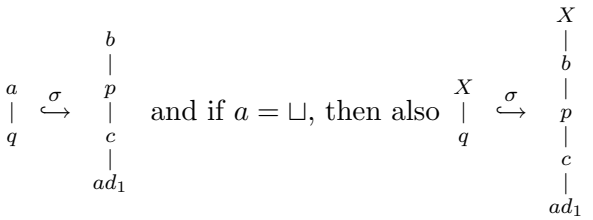


Figure 4.4: Example for the simulation of a TM-transition by a GTRS.

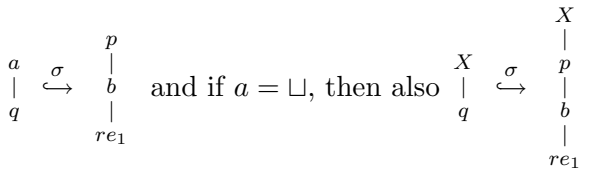
DTM $M = (Q, B, \Gamma, q_{\text{in}}, q_s, \delta)$ as follows:

- $A^M = A_0^M \cup A_1^M \cup A_2^M$ with $A_2^M = \{\bullet\}$, $A_1^M = Q \cup \Gamma \cup \{X\}$, and $A_0^M = A_1^M \cup \{ad_1, ad_2, ad_3, re_1, re_2, err\}$. The use of err will be clarified in the proof of Lemma 4.23.
- As we do not need the transition labels Σ, we let $\Sigma = \{\sigma\}$.
- The initial tree is $t_{\text{in}}^M = t(q_{\text{in}}) =$  .
- The set R^M contains the following rewriting rules.

  (1) For $\delta(q, a) = (p, b, L)$, $c \in \Gamma$:

  (2) For $\delta(q, a) = (p, b, R)$:

(3) For all $a \in \Gamma \cup \{X\}$, $b \in \Gamma$: $a \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} a \\ | \\ b \\ | \\ ad_1 \end{smallmatrix}$, $b \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} b \\ | \\ re_1 \end{smallmatrix}$, $X \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} X \\ | \\ \sqcup \\ | \\ re_1 \end{smallmatrix}$.

(4) $ad_1 \overset{\sigma}{\hookrightarrow} ad_2$, $ad_2 \overset{\sigma}{\hookrightarrow} ad_3$, $re_1 \overset{\sigma}{\hookrightarrow} re_2$.

(5) For all $a \in \Gamma \cup \{X\}$, $b \in \Gamma$, $q \in Q$: $\begin{smallmatrix} q \\ | \\ b \\ | \\ ad_3 \end{smallmatrix} \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} b \\ | \\ q \end{smallmatrix}$, $\begin{smallmatrix} a \\ | \\ b \\ | \\ ad_3 \end{smallmatrix} \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} a \\ | \\ b \end{smallmatrix}$.

(6) For all $a \in \Gamma \cup \{X\}$, $b \in \Gamma$, $q \in Q$: $\begin{smallmatrix} a \\ | \\ b \\ | \\ re_2 \end{smallmatrix} \overset{\sigma}{\hookrightarrow} a$, $\begin{smallmatrix} q \\ | \\ b \\ | \\ re_2 \end{smallmatrix} \overset{\sigma}{\hookrightarrow} q$.

(7) For all $a \in \Gamma$, $p, q \in Q$: $q \overset{\sigma}{\hookrightarrow} \begin{smallmatrix} a \\ | \\ p \\ | \\ err \end{smallmatrix}$, $\begin{smallmatrix} q \\ | \\ err \end{smallmatrix} \overset{\sigma}{\hookrightarrow} q$.

Rule (7) is used to ensure the property from Lemma 4.23 (i), which is necessary in Subsection 4.3.4.

The goal was to construct $\mathcal{R}(M)$ is such a way that we can identify all the paths in $G_{\mathcal{R}(M)}$ that do not correspond to a correct computation of M by a regular set T^M_{err}. This set is defined as follows. A tree t is in T^M_{err} iff

1. t contains the err symbol,
2. t contains more than one ad_i or more than one re_i symbol,
3. t contains a re_i symbol and no ad_i or ad_{i+1} symbol,
4. t contains an ad_2 symbol and no re_i symbol, or
5. t contains subtrees of the form $\begin{smallmatrix} a \\ | \\ re_i \end{smallmatrix}$ and $\begin{smallmatrix} b \\ | \\ ad_j \end{smallmatrix}$ with $a \neq b$.

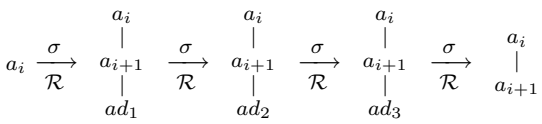
This definition only asks for the presence or absence of certain combinations of symbols. These properties can be tested by a tree automaton and therefore T^M_{err} is regular.

The following lemma gives a precise statement in what sense the DTM M can be simulated by $\mathcal{R}(M)$.

LEMMA 4.23 *(i) For each configuration κ of M: $t^M_{\text{in}} \xrightarrow[\mathcal{R}(M)]{*} t(\kappa)$.*

(ii) For all configurations κ, κ' of M there is a path from $t(\kappa)$ to $t(\kappa')$ in $G_{\mathcal{R}(M)}$ not visiting T^M_{err} iff $\kappa \vdash^*_M \kappa'$.

Proof. (i): Let $\kappa = a_1 \cdots a_k q b_l \cdots b_1$ be a configuration of M. Remember that we identify equivalent configurations and therefore can assume that $k, l > 0$ (by adding $\sqcup$ symbols). The left branch of $t(k)$ can be generated by using rewriting rules from (3), (4), and (5):



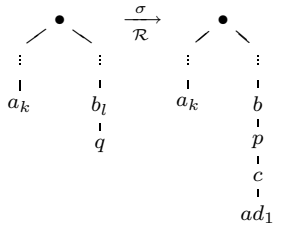
The right branch can be generated by the rewriting rules from (7).

(ii): The correct simulation of M can be carried out by $\mathcal{R}(M)$ without visiting T^M_{err} as sketched in Figure 4.4.

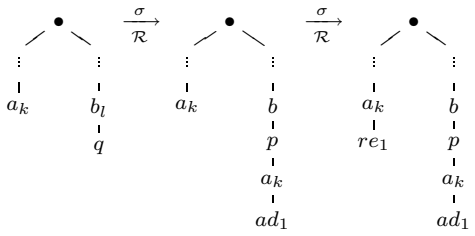
For the other direction let κ, κ' be configurations of M and let π be a path from $t(\kappa)$ to $t(\kappa')$ that does not visit T^M_{err}. We prove the claim by induction on the length of π. Obviously, if π has length 0, then $\kappa = \kappa'$ and therefore $\kappa \vdash^*_M \kappa'$.

So assume that π has length greater than 0 and let $\kappa = a_1 \cdots a_k q b_l \cdots b_1$. We analyze the sequence of rewriting rules that are used on π if $\delta(q, b_l) = (p, b, L)$ and show that this sequence has to follow the scheme from Figure 4.4. For $\delta(q, b_l) = (p, c, R)$ one can proceed analogously.

The first rewriting on π has to insert an ad_1 symbol into the tree. If a re_1 or err symbol would be inserted, then the resulting tree would be in T^M_{err}. Assume that the ad_1 symbol is appended to the left branch. In the next step this ad_1 could be changed into an ad_2 leading to a tree containing an ad_2 symbol and no re symbol, or a transition of the form (1) could be applied to the right branch leading to a tree containing two ad_1 symbols. In both cases the resulting tree would be in T^M_{err}. Therefore, the only possibility for the first step is to use a rewriting rule of the form (1):



The only possibility for the second rewriting on π is to add a re_1 symbol to the left branch, using rule (3). If the c that was added to the right branch in the first step does not equal a_k, then the tree after the second step contains the subtrees $\overset{a_k}{\underset{re_1}{|}}$ and $\overset{c}{\underset{ad_1}{|}}$ with $a_k \neq c$ and therefore would be in T^M_{err}. We have shown that the first to steps on π must be of the following form:



Now it is not difficult to see that the next steps on π must lead to $t(\kappa'')$, where κ'' is the successor configuration of κ. By the induction hypothesis we get $\kappa'' \vdash_M^* \kappa'$ and therefore $\kappa \vdash_M^* \kappa'$. □

After these technical preparations we discuss the remaining reachability problems. All the undecidability proofs use the construction of $\mathcal{R}(M)$ and the set T_{err}^M. Another set of trees used besides T_{err}^M in the following subsections is the set of trees encoding halting configurations of M:

$$T_{\text{stop}}^M = \{t(\kappa) \mid \kappa \text{ is a halting configuration of } M\}.$$

Since we assumed that the transition function of M is completely specified, a tree codes a halting configuration iff it contains the unique halting state q_s. Therefore, T_{stop}^M is regular.

4.3.2 Universal Reachability (AF)

The problem of universal reachability for GTR graphs is the following.

Given: a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and a regular set of trees $T \subseteq T_A$.

Question: Does every maximal path in $G_{\mathcal{R}}$ starting in t_{in} visit T?

THEOREM 4.24 *The problem of universal reachability for GTR graphs is undecidable.*

Proof. Let M be a DTM. Assume that M eventually reaches a halting configuration when started on the empty tape. By Lemma 4.23 (ii) this means that a path through $G_{\mathcal{R}(M)}$ starting in t_{in}^M that never reaches T_{err}^M must eventually reach T_{stop}^M. Thus, all paths through $G_{\mathcal{R}(M)}$ starting in t_{in}^M eventually reach $T_{\text{err}}^M \cup T_{\text{stop}}^M$.

Now assume that M never reaches a halting configuration when started on the empty tape. Then, again by Lemma 4.23 (ii), there is an infinite path through $G_{\mathcal{R}(M)}$ starting in t_{in}^M that never reaches T_{err}^M or T_{stop}^M. Thus, not all paths through $G_{\mathcal{R}(M)}$ starting in t_{in}^M eventually reach $T_{\text{err}}^M \cup T_{\text{stop}}^M$.

Therefore, we can conclude that M stops on the empty tape iff every path through $G_{\mathcal{R}(M)}$ starting in t_{in}^M eventually reaches the regular set $T_{\text{err}}^M \cup T_{\text{stop}}^M$. Since the halting problem for deterministic Turing machines is undecidable, the theorem is proven. □

4.3.3 Constrained Reachability (EU)

The problem of constrained reachability for GTR graphs is the following.

Given: a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and regular sets $T_1, T_2 \subseteq T_A$.

Question: Does there exist a path π in $G_{\mathcal{R}}$ starting in t_{in} such that there exists $i \in \mathbb{N}$ with $\pi(i) \in T_2$ and $\pi(j) \in T_1$ for all $j \leq i$?

This problem is called constrained reachability because we ask if T_2 can be reached under the constraint that the path remains in T_1 until it reaches T_2. The simple reachability question is the special case with $T_1 = T_A$.

THEOREM 4.25 *The problem of constrained reachability for GTR graphs is undecidable.*

Proof. Let M be a DTM. Assume that M eventually reaches a halting configuration when started on the empty tape. By Lemma 4.23 (ii) this means that there is path through $G_{\mathcal{R}(M)}$ starting in t_{in}^M that reaches T_{stop}^M while staying in the complement of T_{err}^M, which is regular because T_{err}^M is regular (Proposition 2.13).

If M does not reach a halting configuration when started on the empty tape, then every path through $G_{\mathcal{R}(M)}$ starting in t_{in}^M must visit T_{err}^M before it can reach T_{stop}^M (again by Lemma 4.23 (ii)).

Thus, M stops on the empty tape iff $\mathcal{R}(M)$ satisfies the constrained reachability problem for $T_{A^M} \setminus T_{\text{err}}^M$ and T_{stop}^M. □

4.3.4 Universal Recurrence (AGF)

The problem of universal recurrence for GTR graphs is the following.

Given: a GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and a regular set $T \subseteq T_A$.

Question: Does every infinite path through $G_{\mathcal{R}(M)}$ that starts in t_{in} infinitely often visit T?

To show the undecidability of universal and constrained reachability we used reductions from the halting problem for deterministic Turing machines since these two problems allowed us to exactly specify the path that simulates the correct behavior of the Turing machine.

The problem of universal recurrence does not allow this exact specification because we are interested in paths that only finitely often visit a certain regular set. If we take this set to be the set T^M_{err}, then a finite number of errors in the simulation of the DTM are allowed. This finite number of errors can be used to generate an arbitrary configuration of the DTM before starting the correct simulation. Therefore, we use a reduction from the following problem, which we call *diverging configuration*.

Given: a Turing machine M.

Question: Does there exist a configuration κ of M such that M does not stop when started in κ?

Note that in the above problem there is no restriction to reachable configurations of M.

LEMMA 4.26 *The problem diverging configuration for deterministic Turing machines is undecidable.*

The proof of this lemma can be found in Appendix A.3.

THEOREM 4.27 *The problem of universal recurrence for GTR graphs is undecidable.*

Proof. Let M be a DTM. If there is a path π in $G_{\mathcal{R}(M)}$ that only finitely often visits $T^M_{\text{err}} \cup T^M_{\text{stop}}$, then there is an $i \in \mathbb{N}$ such that $\pi(i) = t(\kappa)$ for some configuration κ of M and $\pi[i, \infty)$ is an infinite path that does not visit $T^M_{\text{err}} \cup T^M_{\text{stop}}$ at all. With Lemma 4.23 (ii) we can conclude that M does not stop when started in configuration κ.

If there exists a configuration κ of M such that M does not stop when started in κ, then, by Lemma 4.23 (ii), there is an infinite path through $G_{\mathcal{R}(M)}$ starting in $t(\kappa)$ that does not visit $T^M_{\text{err}} \cup T^M_{\text{stop}}$. By Lemma 4.23 (i) we know that there is a path from t_{in} to $t(\kappa)$. The concatenation of these two paths yields an infinite path through $G_{\mathcal{R}(M)}$ starting in t_{in} that visits $T^M_{\text{err}} \cup T^M_{\text{stop}}$ only finitely often.

Therefore, M has no diverging configuration iff $\mathcal{R}(M)$ satisfies the problem of universal recurrence with $T^M_{\text{err}} \cup T^M_{\text{stop}}$. □

4.4 A Temporal Logic for Model-Checking

We have considered all the reachability problems from Section 4.1 independently, except for universal one step reachability, which is covered by one step reachability, and universal constrained reachability, which contains universal reachability as a special case. The goal of this section is to built up a logic with operators for the decidable reachability problems. For our logic we use CTL-like syntax. The operators for one step reachability, reachability, and recurrence are denoted by EX, EF, and EGF. The E in these operators stands for "there is a path" such that a certain property on this path holds. Sometimes, it is reasonable to restrict to paths built up from edges labeled from a proper subset of Σ. Thus, we allow to parametrize this existential quantifier E with a set $\Lambda \subseteq \Sigma$. For this purpose we define, given an RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ and $\Lambda \subseteq \Sigma$, the RGTRS $\mathcal{R}|_\Lambda = (A, \Sigma, R|_\Lambda, t_{\text{in}})$, where $R|_\Lambda$ is obtained from R by removing all rules $T \overset{\sigma}{\hookrightarrow} T'$ with $\sigma \in \Sigma \setminus \Lambda$.

Furthermore, we also allow the past operators X^{-1} and F^{-1} corresponding to X and F.

For a fixed ranked alphabet A and an alphabet Σ for edge labels the formulas of our logic are defined by the following grammar (in CTL-like syntax).

$$\begin{aligned} \phi \quad ::= \quad & T(\mathcal{A}) \mid \neg\phi \mid \phi \vee \phi \mid \\ & E_\Lambda X\phi \mid E_\Lambda X^{-1}\phi \mid \\ & E_\Lambda F\phi \mid E_\Lambda F^{-1}\phi \mid \\ & E_\Lambda GF\phi \end{aligned}$$

for ε-NTA $\mathcal{A}$ and $\Lambda \subseteq \Sigma$.

The semantics $||\phi||_\mathcal{R}$ of such a formula ϕ with respect to an RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ is the set of all trees where ϕ is satisfied with respect to the relation $\underset{\mathcal{R}}{\longrightarrow}$.

- $||T(\mathcal{A})||_\mathcal{R} = T(\mathcal{A})$
- $||\neg\phi||_\mathcal{R} = T_A \setminus ||\phi||_\mathcal{R}$
- $||\phi_1 \vee \phi_2||_\mathcal{R} = ||\phi_1||_\mathcal{R} \cup ||\phi_2||_\mathcal{R}$
- $||E_\Lambda X\phi||_\mathcal{R} = \{t \in T_A \mid t \underset{\mathcal{R}|_\Lambda}{\longrightarrow} ||\phi||_\mathcal{R}\}$ $\quad \left[= \text{pre}_{\mathcal{R}_\Lambda}(||\phi||_\mathcal{R})\right]$
- $||E_\Lambda X^{-1}\phi||_\mathcal{R} = \{t \in T_A \mid t \underset{\mathcal{R}^{-1}|_\Lambda}{\longrightarrow} ||\phi||_\mathcal{R}\}$ $\quad \left[= \text{post}_{\mathcal{R}_\Lambda}(||\phi||_\mathcal{R})\right]$
- $||E_\Lambda F\phi||_\mathcal{R} = \{t \in T_A \mid t \overset{*}{\underset{\mathcal{R}|_\Lambda}{\longrightarrow}} ||\phi||_\mathcal{R}\}$ $\quad \left[= \text{pre}^*_{\mathcal{R}_\Lambda}(||\phi||_\mathcal{R})\right]$
- $||E_\Lambda F^{-1}\phi||_\mathcal{R} = \{t \in T_A \mid t \overset{*}{\underset{\mathcal{R}^{-1}|_\Lambda}{\longrightarrow}} ||\phi||_\mathcal{R}\}$ $\quad \left[= \text{post}^*_{\mathcal{R}_\Lambda}(||\phi||_\mathcal{R})\right]$
- $||E_\Lambda GF\phi||_\mathcal{R} = \{t \in T_A \mid t \overset{\omega}{\underset{\mathcal{R}|_\Lambda}{\longrightarrow}} ||\phi||_\mathcal{R}\}$

For an RGTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$, $t \in T_A$, and a formula ϕ from the above logic we define

$$\mathcal{R}, t \models \phi \text{ iff } t \in ||\phi||_{\mathcal{R}}$$

and if t is not given, then

$$\mathcal{R} \models \phi \text{ iff } t_{\text{in}} \in ||\phi||_{\mathcal{R}}.$$

A consequence of the results from Section 4.2 is the following theorem.

THEOREM 4.28 *For an RGTRS $\mathcal{R}$ and a formula ϕ from the above logic the set $||\phi||_{\mathcal{R}}$ is a regular set of trees and an automaton $\mathcal{A}_\phi$ accepting $||\phi||_{\mathcal{R}}$ can be constructed effectively. In particular it is decidable whether $\mathcal{R} \models \phi$.*

Proof. For the atomic formulas of the form $\phi = T(\mathcal{A})$ the set $||\phi||_{\mathcal{R}}$ is regular by definition. For the Boolean operators we use Propositions 2.12 and 2.13. So we get $\mathcal{A}_{\phi_1 \vee \phi_2} = \mathcal{A}_{\phi_1} \cup \mathcal{A}_{\phi_2}$ and $\mathcal{A}_{\neg\phi} = \overline{\mathcal{A}_\phi}$. Using the results from Section 4.2 we can define the automata for the temporal operators.

- $\mathcal{A}_{E_\Lambda X\phi} = (\mathcal{A}_\phi)_{\text{pre}_\mathcal{R}}$, $\mathcal{A}_{E_\Lambda X^{-1}\phi} = (\mathcal{A}_\phi)_{\text{post}_\mathcal{R}}$,
- $\mathcal{A}_{E_\Lambda F\phi} = (\mathcal{A}_\phi)_{\text{pre}^*_\mathcal{R}}$, $\mathcal{A}_{E_\Lambda F^{-1}\phi} = (\mathcal{A}_\phi)_{\text{post}^*_\mathcal{R}}$, and
- $\mathcal{A}_{E_\Lambda GF\phi} = (\mathcal{A}_\phi)_{\mathcal{R},\omega}$.

For the construction of $(\mathcal{A}_\phi)_{\mathcal{R},\omega}$ we first have to eliminate the ε-transitions from $\mathcal{A}_\phi$ because the algorithm RECUR needs an NTA without ε-transitions as input.

Once we obtained the automaton $\mathcal{A}_\phi$ we can check if t_{in} is in the language $T(\mathcal{A}_\phi)$ to decide whether $\mathcal{R} \models \phi$. □

Although the blow up in the size of the input automaton is polynomial in the constructions for the temporal operators, the iterated application of these constructions results in a blow up that is exponential in the number of nested temporal operators. The construction for the complement automaton is even worse with an exponential blow up, resulting in an automaton with size non-elementary in the number of nested negations. So we have the following complexity.

THEOREM 4.29 *The size of the automaton $\mathcal{A}_\phi$ from Theorem 4.28 is non-elementary in the number of nested negations and exponential in the number of nested temporal operators.*

We end this section with a brief discussion on first-order (FO) logic over GTR graphs. In contrast to temporal logic, which can be used to express global properties of a graph in terms of reachability conditions, FO logic corresponds to Boolean combinations of local properties by Gaifman's locality theorem [Gai82]. So, in general, these two formalisms are not comparable with respect to expressive power. A possibility to combine them is to enrich FO logic with reachability predicates. In [DT90] it is shown that the FO theory of ground tree rewriting systems with a reachability predicate $\xrightarrow[\mathcal{R}]{*}$ is decidable, where a formula $x \xrightarrow[\mathcal{R}]{*} y$ evaluates to true iff x and y are interpreted as trees t_1 and t_2 such that there exists a path from t_1 to t_2. To obtain this decidability result one can make use of tree automata for relations of trees (cf. [CDG+97]). For an FO-formula $\varphi(x_1, \ldots, x_n)$ one can construct an automaton recognizing the n-ary relation of all tuples $(t_1, \ldots, t_n)$ of trees such that $\varphi(t_1, \ldots, t_n)$ evaluates to true. This extended form of FO logic subsumes the temporal logic suggested in this section if the operator EGF is omitted. To also capture recurrence properties one would have to use a kind of recurrence predicate. Our results suggest that the automata theoretic proof for the decidability of FO logic with reachability still goes through if one adds a predicate of the form $x \in EGF(T(\mathcal{A}))$ with the obvious semantics.

4.5 Reachability Games

In this section we study two different reachability games on GTR graphs. Usually, when considering two player games on graphs the set of vertices of the graph is partitioned into two sets, the vertices of Player I and the vertices of Player II. The two players move a token along the edges of the graph, where the "owner" of the current vertex chooses the next edge. In this way, a sequence of vertices is built up and the winner of such a play is determined by a winning condition partitioning the set of possible plays into those that are winning for Player I and those that are winning for Player II. A reachability condition specifies a set of vertices, and the plays that are winning for Player I are exactly those that contain a vertex from this set. So, for Player I it is sufficient to force the play to eventually reach this set specified by the reachability condition.

Throughout this section we omit the alphabet Σ for the edge labels. The terminology for ground tree rewriting systems is transferred to systems without edge labels in the obvious way.

A rather general reachability game on GTR graphs can be defined by

equipping a usual GTRS $\mathcal{R} = (A, \Sigma, R, t_{\text{in}})$ with a partition T_I and T_{II} of T_A and a winning condition T, where all these three sets are regular. The players play on the graph $G_{\mathcal{R}}$ starting in the initial tree. Whenever the current tree t in the play is in T_I, Player I chooses a successor of t in $G_{\mathcal{R}}$ as the next tree, and if t is in T_{II}, then Player II chooses a successor of t in $G_{\mathcal{R}}$ as the next tree. Player I wins a play if it eventually reaches the set T. Player I is said to be the winner of the whole game if he can ensure to eventually reach T, no matter how Player II plays. The crucial question, given such a game, is to determine if Player I is the winner of the game.

We do not give a formal definition of these concepts because one can easily express the universal reachability problem from Section 4.3 with these games. If we choose $T_I = \emptyset$ and $T_{II} = T_A$, then Player I wins the game if all paths starting in t_{in} eventually reach T. Therefore, the problem of determining if Player I can win the game is undecidable.

In the following, we study a slightly different two player game played on GTR graphs, for which we give a more formal description. In these games the set of vertices is not partitioned into vertices of Player I and Player II, but Player I chooses a subset of the outgoing edges of the current vertex, and Player II chooses one of these edges. Of course, Player I is not allowed to choose arbitrary sets of edges. The admissible sets of edges Player I can choose are determined by the rules of the ground tree rewriting system.

A ground tree rewriting game (GTRG) $\mathcal{G}$ is a tuple $\mathcal{G} = (A, R, t_{\text{in}}, T)$, where A is a ranked alphabet, t_{in} is the initial tree, $T \subseteq T_A$ is a regular set, called the winning set for Player I, and R is a finite set of rules of the form $s \hookrightarrow S$ with $s \in T_A$ and S is a finite subset of T_A. The game graph $G_{\mathcal{G}}$ of $\mathcal{G}$ is the graph generated by the GTRS (A, R', t_{in}), where R' contains the rules $s \hookrightarrow s'$ such that there is a rule $s \hookrightarrow S$ in R with $s' \in S$. As mentioned before, we omit the edge labels. By $T(\mathcal{G})$ we denote the set of all trees from the game graph.

A play π of $\mathcal{G}$ is a path through $G_{\mathcal{G}}$ that starts in t_{in} and that is either infinite or ends in a vertex without outgoing edges. This path is built up from moves by the two players as follows. If the play is at a tree t, then Player I chooses a subtree $t^{\downarrow x}$ of t and a rule $t^{\downarrow x} \hookrightarrow \{s_1, \ldots, s_n\}$. Then Player II chooses $s_i \in \{s_1, \ldots, s_n\}$. The next tree in the play is $t[x/s_i]$.

Since we consider reachability games the winning condition is a set T of trees. Player I wins if the play eventually reaches a tree from T. Player II wins if either the play goes on forever without visiting T or if Player I gets stuck before visiting T, i.e., if the play reaches a tree t such that no subtree of t appears on the left hand side of a rule.

Usually, a strategy for a player is a function determining the next move

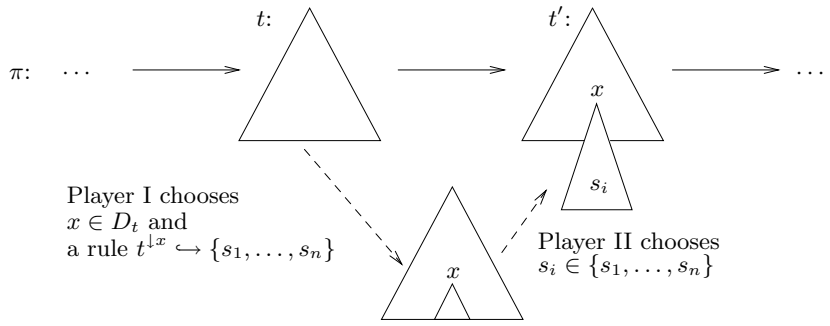


Figure 4.5: The moves carried out by Players I and II.

for the player when given the corresponding initial segment of the play. Because of the simple reachability winning condition we only need to consider the current vertex and not the complete history of the play. So, a strategy for Player I is a function

$$f_I : T(\mathcal{G}) \to \mathbb{N}^* \times R$$

such that for $f_I(t) = (x, s \hookrightarrow S)$ the location x is in D_t and $s = t^{\downarrow x}$. A play π is played according to f_I if for each two successive trees t and t' on π the following holds: if $f_I(t) = (x, s \hookrightarrow S)$, then $t' = t[x/s']$ for some $s' \in S$. A strategy f_I is a winning strategy for Player I if every play π that is played according to f_I is winning for Player I.

A strategy for Player II is a partial function

$$f_{II} : T(\mathcal{G}) \times \mathbb{N}^* \times R \to T_A$$

that is defined for tuples of the form $(t, x, t^{\downarrow x} \hookrightarrow S)$ with $t \in T(\mathcal{G})$, $x \in D_t$, and $t^{\downarrow x} \hookrightarrow S \in R$ such that $f_{II}(t, x, t^{\downarrow x} \hookrightarrow S) \in S$. Informally speaking, if Player I chooses a location x and a rule that can be applied to $t^{\downarrow x}$, then Player II responds with a tree from the right hand side of the rule that should be substituted for $t^{\downarrow x}$. Consequently, a play π is played according to f_{II} if for each two successive trees t and t' on π there is $x \in D_t$ and a rule $t^{\downarrow x} \hookrightarrow S$ such that $t' = t[x/s']$ for $s' = f_{II}(t, x, t^{\downarrow x} \hookrightarrow S)$. A strategy f_{II} is a winning strategy for Player II if every play π that is played according to f_{II} is winning for Player II.

Figure 4.5 illustrates the two types of moves carried out by the two players.

Note that a play does not directly encode the moves of the two players, but a possible sequence of moves, which is not necessarily unique, can be derived from the trees in the play.

Of course, there is at most one player that can have a winning strategy. By a standard attractor construction one can also show that for each GTRG one of the players indeed has a winning strategy. Given a GTRG $\mathcal{G}$ we call the player with the winning strategy the winner of $\mathcal{G}$. The problem of solving a GTRG is to determine the winner, i.e., we are concerned with the following decision problem:

Given: A GTRG $\mathcal{G} = (A, R, t_{\text{in}}, T)$.

Question: Does Player I have a winning strategy?

In the following, we show that this problem is undecidable using the same method as in Section 4.3. But before, we take a look at the solitaire versions of this type of game. By solitaire versions of a two player game we mean the degenerated cases where one of the players does not have any choice.

In contrast to the game mentioned at the beginning of this section, which contained the universal reachability problem as the solitaire game for Player II, the solitaire versions for ground tree rewriting games are solvable.

Solitaire game for Player I: Player II does not have any choice iff all the right hand sides of the rules are singleton sets. In this case, Player I chooses a normal GTR rule of the form $s \hookrightarrow s'$ and the location where it has to be applied. Determining the winner in such a game corresponds to solving the reachability problem from Subsection 4.2.2.

Solitaire game for Player II: If Player I does not have any choice in a GTRG $\mathcal{G}$, this means that for all the trees t in $T(\mathcal{G})$ there is at most one rule of $\mathcal{G}$ that can be applied to t. This situation is even more restrictive than the one from Subsection 3.2.3 (Theorem 3.20) allowing the transformation of GTRS into infix pushdown automata. Hence, the construction used in Theorem 3.20 can be applied to $\mathcal{R}$ even without the use of infix rules, yielding a pushdown automaton M and a set C of configurations corresponding to the winning set T such that Player II is the winner of $\mathcal{R}$ iff there is a path through G_M that does not visit C. This problem for pushdown automata is decidable (cf. [EHRS00, Cac02a]) and therefore GTRGs not admitting any choice for Player I are solvable.

These considerations show that the interaction of the two players is really needed in the following undecidability proof. The proof follows the same

lines as the proofs in Section 4.3 using simulations of Turing machines. Given a DTM $M = (Q, B, \Gamma, q_{\text{in}}, q_s, \delta)$ we construct a GTRG $\mathcal{G}(M)$ such that Player I has a winning strategy in $\mathcal{G}(M)$ iff M eventually stops when started on the empty tape.

Informal description of $\mathcal{G}(M)$

We use the same coding of configurations by trees as in Section 4.3, and we construct $\mathcal{G}(M)$ in such a way that there is exactly one path π_M through $G_{\mathcal{G}(M)}$ corresponding to the correct simulation of M. If the play goes along this path π_M, then Player I wins if on this path there is a tree coding a halting configuration. To force the two players to remain on π_M the game is designed in such a way that the player deciding to leave π_M establishes the possibility to win for the other player. This is the key property of $\mathcal{G}(M)$. In the following example we explain the mechanism used for forcing a desired behavior of the two players. In this example we use trees of the form

$$\begin{array}{ccc} & f & \\ / & & \backslash \\ g & & g \\ \vdots & & \vdots \\ g & & g \\ | & & | \\ c & & d \end{array}$$

We want to force the two players to alternately remove a symbol from the two branches of the tree. It is not sufficient to use the rules

$$\begin{array}{c} g \\ | \\ c \end{array} \hookrightarrow c \text{ and } \begin{array}{c} g \\ | \\ d \end{array} \hookrightarrow d$$

because Player I can use these rules in any order. So, Player I could remove all the symbols from the left branch and then proceed with the right branch. Instead, we use auxiliary symbols and start with the rules

$$(1) \ \begin{array}{c} g \\ | \\ c \end{array} \hookrightarrow \{c_1, c_2\} \text{ and } (2) \ \begin{array}{c} g \\ | \\ d \end{array} \hookrightarrow \{d_1, d_2\}.$$

To force Player I to first use rule (1) and then rule (2) we define that a tree is in the winning set T if it contains c_2 and d, or if it contains d_2 and c_1. If we assume that Player one starts by choosing rule (2), then Player II can choose d_2 as the new subtree. There is no rule to rewrite d_2 and therefore Player I has no choice and must continue with rule (1). Now, Player II can respond with c_2 as the new subtree. The play has reached a tree containing c_2 and d_2 and it did not visit T. Since both, c_2 and d_2, cannot be rewritten,

Player II wins because Player I cannot make a new move. Thus, Player I has to start with rule (1). If Player II chooses c_2 as the new subtree, then the next tree in the play is in T and Player II looses the game. Thus, Player II has to choose c_1 as next subtree. Now Player I can only use (2) as next rule and Player II has to choose d_1 as next subtree because otherwise the new tree in the play would contain d_2 and c_1 and therefore would be in T.

We continue with the rules

$$(3)\ c_1 \hookrightarrow \{c, c_3\} \text{ and } (4)\ d_1 \hookrightarrow \{d, d_3\}.$$

The definition of T is extended to trees containing c_3 and d_1 and to trees containing d_3 and c. With the same argument as above one can see that Player I has to continue with rule (3). Player II has to respond with c as new subtree. Player I must go on with rule (4) and Player II has to choose d as new subtree.

In this way one can enforce a desired order in the choices of Player I, and one can force Player II to choose the desired new subtree. In the formal definition of $\mathcal{G}(M)$ the same scheme is used to ensure the correct simulation of the transitions of M.

Formal definition of $\mathcal{G}(M)$

The ranked alphabet is defined in a similar way as in Section 4.3 but with a different set Aux of auxiliary symbols (of rank 0):

$$\text{Aux} = \{\overleftarrow{L}, \overrightarrow{L}, \overleftarrow{L}_1, \overrightarrow{L}_1, \overleftarrow{L}_2, \overrightarrow{L}_2, \overleftarrow{R}, \overrightarrow{R}, \overleftarrow{R}_1, \overrightarrow{R}_1, \overleftarrow{R}_2, \overrightarrow{R}_2\}.$$

The "L-symbols" are used for the simulation of a TM-transition with the head moving to the left and the "R-symbols" for the simulation of a TM-transition with the head moving to the right. The direction of the arrow on top of the symbols indicates whether the symbol is used on the left hand side of the trees or on the right hand side of the trees.

Alongside the definition the reader should follow the partial plays depicted in Figure 4.6 corresponding to the simulation of the two possible types of transitions of M (head moving to the left and head moving to the right). For the trees above the dashed lines, a possible choice for Player I is shown (in fact the only one that does not lead to an immediate loss for Player I). For each tree Player II can choose one of the successors. The trees below the dashed line belong to a set T_{check}, to be defined later, which is a subset of the winning set for Player I. Hence, Player II has to follow the path above the dashed line.

The rewriting rules of $\mathcal{G}(M)$ are:

(1) For all $b, c \in \Gamma$ and $d \in \Gamma \cup \{X\}$:

$$c \hookrightarrow \{\ \overleftarrow{\overset{c}{|}L}\ ,\ \overleftarrow{\overset{c}{|}L_1}\ \} \text{ and } c \hookrightarrow \{\ \overleftarrow{\overset{\overset{c}{|}}{b}R}\ ,\ \overleftarrow{\overset{c}{|}R_1}\ \}$$

and

$$X \hookrightarrow \{\ \overleftarrow{\overset{X}{|}\sqcup L}\ ,\ \overleftarrow{\overset{X}{|}\sqcup L_1}\ \} \text{ and } X \hookrightarrow \{\ \overleftarrow{\overset{X}{|}bR}\ ,\ \overleftarrow{\overset{X}{|}R_1}\ \}$$

(2) For all M-transitions of the form $\delta(q, a) = (p, b, L)$ and for all $c \in \Gamma$:

$$\overset{a}{\underset{q}{|}} \hookrightarrow \{\ \overrightarrow{\overset{\overset{\overset{b}{|}c}{|}p}{|}L}\ ,\ \overrightarrow{\overset{\overset{b}{|}c}{|}L_1}\ \} \text{ and if } a = \sqcup \text{ also } \overset{X}{\underset{q}{|}} \hookrightarrow \{\ \overrightarrow{\overset{\overset{\overset{X}{|}b}{|}c}{|}pL}\ ,\ \overrightarrow{\overset{\overset{X}{|}b}{|}cL_1}\ \}$$

For all M-transitions of the form $\delta(q, a) = (p, b, R)$ and for all $c \in \Gamma$:

$$\overset{a}{\underset{q}{|}} \hookrightarrow \{\ \overrightarrow{\overset{p}{|}R}\ ,\ \overrightarrow{\overset{b}{|}R_1}\ \} \text{ and if } a = \sqcup \text{ also } \overset{X}{\underset{q}{|}} \hookrightarrow \{\ \overrightarrow{\overset{X}{|}pR}\ ,\ \overrightarrow{\overset{X}{|}bR_1}\ \}$$

(3) For all $b, c \in \Gamma$ and $d \in \Gamma \cup \{X\}$:

$$\overleftarrow{\overset{\overset{d}{|}c}{|}L} \hookrightarrow \{d,\ \overleftarrow{\overset{d}{|}L_2}\ \} \text{ and } \overleftarrow{\overset{b}{|}R} \hookrightarrow \{b,\ \overleftarrow{\overset{b}{|}R_2}\ \}$$

(4) For all $p \in Q$:

$$\overrightarrow{\overset{p}{|}L} \hookrightarrow \{p,\ \overrightarrow{\overset{p}{|}L_2}\ \} \text{ and } \overrightarrow{\overset{p}{|}R} \hookrightarrow \{p,\ \overrightarrow{\overset{p}{|}R_2}\ \}$$

As in Section 4.3 let T^M_{stop} be the set of all trees encoding a halting configuration of the Turing machine. The winning set T for Player I in $\mathcal{G}(M)$ is $T = T^M_{\text{stop}} \cup T_{\text{check}}$, where T_{check} consists of all the trees satisfying one of the following conditions:

1. t contains for some $c \in \Gamma$ the subtrees $\overleftarrow{\overset{c}{|}L}$ and $\overrightarrow{\overset{c}{|}L_1}$.

2. t contains for some $b \in \Gamma$ the subtrees $\overset{b}{\underset{\overleftarrow{R}}{|}}$ and $\overset{b}{\underset{\overrightarrow{R}_1}{|}}$.

3. The set of symbols from Aux occurring in t equals one of the following sets: $\{\overleftarrow{L}_1\}$, $\{\overrightarrow{L}_2\}$, $\{\overleftarrow{R}_1\}$, $\{\overrightarrow{R}_2\}$, $\{\overleftarrow{L}_2, \overrightarrow{L}\}$, or $\{\overleftarrow{R}_2, \overrightarrow{R}\}$.

The initial tree is the one encoding the initial configuration of M on the empty tape.

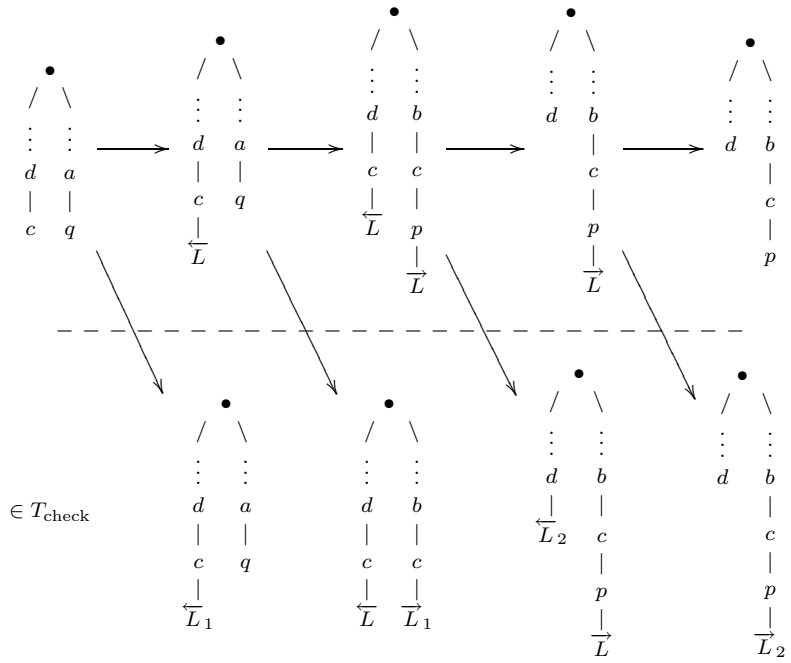
Figure 4.6 shows how Player I can force the simulation of the Turing machine. There are always two possibilities for Player II but one of them is leading to a tree in T. Since T also contains the trees encoding a halting configuration Player I has a winning strategy in $\mathcal{G}(M)$ if M stops when started on the empty tape.

An argument similar to the one from the informal description above can be used to show that Player II has the possibility to win a play as soon as Player I decides not to correctly simulate M because Player I will get stuck without reaching a tree from T.

Hence, Player I has a winning strategy in $\mathcal{G}(M)$ iff M stops when started on the empty tape, leading to the following theorem.

THEOREM 4.30 *The problem of solving ground tree rewriting games is undecidable.*

$\delta(q,a) = (p,b,L)$:



$\delta(q,a) = (p,b,R)$:

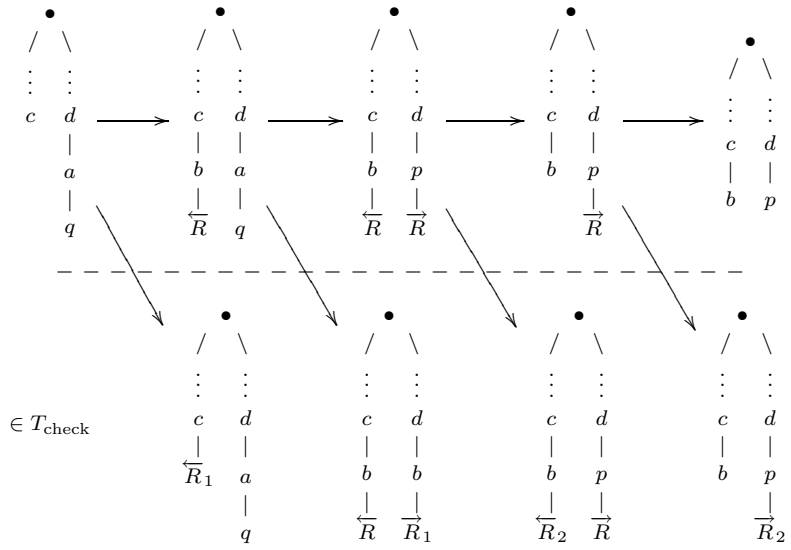


Figure 4.6: Simulation of a DTM with a GTRG

Chapter 5

Conclusion

The central topic of this thesis was the investigation of the class of ground tree rewriting graphs. Ground tree rewriting itself has been studied intensively, but the new aspect in our research was to analyze the transition graphs of ground tree rewriting systems instead of the trees that are generated by a ground tree rewriting system. This new perspective led to new questions, which have not been studied in the classical literature on ground tree rewriting systems. Our analysis was divided into two main parts.

In Chapter 3 we studied structural properties of GTR graphs. The main aspect in this chapter was the characterization of GTR graphs of bounded tree-width as the class of pushdown graphs. This result provided methods for studying the relation of GTR graphs to other classes of graphs. In particular, the combination of our result and the result of Muller and Schupp [MS85] on pushdown graphs allowed to easily separate GTR graphs from other classes of graphs, e.g., prefix recognizable and automatic graphs, as we showed in Section 3.3.

The second main part of our investigation of GTR graphs was concerned with their algorithmic properties. In Chapter 4 we analyzed the decidability and complexity of various reachability problems for RGTR graphs. One step reachability and reachability have already been studied in connection with ground term rewriting. For these problems the contribution of this thesis was to lift the known algorithms to regular ground tree rewriting. The central part and the main contribution of the algorithmic analysis was a polynomial time algorithm for solving the problem of recurrence for RGTR graphs. The decidability results were used to obtain a fragment of temporal logic with a decidable model-checking problem for RGTR graphs. Furthermore, we showed the undecidability for the universal and constrained reachability problems, as well as for reachability games on GTR graphs. This detailed

analysis of the reachability problems stemming from basic temporal operators allowed the statement that the presented fragment of temporal logic is maximal in the sense that adding other temporal operators leads to an undecidable model-checking problem for RGTR (and GTR) graphs.

5.1 Open Problems and Perspectives

We start with a collection of open problems that are directly connected with the problems investigated in this thesis. Some of these problems have already been stated in the text at the appropriate places. Finally, we conclude with some ideas and perspectives for further research in this area.

In Section 3.2 we gave a characterization of pushdown graphs as those GTR graphs that have bounded tree-width. The first question is if this characterization can help to decide for a given GTR graph whether it is a pushdown graph.

QUESTION 5.1 *Is the following problem decidable: Given a GTRS $\mathcal{R}$, is $G_{\mathcal{R}}$ of bounded tree-width?*

In [Col02] a result similar to our result from Section 3.2 is shown for RGTR graphs. It states that a graph is equational iff it is an RGTR graph of bounded tree-width. An open question is whether there is a similar relation between RGTR graphs and prefix recognizable graphs:

QUESTION 5.2 *Is a graph prefix recognizable if it is an RGTR graph of bounded clique-width?*

In Section 3.3 we compared GTR graphs to other classes of graphs. The exact relation between GTR graphs and automatic graphs was left open.

QUESTION 5.3 *Is the class of GTR graphs contained in the class of automatic graphs or are these two classes incomparable?*

In Section 4.4 we presented a logic and an algorithm for model-checking RGTR graphs with this logic. The complexity of this algorithm is non-elementary in the number of nested negations in the formula. It is open whether there is a better algorithm to solve this problem.

QUESTION 5.4 *What is the complexity of model-checking RGTR graphs with the logic presented in Section 4.4?*

These are some of the questions directly related to the topics covered in this thesis. A broader perspective for future research is to generalize the rewriting systems that generate the graphs. In ground tree rewriting (and regular ground tree rewriting) it is not possible to substitute inside the trees and there are no mechanisms to control the order of parallel applications of rewriting rules. For this reason, ground tree rewriting and regular ground tree rewriting are, in more concrete applications, not suitable for modeling real systems.

If one is interested in automated verification, one has to be careful when using extended rewriting systems. At least the reachability problem should remain decidable. An extended form of tree rewriting systems with a decidable reachability problem was presented in [CDGV94]. In the rules of these so called semi-monadic rewrite systems it is allowed to use variables directly below the root of the tree. For example, if X is a variable, then a rewriting rule can be of the form

$$f(X, a) \hookrightarrow g(b, X).$$

Such a rule can be applied to any subtree of the form $f(t, a)$, where t is an arbitrary tree. Applying the rule means to replace the subtree $f(t, a)$ by $g(b, t)$. These rules permit a restricted form of substitution inside the tree. A possibility to obtain even more power for these rewriting systems is to add a mechanism to get more control over the applications of these rewriting rules. As mentioned above, the tree that is substituted for a variable X can be an arbitrary tree. We can restrict this freedom by adding constraints to the variables used in the rewriting rules. For constrained variables one can only substitute trees satisfying these constraints. In this way, we get more control over the applications of the rewriting rules.

Some investigations in this direction revealed that regular constraints are too strong. That is, if we are allowed to specify regular sets of trees for the variables such that the use of the variable is restricted to trees from this regular set, then the reachability problem becomes undecidable. But weaker constraints of the kind "the use of variable X is restricted to those trees that contain a fixed subtree s" might preserve the decidability of the reachability problem.

Acknowledgment

This thesis was written during my time as a member of the research group of my supervisor Professor Wolfgang Thomas at Aachen University. I thank all the people who, in various ways, contributed to this thesis. In partic-

ular, I wish to express my gratitude to Wolfgang Thomas for his support throughout the last years and for suggesting the topic. I am grateful to Antje Nowak, Dietmar Berwanger, Philipp Rohde, Stefan Wöhrle, and Thierry Cachat for their help in the tedious work of proofreading. Their diligent reading and numerous comments uncovered a lot of mistakes, inaccuracies, and unreadable passages.

Appendix

A.1 Computing the Reachable States in Tree Automata

Let $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA. The computation of the reachable states of $\mathcal{A}$ is based on the following definition. We say that a transition $(q_1, \ldots, q_i, a, q)$ is reachable iff all the states $q_1, \ldots, q_i$ are reachable. Then a state q is reachable iff there is a transition $(q_1, \ldots, q_i, a, q)$ that is reachable. We distinguish between ε-transitions and normal transitions and therefore define $\Delta_\varepsilon = \Delta \cap (Q \times Q)$ and $\Delta_{\neg\varepsilon} = \Delta \setminus \Delta_\varepsilon$.

To compute the set of reachable states in time linear in the size of $\mathcal{A}$ we use the following data structures:

- For a transition $\alpha = (q_1, \ldots, q_i, a, q) \in \Delta_{\neg\varepsilon}$ let
  - $\text{source}(\alpha) = \{q_1, \ldots, q_i\}$
  - $\text{depend}(\alpha) = |\text{source}(\alpha)|$,
  - $\text{influence}(\alpha) = \{q\}$.
- For a state $q \in Q$ let

$$\text{influence}(q) = \{\alpha \in \Delta_{\neg\varepsilon} \mid q \in \text{source}(\alpha)\} \cup \{p \in Q \mid (q, p) \in \Delta_\varepsilon\}.$$

The algorithm explores the reachable states and transitions in a "bottom-up manner". Note that $\text{depend}(\alpha) = 0$ for transitions of the from (a, q), which are used by the automaton at the leafs of the input trees. Since these transitions do not depend on any states they are marked as reachable in the beginning. The algorithm maintains a set X of states and transitions that are already recognized as reachable and a queue containing those states and transitions for which the elements they influence still have to be analyzed. The operations on the queue are

- head(queue) returning the first element of the queue and deleting it,

Algorithm: REACHABLE_STATES

INPUT: ε-NTA $\mathcal{A} = (Q, A, \Delta, F)$

1. **for each** $\alpha \in \Delta_{\neg\varepsilon}$ **do**
2. **if** depend(α) $= 0$ **then** $X := X \cup \{\alpha\}$; append(queue,α) **endif**
3. **endfor**
4. **while** queue $\neq \emptyset$ **do**
5. $u =$ head(queue)
6. **for each** $v \in$ influence(u) **do**
7. **if** $v \notin X$ **then**
8. **if** $v \in Q$ **then** $X = X \cup \{v\}$; append(queue,v) **endif**
9. **if** $v \in \Delta$ **then**
10. depend(v) $=$ depend(v) $- 1$
11. **if** depend(v) $= 0$ **then** $X = X \cup \{v\}$; append(queue,v) **endif**
12. **endif**
13. **endif**
14. **endfor**
15. **endwhile**

OUTPUT: X

Figure A.1: An algorithm to compute the set of reachable states of an ε-NTA

- append(queue,v) appending v to the queue.

The algorithm is shown in Figure A.1. For a transition α, every time a state q that influences α is found to be reachable, the number depend(α) is decreased. If depend(α) reaches the value 0, then all states in source(α) are reachable and therefore α is reachable. It is not difficult to show the correctness of this algorithm. The running time is given in the following theorem.

THEOREM A.5 *The algorithm* REACHABLE_STATES($\mathcal{A}$) *computes the set of reachable states of* $\mathcal{A}$ *in time* $\mathcal{O}(|\mathcal{A}|)$.

Proof. First note that each element from $Q \cup \Delta_{\neg\varepsilon}$ will be in the queue at most once. Hence, the total running time is dominated by the sum of the sizes of the influence sets. For each $\alpha \in \Delta_{\neg\varepsilon}$ we have $|\text{influence}(\alpha)| = 1$.

To estimate the sum of the sizes of all influence(q) sets note that each transition $\alpha \in \Delta_{\neg\varepsilon}$ occurs in at most $|\text{source}(\alpha)|$ different influence-sets. Furthermore, each ε-transition adds exactly one state to the influence set of the state it departs from. So we get

$$\begin{aligned}\sum_{v\in Q\cup\Delta_{\neg\varepsilon}} |\text{influence}(v)| &= \sum_{q\in Q} |\text{influence}(q)| + \sum_{\alpha\in\Delta_{\neg\varepsilon}} 1 \\ &\leq \sum_{\alpha\in\Delta_{\neg\varepsilon}} |\text{source}(\alpha)| + \sum_{\alpha\in\Delta_{\varepsilon}} 1 + \sum_{\alpha\in\Delta_{\neg\varepsilon}} 1 \\ &= \left(\sum_{\alpha\in\Delta_{\neg\varepsilon}} |\text{source}(\alpha)| + 1\right) + |\Delta_{\varepsilon}| \leq |\mathcal{A}|.\end{aligned}$$

□

A.2 Time Complexity of REACH

In Chapter 4 (Subsection 4.2.2) we defined the automaton $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ for an ε-NTA and an RGTRS $\mathcal{R}$ as the output of the algorithm REACH (Figure 4.2). Here we are going to analyze the complexity of this algorithm.

Let $\mathcal{A} = (Q, A, \Delta, F)$ be an ε-NTA and let $\mathcal{R} = (A, \Sigma, R, t_{\mathrm{in}})$ be an RGTRS with $R = \{T_1 \overset{\sigma_1}{\hookrightarrow} T_1', \ldots, T_m \overset{\sigma_m}{\hookrightarrow} T_m'\}$, where the sets T_i, T_i' are accepted by NTAs $\mathcal{A}_i = (Q_i, A, \Delta_i, F_i)$ and $\mathcal{A}_i' = (Q_i', A, \Delta_i', F_i')$, respectively. By $\mathcal{B}'$ we denote the NTA $\mathcal{B}' = \bigcup_{i=1}^m \mathcal{A}_i'$.

The main problem for the complexity of REACH is that in each iteration of the **while**-loop we have to find an index i and a state p such that $T_i' \xrightarrow[\mathcal{B}_j]{*} p$. This can be done by computing the set of reachable states in the automaton $\mathcal{B}' \times \mathcal{B}_j$, if $\mathcal{B}_j$ denotes the automaton computed after the jth iteration of the **while**-loop in REACH. The size of the automaton $\mathcal{B}_j$ is bounded by the size of $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$, i.e., $|\mathcal{B}_j| \in \mathcal{O}(|\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}|)$. Therefore, such a computation takes time $\mathcal{O}(|\mathcal{B}'| \times |\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}|)$ which is $\mathcal{O}(|\mathcal{R}|^2(|\mathcal{A}| + |\mathcal{R}|))$. Roughly speaking, the **while**-loop in REACH may be executed $\mathcal{O}(|\mathcal{R}|(|\mathcal{A}| + |\mathcal{R}|))$ times. In total, the order of the time complexity of this naive implementation is $\mathcal{O}(|\mathcal{A}|^2|\mathcal{R}|^3 + |\mathcal{A}||\mathcal{R}|^4 + |\mathcal{R}|^5)$, a polynomial of degree 5.

The implementation we propose uses that transitions are added to $\mathcal{B}_j$, but no transitions are removed. Therefore, the set of reachable states in $\mathcal{B}' \times \mathcal{B}_j$ only increases. The idea of the implementation is as follows:

(1) Compute the reachable states of $\mathcal{B}' \times \mathcal{B}_j$ (where initially $j = 0$).

(2) Whenever a state (q', p) with $q' \in F_i'$ is recognized as reachable (that is $T_i' \xrightarrow[\mathcal{B}_j]{*} p$) add the transitions $F_i \times \{p\}$ to $\mathcal{B}_j$ and all the corresponding transitions to $\mathcal{B}' \times \mathcal{B}_j$. Continue with (1).

The main point is that after step (2) we do not completely restart the computation of reachable states but keep the states that are already marked as reachable. For this purpose we modify the algorithm from Figure A.1. We use the following notations:

- $\mathcal{B} = \mathcal{A} \cup \bigcup_{i=1}^m \mathcal{A}_i$.
- $\mathcal{B}' = \bigcup_{i=1}^m \mathcal{A}_i'$ with $\mathcal{B}' = (Q', A, \Delta', F')$.
- $\mathcal{C} = \mathcal{B}' \times \mathcal{B}$ with components $\mathcal{C} = (Q^{\mathcal{C}}, A, \Delta^{\mathcal{C}}, F^{\mathcal{C}})$.

The set of reachable states has to be computed for $\mathcal{C}$, therefore the data structures that refer to the automaton $\mathcal{A}$ in Appendix A.1 (like depend

and influence) do now refer to $\mathcal{C}$. We use an array 'inserted' where an entry inserted$[i,p]$ for $i \in \{1,\dots,m\}$ and state p of $\mathcal{B}$ is set to true iff the transitions $F_i \times \{p\}$ were inserted in $\mathcal{B}$. Initially all these entries are false.

The algorithm is shown in Figure A.2. The main part corresponds to the algorithm REACHABLE_STATES of Appendix A.1. The difference is in lines 10–12. If a state $(q',p) \in \mathcal{C}$ with $q' \in F_i'$ is marked as reachable, then the procedure insert(i,p) is called. This procedure adds the transitions $F_i \times \{p\}$ to $\mathcal{B}$ and updates the influence-sets of $\mathcal{C}$. Furthermore, it checks for $q' \in Q'$ and $q \in F_i$ whether (q',q) is already marked as reachable. In this case (q',p) is added to the queue and is marked as reachable because there is a new ε-transition from (q',q) to (q',p).

With these considerations it is not difficult to see that after termination the automaton $\mathcal{B}$ corresponds to the automaton $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$. We are interested in the complexity of this algorithm.

THEOREM A.6 *The algorithm from Figure A.2 computes the ε-NTA $\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}$ in time $\mathcal{O}(|\mathcal{R}|^2(|\mathcal{A}|+|\mathcal{R}|))$.*

Proof. As for the algorithm REACHABLE_STATES all the statements in the **while**-loop are executed at most $\mathcal{O}(|\mathcal{B}'| \cdot |\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}|)$ times.

In the **while**-loop the procedure insert is called, which itself contains a loop. But the statements in lines 3–6 of insert are executed in the worst case for each $q' \in Q'$, each $q \in \dot{\bigcup}_{i=1}^{m} F_i$, and each $p \in Q \dot{\cup} \dot{\bigcup}_{i=1}^{m} Q_i$, i.e., $O(|\mathcal{R}| \cdot |\mathcal{R}| \cdot (|\mathcal{A}|+|\mathcal{R}|))$ times. Since this estimation for the procedure insert is independent of the number of executions of the statements in the **while**-loop and since $|\mathcal{A}_{\mathrm{pre}^*_{\mathcal{R}}}| \in \mathcal{O}(|\mathcal{R}| \cdot (|\mathcal{A}|+|\mathcal{R}|))$, we get a total complexity of $\mathcal{O}(|\mathcal{R}|^2(|\mathcal{A}|+|\mathcal{R}|))$. □

INPUT: As for REACH in Subsection 4.2.2

1. **for each** $\alpha \in \Delta^{\mathcal{C}}_{\neg\varepsilon}$ **do**
2. **if** depend(α) = 0 **then** $X := X \cup \{\alpha\}$; append(queue,α) **endif**
3. **endfor**
4. **while** queue $\neq \emptyset$ **do**
5. u = head(queue)
6. **for each** $v \in$ influence(u) **do**
7. **if** $v \notin X$ **then**
8. **if** $v \in Q^{\mathcal{C}}$ **then**
9. $X = X \cup \{v\}$; append(queue,v)
10. **if** $v = (q', p)$ with $q' \in F'_i$ and not inserted$[i, p]$ **then**
11. insert(i, p)
12. inserted$[i, p]$:= true
13. **endif**
14. **endif**
15. **if** $v \in \Delta^{\mathcal{C}}$ **then**
16. depend(v) = depend(v) $-$ 1
17. **if** depend(v) = 0 **then** $X = X \cup \{v\}$; append(queue,v) **endif**
18. **endif**
19. **endif**
20. **endfor**
21. **endwhile**

OUTPUT: $\mathcal{B}$

---

procedure insert(i, p):

1. $\Delta_{\mathcal{B}} := \Delta_{\mathcal{B}} \cup (F_i \times \{p\})$
2. **for each** $q \in F_i, q' \in Q'$ **do**
3. influence(q', q) = influence$(q', q) \cup \{(q', p)\}$
4. **if** $(q', q) \in X$ and $(q', p) \notin X$ **then**
5. $X := X \cup \{(q', p)\}$
6. append(queue,(q', p))
7. **endif**
8. **endfor**

Figure A.2: A possible implementation of REACH (for the notations see explaining text)

A.3 Undecidability of "Diverging Configuration"

In Subsection 4.3.4 we used the problem "diverging configuration" for Turing machines to show the undecidability of the problem "universal recurrence" for ground tree rewriting systems. The problem "diverging configuration" for Turing machines is the following:

Given: a Turing machine M.

Question: Does there exist a configuration κ of M such that M does not stop when started in κ?

Here we show the undecidability of this problem. Note that we cannot apply Rice's theorem because a diverging configuration needs not to be reachable in M. The proof presented below is due to an idea of Jacques Duparc.

Proof of Lemma 4.26. We use a reduction from the halting problem for deterministic Turing machines, i.e., for a DTM M we construct a DTM M' such that M' has a diverging configuration iff the initial configuration of M (on the empty tape) is diverging.

So, no matter in which configuration M' is started, it shall check how M behaves when started on the empty tape. The basic idea is that M' starts simulating one step of M on the empty tape. Then M' resets its M-simulation and starts simulating two steps of M on the empty tape and so on. To keep track of how many steps of M have to be simulated, M' maintains a counter. For simplicity, we use a unary counter. The content of the tape of M' during such a simulation of M looks as follows:

$$\triangleright \underbrace{0 \cdots 0}_{n_1} \underbrace{1 \cdots 1}_{n_2} \$ a_1 \cdots a_k \begin{pmatrix} q \\ b_1 \end{pmatrix} b_2 \cdots b_l \triangleleft$$

The meaning of the part to the left of \$ is that M' aims at simulating n_1+n_2 steps of M and has already simulated n_1 of these n_1+n_2 steps. The part to the right of \$ encodes the configuration of M. In the representation of M-configurations we use a new set of symbols that does not interfere with the symbols used in M'. Even for the blank symbol of M we introduce a new symbol to be able to distinguish the blank symbols of M and the empty parts of the M'-tape. In the following $\sqcup$ denotes the blank symbol of M' and $\sqcap$ denotes the blank symbol of M used in the encoded configurations.

For the construction of M' we have to keep in mind that we need to consider all possible configurations of M'. As long as M' is in a configuration of the above form such that the encoded M-configuration is reachable from

the initial configuration of M, no problems arise. The two possible sources for a "misbehavior" of M' are:

(i) M' is started in a configuration that is not of the above form.

(ii) M' is started in a configuration of the above form, but the encoded M-configuration is not reachable from the initial configuration of M.

As we will see from the description below, the "check phase" of M' takes care of (i), and the "reset phase" prevents M' from diverging unintentionally in a situation of type (ii).

Now we describe the behavior of M'. In general, we say that M' stops as soon as something "unexpected" happens. For example, if we write "M' returns to the $\triangleright$-symbol", then it might happen that M' was started in a configuration without $\triangleright$-symbol. In this case, M' will eventually find a $\sqcup$-symbol instead of the $\triangleright$-symbol and stop. Similarly, in all the situations that are not explicitly covered by our description, M' stops. This is to ensure that M' does not diverge from malformed configurations.

Assume that the reading head of M' is at the left border on the $\triangleright$-symbol. Then M' proceeds in several phases.

Check phase: M' checks whether the content of the tape is of the form described above. If it is not, then M' stops. Otherwise, M' returns to the $\triangleright$-symbol and proceeds with the simulation phase.

Simulation phase: M' finds the first 1 on the tape and flips it into a 0. If M' meets \$ without finding a 1 before, then it returns to the $\triangleright$-symbol and continues with the reset phase. Otherwise, M' moves on to \$ and then to the field coding the state of M and the position of the reading head of M. At that point M' simulates one step of the M computation. If M reaches a halting configuration in this simulation step, then M' stops. Otherwise, M' returns to the $\triangleright$-symbol and reenters the check phase.

Reset phase: M' increases the number of 0-symbols on the tape by first moving to \$, replacing \$ by 0, moving one position to the right, and writing \$ onto this field. Then M' moves to the $\triangleleft$-symbol, starts deleting everything to the left until it meets \$, writes $\begin{pmatrix} q_0 \\ \sqcap \end{pmatrix} \triangleleft$ to the right of \$, moves back to \$, and then back to $\triangleright$ while flipping all 0 between \$ and $\triangleright$ into 1.

The intention in this phase is to pass from a configuration

$$\triangleright \underbrace{0 \cdots 0}_{n} \$ a_1 \cdots a_k \begin{pmatrix} q \\ b_1 \end{pmatrix} b_2 \cdots b_l \triangleleft,$$

meaning that n steps of M have been simulated, to the configuration

$$\triangleright \underbrace{1 \cdots 1}_{n+1} \$ \begin{pmatrix} q_0 \\ \sqcap \end{pmatrix} \triangleleft,$$

meaning that now $n+1$ steps of M on the empty tape will be simulated.

Assume that M does not stop when started on the empty tape. It is clear that M' diverges from the configuration

$$\triangleright 1 \$ \begin{pmatrix} q_0 \\ \sqcap \end{pmatrix} \triangleleft$$

because M' simulates a continuously increasing number of steps of M. All the intermediate configurations have the intended form. So, M' does not stop because nothing unexpected happens and M never reaches a halting configuration.

On the other hand, if M terminates when started on the empty tape, then M' has no diverging configuration. Either M' is started in a malformed configuration, then this will be detected and M' stops. Or M' is started in a configuration of the desired form, then the counter will eventually be reset and henceforth M' simulates the behavior of M on the empty tape. When M finally reaches a halting configuration in this simulation, then M' stops. □

Bibliography

[AJNO02] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d' Orso. Regular tree model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 555–568. Springer, 2002. 10

[Bar98] Klaus Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science, MFCS '98*, volume 1450 of *Lecture Notes in Computer Science*, pages 543–552. Springer, 1998. 4, 6, 65

[BCMS01] Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier, 2001. 4

[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 Society Press, 2000. 4, 30, 65, 71

[BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000. 9

[Blu01] Achim Blumensath. Prefix-recognizable graphs and monadic second order logic. Technical Report AIB-2001-06, RWTH Aachen, May 2001. 4, 9, 63

[Bra69] Walter S. Brainerd. Tree generating regular systems. *Information and Control*, 14:217–231, 1969. 6, 8, 16, 22

[BT02] Ahmed Bouajjani and Tayssir Touili. Extrapolating tree transformations. In *Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 539–554. Springer, 2002. 10

[Büc64] J. Richard Büchi. Regular canonical systems. *Archiv für Mathematische Grundlagenforschung*, 6:91–111, 1964. 3, 16

[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

[Cac02a] Thierry Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming, ICALP 2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 704–715. Springer, 2002. 3, 127

[Cac02b] Thierry Cachat. Uniform solution of parity games on prefix-recognizable graphs. In *Proceedings of the 4th International Workshop on Verification of Infinite-State Systems, INFINITY 2002*, 2002. To appear in volume 68(6) of ENTCS (Preeproceedings available on `http://www.fi.muni.cz/infinity02/`). 4

[Cau92a] Didier Caucal. Monadic theory of term rewritings. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science, LICS '92*, pages 266–273. IEEE Computer Society Press, 1992. 5

[Cau92b] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992. 3

[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. 3, 4, 7, 30, 63

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sopie Tison, and Marc

Tommasi. *Tree Automata Techniques and Applications*. (`http://www.grappa.univ-lille3.fr/tata`), 1997. 6, 8, 20, 23, 27, 88, 124

[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. 8, 88, 135

[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. 6, 8

[CK01] Didier Caucal and Teodor Knapik. An internal presentation of regular graphs by prefix-recognizable graphs. *Theory of Computing Systems*, 34(4):299–336, 2001. 4

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000. 37, 55, 58

[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. 9, 28, 62, 65, 134

[Cou89] Bruno Courcelle. The monadic second order logic of graphs II: Infinite graphs of bounded width. *Mathematical System Theory*, 21:187–222, 1989. 4, 30, 65

[Cou00] Bruno Courcelle. Clique-width of countable graphs: a compactness property. In *Proceedings of the 6th International Conference on Graph Theory*, volume 5 of *Electronic Notes in Discrete Mathematics*, 2000. 9, 55, 56, 58

[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. 6

[Die00] Reinhard Diestel. *Graph Theory*. Springer, second edition, 2000. 4, 6, 36, 39

[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 '90*, pages 242–248. IEEE Computer Society Press, 1990. 6, 124

[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. 3, 127

[EK95] Javier Esparza and Astrid Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *Proceedings of the 7th International Conference on Computer Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 1995. 5, 8, 114

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

[Eme90] E. Allen Emerson. Temporal and modal logic. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990. 1, 7, 85, 86

[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, 85

[Eng99] Joost Engelfriet. Derivation trees of ground term rewriting systems. *Information and Computation*, 152(1):1–15, 1999. 6, 8, 26

[EP00] Javier Esparza and Andreas Podelski. Efficient algorithms for pre^* and $post^*$ on interprocedural parallel flow graphs. In *Proceedings of the 27th ACM SIGPLAN-SIGACT on Principles of Programming Languages, POPL '2000*, pages 1–11. ACM Press, 2000. 5

[FS93] Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993. 4, 65

[Gai82] Haim Gaifman. On local and non-local properties. In *Proceedings of the Herbrand Symposium: Logic Colloquium '81*, pages 105–135. North-Holland, 1982. 124

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984. 6, 8, 20

[GW00] Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2000*, volume 1928 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000. 58

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979. 7, 31, 74, 75, 114

[KV00] Orna Kupferman and Moshe Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*. Springer, 2000. 3, 4

[Löd02a] Christof Löding. Ground tree rewriting graphs of bounded tree width. In *Proceedings of the 19th Annual Symposioum on Theoretical Aspects of Computer Science, STACS 2002*, volume 2285 of *Lecture Notes in Computer Science*, pages 559–570. Springer, March 2002. 9, 37

[Löd02b] Christof Löding. Model-checking infinite systems generated by ground tree rewriting. In *Proceedings of Foundations of Software Science and Computation Structures, FoSSaCS 2002*, vol-

ume 2303 of *Lecture Notes in Computer Science*, pages 280–294. Springer, April 2002. 8, 88

[LS00] Denis Lugiez and Philippe Schnoebelen. Decidable first-order transition logics for PA-processes. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming, ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2000. 5

[May98] Richard Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, Technical University Munich, 1998. 4

[May00] Richard Mayr. Process rewrite systems. *Information and Computation*, 156(1–2):264–286, 2000. 4, 5, 63

[May01] Richard Mayr. Decidability of model checking with the temporal logic EF. *Theoretical Computer Science*, 256:31–62, 2001. 4, 5

[Mor99] Christophe Morvan. On rational graphs. In *Proceedings of the Third International Conference on Foundations of Software Science and Computation Structures, FoSSaCS '99*, volume 1784 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 1999. 4

[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. 2, 3, 28, 29, 31, 32, 33, 37, 54, 82, 133

[MS01] Christophe Morvan and Colin Stirling. Rational graphs trace context-sensitive languages. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science, MFCS 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 548–559. Springer, 2001. 7, 30

[Ris02] Chloé Rispal. The synchronized graphs trace the context-sensitive languages. In *Proceedings of the 4th International Workshop on Verification of Infinite-State Systems, INFINITY 2002*, 2002. To appear in volume 68(6) of ENTCS (Preeproceedings available on `http://www.fi.muni.cz/infinity02/`). 7, 30

[Tho88] Robin Thomas. The tree-width compactness theorem for hypergraphs. Manuscript, 1988. 39

[Tho89] Carsten Thomassen. Configurations in graphs of large minimum degree, connectivity or chromatic number. In *Proceedings of the 3rd International Conference on Combinatorial Mathematics*, volume 555 of *Ann. New Yrk Acad. Sci.*, pages 402–412, 1989. 39

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In C. Puech E. W. Mayr, editor, *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, STACS '95*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995. 3, 86

[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. 4, 7, 30

[Tis89] Sophie Tison. Fair termination is decidable for ground systems. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications, RTA 89*, volume 355 of *Lecture Notes in Computer Science*, pages 462–476. Springer, 1989. 6

[Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266, 1996. 1

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In *Proceedings of the 8th International Conference on Computer Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996. 3

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. 3

Index

T

W