Decision Problems over Infinite Graphs

Higher-order Pushdown Systems and Synchronized Products

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-Mathematiker
Stefan Wöhrle
aus Offenburg

Berichter: Univ.-Prof. Dr. Wolfgang Thomas
Dr. habil. Didier Caucal

Tag der mündlichen Prüfung: 30. Juni 2005

Abstract

The extension of formal verification methods to infinite models requires classes of graphs which are finitely representable and for which the model checking problem is decidable. We consider three approaches to define classes of finitely representable graphs: internal representations as configuration graphs of higher-order pushdown systems, transformational representations by application of operations which preserve the decidability of the model checking problem, and by composition from components using synchronized products.

In the first part of the thesis we show that the hierarchy of higher-order pushdown graphs coincides with the Caucal hierarchy of graphs. We thus obtain transformational representations of higher-order pushdown graphs and can conclude that they enjoy a decidable monadic second-order theory.

In the second part of the thesis investigate synchronized products of finitely representable infinite graphs and show that the decidability of an extension of first-order logic with reachability predicates is preserved under the formation of finitely synchronized products. This result is complemented by undecidability results for extensions of the admissible product operations as well as the expressive power of the logic under consideration.

Zusammenfassung

Die Erweiterung formaler Verifikationsmethoden auf unendliche Systeme erfordert die Untersuchung von endlich darstellbaren Graphklassen, für die das Model Checking Problem entscheidbar ist. Wir betrachten drei Zugänge, um solche Graphklassen zu definieren: interne Darstellungen als Konfigurationsgraphen von Kellerautomaten höherer Ordnung, Darstellungen durch Transformationen, die die Entscheidbarkeit der monadischen Theorie erhalten, und die Komposition von Komponenten mittels synchronisierter Produkte.

Im ersten Teil der Dissertation zeigen wir, dass die Hierarchie von Graphen, die durch Kellerautomaten höherer Ordnung erzeugt werden, mit der mittels Transformationen defininierten Caucal Hierarchie übereinstimmt. Wir können somit schließen, dass die monadische Theorie dieser Graphen entscheidbar ist.

Im zweiten Teil der Arbeit untersuchen wir synchronisierte Produkte endlich darstellbarer Graphen im Bezug auf das Model Checking Problem. Wir betrachten unter anderem die Logik der ersten Stufe erweitert um Erreichbarkeitsprädikate und zeigen, dass nur die Bildung endlich synchronisierter Produkte die Entscheidbarkeit dieser Logik erhält. Dieses Ergebnis wird durch Unentscheidbarkeitsresultate für Varianten der zulässigen Produktoperationen und der betrachteten Logiken ergänzt.

Contents

Chapter 1

Introduction

Several techniques have been invented to assist developers in assuring the correctness of computing systems. Among the automatic formal verification methods, see e.g [Pel01] for an overview, the model checking paradigm introduced in [CE81] was most successful.

$$\begin{tikzcd} \boxed{S} \arrow[r, leftrightarrow] & \boxed{M(S)} \arrow[r, leftrightarrow] & \boxed{\varphi} \end{tikzcd}$$

Figure 1.1: The model checking paradigm

The idea of model checking is depicted in Figure 1.1. Given a computing system S, a formal model $M(S)$ is defined. An expected behavior of the system model is formalized by a formula φ in a certain specification logic, and an automatic procedure, i.e. an algorithm, verifies whether the model $M(S)$ satisfies ($\models$) the formula φ.

Conditions posed on the behavior of a system are often Boolean combinations of rather simple requirements like

- Guaranty: “Eventually something happens.”
- Safety: “Something never happens.”
- Recurrence: “Something happens again and again.”
- Fairness: “If something happens infinitely often, then something else happens infinitely often.”

Temporal logics like LTL and CTL allow to express such properties. They play a prominent role as specification languages in model checking, see e.g. [CGP99], but also non-temporal logics like the modal μ-calculus or monadic second-order logic (MSO) are considered. The system models are usually Kripke structures or transition systems [Arn94]. Both can be seen as edge and/or vertex labeled graphs where vertices represent internal states of the system, their labels represent certain properties of these states, and an edge labeled by a symbol a between a vertex v and a vertex w implies that there is some action a which causes the system to change its state from v to w.

Model checking of finite state-based systems is nowadays well understood, see [CGP99, Pel01] and references therein. It is applied in practice even for large scale models. However, infinite state spaces naturally arise when modeling real world systems. Real world systems usually contain potentially infinite structures such as call stacks, counters, communication queues or channels. Thus the generation of finite models always requires abstraction techniques which may constrain the applicability of results obtained by a model checker.

In the last years considerable effort has been spent to transfer the results known about finite models at least partially to infinite models. The present thesis is a contribution to this track of research.

The extension of the model checking paradigm to infinite state systems raises two problems which are not present for finite ones: the system must be finitely representable to be handled by an algorithm which decides whether $M(S)$ satisfies a specification φ, and such an algorithm must exist, i.e. the model checking problem "$M(S) \models \varphi$?" must be decidable.

It is quite obvious that for keeping the model checking problem decidable there is a tradeoff between the richness of a class of models and the expressive power of the logic under consideration. Expressive logics like monadic second-order logic are already undecidable over very simple system models such as the infinite grid, whereas the quantifier free fragment of first-order logic is decidable even for recursive graphs, i.e. graphs which are described by an algorithm which decides whether an element of an infinite domain codes a vertex and whether there is an edge between two given vertices.

However, the quantifier free fragment of first-order logic cannot be regarded as a logic suitable for model checking. None of the natural conditions stated above can be expressed. This is due to the fact that they all speak about reachable (respectively non-reachable) states of the system. The reachability problem can be defined as follows:

Input:	A finite representation of a graph G, vertices u and v of G.
Problem:	Is there a path from u to v in G?

We will consider only those logics as adequately expressive for model checking in which the reachability problem can be formalized.

In this thesis we will address problems which arise for model checking of infinite systems. We analyze classes of graphs which are finitely representable and for which the model checking problem is decidable for a logic which is expressive enough to allow a formalization of the reachability problem, for example monadic second-order logic or transitive closure logic.

There are several approaches to define classes of finitely representable graphs in the literature. Infinite graphs may be described

- as solutions of systems of (graph-) equations,
- by finite systems like automata or rewriting systems,
- by application of finite sequences of operations to a finite initial system.

In [CC03] these representation schemes are called equational, internal and transformational respectively. Overviews over these approaches are given in [Cou90a], [Blu01] and [Tho03]. In this thesis we will consider internal and transformational representations of graphs.

Internal Representations

An internal representation of a graph is given by a family $(\mathcal{A}_i)_{1\leq i\leq n}$ of finite devices like automata, rewriting systems or Turing machines. If $n = 1$ only a single $\mathcal{A}$ is given and the graph represented by $\mathcal{A}$ is the configuration graph of $\mathcal{A}$. If $n > 1$ one of the systems $\mathcal{A}_i$ is used to decide whether an element of a certain infinite domain codes a vertex of the graph, and the other ones to decide the edge relations of the graph represented.

Examples of classes of configuration graphs which have been considered are those of pushdown systems [MS85], which are also called context free graphs, the graphs of linear bounded Turing machines [KP99], graphs generated by prefix rewriting systems [Cau92], prefix recognizable graphs [Cau96], or configuration graphs of tree rewriting systems [DT90, Löd02, Löd03, Col02].

Examples of graph classes where finite devices are used to decide the vertex set and the edge relations are the synchronized rational or automatic graphs [FS93, BG00], the rational graphs [Mor00] or the class of recursive graphs [EGNR98].

Already in [Büc64] J.R. Büchi showed that the reachability problem is decidable for configuration graphs of pushdown systems. D.E. Muller and P.E. Schupp [MS85] extended this result and showed that the model checking problem for monadic second-order logic is decidable for context free graphs. Prefix recognizable graphs [Cau96] include the context free graphs and still enjoy a decidable monadic second-order theory. For automatic graphs the first-order theory is still decidable [BG00], while it is undecidable for rational [Tho02] as well as recursive graphs.

In the first part of this thesis (Chapter 3) we consider a strong generalization of the class of pushdown graphs, graphs represented by higher-order pushdown systems [Gre70, Mas76, KNU02]. We will show that these graphs have a decidable model-checking problem with respect to monadic second-order logic. Higher-order pushdown systems extend the usual pushdown systems in the following sense. While an ordinary pushdown system uses a stack as an internal data structure, higher-order pushdown systems use higher-order stacks. These are stacks whose entries are not single letters as on level one but again stacks of lower levels. A higher-order pushdown stack is thus a stack of stacks of stacks .... In addition to the standard operations which only allow to modify the top symbol of a stack, on level n it is also possible to copy and remove the topmost stacks of level $k < n$ in one step.

Transformational Representations and the Caucal Hierarchy

A transformational representation of a graph consists of a finite sequence of computable operations which is applied to an initial graph which is either finite or finitely representable. Of particular interest in this setting are operations which are compatible with a logic $\mathcal{L}$. Such operations preserve the decidability of $\mathcal{L}$ when applied to graphs with a decidable $\mathcal{L}$-theory.

In [Cau02] a hierarchy of finitely represented graphs was introduced by alternating the application of inverse rational mappings and unfoldings. Starting from the initial class $\mathrm{Tree}(0)$ of all finite trees, the classes $\mathrm{Graph}(n)$ and $\mathrm{Tree}(n)$ for $n \geq 0$ are defined. $\mathrm{Graph}(n)$ contains all graphs which can be defined by an inverse rational mapping from a tree in $\mathrm{Tree}(n)$ and $\mathrm{Tree}(n+1)$ contains all unfoldings of graphs in $\mathrm{Graph}(n)$. The classes $(\mathrm{Graph}(n))_{n\geq 0}$ and $(\mathrm{Tree}(n))_{n\geq 0}$ are called the Caucal hierarchy of graphs and trees respectively.

Both operations, the unfolding of a graph (from a definable vertex) and inverse rational mappings are monadic second-order compatible [CW98, Cau92]. Thus the model checking problem for monadic second-order logic is decidable for graphs in the Caucal hierarchy.

The first level of the graph hierarchy is well known. $\mathrm{Graph}(1)$ contains exactly the prefix recognizable graphs [Bar97, Blu01] which can be in turn characterized as the graphs which are the ε-closure of configuration graphs of pushdown automata (shortly pushdown graphs) [Sti].

Goal of Part 1 of the Thesis

In Chapter 3 we extend the internal description of graphs from $\mathrm{Graph}(1)$ as pushdown graphs to higher levels of the hierarchy by showing that a graph is in $\mathrm{Graph}(n)$ if, and only if, it is the ε-closure of a higher-order pushdown system of level n. This allows us to conclude that the model checking problem for higher-order pushdown graphs with respect to monadic second-order logic is decidable.

We furthermore show that the same hierarchy $(\mathrm{Graph}(n))_{n\geq 0}$ of graphs is obtained if unfoldings and inverse rational mappings are replaced by the more general treegraph operation and monadic second-order transductions. Both transductions and the treegraph operation are variants of the classical logical constructs adapted for graph structures. Transductions [Cou94] are multi-dimensional interpretations [EF95] and the treegraph operation [Col04] is a variant of the iteration of structures [She75, Sem84, Wal02]. Both operations are monadic second-order compatible.

Compositional Representations

In Chapter 4 we pursue another methodology on how to finitely represent infinite graphs by specifying operations which combine finite or finitely representable graphs into new ones. We call this the compositional approach. The idea of describing systems by composition from components is not new and is also present in the equational approach where disjoint sums and gluing of graphs at certain nodes is possible [Cou90a].

Instead of considering sum like operations as mentioned above we consider product like operations such as asynchronous or synchronous products of graphs. These operations are usually used to model concurrency and interaction between components of a system. For finite systems product operations of different kinds appear for example in the calculus of communicating systems CCS [Mil85] or in the CSP model of communicating sequential processes [Hoa78]. Products of infinite state systems have for example been considered in [May98] and [Rab05]. We will follow the definition of [Arn94, Arn02] when we deal with synchronized products of graphs in Chapter 4.

The infinite grid serves as a standard example of a graph obtained by the asynchronous product of two well behaved structures, namely two semi-infinite lines which represent the positive integers with successor relation.

Since the monadic second-order theory of the infinite grid is undecidable while it is decidable for the positive integers with successor relation this immediately implies that the formation of an asynchronous product is not monadic second-order compatible.

All of the graph classes with a decidable monadic second-order theory mentioned above contain the semi-infinite line. Thus none of these classes is closed under (asynchronous) products.

We investigate product operations which extend the asynchronous products and which are compatible with a logic $\mathcal{L}$ in which reachability can be expressed. We thus have to consider both less expressive logics and less complex graph classes. In particular we are interested in logics and operations in which the evaluation of a formula in the compound can be reduced to the evaluation of formulas in the components.

In model theory such composition theorems have been investigated for a long time. The first outstanding result was obtained by S. Feferman and R.L. Vaught in [FV59] where composition theorems for products and sums of relational structures and first-order logic were given. The latter result on sums of structures was extended in a series of papers [She75, Gur85] to monadic second-order logic and monadic second-order logic with counting [Cou90b]. Composition theorems for more general sum-like operations were proved in [Rab] and [She96].

In [Rab05, Rab01] product like operations are considered and a composition theorem for modal logic is shown. It is additionally shown that for general product like operations and for any extension of modal logic in which the reachability modality EF can be expressed, the composition theorem fails. This extension is only possible for the restricted case of asynchronous products, see again [Rab05]. A nice survey on Feferman-Vaught like results and their algorithmic applications is given in [Mak04].

Goal of Part 2 of the Thesis

All the results mentioned above do not match the premises posed before. We want to solve the model checking problem for products of graphs rather than sums, and the logic must be expressive enough to allow the formalization of the reachability problem. In Chapter 4 we propose a restricted class of synchronized product operations for graphs for which we can show a composition theorem for FO(R), first-order logic extended by reachability predicates. We furthermore show that any extension of either the product operations or the expressive power of the logic leads to undecidability results.

Detailed Overview of the Thesis

The thesis consists of two parts which can be read separately. In Chapter 3 we consider the class of higher-order pushdown graphs while in Chapter 4 we investigate general synchronized products.

In Chapter 2 we give an overview of the basic notions used in both parts. We introduce edge labeled directed graphs as our system models and give the definitions of the specification logics we consider. In particular we define extensions FO(R) and FO(Reg) of first-order logic by reachability predicates and regular reachability predicates respectively.

In Section 2.2 we review more carefully the classes of finitely representable infinite graphs mentioned in the introduction. In Section 2.3 we define compatible operations on graphs like inverse rational mappings and transductions, iteration of structures and unfolding, as well as synchronized products. In Section 2.4 we formally introduce the Caucal hierarchy.

Higher-order Pushdown Graphs

In Chapter 3 we consider the class of higher-order pushdown graphs. A higher-order pushdown graph of level n is obtained as the ε-closure of the configuration graph of a higher-order pushdown system of level n.

In Section 3.1 we introduce higher-order pushdown systems. The presentation follows [KNU02]. From results of [Eng91] it follows that the hierarchy of higher-order pushdown graphs is strict level by level.

Since a higher-order pushdown system of level one is just a pushdown system we know that the level one higher-order pushdown graphs are exactly the graphs on the first level of the Caucal hierarchy. The main result of Chapter 3, presented in Theorem 3.23, extends this internal characterization of graphs in the Caucal hierarchy to every level: A graph is of level n in the Caucal hierarchy if, and only if, it is a higher-order pushdown graph of level n.

The proof of Theorem 3.23 is rather involved. The direction from right to left as presented in Section 3.3 makes use of closure properties of the Caucal hierarchy which are of their own interest.

In Section 3.2 we show that every level of the Caucal hierarchy is closed under monadic second-order transductions. We further show that by applying the treegraph operation, a variant of the iteration of structures, to a graph of level n one obtains a graph of level $n + 1$ of the hierarchy. This yields just another characterization of the graph hierarchy in terms of the two strongest MSO-compatible operations we know of (see Theorem 3.10): The Caucal hierarchy of graphs is equal to the hierarchy obtained from the class of finite graphs by applying the treegraph operation and MSO-transductions.

As a byproduct we obtain for every level n of the hierarchy a deterministic generator of level n, i.e. a graph from which every other graph on this level can be obtained by an MSO-transduction. It is the graph obtained by an $n-1$-fold application of the treegraph operation to the infinite binary tree Δ_2.

The results of Section 3.2 benefited a lot from previous work of A. Carayol and T. Colcombet [CC03].

In Section 3.4 we show that every graph of level n in the Caucal hierarchy is a higher-order pushdown graph of level n. For this we introduce a strengthened model of higher-order pushdown systems where the equality of the two topmost stacks on every level can be checked.

In Subsection 3.4.1 we show that for every $n \in \mathbb{N}$ these higher-order pushdown systems with equality pop generate the same graphs as standard higher-order pushdown systems. There the simulation of a higher-order pushdown system with equality pop operations by a standard one is technically involved, see pages 58–74. It makes use of the fact that all relevant changes to the stack content can be recorded to check equality even in the standard model.

In Subsection 3.4.2 and Subsection 3.4.3 we use higher-order pushdown systems with equality pop to construct systems which generate the unfolding of a graph and its image under an inverse rational mapping. Here we use the existence of unique inverse stack operations introduced with the equality-pop operations to handle backward edges in the regular expression defining an inverse rational mapping.

The equivalence results for higher-order pushdown graphs are summarized in Corollary 3.24. They allow us to transfer results known for Caucal graphs to higher-order pushdown graphs and back. In particular we obtain in the one direction that the monadic second-order theory of every higher-order pushdown graph is decidable, and in the other one that the Caucal hierarchy is indeed strict.

The results presented in Chapter 3 have been published with A. Carayol in [CW03].

Synchronized Products of Graphs

In Chapter 4 we investigate general synchronized products as introduced in Subsection 2.3.3, including asynchronous and direct products, as composition operations on graphs.

In Section 4.1 we investigate FO(TC)$_{(1)}$, transitive closure logic for graphs, over the infinite grid. We show that in general FO(TC)$_{(1)}$ is undecidable over the infinite grid while it becomes decidable if we disallow the nesting of transitive closure operators. The latter result is obtained by a reduction to Presburger arithmetic which is known to be decidable [Pre30, Pre91]. The reduction itself heavily relies on the regularity of the infinite grid and the locality of first-order logic [Han65].

In Section 4.2 we further restrict the expressive power of the specification language and consider the reachability logic FO(R). We introduce a semantically defined class of restricted synchronized product operations which we call finitely synchronized and where in each component only finitely many vertices may be involved in a synchronization.

A typical example of such a finitely synchronized product is a product of pushdown systems where a synchronization of the components is only possible in configurations where the stack is empty.

We show that finitely synchronized products preserve the decidability of the FO(R)-theory by proving a composition theorem which reduces the evaluation of an FO(R) formula in the product to the evaluation of finitely many FO(R) formulas in the components and a Boolean formula which combines these truth values (Theorem 4.5).

In Section 4.3 we show that this result is optimal in the following sense. First we show that if we extend the expressive power of the logic slightly, i.e. if we consider FO(Reg) and thus allow regular path descriptions to be formalized, then already asynchronous products, and hence also finitely synchronized products, are not compatible anymore. Secondly we show that if the finitely synchronized products are slightly generalized to semi-finitely synchronized products where in at most one component of the product infinitely many vertices may participate in a synchronization, then again the decidability of FO(R) is lost.

A preliminary version of the results presented in Chapter 4 was published with W. Thomas in [WT04].

Perspectives

Both research topics we present in the two parts of the thesis deserve further studies.

The results we present in Chapter 4 on the model checking problem for extensions of first-order logic and synchronized products is rather complete. Nevertheless other composition operations and other logics should be investigated with respect to their compatibility, in order to extend the methods which can be used for modeling computational systems.

For higher-order pushdown graphs which we present in Chapter 3 less is known. While we are familiar with the higher-order pushdown graphs of level one, almost nothing is known about their structure on higher levels.

In the conclusion of this thesis in Chapter 5 we present some open questions and perspectives in this field.

Chapter 2

Preliminaries and Background Theory

In this chapter we introduce the basic notions used in this thesis while we review classes of infinite graphs which have been previously considered. We start with the definition on graph structures, the specification logics, and the model checking problem.

2.1 Graph Structures and Specification Logics

We now present the structures we use as system models and the specification logics we consider. Often systems are modeled as Kripke structures or transition systems. Both models are nothing else than edge labeled and/or vertex labeled graphs. We use as system models edge labeled directed graphs which, due to our choice of specification languages presented below, are treated as relational structures.

2.1.1 Graph Structures

Let Σ be a finite set. A *Σ-labeled directed graph* G is a tuple $(V^G, (E_a^G)_{a\in\Sigma})$ where V^G is an at most countable set of vertices, and for $a \in \Sigma$ the set of a-labeled edges of G is denoted by E_a^G. We assume that there are no isolated vertices in G, i.e. for every $v \in V^G$ there exists an $w \in V^G$ such that $(v, w) \in E_a^G$ or $(w, v) \in E_a^G$ for some $a \in \Sigma$.

We fix a countable set Λ as a reservoir of names for edge labels and henceforth only consider Σ-labeled graphs with $\Sigma \subseteq \Lambda$. We do not distinguish between isomorphic graphs. If the graph G and the set of edge labels Σ is clear from the context we drop the superscript ^{G} and speak just of a labeled graph.

A graph is called *deterministic* if $(v, w) \in E_a$ and $(v, w') \in E_a$ implies

$w = w'$ for all $v, w, w' \in V$ and $a \in \Sigma$.

A deterministic graph which plays an important role in the second part of this thesis is the *infinite grid* depicted in Figure 2.1. It is the graph structure $\mathcal{G} = (V, S_1, S_2)$ with vertex set $V := \{(i,j) \mid i,j \geq 0\}$ and edge relations $S_1 := \{((i,j),(i+1,j)) \mid i,j \geq 0\}$ and $S_2 := \{((i,j),(i,j+1)) \mid i,j \geq 0\}$. We deviate from the usual naming convention for edges here to emphasize that they represent the successor relations in the components.

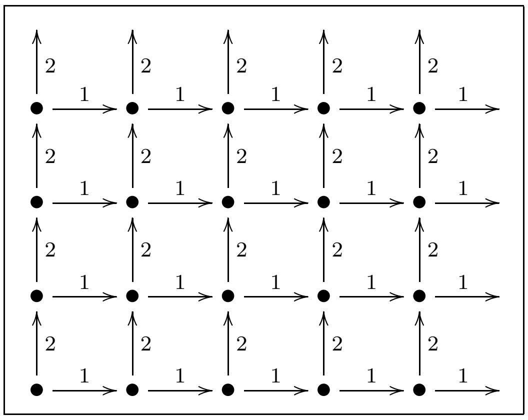


Figure 2.1: The infinite grid $\mathcal{G}$

A *path* from a vertex u to a vertex v labeled by $w = a_1 \dots a_{n-1}$ is a sequence $v_1 a_1 v_2 a_2 \dots a_{n-1} v_n \in V(\Sigma V)^*$ such that $v_1 = u$, $v_n = v$ and $(v_i, v_{i+1}) \in E_{a_i}$ for every $i \in \{1, \dots, n-1\}$. In this case we will also write $u \xrightarrow{w} v$. An (unranked) *tree* $T = (V^T, (E_a^T)_{a \in \Sigma})$ is a graph containing a vertex r called the *root* such that for any vertex $v \in V^T$ there exists a unique path from r to v.

Important trees are the *complete infinite m-branching trees* over the signature $\Sigma := \{0, \dots, m-1\}$ which we denote by Δ_m.

We view a Σ-labeled graph $G = (V^G, (E_a^G)_{a \in \Sigma})$ as a relational structure over the signature consisting of the binary relation symbols E_a for $a \in \Sigma$.

For $W \subseteq V$ we denote by $G \restriction W$ the substructure of G induced by W, i.e $G \restriction W := (W, (E'_a)_{a \in \Sigma})$ where $E'_a := E_a^G \cap (W \times W)$ for every $a \in \Sigma$.

2.1.2 Specification Logics

There are various logics which have been considered as specification languages for computational systems like (multi-) modal logic, temporal logics like LTL or CTL, the modal μ-calculus or first-order logic and monadic second-order logic. Some of them, like modal logic and first-order logic, lack the possibility to express that a certain state of the system can be reached from another state. We regard this reachability property as central for the verification of computational systems and therefore do only consider extensions of classical first-order logic in which it can be expressed.

First-order Logic and Monadic Second-order Logic. We recall the definition of *monadic second-order logic (MSO)* over graphs structures with the standard syntax and semantics (see e.g. [EF95]). Atomic formulas are of the form $E_a xy$ for $a \in \Sigma$, $x = y$ or Xy where $x, y, \ldots$ denote first-order variables and $X, Y, \ldots$ second-order variables. We allow the full spectrum of Boolean connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ as well as existential $\exists$ and universal $\forall$ quantification over both first-order and second-order variables. *First-order logic FO* is the restriction of MSO to formulas which do not contain second-order variables and quantifiers.

We write $\varphi(X_1, \ldots, X_n, y_1, \ldots, y_m)$ to denote that the free variables of the formula φ are among $X_1, \ldots, X_n$ (second-order) and $y_1, \ldots, y_m$ (first-order) respectively. A formula without free variables is a *sentence*.

Let G be a graph and $U_i \subseteq V^G$ for $1 \leq i \leq n$ and $v_j \in V^G$ for $1 \leq j \leq m$ be a valuation of the free variables of $\varphi(X_1, \ldots, X_n, y_1, \ldots, y_m)$. We write $G \models \varphi[U_1, \ldots, U_n, v_1, \ldots, v_m]$ to express that φ holds in G with the given valuation.

We often denote tuples of variables and tuples of (sets of) elements by $\bar{X}, \bar{y}, \ldots$ and $\bar{U}, \bar{v}, \ldots$ respectively.

Transitive Closure Logic. *Transitive closure logic FO(TC)* is defined by extending FO with formulas of the type

$$\psi(\bar{s},\bar{t},\bar{z}) := [\mathrm{TC}_{\bar{x},\bar{y}}\,\varphi(\bar{x},\bar{y},\bar{z})]\,\bar{s},\bar{t}$$

where $\varphi(\bar{x},\bar{y},\bar{z})$ is a FO(TC)-formula, $\bar{x},\bar{y}$ and $\bar{z}$ are disjoint tuples of free variables, $\bar{x}$ and $\bar{y}$ are of the same length $k > 0$ and $\bar{s},\bar{t}$ are tuples of variables of length k. The variables $\bar{x}$ and $\bar{y}$ in φ are bound by the TC-operator, hence in the notation $[\mathrm{TC}_{\bar{x},\bar{y}}\,\varphi(\bar{x},\bar{y},\bar{z})]\,\bar{x},\bar{y}$ the variables $\bar{x}$ and $\bar{y}$ inside the square brackets are bound while the variables at the end of the formula occur free.

Let G be a graph, let $\bar{c}$, $\bar{d}$, and $\bar{e}$ be the interpretations of the variables $\bar{z}$, $\bar{s}$, and $\bar{t}$ in φ. Let E be the relation on k-tuples defined by $E(\bar{c}) := \{(\bar{a},\bar{b}) \mid G \models \varphi[\bar{a},\bar{b},\bar{c}]\}$, and $E'(\bar{c})$ be its transitive closure, i.e. $(\bar{a},\bar{b}) \in E'(\bar{c})$ iff there exists a sequence $\bar{f}_0,\bar{f}_1,\ldots,\bar{f}_l$ such that $\bar{f}_0 = \bar{a}$, $(\bar{f}_i,\bar{f}_{i+1}) \in E(\bar{c})$ for $1 \leq i < l$, and $\bar{f}_l = \bar{b}$. The semantics of the FO(TC)-formula $\psi(\bar{s},\bar{t},\bar{c})$ is defined with respect to the relation E' by

$$G \models \psi[\bar{d},\bar{e},\bar{c}] \Leftrightarrow (\bar{d},\bar{e}) \in E'(\bar{c}).$$

We call the variables $\bar{z}$ *parameters* for the transitive closure operator. By FO(TC)$_{(k)}$ be denote the fragment of FO(TC) where the transitive closure operation is only allowed to define relations over tuples of length less than or equal to k, i.e. the length of the tuples $\bar{x},\bar{y}$ in the definition above is bounded by k. In FO(TC)$_{(1)}$ only binary relations, i.e. edges in a graph, can be defined using a transitive closure operator. FO(TC)$_{(1)}$ evaluated over graphs structures is a sublogic of MSO by the following equivalence

$$\begin{aligned} &G \models [\mathrm{TC}_{x,y}\,\varphi(x,y,\bar{z})]\,x,y \Leftrightarrow \\ &\qquad\qquad G \models \exists X\,\big(\theta(X,x,y,\bar{z}) \wedge Xy \wedge \forall Y\big(\theta(Y,x,y,\bar{z}) \to X \subseteq Y\big)\big) \end{aligned}$$

where

$$\theta(X,x,y,\bar{z}) := Xx \wedge \forall u \forall v \big(Xu \wedge \varphi(u,v,\bar{z})\big) \to Xv.$$

For finite models the arity hierarchy $(\text{FO(TC)}_{(k)})_{k\geq 0}$ is strict [Gro96].

By FO(TC)$^{l}_{(k)}$ we denote the fragment of FO(TC)$_{(k)}$ where the nesting depth of transitive closure operations is bounded by l.

Reachability Logics. In transitive closure logic we can express that from a vertex x a vertex y is reachable via a path with labels from some set $\Sigma' \subseteq \Sigma$ by

$$\mathrm{Reach}_{\Sigma'}(x,y) := \Big[\mathrm{TC}_{x,y}\,\big(x = y \vee \bigvee_{a\in\Sigma'} E_a xy\big)\Big]x,y.$$

We call the restriction of FO(TC) where the only transitive closure formulas allowed are of the form $\mathrm{Reach}_{\Sigma'}(x,y)$ for $\Sigma' \subseteq \Sigma$ *reachability logic* and denote it by FO(R).

The expressive power of the reachability predicates in FO(R) is limited, e.g. we cannot express that there is a path between vertex v and w in the graph whose labels form a word in a regular language.

We denote by FO(Reg) the extension of first-order logic by reachability predicates $\mathrm{Reach}_r(x,y)$ where r is a regular expression over Σ and where $G \models \mathrm{Reach}_r[v,w]$ if there is a path in G from v to w labeled by a word contained in the language described by r.

Again $\mathrm{Reach}_r(x,y)$ can be defined in FO(TC)$_{(1)}$ and hence we obtain a reachability logic between FO(R) and MSO in expressive power.

2.1.3 The Model Checking Problem

Among the various decision problems for specification logics and system models like satisfiability, logical equivalence, bisimulation invariance ... we discuss only decision problems which are referred to as *model checking problems*.

The solution of a model checking problem is an algorithm that solves the following question:

Input:	A graph G in a class $\mathcal{C}$, a formula φ in a logic $\mathcal{L}$.
Problem:	Does $G \models \varphi$ hold?

Note that we look at the *uniform* version of the model checking problem where both the graph under consideration and the formula to be checked are part of the input. For this we have to require that every infinite graph in the class $\mathcal{C}$ has a finite representation.

For complexity analyses of the model checking algorithms we define the size of the input as the size of the representation of the graph plus the length of the formula. This is usually referred to as *combined complexity*. As an underlying computation model we use random access machines with addition and multiplication as basic operations with a uniform cost measure, see e.g. [AHU74].

2.2 Graphs Represented by Finite Systems

In textbooks on automata theory like [HU79] finite automata are treated as a means to describe formal languages. They can however also be viewed as finite representations of possibly infinite graphs, namely their transition graphs. One of the first classes of such graphs which have been studied are the transition graphs of pushdown systems.

2.2.1 Pushdown Systems and Prefix Rewriting Graphs

A *pushdown system* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta)$ where Q is a finite set of states, Σ is a set of transition labels, Γ is a stack alphabet, q_0 is an initial state and $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times \Gamma^* \times Q$ is a finite transition relation.

A *configuration* of $\mathcal{A}$ is a pair $(q, [w])$ where $q \in Q$ is a state and $w = \gamma_m \dots \gamma_1 \in \Gamma^*$ is a stack content. The initial configuration is $(q_0, [\varepsilon])$. The pushdown system $\mathcal{A}$ can reach a configuration $(q', [w''\gamma_{m-1} \dots \gamma_1])$ from $(q, [\gamma_m \dots \gamma_1])$ via an a-labeled edge if $(q, a, \gamma_m, w'', q') \in \Delta$.

The *configuration graph* $\mathcal{C}(\mathcal{A})$ of $\mathcal{A}$ is the graph of all configurations reachable from the initial configuration with the incident edges as defined above.

A typical example of a configuration graph of a pushdown system is depicted in Figure 2.2. This graph is generated by a pushdown system with two states $Q := \{q_0, q_1\}$, input alphabet $\Sigma := \{a.b\}$, stack alphabet $\Gamma := \{a\}$ and transitions

$$\Delta := \{(q_0, a, \varepsilon, a, q_0), (q_0, a, a, aa, q_0), (q_0, b, a, \varepsilon, q_1)(q_1, b, a, \varepsilon, q_1)\}.$$

$$\begin{tikzcd}
(q_0,[\varepsilon]) \arrow[r, "a"] & (q_0,[a]) \arrow[r, "a"] \arrow[d, "b"] & (q_0,[aa]) \arrow[r, "a"] \arrow[d, "b"] & (q_0,[aaa]) \arrow[r, "a"] \arrow[d, "b"] & (q_0,[aaaa]) \arrow[d, "b"] & \cdots \\
 & (q_1,[\varepsilon]) & (q_1,[a]) \arrow[l, "b"'] & (q_1,[aa]) \arrow[l, "b"'] & (q_1,[aaa]) \arrow[l, "b"'] & \cdots
\end{tikzcd}$$

Figure 2.2: Configuration graph of a pushdown system

Configuration graphs of pushdown systems have been investigated by D.E. Muller and P.E. Schupp in [MS85]. In particular they showed that MSO-theory of configuration graphs of pushdown systems is decidable. A result of J.R. Büchi on canonical systems presented in [Büc64] already implies that the reachability problem is decidable for pushdown systems. Efficient algorithms for this problem and its application to model checking are considered in [EHRS00] and [BEM97].

In [Cau92] D. Caucal characterized configuration graphs of pushdown systems as graphs generated by prefix rewriting systems. A *prefix rewriting system* is a tuple (R, w_0) where R is a finite subset of $\Gamma^* \times \Sigma \times \Gamma^*$ and $w_0 \in \Gamma^*$ is an initial word. The graph generated by such a rewriting system is the graph consisting of all words reachable from w_0, with an a-labeled edge from w to w' if there is a rule $(u, a, v) \in R$ and a $w'' \in \Gamma^*$ such that $w = uw''$ and $w' = vw''$.

The graph shown in Figure 2.2 is generated by the prefix rewriting system (R, w_0) where $w_0 = q_0$ and $R \subseteq (Q \cup \Gamma)^* \times \Sigma \times (Q \cup \Gamma)^*$ is given by $R := \{(q_0, a, q_0a), (q_0a, b, q_1), (q_1a, b, q_1)\}$.

2.2.2 Prefix Recognizable Graphs and Pushdown Graphs

The prefix rewriting approach was extended by D. Caucal in [Cau96]. He defined the class of prefix recognizable graphs, which include the prefix rewriting graphs and still enjoy a decidable MSO-theory.

A *recognizable* Σ-labeled graph G with vertices in Γ^* is a graph such that for every $a \in \Sigma$ the set of a-labeled edges E_a is a finite union of sets $U \times V$ where U and V are regular (i.e. recognizable) subsets of Γ^*. Every set of prefix rewriting rules thus defines a finite recognizable graph and vice versa. The *right closure* of a recognizable graph G is the graph $G.\Gamma^*$ with an a-labeled edge between w and w' if there is a w'' and a $(u, v) \in E_a^G$ such that $w = uw''$ and $w' = vw''$. A *prefix recognizable graph* is a graph which can be obtained from the right closure of a recognizable graph by restricting the set of vertices to a regular language.

Prefix recognizable graphs constitute one of the most studied and best understood classes of infinite graphs. Analogously to prefix rewriting graphs the prefix recognizable graphs can be characterized by pushdown systems where ε-transitions are factored out.

To define the factorization we assume, for the remainder of this thesis, that pushdown systems are *normalized* , i.e. for every state in Q and every stack symbol in $\Gamma \cup \{\varepsilon\}$ only ε-transitions or only non-ε-transitions (labeled with symbols from Σ) are possible.

Let $\mathcal{A}$ be a pushdown system. The *ε-closure* of the configuration graph $C(\mathcal{A})$ of $\mathcal{A}$ is the graph G obtained from $C(\mathcal{A})$ by first adding an a-labeled edge between vertices v and w if there is an a-labeled path from v to w in $C(\mathcal{A})$ and second by removing all vertices with outgoing ε-transitions, i.e. vertex v is contained in the ε-closure of $C(\mathcal{A})$ iff there there is no w such that $(v, w) \in E_\varepsilon^{C(\mathcal{A})}$. We call G the graph *generated by* the pushdown system $\mathcal{A}$, or shortly a *pushdown graph* .

C. Stirling showed [Sti] that prefix recognizable graphs are exactly the pushdown graphs, i.e. every prefix recognizable graph can be obtained as the ε-closure of a configuration graph of a pushdown system.

While the degree of every prefix rewriting graph is bounded, this no longer holds for prefix recognizable graphs. This can be easily seen using their characterization as ε-closures of the configuration graphs of pushdown systems. If we replace in the graph depicted in Figure 2.2 all but the leftmost label a and all labels b on the vertical edges of the graph by ε and then compute the ε-closure, we obtain the graph depicted in Figure 2.3.

$$\begin{tikzcd}
(q_0,[\varepsilon]) \arrow[dr, "a"'] \arrow[drr, "a"] \arrow[drrr, "a"] \arrow[drrrr, "a"] & & & & & \\
& (q_1,[\varepsilon]) & (q_1,[a]) \arrow[l, "b"] & (q_1,[aa]) \arrow[l, "b"] & (q_1,[aaa]) \arrow[l, "b"] & \cdots\cdots
\end{tikzcd}$$

Figure 2.3: A prefix recognizable graph

2.2.3 Automatic Graphs and Rational Graphs

Prefix recognizable graphs can be specified by finite automata for the languages U and V which appear in the definition of the edge relations. Instead of supplying separate automata for U and V one can also imagine to provide a single automaton with two heads which move either synchronously or asynchronously over the words. There is an edge between vertices u and v if the pair (u, v) is accepted by the two-head automaton. If synchronous two-head automata are used to describe the edge relation the class of *synchronized rational graphs* [FS93] or *automatic graphs* [BG00] is obtained. If the two-head automata which define the edge relation are allowed to work asynchronously we obtain the class of *rational graphs* [Mor00].

By definition every automatic graph is also a rational graph. A. Blumensath showed in [Blu01] that every prefix recognizable graph is an automatic graph. We obtain a hierarchy of graph classes whose strictness follows from decidability results. The MSO-theory of prefix recognizable graphs is decidable while there is an automatic graph for which even reachability is undecidable [Blu99, Tho02]. This result follows from the fact that configuration graphs of Turing machines are automatic. For automatic graphs the FO-theory is still decidable [BG00] while again there is a single rational graph for which it is undecidable [Tho02].

2.2.4 Tree Rewriting Graphs

Another generalization of the prefix rewriting approach is to consider tree rewriting systems instead of rewriting on words. The class of graphs which can be represented by ground tree rewriting systems has been recently studied by C. Löding in [Löd02, Löd03].

A *ground tree rewriting system* is a tuple $\mathcal{R} = (A, \Sigma, R, t_0)$ where A is a ranked alphabet , Σ is a set of labels for the rules, R is a finite set of rewriting rules, and t_0 is a finite tree over A. We denote the set of all finite ranked trees over A by T_A. A *rewriting rule* r is of the form $t \xrightarrow{b} t'$ with $t, t' \in T_A$ and $b \in \Sigma$. A rule r is applicable to a tree s if there is a node such that the subtree s_1 of s rooted at this node equal to t, and the result of an application of r to s is a tree s' obtained from s by replacing s_1 with t'. $\mathcal{R}$ generates a Σ-labeled graph whose vertices are the trees which can be obtained from t_0 by applying rewriting rules from R, with a b-labeled edge between s and s' if s' results from s by an application of a rule of the form $t \xrightarrow{b} t' \in R$.

Note that the nodes of a graph generated by a tree rewriting system are ranked trees in contrast to the (unranked) trees defined in Subsection 2.1.1 as special graph structures.

It is very easy to generate the infinite grid $\mathcal{G}$ as introduced in Subsection 2.1.1 by a tree rewriting system using the fact that a vertex (i, j) can be represented by a tree with two branches, the left one of length i and the right one of length j. The rules of the tree rewriting system then generate the successor relations by extending the respective branch of the tree. The initial tree for the rewriting system is the tree

$$t_0 := \begin{tikzcd}[column sep=small] & \bullet \arrow[dl, dash] \arrow[dr, dash] & \\ a & & b \end{tikzcd}$$

and the rewriting rules are

$$a \xrightarrow{1} \begin{matrix} c \\ \mid \\ a \end{matrix} \qquad \text{and} \qquad b \xrightarrow{2} \begin{matrix} c \\ \mid \\ b \end{matrix}$$

Here the ranked alphabet is given by $A_0 := \{a, b\}$, $A_1 := \{c\}$ and $A_2 := \{\bullet\}$.

Similar to prefix rewriting systems one can extend the definition of ground tree rewriting systems to *regular ground tree rewriting systems* by allowing regular tree languages on both sides of the rewriting rules and by appropriately modifying the applicability of rules.

C. Löding showed in [Löd02] that the model checking problem for the class of regular ground tree rewriting graphs and a temporal logic encompassing reachability and recurrence operators is decidable. More precisely, the reachability problem for regular ground tree rewriting systems is defined as follows:

Input:	A GTRS $\mathcal{R} = (A, \Sigma, R, t_0)$, a regular set $T \subseteq T_A$.
Problem:	Does $\mathcal{R}$ reach some $t \in T$ from t_0?

Thus the decidability result does not exactly match the definition of the reachability predicates Reach in FO(R) where path labels can be restricted to a set $\Sigma' \subseteq \Sigma$ and arbitrary initial trees may be considered. However the underlying algorithm can be easily modified to handle restrictions on the set of allowed edge labels (i.e. restrictions on the set of rewriting rules) and changes of the initial tree.

Together with the result of M. Dauchet and S. Tison [DT90] that the first-order theory of GTRS is decidable we can conclude that that the model checking problem for these graphs and FO(R) is indeed decidable.

In [Col02] T. Colcombet studied graphs generated by *recognizable ground tree rewriting systems*. These systems are closely related to the ground tree rewriting systems of [Löd02] but work on yet another tree model, namely typed and ranked trees, which disallows a direct comparison of the two approaches. Graphs of recognizable ground tree rewriting systems enjoy a decidable FO(R)-theory, too.

The class of regular ground tree rewriting graphs obviously extends the class of prefix rewriting graphs. In [Löd03] it was shown that every ground tree rewriting graph of bounded out-degree is automatic. Also an example of an automatic graph which is not a ground tree rewriting graph was given. However, until now the exact relation between these two classes is still unknown.

2.3 Finite Representations via Operations

Another approach to represent an infinite graph structure G is to define it by (a sequence of) computable operations from finite or finitely representable graphs. A model checking algorithm working on such a graph representation will then make use of the algorithms which define G.

In [Cau96] prefix recognizable graphs are introduced in such a way, as structures obtained from the complete infinite binary tree by the application of operations which preserve the decidability of the MSO-theory: an inverse rational mapping followed by a rational restriction.

2.3.1 Inverse Rational Mappings and Transductions

Let $\bar{\Sigma}$ be a set of symbols disjoint from but in bijection with Σ. We extend every Σ-labeled graph G to a $(\Sigma \cup \bar{\Sigma})$-labeled graph $\bar{G}$ by adding reverse edges $E_{\bar{a}} := \{(u,v) \mid (v,u) \in E_a\}$. Let $\Gamma \subseteq \Lambda$ be a finite set of edge labels. A *rational mapping* is a mapping $h : \Gamma \to \mathcal{P}(\Sigma \cup \bar{\Sigma})^*$ which assigns to every symbol from Γ a regular (rational in the French community) subset of $(\Sigma \cup \bar{\Sigma})^*$. A rational mapping h is applied to a Σ-labeled graph G by the inverse to obtain a Γ-labeled graph $h^{-1}(G)$ with $(u,v) \in E_b$ if there is a path from u to v in $\bar{G}$ labeled by a word in $h(b)$. The set of vertices of $h^{-1}(G)$ is given implicitly by the edge relations. We also speak of $h^{-1}(G)$ as the graph obtained from G by the *inverse rational mapping* h^{-1}. If $h(a)$ is finite for every $a \in \Gamma$ then h is called a *finite mapping*.

The existence of a path in a graph labeled by a word in a regular language can be expressed by a monadic second-order formula. Indeed, an inverse rational mapping is a special case of an MSO-definable interpretation or, more general, an MSO-definable transduction.

An *MSO-interpretation* of Γ in Σ is a family $\mathcal{I} = (\varphi_a(x,y))_{a\in\Gamma}$ of MSO-formulas over Σ. Applying an MSO-interpretation $\mathcal{I}$ of Γ in Σ to a Σ-labeled graph G we obtain a Γ-labeled graph $\mathcal{I}(G)$ where for every $a \in \Gamma$ the edge relation $E_a^{\mathcal{I}(G)}$ is given by the pairs of vertices for which $\varphi_a(x,y)$ is satisfied in G, and $V^{\mathcal{I}(G)}$ is given implicitly as the set of all vertices which occur in a relation $E_a^{\mathcal{I}(G)}$ for some $a \in \Gamma$. Note that the addition of an MSO-formula $\delta(x)$ to $\mathcal{I}$ defining the vertex set explicitly does not increase the power of an interpretation under our premise that there are no isolated vertices in graphs.

In finite model theory one usually considers more general multi-dimensional interpretations, see e.g. [EF95]. For graph structures B. Courcelle [Cou94] established transductions as an equivalent of multi-dimensional interpretations, using another graph transformation.

Let $G = (V, (E_a)_{a\in\Sigma})$ be a Σ-labeled graph and K be a finite subset of Λ disjoint from Σ. A *K-copying operation* for Σ associates to G a $(\Sigma \cup K)$-labeled graph $G' = (V', (E'_a)_{a\in\Sigma\cup K})$ where $V' = V \cup (V \times K)$, $E'_a := E_a$ for $a \in \Sigma$, and $E'_b := \{(v, (v, b)) \mid v \in V\}$ for $b \in K$. An *MSO-transduction* $\mathcal{T} = (K, \mathcal{I})$ from Σ to Γ is a K-copying operation for Σ followed by an MSO-interpretation $\mathcal{I}$ of Γ in $\Sigma \cup K$.

We say that a graph G' is *MSO-definable* (or just *definable*) in a graph G if there exists an MSO-transduction $\mathcal{T}$ such that $G' = \mathcal{T}(G)$.

MSO-transductions have appeared in many forms in the literature. The decomposition of a transduction into a copying operation and an (one-dimensional) interpretation follows [Cou94]. The most important property of transductions is that they preserve the decidability of the MSO-theory of the structures under consideration.

K. Barthelmann [Bar97] characterized the prefix recognizable graphs as the graphs which can be obtained from the infinite binary tree by MSO-definable transductions. A simpler characterization in terms of interpretations was given by A. Blumensath in [Blu01].

2.3.2 Iteration of Structures and Unfolding

Another operation which attracted a lot of attention is the iteration of structures. It was first mentioned in [She75] in a slightly weaker version (without the clone predicate cl defined below).

Let $G = (V, (E_a)_{a\in\Sigma})$ be a graph. The *iteration* of G is the structure $G^* = (V^*, (E^*_a)_{a\in\Sigma}, \mathrm{son}, \mathrm{cl})$ whose universe consists of all finite sequences of vertices in V, $E^*_a := \{(wu, wv) \mid (u, v) \in E_a,\ w \in V^*\}$ for every $a \in \Sigma$, $\mathrm{son} := \{(w, wu) \mid u \in V,\ w \in V^*\}$, and $\mathrm{cl} := \{wuu \mid u \in V,\ w \in V^*\}$.

Muchnick's Theorem, first stated in [Sem84], claims that the decidability of the MSO-theory of the iteration structure can be reduced to the decidability of the MSO-theory of the non-iterated structure. The first complete proof of this fact was given by I. Walukiewicz in [Wal02].

Note that the iteration of a graph is, due to the clone predicate, not a graph structure. This is why in the following we will use a variant of the iteration of structures, the treegraph operation [Col04], which preserves the graph structure and thus is better suited for our setting.

Let $G = (V, (E_a)_{a\in\Sigma})$ be a graph and $\sharp \notin \Sigma$ be a new edge label. The *treegraph* of G with respect to $\sharp$ is the graph $\mathrm{Treegraph}(G, \sharp) := (V^*, (E^*_a)_{a\in\Sigma\cup\{\sharp\}})$ whose vertices are the finite sequences of vertices in V, $E^*_a := \{(wu, wv) \mid (u, v) \in E_a,\ w \in V^*\}$ for every $a \in \Sigma$ and $E^*_\sharp := \{(wu, wuu) \mid u \in V,\ w \in V^*, \}$.

Since $\mathrm{Treegraph}(G, \sharp) \models E_\sharp xy[u, v] \Leftrightarrow G^* \models \mathrm{son}(x, y) \wedge \mathrm{cl}(y)[u, v]$ the treegraph of G is always definable in the iteration G^*. Conversely, if G is connected we can define G^* in $\mathrm{Treegraph}(G, \sharp)$ by the the following equivalences:

$$G^* \models \mathrm{cl}(x)[u] \Leftrightarrow \mathrm{Treegraph}(G, \sharp) \models \exists z E_\sharp zx[u],$$
$$G^* \models \mathrm{son}(x, y)[u, v] \Leftrightarrow \mathrm{Treegraph}(G, \sharp) \models \exists z E_\sharp zy \wedge \mathrm{Reach}_{\Sigma\cup\bar{\Sigma}}(z, y)[u, v].$$

Here the MSO-formula $\mathrm{Reach}_{\Sigma\cup\bar{\Sigma}}(z, x)$ expresses that there is a path from z to y in which edges may be traversed in either direction (indicated by barred edge labels as for rational mappings).

A more natural operation on graphs than the treegraph operation is the unfolding. The *unfolding* $\mathrm{Unf}(G, r)$ of a graph $G = (V^G, (E^G_a)_{a\in\Sigma})$ from a node $r \in V^G$ is the tree $T = (V^T, (E^T_a)_{a\in\Sigma})$ where V^T is the set of all paths starting from r in G and for all $a \in \Sigma$, $(w, w') \in E^T_a$ iff $w' = w \cdot a \cdot v$ for some $v \in V^G$.

B. Courcelle and I. Walukiewicz showed in [CW98] that the unfolding $\mathrm{Unf}(G, r)$ from a definable vertex r is definable in the iteration G^*. Here a vertex r is called (MSO-) definable if there exists an (MSO-) formula $\psi(x)$ such that r is the unique vertex for which $G \models \psi[r]$. Since only the connected component of G which contains r matters, $\mathrm{Unf}(G, r)$ is also definable in $\mathrm{Treegraph}(G, \sharp)$.

It is possible to iterate the application of these operations on finite or finitely representable graphs without losing the decidability of the monadic second-order theory.

2.3.3 Synchronized Products

Synchronized products appear naturally if concurrency and interaction between computational processes which are modeled as graphs are considered [Arn94, Arn02]. The formation of a synchronized product of a family of finite or finitely representable graphs is a computable operation. Thus it may also be seen as a means to describe infinite graph structures.

For $1 \leq i \leq n$ let $G_i := (V_i, (E^i_a)_{a\in\Sigma_i})$ be a Σ_i-labeled graph. We assume that Σ_i is partitioned into a set Σ^l_i of *local* labels and a set Σ^s_i of *synchronizing* labels, and to avoid notational complication we require the sets of local labels to be pairwise disjoint. A *synchronization constraint* is a tuple $\bar{c} = (c_1, \ldots, c_n) \in \times_{1\leq i\leq n}\tilde{\Sigma}^s_i$, where $\tilde{\Sigma}^s_i := \Sigma^s_i \cup \{\varepsilon\}$. We also call sets $C \subseteq \times_{1\leq i\leq n}\tilde{\Sigma}^s_i$ synchronization constraints.

Applied to a family $(G_i)_{1\leq i\leq n}$ of graphs a synchronization constraint C defines a product graph with vertex set $\times_{1\leq i\leq n}V_i$. An asynchronous edge labeled by $a \in \Sigma^l_i$ is applied only in the i-th component of a vertex $\bar{v}$ of the product graph while the other components stay fixed. For synchronizing edges we distinguish explicitly between the components where a joint change is issued and the components where the vertex does not change. To describe the latter we use the label ε in the definition of the constraint C above. If $\bar{c} \in C$, a $\bar{c}$-labeled transition induces a simultaneous change in the components i where $c_i \neq \varepsilon$ while the vertices do not change in the other components.

Formally, the *synchronized product* of $(G_i)_{1\leq i\leq n}$ defined by C is the graph G with vertex set $V := \times_{1\leq i\leq n}V_i$, asynchronous edges with labels $a \in \bigcup_{1\leq i\leq n}\Sigma^l_i$ defined by $E^G_a\bar{v}\bar{w}$ if $E^i_a v_i w_i$ and $v_j = w_j$ for $j \neq i$, and synchronized edges with labels $\bar{c} \in C$ defined by $E^G_{\bar{c}}\bar{v}\bar{w}$ if $E^i_{c_i} v_i w_i$ for every $1 \leq i \leq n$. Here the relations E^i_ε are interpreted as $E^i_\varepsilon := \{(v, v) \mid v \in V_i\}$. We denote the set of local transitions labels $\bigcup_{1\leq i\leq n}\Sigma^l_i$ of G by Σ^l, and the set $C \cup \Sigma^l$ of all transition labels by Σ.

Note that we slightly deviate from the definition in [Arn94] since we require the sets of local labels and synchronizing labels to be disjoint, and implicitly assume an asynchronous behavior of edges labeled with local symbols.

A product is *asynchronous* if $C = \emptyset$. It is *completely synchronized* if $\Sigma^l_i = \emptyset$ and $\Sigma^s_1 = \Sigma^s_i$ for every $1 \leq i \leq n$ and $C := \{(a, \ldots, a) \mid a \in \Sigma^s_1\}$.

In contrast to the operations considered in the previous two subsections not even the formation of an asynchronous product does preserve the decidability of the monadic second-order theory of the graphs under consideration. Just consider the infinite grid $\mathcal{G} = (\omega \times \omega, S_1, S_2)$ whose monadic second-order theory is undecidable, but which can be viewed as the asynchronous product of two copies $\mathcal{N}_1 = (\omega, S_1)$ and $\mathcal{N}_2 = (\omega, S_2)$ of the positive integers with successor relations $S_1, S_2 := \{(j, j+1) \mid j \geq 0\}$.

In Chapter 4 we will address this problem. There we will introduce semantically restricted synchronized products and study them with respect to the model checking problem for logics in expressive power between first-order logic and monadic second-order logic.

2.4 The Caucal Hierarchy

As stated at the beginning of Section 2.3 D. Caucal introduced the prefix recognizable graphs as the graphs which can be obtained by applying an inverse rational mapping to the infinite binary tree Δ_2. Δ_2 in turn can be seen as the unfolding of the one element graph with self loops labeled 0 and 1, see Figure 2.4. Prefix recognizable graphs thus can be understood as the graphs obtained from this one element structure by applying two operations which preserve the decidability of the MSO-theory.

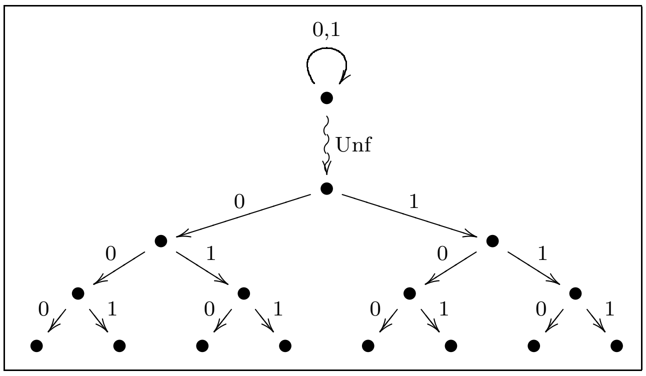


Figure 2.4: One-element graph with two self loops and its unfolding Δ_2

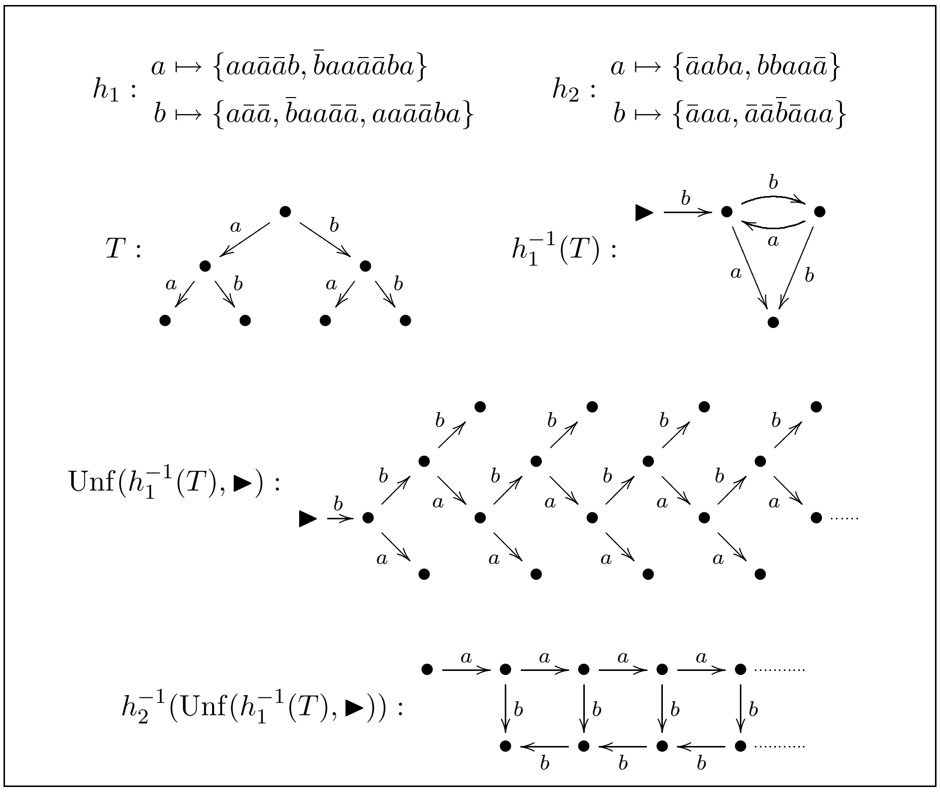
In [Cau02] D. Caucal proposed to iterate this schema and to define a whole hierarchy of graph classes and corresponding tree classes. The starting point is the class $\text{Tree}(0)$ of all finite Σ-labeled trees for $\Sigma \subseteq \Lambda$.

For $n \geq 0$ let

$$\begin{aligned}\mathrm{Graph}(n) &:= \big\{h^{-1}(T) \mid T \in \mathrm{Tree}(n),\ h \text{ a rational mapping}\big\},\\ \mathrm{Tree}(n+1) &:= \big\{\,\mathrm{Unf}(G,r) \mid G \in \mathrm{Graph}(n),\ r \in V^G\big\}.\end{aligned}$$

The classes of the *Caucal graphs* $(\mathrm{Graph}(n))_{n\geq 0}$ respectively *Caucal trees* $(\mathrm{Tree}(n))_{n\geq 0}$ form the *Caucal hierarchy* .

Figure 2.5 (see also [Cau02]) shows an example of a sequence of graphs obtained from a finite tree by applying inverse rational mappings and unfoldings iteratively. Starting from a finite tree a finite graph is obtained. Its unfolding is a regular tree of finite degree (A tree is called *regular* if it has only finitely many non-isomorphic subtrees.). The application of another inverse rational mapping yields a pushdown graph.



$$h_1 : \begin{array}{l} a \mapsto \{aa\bar{a}\bar{a}b, \bar{b}aa\bar{a}\bar{a}ba\} \\ b \mapsto \{a\bar{a}\bar{a}, \bar{b}aa\bar{a}\bar{a}, aa\bar{a}\bar{a}ba\} \end{array} \qquad h_2 : \begin{array}{l} a \mapsto \{\bar{a}aba, bbaa\bar{a}\} \\ b \mapsto \{\bar{a}aa, \bar{a}\bar{a}\bar{b}\bar{a}aa\} \end{array}$$

T :
$h_1^{-1}(T)$:
$\mathrm{Unf}(h_1^{-1}(T), \blacktriangleright)$:
$h_2^{-1}(\mathrm{Unf}(h_1^{-1}(T), \blacktriangleright))$:
a
b

Figure 2.5: Sequence of graphs in the Caucal hierarchy

The example shown in Figure 2.5 is typical. Every finite graph G with m vertices can be defined by an inverse rational mapping in a complete finite binary tree with more than m vertices. Hence we obtain that

$\mathrm{Graph}(0)$ is the class of all finite graphs. The class $\mathrm{Tree}(1)$ consists of unfoldings of finite graphs which are the regular trees of finite degree. On the other hand every regular tree of finite degree can be defined in the infinite binary tree Δ_2 by an MSO-interpretation. Using the characterization of prefix recognizable graphs from [Bar97, Blu01], we obtain that the graphs in $\mathrm{Graph}(1)$ are prefix recognizable and thus are pushdown graphs [Sti].

In [Cau02] D. Caucal studied a subhierarchy of the tree hierarchy. He considered only trees which represent terms over a ranked alphabet A. He extended the notion of rational mappings to cover also the case of graphs with vertex labels in A and considered only rational mappings which preserve the determinism of trees. The term hierarchy was then defined, starting from regular term trees, by iteratively applying the unfolding operation and inverse deterministic rational mappings, thereby restricting each class to trees which represent terms. A precise definition of the term hierarchy can be found in [Cau02].

D. Caucal established that the term hierarchy contains the hierarchy of terms that are solutions of safe higher-order schemes [Dam82, KNU02] and with the hierarchy of terms which are obtained by iterated first-order substitutions from the regular terms [CK02].

Chapter 3

Graphs of Higher-order Pushdown Systems

In this chapter we consider graphs which can be represented by higher-order pushdown systems. A higher-order pushdown system generalizes a usual pushdown system in the sense that is allows a higher-order pushdown stack, i.e. a stack whose entries are again (higher-order) stacks. Higher-order pushdown systems can be naturally sorted by the nesting depth of stacks allowed and the graphs these systems generate again form a hierarchy.

In the next section we will introduce higher-order pushdown systems formally. Then we will the work towards the main result of this chapter: The Caucal hierarchy of graphs coincides with the hierarchy of graphs generated by higher-order pushdown systems.

To accomplish this result we first have to show two closure properties of the Caucal hierarchy, namely its closure under monadic second-order definable transductions and the treegraph operation. As a side effect of these considerations we obtain yet another characterization of the Caucal hierarchy of graphs in terms of iteratively applying the treegraph operation and MSO-definable transductions. These results are presented in Section 3.2.

In Section 3.3 we show that every graph generated by a higher-order pushdown system is a graph on the corresponding level of the Caucal hierarchy. The converse result is presented in Section 3.4.

In the last section we summarize some properties of the classes of higher-order pushdown graphs and by the equivalence shown before also of the classes of graphs in the Caucal hierarchy.

The results in this chapter originated in a close collaboration with A.

Carayol and have been published in [CW03].

3.1 Higher-order Pushdown Systems

In this section we define higher-order pushdown systems which extend the pushdown systems presented in Subsection 2.2.1 and which work with an internal memory of a higher-order pushdown stack instead of a pushdown stack (of level 1). Our definition of a higher-order pushdown stack follows the one given in [KNU02].

Let Γ be a finite set of stack symbols. A *level 1 pushdown stack* is a sequence $s := [\gamma_m, \dots, \gamma_1]$ where $\gamma_m, \dots, \gamma_1 \in \Gamma$. For $n \geq 2$ a *pushdown stack of level* n is inductively defined as a sequence $[s_r, \dots, s_1]$ of pushdown stacks $s_r, \dots, s_1$ of level $n-1$. We denote the empty stack of level 1 by $[\varepsilon]$. The empty stack of level $n \geq 2$ is defined to be the stack of level n which contains only the empty stack of level $n-1$ and is denoted by $[\varepsilon]^n$.

Next we define the instructions which are used to modify higher-order pushdown stacks. For level one stacks $[\gamma_m, \dots, \gamma_1]$ these are the instructions

$$\begin{aligned}
\mathrm{push}_1^{\gamma}([\gamma_m, \dots, \gamma_1]) &:= [\gamma, \gamma_m, \dots, \gamma_1] \text{ for every } \gamma \in \Gamma \\
\mathrm{pop}_1([\gamma_m, \gamma_{m-1}, \dots, \gamma_1]) &:= [\gamma_{m-1}, \dots, \gamma_1]
\end{aligned}$$

For a stack $[s_r, \dots, s_1]$ of level $n \geq 2$ we define the instructions

$$\begin{aligned}
\mathrm{push}_1^{\gamma}([s_r, \dots, s_1]) &:= [\mathrm{push}_1^{\gamma}(s_r), s_{r-1}, \dots s_1] \text{ for every } \gamma \in \Gamma \\
\mathrm{push}_n([s_r, \dots, s_1]) &:= [s_r, s_r, \dots, s_1] \\
\mathrm{push}_k([s_r, \dots, s_1]) &:= [\mathrm{push}_k(s_r), s_{r-1}, \dots, s_1] \text{ for } 2 \leq k < n \\
\mathrm{pop}_n([s_r, \dots, s_1]) &:= [s_{r-1}, \dots, s_1] \\
\mathrm{pop}_k([s_r, \dots, s_1]) &:= [\mathrm{pop}_k(s_r), s_{r-1}, \dots, s_1] \text{ for } 1 \leq k < n
\end{aligned}$$

The operation pop_n for $n \geq 2$ is undefined on stacks of level n which consist only of one element, a pushdown stack of level $n-1$. Therefore also the operations pop_k for $2 \leq k < n$ are undefined on stacks of level n whose topmost level k stack consists of only one element. This ensures that the operations defined above preserve the structure of higher-order stacks.

The instruction push_1^γ adds the symbol γ to the topmost level 1 stack, while push_k duplicates the topmost level $k-1$ stack. pop_1 removes the top symbol of the topmost level one stack while pop_k for $1 < k \leq n$ removes the topmost level $k-1$ stack completely.

Figure 3.1 shows a sequence of instructions applied to the empty stack of level 3. First the letter a is pushed onto the stack. Then, using the push_2 instruction, the top stack of level 1 is copied and letter b is added by push_1^b to the now topmost level 1 stack. The first instruction in the second line, push_3, copies the topmost level 2 stack. The top level 1 stack of this copy is removed by pop_2, and finally the top level 1 stack (and coincidentally the top level 2 stack) is emptied with the instruction pop_1. Note that pop_3 is not applicable to any of the stacks in the first line, and that the two level 3 stacks in the rightmost column are different since the empty top level 2 stack in the second line has not been explicitly removed by a pop_3 instruction.

$$[[[\varepsilon]]] \xrightarrow{\mathrm{push}_1^a} [[[a]]] \xrightarrow{\mathrm{push}_2} \left[\left[\begin{matrix}{[a]}\\ {[a]}\end{matrix}\right]\right] \xrightarrow{\mathrm{push}_1^b} \left[\left[\begin{matrix}{[ba]}\\ {[a]}\end{matrix}\right]\right]$$

$$\xrightarrow{\mathrm{push}_3} \left[\begin{matrix}\left[\begin{matrix}{[ba]}\\ {[a]}\end{matrix}\right]\\ \left[\begin{matrix}{[ba]}\\ {[a]}\end{matrix}\right]\end{matrix}\right] \xrightarrow{\mathrm{pop}_2} \left[\begin{matrix}\left[\begin{matrix}{[a]}\end{matrix}\right]\\ \left[\begin{matrix}{[ba]}\\ {[a]}\end{matrix}\right]\end{matrix}\right] \xrightarrow{\mathrm{pop}_1} \left[\begin{matrix}\left[\begin{matrix}{[\varepsilon]}\end{matrix}\right]\\ \left[\begin{matrix}{[ba]}\\ {[a]}\end{matrix}\right]\end{matrix}\right]$$

Figure 3.1: An example sequence of stack operations

We denote by Instr_n the set of instructions which can be applied on level n. To simplify some of the constructions in this chapter we add an identity function denoted by $-$ to Instr_n which does not change the stack content.

The applicability of transitions of a higher-order pushdown system as defined below depends on the topmost symbol of the topmost level one stack which is formally defined by $\mathrm{top}([\varepsilon]) := \varepsilon$, $\mathrm{top}([\gamma_m, \ldots, \gamma_1]) := \gamma_m$ for $m \geq 1$, and $\mathrm{top}([s_r, \ldots, s_1]) := \mathrm{top}(s_r)$.

For $n \geq 1$, a *higher-order pushdown system of level* n is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta)$ where Q is a finite set of states, Σ is an input alphabet, Γ is a stack alphabet, $q_0 \in Q$ is an initial state, and $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup$

$\{\varepsilon\}) \times Q \times \mathrm{Instr}_n$ is a transition relation. A *configuration* of $\mathcal{A}$ is a pair $(q, [s_r, \ldots, s_1])$ where q is a state of $\mathcal{A}$ and $[s_r, \ldots, s_1]$ is a stack of level n. The initial configuration $(q_0, [\varepsilon]^n)$ consists of the initial state q_0 and the empty level n stack $[\varepsilon]^n$. $\mathcal{A}$ can reach a configuration $(q', [s'_{r'}, \ldots, s'_1])$ from $(q, [s_r, \ldots, s_1])$ via an a-labeled edge for some $a \in \Sigma \cup \{\varepsilon\}$ if there is a transition $(q, a, \mathrm{top}([s_r, \ldots, s_1]), q', i) \in \Delta$ such that $i([s_r, \ldots, s_1]) = [s'_{r'}, \ldots, s'_1]$. In particular a transition $(q, a, \mathrm{top}([s_r, \ldots, s_1]), q', i)$ can only be executed in configuration $(q, [s_r, \ldots, s_1])$ if $i([s_r, \ldots, s_1])$ is defined.

A higher-order pushdown system of level 0 is just a finite state system, i.e. of the form $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ where Q, Σ and q_0 are as above and $\Delta \subseteq Q \times \Sigma \times Q$. We denote by $\mathrm{HOPDS}(n)$ the class of all higher-order pushdown systems of level n.

As in Subsection 2.2.1 we define the *configuration graph* $\mathcal{C}(A)$ of $\mathcal{A}$ as the graph of all configurations reachable from the initial configuration of $\mathcal{A}$ with the incident edges as defined above.

A graph G is *generated by a higher-order pushdown system* $\mathcal{A}$ of level n if G is the ε-closure of the configuration graph of $\mathcal{A}$. In this case we call G a *higher-order pushdown graph of level* n. The class of all higher-order pushdown graphs of level n is denoted by $\mathrm{HOPDG}(n)$.

The graph classes $\mathrm{HOPDG}(n)$ for $n \geq 0$ form a hierarchy of graphs starting with the class of finite graphs $\mathrm{HOPDG}(0)$ and the class of pushdown graphs $\mathrm{HOPDG}(1)$. It thus coincides with the Caucal hierarchy of graphs on the first two levels. In the remainder of this chapter we will extend this characterization to every level.

Higher-order pushdown systems have already been considered in the 70's. In [Gre70] S.A. Greibach attributes the idea of higher-order pushdown stacks to A.V. Aho and J.D. Ullman, but apparently their investigations were never published. In [Mas76] A.N. Maslov introduced higher-order pushdown systems under the name multilevel stack automata.

Multilevel stack automata differ only slightly from the iterated pushdown automata considered by W. Damm and A. Goerdt [DG86] and J. Engelfriet [Eng91]. In particular, both automaton models work with higher-order stacks where the entries of a higher-order stack of level n are pairs of a single stack symbol and a higher-order stack of level $n-1$. More formally, such an iterated pushdown stack of level n is a sequence $[(\gamma_m, s_m) \ldots (\gamma_1, s_1)]$ where $\gamma_i \in \Gamma$ and s_i is an iterated stack of level $n-1$

for every $1 \leq i \leq m$. The operations on iterated pushdown stacks modify the extra stack symbols in addition to changes of the corresponding stacks. The top function is redefined and returns the n-tuple of leading symbols from the topmost level n stack. For a precise definition see [Eng91].

Following an argument presented in [KNU02] it is easy to see that higher-order pushdown systems generate the same classes of graphs as iterated pushdown automata do. The additional labels of an iterated pushdown stack can be stored on the top level one stack of a higher-order pushdown stack and every transition of an iterated stack automaton can be simulated by a sequence of transitions of a higher-order pushdown system.

Higher-order pushdown stacks should not be confused with nested stacks which have been introduced by A.V. Aho in [Aho69]. Indeed nested stack automata can be simulated by higher-order pushdown systems of level 2 and vice versa, see [Eng91].

In [Mas76], [DG86] and [Eng91] higher-order pushdown systems have been used as language acceptors. To accept a language by a higher-order pushdown system $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta)$ we equip the system with a set $F \subseteq Q$ of final states and call $\mathcal{A}' = (Q, \Sigma, \Gamma, q_0, \Delta, F)$ a *higher-order pushdown automaton* to distinguish it from the higher-order pushdown system $\mathcal{A}$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}'$ if there is a path in $C(\mathcal{A})$ labeled by w from the initial configuration $(q_0, [\varepsilon]^n)$ to a configuration (q, s) with $q \in F$. The set $L := \{w \mid \mathcal{A}' \text{ accepts } w\}$ is the *language* accepted by $\mathcal{A}'$.

Analogously to the acceptance by final states stated above we can also define the acceptance of a word w "by empty stack" by adding bottom symbols $\perp_1, \ldots, \perp_n$ to the stack alphabet (one for each level) and requiring that there is a path in $C(\mathcal{A})$ labeled by w from the initial configuration $(q_0, [\perp_n \ldots \perp_1]^n)$ to a configuration $(q, [\varepsilon]^n)$ with empty stack.

Using the same techniques as for level one (see e.g. [HU79]) one can show that the class of languages accepted by higher-order pushdown automata of level n with final states is the same as the class of languages accepted higher-order pushdown automata of level n with empty stack. This is most easily seen using the iterated pushdown model of [DG86, Eng91] where adding a single additional symbol $\perp$ suffices.

Example 3.1 We construct a higher-order pushdown automaton $\mathcal{A}$ accepting the language $L := \{a^n b^{2^n} \mid n \geq 0\}$ by empty stack. The automaton implements a binary counter of length n using the level 2 stack. The length of the counter is stored on in the bottom level one stack, the binary number represented by the stack consists of the topmost symbols all level one stacks (with the least important bit on the top). Instead of a formal definition of $\mathcal{A}$ we describe its behavior using the sample configuration sequence on input a^3b^8 given in Figure 3.2.

$$q_0, [[\varepsilon]] \overset{aaa}{\rightsquigarrow} q_1, [[000]] \overset{\varepsilon}{\rightsquigarrow} q_2 \begin{bmatrix} [0] \\ {[00]} \\ {[000]} \end{bmatrix}$$

$$\overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [1] \\ {[00]} \\ {[000]} \end{bmatrix} \overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [0] \\ {[10]} \\ {[000]} \end{bmatrix} \overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [1] \\ {[10]} \\ {[000]} \end{bmatrix}$$

$$\overset{b}{\rightsquigarrow} q_2 \begin{bmatrix} [0] \\ {[00]} \\ {[100]} \end{bmatrix} \overset{b^4}{\rightsquigarrow} q_2 \begin{bmatrix} [1] \\ {[10]} \\ {[100]} \end{bmatrix} \overset{\varepsilon}{\rightsquigarrow} q_f, [[\varepsilon]]$$

Figure 3.2: Configuration sequence on input a^3b^8

Starting from the initial state q_0 the automaton builds while reading the block of a's a bottom level one stack of the same length as the block of a's. It nondeterministically guesses the end of this block, enters state q_1 and creates the rightmost level two stack of line one. The top symbols of the level one stacks represent the number of b's read so far. The counter for the b's is increased in state q_2 in the following way:

- If the topmost symbol of the topmost level one stack is 0 it is replaced by 1.
- Otherwise the level one stacks are popped until one with topmost symbol 0 is reached. This 0 is replaced by 1 and the tower of level one stacks consisting just of 0's is recreated.

$\mathcal{A}$ guesses the end of the block of b's and then verifies that all level one stacks have 1 as topmost symbol. Note that for this it is necessary to be able to identify the bottom level one stack. This can be achieved for example by creating a bottom level one stack $[01]$ in the very beginning which is not affected by the counting procedure described above.

Recall that we defined a higher-order pushdown automaton of level 0 to be just a finite automaton. Thus the classes of languages accepted by higher-order pushdown automata of level zero, one and two are the regular languages, the context-free languages and the indexed languages. A.N. Maslov [Mas76] showed that higher-order pushdown automata on level n accept the languages generated by indexed grammars of level n. W. Damm and A. Goerdt [DG86] proved that a language is an OI-language of level n if, and only if, it is accepted by a higher-order pushdown automaton of level n. J. Engelfriet [Eng91] characterized the languages accepted by higher-order pushdown automata of level n by deterministic non-elementary time complexity classes thereby showing that the OI-hierarchy of languages is strict level by level.

There is a close connection between languages which can be accepted by higher-order pushdown automata of level n and the graphs generated by higher-order pushdown systems of level n. It is a simple fact that every higher-order pushdown automaton of level n accepting a language L can be transformed into one in which every final state is a sink and which still accepts L. In the graph generated by this automaton the set of path labels from the initial configuration to the leafs of the graph (vertices without an outgoing edge), i.e. the set of *traces* of the graph, is exactly L. The converse also holds. For every higher-order pushdown graph G of level n there is a higher-order pushdown automaton of level n which accepts the traces of the graph. To see this it suffices to ensure that in the higher-order pushdown system generating G a transition $(p, a, \alpha, q, i) \in \Delta$ is applicable in every configuration (q, s) with $\mathrm{top}(s) = \alpha$, and that if $(p, a, \alpha, q, i) \in \Delta$ then also $(p, a', \beta, q', i') \in \Delta$ for every $\beta \in \Gamma$. The first property can be ensured using bottom symbols on every level of the stack, the second property is achieved by adding corresponding ε-transitions which lead to an ε-loop and then normalizing the resulting system to ensure that the ε-closure can be computed. Then all states p for which no transition $(p, a, \alpha, q, i) \in \Delta$ exists can be marked as final and the resulting higher-order pushdown automaton accepts exactly the traces of G.

We state this correspondence in the following proposition.

Proposition 3.2 *For every $n \geq 0$ and every language $L \subseteq \Sigma^*$ it holds that L is accepted by a higher-order pushdown automaton on level n iff L is the set of traces of a higher-order pushdown graph of level n.* □

The correspondence between the traces of graphs and languages accepted by higher-order pushdown automata allows us to transfer the hierarchy results of [Eng91] to the graph classes. This is done in Section 3.5.

3.2 The Treegraph Operation and Transductions

In this section we show that the Caucal hierarchy of graphs is closed under the application of the treegraph operation. More precisely we show that for any graph $G \in \mathrm{Graph}(n)$ we have $\mathrm{Treegraph}(G, \sharp) \in \mathrm{Graph}(n+1)$. This result will be used in the following section to show that every higher-order pushdown graph is a graph in the Caucal hierarchy.

Furthermore we show that the Caucal hierarchy of graphs coincides with the hierarchy obtained from the class of finite graphs by iteratively applying the treegraph operation and MSO-definable transductions. We thus get a characterization in terms of more powerful operations than the unfolding and rational mappings.

The characterization of the Caucal hierarchy of graphs in terms of the treegraph operation and MSO-definable transductions is split into two parts. In the next subsection we introduce a subhierarchy of the Caucal hierarchy where the tree classes are restricted to deterministic trees. We will show that in this case the corresponding graph classes are closed under MSO-definable transductions. Then we show, for the case of deterministic graphs, how to simulate the treegraph operation using unfolding, inverse rational mappings, and MSO-definable markings.

In Subsection 3.2.2 we then show that every graph in the Caucal hierarchy can be obtained by a transduction from a deterministic tree.

Combining these results enables us to show that the Caucal hierarchy of graphs coincides with the subhierarchy defined in Section 3.2.1 and thus with the hierarchy obtained by iterating the treegraph operation and transductions.

The results in this section benefited a lot from previous work of Carayol and Colcombet [CC03].

3.2.1 A Deterministic Subhierarchy

In this section we introduce a restriction of the Caucal hierarchy where we consider only unfoldings of deterministic graphs.

By DGraph we denote the class of of all deterministic graphs. We define the *deterministic* Caucal hierarchy. Starting from the class $\mathrm{Tree}^d(0)$ of all finite deterministic trees let

$$\begin{aligned}\mathrm{Graph}^d(n) &:= \{h^{-1}(T) \mid T \in \mathrm{Tree}^d(n),\ h^{-1} \text{ a rational mapping}\},\\ \mathrm{Tree}^d(n+1) &:= \{\,\mathrm{Unf}(G,r) \mid G \in \mathrm{Graph}^d(n) \cap \mathrm{DGraph},\ r \in V^G\}.\end{aligned}$$

for $n \geq 0$. Note that by definition $\mathrm{Tree}^d(n)$ contains only deterministic trees while $\mathrm{Graph}^d(n)$ also contains non-deterministic graphs.

Since every finite non-deterministic tree can be made deterministic by renaming edges (thereby enlarging the set of edge labels) it is easy to see that $\mathrm{Graph}^d(0) = \mathrm{Graph}(0)$, i.e. $\mathrm{Graph}^d(0)$ is the class of finite graphs.

The restriction to unfoldings of deterministic graphs allows us to apply partial commutation results from [CC03] to prove the closure of the $\mathrm{Graph}^d(n)$ classes under transductions. For the Tree^d classes we obtain a similar result restricted to the special case of monadic second-order definable markings.

Let $G = (V, (E_a)_{a\in\Sigma})$ be a graph. The *marking* of a set $W \subseteq V$ of G by a symbol $\sharp \notin \Sigma$ results in the graph $\mathcal{M}_\sharp(G, W) := (V', (E'_a)_{a\in\Sigma\cup\{\sharp\}})$ where $V' := V \cup (V \times \{\sharp\})$, $E'_a := E_a$ and $E'_\sharp := \{(v,(v,\sharp)) \mid v \in W\}$. A marking $\mathcal{M}_\sharp(G, W)$ is *MSO-definable* (shortly an *MSO-marking*) if there exists an MSO-formula $\psi(x)$ such that $W := \{v \in V \mid G \models \psi[v]\}$. Note that every MSO-marking is an MSO-transduction. If the marking symbol and the set of vertices which are marked are clear from the context or do not matter we denote a marking just by $\mathcal{M}$.

Proposition 3.3 ([CC03])

1. *For every deterministic graph G, every vertex $r \in V^G$ and every MSO-transduction $\mathcal{T}$ there exists an MSO-transduction $\mathcal{T}'$, a vertex $r' \in V^{\mathcal{T}'(G)}$ and a rational mapping h such that*

$$\mathcal{T}(\mathrm{Unf}(G,r)) = h^{-1}(\mathrm{Unf}(\mathcal{T}'(G), r')).$$

2. *For every deterministic graph G, every vertex $r \in V^G$ and every MSO-marking $\mathcal{M}$ there exists an MSO-transduction $\mathcal{T}$ and a vertex $r' \in V^{\mathcal{T}(G)}$ such that*

$$\mathcal{M}(\mathrm{Unf}(G, r)) = \mathrm{Unf}(\mathcal{T}(G), r').$$

In either case $\mathcal{T}'(G)$ respectively $\mathcal{T}(G)$ is again a deterministic graph.

Proof (Sketch). We shortly sketch the proof of first statement of the proposition from which the second part follows.

Let G be a deterministic graph, r a vertex of G, and $\mathcal{T} = (K, \mathcal{I})$ a transduction. Since the copying-operation K commutes with the unfolding, i.e. $K(\mathrm{Unf}(G, r)) = \mathrm{Unf}(K(G), r)$, and preserves the determinism of the graph it suffices to consider only the interpretation $\mathcal{I}$. For simplification we assume that $\mathcal{I}$ consists of a single monadic second-order formula $\varphi(x, y)$. For $\varphi(x, y)$ there exists a parity automaton $\mathcal{A}_\varphi$ which accepts exactly those trees T with $T \models \varphi[u, v]$, see e.g [Tho97]. Thereby the positions of x and y are marked by additional monadic predicates. For every vertex $w \in V^T$ let $\theta(w)$ be the set of transitions of $\mathcal{A}_\varphi$ which start a successful run on the subtree rooted at w for some interpretation of the additional predicates. Due to the existence of regular runs [CW98] we have that $\theta(w) = \theta(w')$ if the subtrees rooted at w and w' are isomorphic. Thus the function θ is compatible with the unfolding operation. Moreover, since the existence of a successful run on a subtree starting with a certain transition can be checked on G using an MSO-formula, the information provided by θ can be attached to G using an MSO-transduction $\mathcal{T}'$. Since every MSO-marking a can be seen as a special MSO-transduction this already proves Part 2 of the proposition.

For Part 1 we additionally assume that the states of $\mathcal{A}_\varphi$ signal whether x or y are expected in a subtree, i.e. for every tree T with additional monadic predicates for x and y, if in a successful run of $\mathcal{A}_\varphi$ state q is assumed at a vertex w of T, then the subtree rooted at w contains exactly the monadic predicates for x and y signalled by q.

The rational mapping h now connects vertices u, v of $\mathrm{Unf}(\mathcal{T}'(G), r')$ if $\theta(u)$ contains a state signaling that u is interpreted as x, $\theta(v)$ contains a state signaling that v is interpreted as y, and the labeling of the vertices on the path from u to v by θ is compatible with the transition relation of $\mathcal{A}_\varphi$. □

Using Proposition 3.3 we can prove the following closure properties of the classes of the deterministic Caucal hierarchy.

Proposition 3.4

1. *For every $n \geq 0$, every $G \in \mathrm{Graph}^d(n)$ and every MSO-transduction $\mathcal{T}$ we have $\mathcal{T}(G) \in \mathrm{Graph}^d(n)$ too.*
2. *For every $n \geq 0$, every $T \in \mathrm{Tree}^d(n)$ and every MSO-marking $\mathcal{M}$ we have $\mathcal{M}(T) \in \mathrm{Tree}^d(n)$ too.*

Proof. The proof proceeds by induction on the level n of the deterministic Caucal hierarchy. For $n = 0$ there is nothing to show. Just note for 1. that $\mathrm{Graph}^d(0)$ is the class of all finite graphs and for 2. that a marking preserves the determinism as well as the tree structure.

For the induction step we first prove 1. Let $G \in \mathrm{Graph}^d(n)$ and $\mathcal{T}$ be a transduction. By definition of $\mathrm{Graph}^d(n)$ there exists a rational mapping h, a graph $G' \in \mathrm{Graph}^d(n-1)$ and a vertex $r \in V^{G'}$ such that $G = h^{-1}(\mathrm{Unf}(G', r))$. Since inverse rational mappings are special transductions and these are closed under composition, there exists a transduction $\mathcal{T}'$ such that

$$\mathcal{T}(G) = (\mathcal{T} \circ h^{-1})(\mathrm{Unf}(G', r)) = \mathcal{T}'(\mathrm{Unf}(G', r)).$$

By Proposition 3.3 there exists another transduction $\mathcal{T}''$, a rational mapping h_1 and a vertex $r' \in V^{\mathcal{T}(G')}$ such that

$$\mathcal{T}'(\mathrm{Unf}(G', r)) = h_1^{-1}(\mathrm{Unf}(\mathcal{T}''(G'), r')).$$

By the induction hypothesis $\mathcal{T}''(G') \in \mathrm{Graph}^d(n-1)$. Since by Proposition 3.3 the transduction $\mathcal{T}''$ preserves the determinism of G' we obtain that $\mathrm{Unf}(\mathcal{T}''(G'), r') \in \mathrm{Tree}^d(n)$ and hence $\mathcal{T}(G) \in \mathrm{Graph}^d(n)$.

For the proof of 2. let $T \in \mathrm{Tree}^d(n)$ and $\mathcal{M}$ be a marking. By definition there exists a deterministic graph $G \in \mathrm{Graph}^d(n-1)$ and a vertex $r \in V^G$ such that $T = \mathrm{Unf}(G, r)$. By Proposition 3.3 there exist a transduction $\mathcal{T}$ and a vertex $r' \in V^{\mathcal{T}(G)}$ such that

$$\mathcal{M}(T) = \mathcal{M}(\mathrm{Unf}(G, r)) = \mathrm{Unf}(\mathcal{T}(G), r').$$

$\mathcal{T}$ preserves the determinism of G. Since by Part 1 of the proposition $\mathcal{T}(G) \in \mathrm{Graph}^d(n-1)$ we obtain $\mathcal{M}(T) \in \mathrm{Tree}^d(n)$. □

As explained in Subsection 2.3.2 the unfolding of a connected graph G from a definable vertex is MSO-definable in the treegraph of G. In the case of deterministic trees we can show a converse result: It is possible to define the treegraph of a deterministic tree using the unfolding operation and MSO-transductions.

Lemma 3.5 *Let Σ be a finite set of edge labels. Then there exist finite mappings h_1 and h_2 and an MSO-definable marking $\mathcal{M}$ such that for every deterministic Σ-labeled tree T with root r*

$$\mathrm{Treegraph}(T, \sharp) = h_2^{-1}(\mathcal{M}(\mathrm{Unf}(h_1^{-1}(T), r))).$$

Proof. Let $\sharp$ be a label not in Σ and $\tilde{\Sigma}$ be a set of edge labels disjoint but in bijection to Σ. The finite mapping h_1 adds backward edges labeled by elements of $\tilde{\Sigma}$ and a loop labeled by $\sharp$ to every vertex. Formally $h_1(a) := \{a\}$, $h_1(\tilde{a}) := \{\bar{a}\}$ for every $a \in \Sigma$ and $h_1(\sharp) := \{\varepsilon\}$. Obviously h_1 preserves the determinism of T. The marking $\mathcal{M}$ marks all the vertices v in $\mathrm{Unf}(h_1^{-1}(T), r)$ by $\$ \notin \Sigma$ whose unique path from r to v does not contain an infix $a\tilde{a}$ or $\tilde{a}a$ for $a \in \Sigma$. $\mathcal{M}$ is obviously MSO-definable. The finite mapping h_2 erases all unmarked vertices and reverses the edges labeled by a symbol from $\tilde{\Sigma}$. Formally h_2 is given by $h_2(\sharp) = \{\sharp\}$ and $h_2(a) = \big\{\$\bar{\$}a\$\bar{\$}, \$\bar{\$}\bar{\tilde{a}}\$\bar{\$}\big\}$ for $a \in \Sigma$. □

Figure 3.3 illustrates the construction above. The upper graph shows an initial segment of the semi-infinite line after applying h_1^{-1}. The lower graph shows its unfolding. The filled dots represent the vertices marked by $\mathcal{M}$.

Note that statement of the previous lemma is uniform in the sense that the rational mappings and the marking only depend on set of edge labels but not on the tree under consideration.

Lemma 3.5 allows us now to show that the deterministic graph hierarchy is closed under the application of the treegraph operation.

Proposition 3.6 *For every $n \geq 0$ and every $G \in \mathrm{Graph}^d(n)$ we have $\mathrm{Treegraph}(G, \sharp) \in \mathrm{Graph}^d(n+1)$.*

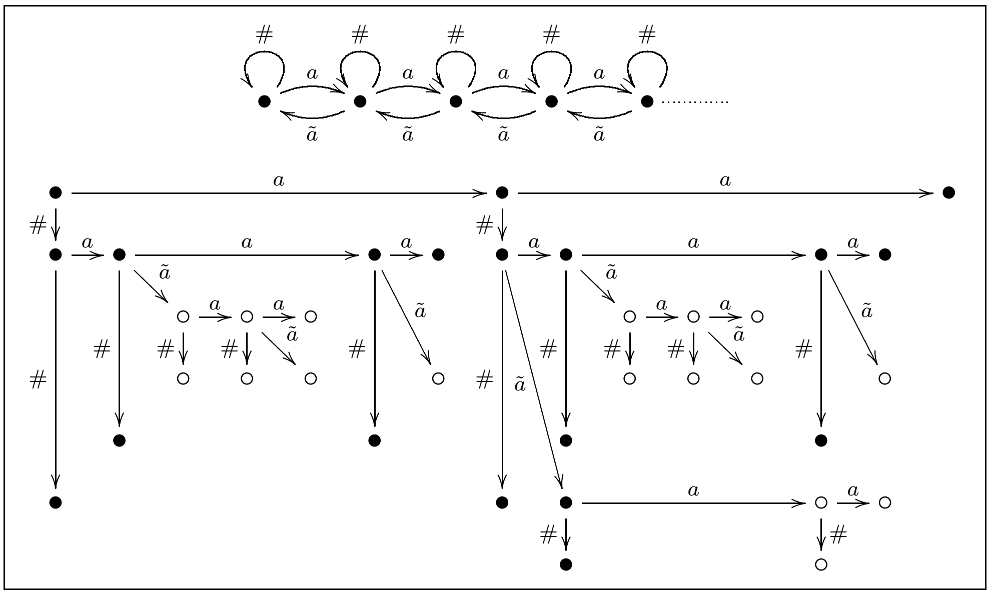


Figure 3.3: The semi-infinite line after applying h_1 and its unfolding

Proof. Let $G \in \mathrm{Graph}^d(n)$ be a Γ-labeled graph and $\# \notin \Gamma$. Then there exists Σ-labeled tree $T \in \mathrm{Tree}^d(n)$ and a rational mapping $h : \Gamma \to \mathcal{P}(\Sigma \cup \bar{\Sigma})^*$ such that $G = h^{-1}(T)$. We extend h to a rational mapping $h_\#$ with domain $\Gamma \cup \{\#\}$ by setting $h_\#(a) := h(a)$ for every $a \in \Gamma$ and $h_\#(\#) := \{\#\}$. Then

$$\mathrm{Treegraph}(G, \#) = \mathrm{Treegraph}(h^{-1}(T), \#) = h_\#^{-1}(\mathrm{Treegraph}(T, \#))$$

and by Lemma 3.5

$$h_\#^{-1}(\mathrm{Treegraph}(T, \#)) = h_\#^{-1}(h_2^{-1}(\mathcal{M}(\mathrm{Unf}(h_1^{-1}(T, r))))).$$

Furthermore $\mathrm{Unf}(h_1^{-1}(T, r))$ is a deterministic tree in $\mathrm{Tree}^d(n+1)$ and hence by part two of Proposition 3.3 we have $\mathcal{M}(\mathrm{Unf}(h_1^{-1}(T, r)))) \in \mathrm{Tree}^d(n+1)$. Thus $\mathrm{Treegraph}(G, \sharp) \in \mathrm{Graph}^d(n+1)$. $\square$

3.2.2 Deterministic Trees Suffice

We now show that deterministic trees indeed suffice to generate the complete Caucal hierarchy. This allows us to transfer the results of the previous subsection to prove that the Caucal hierarchy coincides with the hierarchy defined by iteratively applying the treegraph operation and MSO-definable transductions.

Lemma 3.7 $\mathrm{Graph}(n) = \mathrm{Graph}^d(n)$ *for every* $n \geq 0$.

Proof. The inclusion from right to left is trivial. For the converse we have to show that for every $n \geq 0$ and every $G \in \mathrm{Graph}(n)$ there exists a $T \in \mathrm{Tree}^d(n)$ and a rational mapping h such that $G = h^{-1}(T)$. Again we proceed by induction on n. By definition of $\mathrm{Graph}(n)$ there exists a rational mapping g_1 and a (possibly non-deterministic) tree $T \in \mathrm{Tree}(n)$ such that $G = g_1^{-1}(T)$. Since the composition of rational mappings is again a rational mapping it suffices to show that $T \in \mathrm{Graph}^d(n)$.

For $n = 0$ there is almost nothing to show. Since the degree of T is bounded by some $m \in \mathbb{N}$ we can replace every edge label $a \in \Sigma$ by m copies $a_1, \ldots, a_m$, convert T into a deterministic tree T' by replacing multiple a-labeled edges by corresponding copies, and by replacing every symbol a which occurs in a regular expression defining g_1 by $(a_1 + \ldots + a_m)$.

For $n > 0$ we may assume by the induction hypothesis that $T = \mathrm{Unf}(g_2^{-1}(T'), s)$ for some rational mapping g_2, some $T' \in \mathrm{Tree}^d(n-1)$ and $s \in V^{g_2^{-1}(T')}$. Now we can replace the unfolding by an application of treegraph and a transduction: $T = \mathcal{T}(\mathrm{Treegraph}(g_2^{-1}(T'), \sharp))$. As in the proof of Proposition 3.6 we can extend g_2 by $g_2(\sharp) := \{\sharp\}$ to swap the treegraph operation and g_2. This allows us to apply Lemma 3.5 to obtain

$$\begin{aligned} T = \mathcal{T}(\mathrm{Treegraph}(g_2^{-1}(T'), \sharp)) &= (\mathcal{T} \circ g_2^{-1})(\mathrm{Treegraph}(T', \sharp)) \\ &= (\mathcal{T} \circ g_2^{-1} \circ h_2^{-1} \circ \mathcal{M})(\mathrm{Unf}(h_1^{-1}(T'), r)) \end{aligned}$$

where h_1 and h_2 denote the rational mappings of Lemma 3.5 and r is the root of T'. $h_1^{-1}(T')$ is a deterministic graph in $\mathrm{Graph}^d(n-1)$, hence $\mathrm{Unf}(h_1^{-1}(T'), r) \in \mathrm{Tree}^d(n) \subseteq \mathrm{Graph}^d(n)$. Since $\mathcal{T} \circ g_2^{-1} \circ h_2^{-1} \circ \mathcal{M}$ is again an MSO-transduction we can conclude using Proposition 3.4 that $T \in \mathrm{Graph}^d(n)$. $\square$

Combining Lemma 3.7, Proposition 3.6 and Proposition 3.4 we obtain the following closure properties of Caucal hierarchy.

Corollary 3.8

1. *For every $n \geq 0$ and every $G \in \mathrm{Graph}(n)$ we have $\mathrm{Treegraph}(G, \sharp) \in \mathrm{Graph}(n+1)$.*
2. *For every $n \geq 0$, every $G \in \mathrm{Graph}(n)$ and every MSO-transduction $\mathcal{T}$ we have $\mathcal{T}(G) \in \mathrm{Graph}(n)$.* □

From the proof of Lemma 3.7 we can extract another important property of graphs in the Caucal hierarchy. It suffices to consider unfoldings of graphs from MSO-definable vertices.

Corollary 3.9 *For every graph $G \in \mathrm{Graph}(n)$ there is a graph $G' \in \mathrm{Graph}(n-1)$, an inverse rational mapping h, and an MSO-definable vertex r such that $G = h^{-1}(\mathrm{Unf}(G', r))$.*

Proof. As in the proof of Lemma 3.7 is suffices to show the claim only for trees in $\mathrm{Tree}(n)$ instead of graphs in $\mathrm{Graph}(n)$. So let $T \in \mathrm{Tree}(n)$. By Lemma 3.7 we may assume that $T = \mathrm{Unf}(g^{-1}(T'), s)$ for some rational mapping g, some $T' \in \mathrm{Tree}^d(n-1)$ and $s \in V^{g^{-1}(T')}$. Since T' is deterministic the vertex s is MSO-definable in T' and hence there exists an MSO-marking $\mathcal{M}$ which marks s in T' by an edge labeled $\sharp$. By Proposition 3.4 $\mathcal{M}(T', \sharp) \in Tree^d(n-1)$. Extending g by $g(\sharp) := \{\sharp\}$ we obtain the graph $G' := g^{-1}(\mathcal{M}(T', \sharp))$ in which s is MSO-definable. The rational mapping h is then used to remove the marking of s from $\mathrm{Unf}(G', s)$. □

This corollary justifies that in the definition of the Caucal hierarchy on Page 36 we allow to unfold a graph G from any vertex: For every vertex $s \in V^G$ we can find another graph G' on the same level and a vertex $s' \in V^{G'}$ such that s' is definable in G' and $\mathrm{Unf}(G, s)$ is equal to $\mathrm{Unf}(G', s')$ (up to the marking of s').

With this result and the fact that the unfolding of a graph can be defined in its treegraph by an MSO-transduction we can conclude that the graph hierarchy defined by iteratively applying the treegraph operation and MSO-transductions coincides with the Caucal hierarchy of graphs. As a direct consequence of this characterization we obtain that every graph in the Caucal hierarchy has a decidable MSO-theory.

Actually we know even more. We can add counting quantifiers $|X| = p \mod q$ for all $p < q$ and an infinity quantifier $|X| \geq \omega$ to MSO which express that a X is of cardinality $p \mod q$ respectively X is infinite to obtain the logic MSO+C. A. Blumensath [Blu03] observed that MSO-transductions preserve the decidability of MSO+C and A. Blumensath and S. Kreutzer in [BK] extended Muchnik's Theorem to MSO+C.

We summarize these results in the following theorem

Theorem 3.10

1. *The Caucal hierarchy of graphs is equal to the hierarchy obtained from the class of finite graphs by iteratively applying the treegraph operation and MSO-definable transductions.*

2. *Every graph in the Caucal hierarchy has a decidable MSO+C theory.*

□

Part 1 of Theorem 3.10 in particular implies that for every tree $T \in \mathrm{Tree}(n)$ and every MSO-interpretation $\mathcal{I}$ there exists a tree $T' \in \mathrm{Tree}(n)$ and a rational mapping h such that $\mathcal{I}(T) = h^{-1}(T')$ and vice versa for every $T' \in \mathrm{Tree}(n)$ and every rational mapping h there exists a tree $T \in \mathrm{Tree}(n)$ and an MSO-interpretation $\mathcal{I}$ such that $\mathcal{I}(T) = h^{-1}(T')$.

Since inverse rational mappings are special MSO-interpretations, in the second case the tree T can be chosen to be the same as T'. This does not hold for the first case. With respect to single trees MSO-interpretations are more expressive than inverse rational mappings. To see this we continue the example depicted in Figure 2.5. Figure 3.4 shows the unfolding of the pushdown graph of Figure 2.5 from the leftmost vertex.

With an MSO-interpretation we can characterize the vertex ▶ as the only vertex which has no incoming edge and the vertices ▼ as the ones which have no outgoing edge. Thus it is possible using an MSO-interpretation to connect ▶ with every vertex ▼ by an edge labeled c. The path language described by this MSO-interpretation in the graph

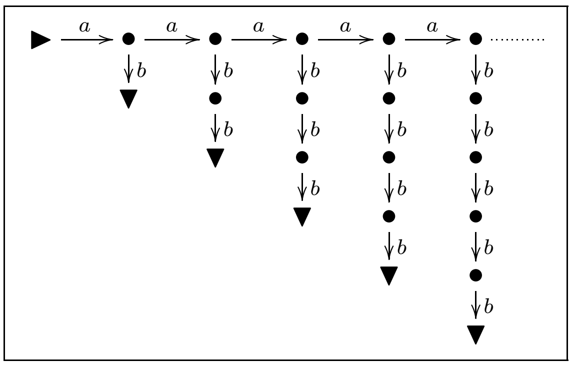


Figure 3.4: Unfolding of a pushdown graph

is $\{a^n b^n \mid n \geq 1\}$ which is not regular. Thus there is no regular expression r which can describe the set of paths from ▶ to the vertices ▼ and therefore there is no rational mapping which can add the corresponding edges.

3.3 From HOPDG to Caucal Graphs

In this section we will show that every higher-order pushdown graph of level n is a graph of level n of the Caucal hierarchy. For this purpose will identify for every higher-order pushdown system $\mathcal{A}$ a graph $G_{\mathcal{A}}$ whose vertices are (almost) in one-to-one correspondence with the configurations of $\mathcal{A}$ and then show how the edges of the configuration graph of $\mathcal{A}$ can be defined using an inverse rational mapping.

A construction along these lines was already proposed by T. Cachat in [Cac03] in the context of game simulations. The graphs he used have the disadvantage that many vertices represent the same configuration of the higher-order pushdown system.

With the results of the previous section as a background we are able to propose as base graphs the family Δ_m^n of graphs obtained by an $(n-1)$-fold application of the treegraph operation to the infinite m-ary tree Δ_m. Formally for $m, n \geq 1$ the graph Δ_m^n is defined as

$$\Delta_m^n := \text{Treegraph}(\text{Treegraph}(\dots \text{Treegraph}(\Delta_m, \#_2) \dots, \#_{n-1}), \#_n).$$

Note that $\Delta_m^1 = \Delta_m$ for every m. By Corollary 3.8 we know that $\Delta_m^n \in \text{Graph}(n)$ for every $m, n \geq 1$.

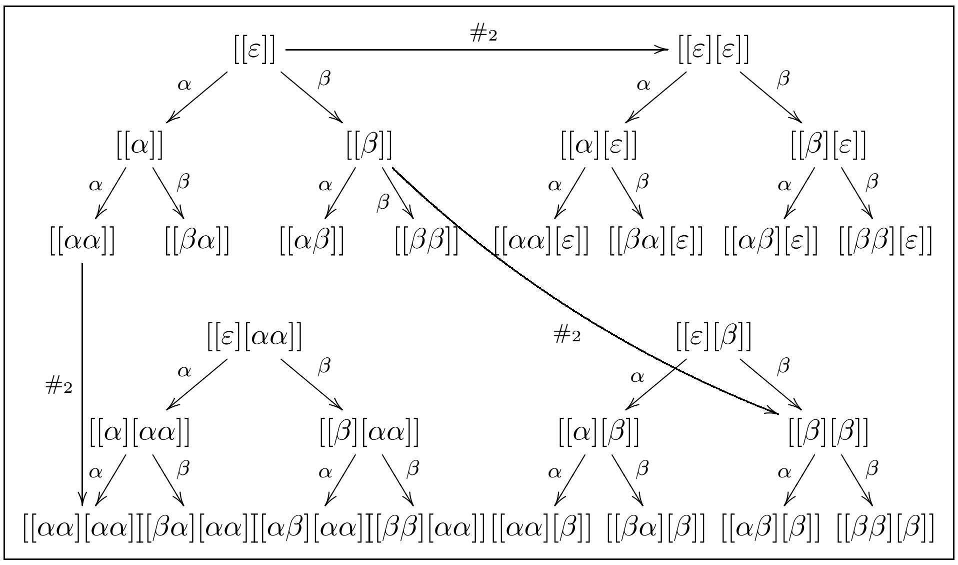


Figure 3.5: Initial segment of Δ_2^2.

The nodes of Δ_m^n are in one-to-one correspondence to higher-order pushdown stacks of level n over a stack alphabet Γ of size m. Here and in the following we assume that Γ is equipped with a linear order on its elements and we identify an edge label $j \leq m$ with the jth element of Γ. Figure 3.5 shows an initial segment of Δ_2^2 where $\Gamma = \{\alpha, \beta\}$. The labels of the vertices depict the level 2 stacks the vertices are identified with (with the top of the stack to the left). We will use the edges labeled $\#_j$ to model the operations push_j and pop_j for $j \geq 2$.

Proposition 3.11 *For every $n \geq 1$, if $G \in \mathrm{HOPDG}(n)$ then $G \in \mathrm{Graph}(n)$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \Gamma, q_i, \Delta) \in \mathrm{HOPDS}(n)$ and G be the graph generated by $\mathcal{A}$. Let $|\Gamma| = m$. W.l.o.g we assume that $Q \cap \Gamma = \emptyset$. As argued above we can identify every vertex of Δ_m^n with a pushdown stack of level n over Γ. To handle the state component of a configuration we apply a Q-copying operation to Δ_m^n. Furthermore we add a loop labeled $e \notin \Sigma$ to all the vertices of Δ_m^n which represent stacks with empty top level 1 stack. These are the vertices of Δ_m^n which do not have an incoming edge labeled by a symbol from Γ and therefore are MSO-definable. Thus the resulting graph $\Delta_m^n(Q)$ is obtained from Δ_m^n by an MSO-transduction and therefore $\Delta_m^n(Q) \in \mathrm{Graph}(n)$. The vertices added by the transduction to $\Delta_m^n(Q)$ are in one-to-one correspondence to the configurations of $\mathcal{A}$.

We define now formally how simulate the behavior of $\mathcal{A}$ on $\Delta_m^n(Q)$. For this we translate every transition $(q, a, \alpha, q', i) \in \Delta$ with $\alpha \in \Gamma$ into a regular expression which will be added to the definition of $h(a)$:

- For $(q, a, \alpha, q', \mathrm{push}_1^\beta) \in \Delta$ add $\bar{q}\bar{\alpha}\alpha\beta q'$ to $h(a)$.
- For $(q, a, \alpha, q', \mathrm{push}_j) \in \Delta$ add $\bar{q}\bar{\alpha}\alpha\#_j q'$ to $h(a)$.
- For $(q, a, \alpha, q', \mathrm{pop}_1) \in \Delta$ add $\bar{q}\bar{\alpha}q'$ to $h(a)$.
- For $(q, a, \alpha, q', \mathrm{pop}_j) \in \Delta$ add $\bar{q}\bar{\alpha}(\Gamma + \bar{\Gamma} + \{\sharp_k, \bar{\sharp}_k \mid k < j\})^*\bar{\#}_j q'$ to $h(a)$.

To handle the transitions of $\mathcal{A}$ on an empty top level 1 stack, i.e. for $\alpha = \varepsilon$, we have to replace every occurrence of α and $\bar{\alpha}$ in the regular expression on the right hand side of the rules above by e. Since transitions with pop_1 instructions are not applicable on higher-order stacks with empty top level 1 stack these have to be ignored.

It follows directly from the construction that for every $a \in \Sigma \cup \{\varepsilon\}$ and all configurations (q, s), (q', s') it holds that $\mathcal{A}$ can reach (q', s') from (q, s) via an edge labeled by $a \in \Sigma \cup \{\varepsilon\}$ iff for the vertices v_1, v_2 representing (q, s) respectively (q', s') we have $v_1 \xrightarrow{h(a)} v_2$ in $\Delta_m^n(Q)$.

We now have to restrict $h^{-1}(\Delta_m^n(Q))$ to the vertices which represent configurations which can be reached from the initial configuration $(q_i, [\varepsilon]^n)$. This set of vertices can be characterized as the smallest set which contains $(q_i, [\varepsilon]^n)$ and which is closed under $(\Sigma \cup \{\varepsilon\})$-successors. $(q_i, [\varepsilon]^n)$ is definable as the q_i successor of the only vertex w which has the following properties:

- w has no incoming edge labeled by a symbol from Γ.
- From w no backward edge labeled $\#_j$ for some j can be reached via a path which traverses only Γ-labeled edges (in either direction).

Thus there exists an MSO-marking $\mathcal{M}_\&$ which marks the set of configurations reachable from $(q_i, [\varepsilon]^n)$.

To finish the proof we have to compute the ε-closure of the graph restricted to vertices marked by $\&$. We use another MSO-marking $\mathcal{M}_\$$ to mark every vertex which has no outgoing ε-edges, i.e. those vertices which are not removed due to the ε-closure (Note that $h^{-1}(\Delta_m^n(Q))$

contains ε as a usual edge label.). Then we apply the inverse rational mapping h_ε defined by $h_\varepsilon(a) = \&\bar{\&}a\varepsilon^*\$\bar{\$}$ for every $a \in \Sigma$ to $(\mathcal{M}_\$ \circ \mathcal{M}_\&)(h^{-1}(\Delta^n_m(Q)))$ to obtain the ε-closure of the configuration graph of $\mathcal{A}$, i.e to obtain the graph G generated by $\mathcal{A}$. □

3.4 From Caucal Graphs to HOPDG

We now turn to the converse direction. We show that every graph G on level n of the Caucal hierarchy is generated by a higher-order pushdown system of level n.

To achieve this result some preliminary work is necessary. First we introduce a variant of the higher-order pushdown systems where, for every k, a stack of level k can only be popped if the two top level k stacks coincide. In Subsection 3.4.1 we show that these automata with *equality pop*[1] generate the same classes of higher-order pushdown graphs as the standard model.

With this strengthened model we can proceed towards the main result. In Subsection 3.4.2 we show how to construct from a higher-order pushdown system $\mathcal{A}$ of level n a higher-order pushdown system of level $n+1$ which generates the unfolding of the graph generated by $\mathcal{A}$ from a certain vertex.

In Section 3.4.3 we finally show that the application of an unfolding followed by an inverse rational mapping to a higher-order pushdown graph of level n yields a higher-order pushdown graph of level $n+1$.

3.4.1 HOPDS with Equality Pop

We now introduce a variant of the higher-order pushdown systems by extending the pop_k operations with a built-in equality test: for $k > 1$ the operation $\mathrm{pop}^=_k$ can only be applied if the two top level k stacks coincide.

Let $[s_r, \ldots, s_1]$ be a stack of level n. We define the operations $\mathrm{pop}^=_k$

[1] In [CW03] we called these automata “weak-popping”, a term which suggests a weaker popping operation, not one strengthened with a built in equality test.

for $1 \leq k \leq n$ by setting

$$\begin{aligned}
\mathrm{pop}_n^=([s_r, s_{r-1}, \ldots, s_1]) &:= [s_{r-1}, \ldots, s_r] \text{ if } s_r = s_{r-1} \text{ and } n > 1, \\
\mathrm{pop}_k^=([s_r, \ldots, s_1]) &:= [\mathrm{pop}_k^=(s_r), s_{r-1}, \ldots, s_1] \text{ for } 1 < k < n, \\
\mathrm{pop}_1^=([s_r, \ldots, s_1]) &:= [\mathrm{pop}_1(s_r), s_{r-1}, \ldots, s_1].
\end{aligned}$$

Otherwise $\mathrm{pop}_k^=$ is undefined. A *higher-order pushdown system with equality pop* of level n is a higher-order pushdown system of level n where in Instr_n the instruction pop_k is replaced by $\mathrm{pop}_k^=$ for $1 \leq k \leq n$. We denote this set of instructions by $\mathrm{Instr}_n^=$.

We show that the higher-order pushdown systems with equality pop generate the same classes of graphs as the standard model. The idea how to simulate a pop instruction by $\mathrm{pop}^=$ instructions is straightforward. The automaton with equality pop operation just guesses the correct stack content to be able to apply $\mathrm{pop}^=$.

For a pop_2 instruction this amounts to emptying the top-most stack of level 1 using pop_1 instructions and nondeterministically recreating the stack content using push_1 instructions to apply $\mathrm{pop}_2^=$. For a pop_3 instruction the automaton empties the top-most stack of level 2 and recreates nondeterministically the correct content of this level 2 stack using the simulation of pop_2 by $\mathrm{pop}_2^=$ both during emptying the stack and recreation of the correct content.

Figure 3.6 shows an example sequence to enable a $\mathrm{pop}_3^=$ instruction. First letter c is pushed onto the top level 1 stack to be able to apply $\mathrm{pop}_2^=$. Then the top level 1 stack (and also the top level 2 stack) is emptied using level 1 instructions. $[abc]$ is recreated, pushed and, again using level 1 instructions, the level 2 stack is finally recreated to be able to apply $\mathrm{pop}_3^=$.

A formal description of the transitions needed to allow the simulation of pop_k operations by $\mathrm{pop}_k^=$ operations is given in the proof of the following proposition.

Proposition 3.12 *If $\mathcal{A}$ is a higher-order pushdown system of level n then there exists a higher-order pushdown system $\mathcal{B}$ with equality pop of level n such that $\mathcal{A}$ and $\mathcal{B}$ generate the same graph.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta)$ be a (standard) higher-order pushdown system of level n. We will construct a higher-order pushdown system $\mathcal{B}$ with equality pop from $\mathcal{A}$ by replacing every transition $\delta =$

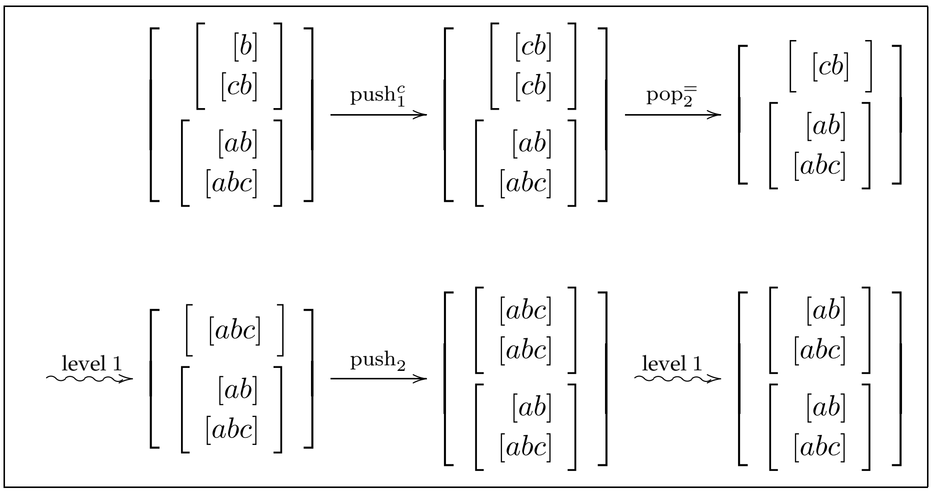


Figure 3.6: Recreation of a stack to apply $\mathrm{pop}_3^=$

$(q, a, \alpha, q', i) \in \Delta_{\mathcal{A}}$ with $i = \mathrm{pop}_k$ for some $1 \leq k \leq n$ with a set $[\delta]$ of transitions with only $\mathrm{pop}^=$ instructions.

Let $\delta = (q, a, \alpha, q', \mathrm{pop}_k) \in \Delta_{\mathcal{A}}$. We proceed by induction on the level k of the pop instruction. For $k = 1$ $\mathrm{pop}_1^=$ is defined as pop_1 and there is nothing to show. For $k = 2$ and $\alpha \neq \varepsilon$ we define

$$\begin{aligned} [(q, a, \alpha, q', \mathrm{pop}_2)] := {} & (q, \varepsilon, \alpha, p, \mathrm{pop}_1^=) && (3.1)\\ & \cup \{(p, \varepsilon, \beta, p, \mathrm{pop}_1^=) \mid \beta \in \Gamma\} && (3.2)\\ & \cup \{(p, \varepsilon, \beta, p, \mathrm{push}_1^\gamma) \mid \gamma \in \Gamma,\ \beta \in \Gamma \cup \{\varepsilon\}\} && (3.3)\\ & \cup \{(p, a, \beta, q', \mathrm{pop}_2^=) \mid \beta \in \Gamma \cup \{\varepsilon\}\} && (3.4) \end{aligned}$$

where p is a new state for every transition δ. The transition in Line 3.1 checks whether α is the top level 1 stack symbol and removes it, the transitions in Line 3.2 enable the pushdown system to empty the top level 1 stack, while the transitions in Line 3.3 allow to recreate the correct content of this stack to enable an a-labeled transition of Line 3.4 to remove the top level 1 stack completely.

If $\alpha = \varepsilon$, i.e. the transition δ is applied to remove an empty top level 1 stack we have to replace Lines 3.1 and 3.2 by a single transition $(q, a, \varepsilon, p, -)$ which allows the pushdown system to change its state to p without changing the stack content.

It is easy to see that the construction ensures that from a configuration (q, s) the configuration (q', s') can be reached via the transition $\delta = (q, a, \alpha, q', \mathrm{pop}_2)$ iff there is a sequence of transitions in $[\delta]$ such that

(q', s') is reached from (q, s) via a path labeled by a.

We generalize this idea of first emptying a stack and then recreating a correct stack content to simulate higher-order pop instructions by $\mathrm{pop}^=$ instructions.

Let $k > 2$ and assume that $[\delta]$ has been defined for all transitions δ with pop_j instructions for $j < k$. We set

$$\begin{aligned}
[(q, a, \alpha, q', \mathrm{pop}_k)] := {} & [(q, \varepsilon, \alpha, p, \mathrm{pop}_{k-1})] && (3.5)\\
& \cup \bigcup_{\substack{\beta\in\Gamma\cup\{\varepsilon\}\\ j<k}} [(p, \varepsilon, \beta, p, \mathrm{pop}_j)] && (3.6)\\
& \cup \{(p, \varepsilon, \beta, p, \mathrm{push}_j) \mid \beta \in \Gamma \cup \{\varepsilon\}, j < k\} && (3.7)\\
& \cup \{(p, \varepsilon, \beta, p, \mathrm{push}_1^\gamma) \mid \gamma \in \Gamma,\ \beta \in \Gamma \cup \{\varepsilon\}\} && (3.8)\\
& \cup \{(p, a, \beta, q', \mathrm{pop}_k^=) \mid \beta \in \Gamma \cup \{\varepsilon\}\} && (3.9)
\end{aligned}$$

and assume that all states added during the recursive construction are new.

The transitions in Lines 3.5 and 3.6 allow to empty the top stack of level $k-1$, the transitions in Lines 3.6, 3.7 and 3.8 enable the pushdown system to recreate the correct stack content to apply one of the transitions of Line 3.9 to delete the topmost level k stack.

If $[\delta]$ is extended to be the identity function on transitions δ without pop operations we can define the higher-order pushdown system $\mathcal{B}$ with equality pop which generates the same graph as $\mathcal{A}$ by setting $\mathcal{B} := (\bar{Q}, \Sigma, \Gamma, q_0, \bar{\Delta})$ where $\bar{\Delta} := \bigcup_{\delta\in\Delta}[\delta]$ and $\bar{Q}$ contains Q and all the states which have to be added during the construction of $\bar{\Delta}$. $\square$

The simulation of a higher-order pushdown system with equality pop instructions by one with the usual pop instructions is slightly more complicated. We have to ensure that the transitions which simulate a $\mathrm{pop}_k^=$ instruction can only be applied if the two top level k stacks coincide.

For this purpose we will enrich the stack alphabet with new symbols which state which instructions have to be executed to recreate a previous stack content. Such an encoding is only possible due to the simple structure of $\mathrm{Instr}_n^=$ which ensures that for every stack s of level n there is a unique shortest sequence of instructions which creates s from $[\varepsilon]^n$. Before we proceed to the proof of this proposition we collect some necessary facts in a series of lemmas.

We assume w.l.o.g. that $\mathrm{Instr}_n^=$ does not contain the identity instruction $-$. From the definition of $\mathrm{Instr}_n^=$ we then can derive the following: Firstly, every instruction $i \in \mathrm{Instr}_n^=$ is injective, i.e. if i is applicable to stacks s and t and $s \neq t$ then $i(s) \neq i(t)$. Secondly, instructions define distinct operations on the stack, i.e. if $i, j \in \mathrm{Instr}_n^=$, $i \neq j$ and both are applicable to a stack s then $i(s) \neq j(s)$.

Furthermore the restriction to equality pop-instructions allows the definition of inverse instructions. Let $i \in \mathrm{Instr}_n^=$. By i^{-1} we denote the *inverse instruction* of i, i.e. $(\mathrm{pop}_k^=)^{-1} := \mathrm{push}_k$ and $\mathrm{push}_k^{-1} := \mathrm{pop}_k^=$ for $2 \leq k \leq n$, $(\mathrm{push}_1^\gamma)^{-1} := \mathrm{pop}_1$ and, slightly abusing notation, $\mathrm{pop}_1^{-1}(s) := \mathrm{push}_1^\gamma(s)$ if $\mathrm{top}(s) = \gamma$. Obviously we have that $i^{-1}(i(s)) = s$ for every stack s and all instructions i applicable to s. Note that with the standard definition of Instr_n there are no inverse instructions for pop_k for $k \geq 2$.

The notion of the distance of two stacks will play a crucial role in the following statements. Let s and t be stacks of level n. The *distance* m of t from s is the length of the shortest sequence $s = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \ldots \xrightarrow{i_m} s_{m+1} = t$ with $i_k \in \mathrm{Instr}_n^=$ for $1 \leq k \leq m$.

First we consider the distance of *substacks* of a stack s of level n. The substack property is inductively defined. If $n = 1$ and $s = [\gamma_k, \ldots \gamma_1]$ ($\gamma_j \in \Gamma$) then every stack $[\gamma_j, \ldots, \gamma_1]$ for $1 \leq j \leq k$ of level 1 is a substack of s. If $n > 1$ and $s = [s_k, \ldots, s_1]$ is a stack of level n then for every $1 \leq j \leq k$ the stack $[s_j, \ldots, s_1]$ is a substack (of level n) of s and every substack of s_j is a substack (of level less than n) of s. In the following lemma we identify a substack t of level less than n of s with its corresponding stack of level n.

Lemma 3.13 *If s is a stack of level n of distance m from $[\varepsilon]^n$ and $t \neq s$ is a substack of s then t is of distance less than m from $[\varepsilon]^n$.*

Proof. The proof is an easy induction on the level n of the stack s. If $n = 1$ the stated property is obvious, so assume that the lemma holds for all $n' < n$ and that $s = [s_k, \ldots, s_1]$ is a stack of level n. It suffices to show for every $1 \leq j < k$ that the substacks $[s_j, \ldots, s_1]$ of level n are of distance less than m from $[\varepsilon]^n$. The claim for substacks of level less than n then follows by the induction hypothesis. To see that the substacks $[s_j, \ldots, s_1]$ are of distance less than m from $[\varepsilon]^n$ just notice that for $1 \leq j < k$ the stack $[s_j, s_j, \ldots, s_1]$ is contained in any sequence of stacks from $[\varepsilon]^n$ to s and hence the distance of $[s_j, \ldots, s_1]$ from $[\varepsilon]^n$ is less than m. □

Now we can now show that for any stack s the shortest path generating s from $[\varepsilon]^n$ is unique.

Proposition 3.14 *For every stack s of level n the shortest path from $[\varepsilon]^n$ to s is unique.*

Proof. We show the following claim by induction on m.

Claim 1. If t is a stack of distance $m-1$ from $[\varepsilon]^n$ and $s := i(t)$ is of distance m from $[\varepsilon]^n$ then $j(s)$ is of distance $m+1$ from $[\varepsilon]^n$ for all instructions $j \in \mathrm{Instr}_n^=$, $j \neq i^{-1}$ which are applicable to s.

From this the statement of the proposition follows. Assume that s is a stack of distance m from $[\varepsilon]^n$ and there exist two distinct paths $[\varepsilon]^n = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \ldots \xrightarrow{i_m} s_{m+1} = s$ and $[\varepsilon]^n = t_1 \xrightarrow{j_1} t_2 \xrightarrow{j_2} \ldots \xrightarrow{j_m} t_{m+1} = s$ of length m from $[\varepsilon]^n$ to s. Then there exists a $1 \leq k \leq m$ such that $s_{k+1} = t_{k+1}$ but $s_k \neq t_k$, in particular we have that $i_k(s_k) = s_{k+1} = t_{k+1} = j_k(t_k)$ for instructions $i_k \neq j_k$. Thus $j_k^{-1} \neq i_k^{-1}$ and hence by Claim 1 we have that $j_k^{-1}(i(s_k)) = t_k$ is of distance k from $[\varepsilon]^n$ which contradicts that $[\varepsilon]^n = t_1 \xrightarrow{j_1} t_2 \xrightarrow{j_2} \ldots \xrightarrow{j_{k-1}} t_k$, i.e. t_k is of distance $k-1$ from $[\varepsilon]^n$.

Thus it suffices to show Claim 1 for stacks s of distance m from $[\varepsilon]^n$. For this we need the property that the application of a non-inverse instruction to stacks of distance m from $[\varepsilon]^n$ leads to distinct stacks (of distance $m+1$ from $[\varepsilon]^n$). This property is formalized in Claim 2 below. The proofs of the Claim 1, Claim 2 and the proposition are nested: the statement of Claim 1 for stacks s of distance m from $[\varepsilon]^n$ relies on the statement of Claim 2 on stacks s and t of distance m from $[\varepsilon]^n$ which again relies on the statement of the proposition for stacks of distance $m' < m$ from $[\varepsilon]^n$.

Claim 2. Let $m > 0$, s', t' be stacks of level n of distance $m-1$ from $[\varepsilon]^n$ and $i', j' \in \mathrm{Instr}_n^=$ such that $s := i'(s')$, $t := j'(t')$ are both of distance m from $[\varepsilon]^n$ and $s \neq t$. Then for all $i, j \in \mathrm{Instr}_n^=$ with $i \neq (i')^{-1}$, $j \neq (j')^{-1}$ which are applicable to s respectively t we have $i(s) \neq t$ and $i(s) \neq j(t)$.

Proof of Claim 1. Let t be stack of distance $m-1$ from $[\varepsilon]^n$ and $s := i(t)$ be of distance m from $[\varepsilon]^n$. Let $j \in \mathrm{Instr}_n^=$, $j \neq i^{-1}$ be applicable to s and assume that $j(s)$ is of distance $k < m+1$ from $[\varepsilon]^n$. Applying Claim 2 to $j(s)$ and s we can conclude that $j(s)$ is not of distance m from $[\varepsilon]^n$. Thus two cases remain to be considered. If $k < m-1$ then $s = j^{-1}(j(s))$ is of distance $< k+1 < m$ which contradicts that s is of distance m from $[\varepsilon]^n$. If $k = m-1$, i.e. t and $j(s)$ have the same distance from $[\varepsilon]^n$, then the equation $j^{-1}(j(s)) = s = i(t)$ contradicts Claim 2 since $j(s) = j(i(t)) \neq i^{-1}(i(t))$.

Proof of Claim 2. We proceed by induction on the length m of the shortest paths to s respectively t.

If $m = 1$, $s := i'([\varepsilon]^n)$, $t := j'([\varepsilon]^n)$ and $s \neq t$ we know that $i' \neq j'$ and that the paths of length one from $[\varepsilon]^n$ to s respectively t are unique. Let $i \in \mathrm{Instr}_n^=$, $i \neq (i')^{-1}$ be applicable to s and $j \in \mathrm{Instr}_n^=$, $j \neq (j')^{-1}$ be applicable to t. Since different instructions applied to a stack lead to different stacks and $i \neq (i')^{-1}$, $j \neq (j')^{-1}$ we know that $i(s) \neq [\varepsilon]^n$ and $j(t) \neq [\varepsilon]^n$. It is easy to verify by a case distinction that $i(s) \neq j(t)$ for all $i, j \in \mathrm{Instr}_n^=$ which are applicable to s respectively t and that all these stacks are of distance two from $[\varepsilon]^n$.

Now let $m > 1$ and assume that Proposition 3.14, Claim 1 and Claim 2 hold for all $m' < m$. Let s', t' be stacks of level n of distance $m-1$ from $[\varepsilon]^n$ and $i', j' \in \mathrm{Instr}_n^=$ be instructions such that $s := i'(s')$ and $t := j'(t')$ are both of distance m from $[\varepsilon]^n$ and $s \neq t$. Let $i, j \in \mathrm{Instr}_n^=$, $i \neq (i')^{-1}$, $j \neq (j')^{-1}$ be applicable to s respectively t. Let $s = [s_k, \ldots, s_1]$ and $t = [t_l, \ldots, t_1]$ where $k, l \geq 1$ and $s_k, \ldots, s_1$ as well as $t_l, \ldots, t_1$ are stacks of level $n-1$.

We first show that $i(s) \neq t$. Assume $i(s) = t$. Since both $i(s)$ and t are of distance m from $[\varepsilon]^n$ we can conclude that $i \neq \mathrm{push}_n$ and $i \neq \mathrm{pop}_n^=$. Hence it follows that $k = l$ and it suffices to consider the top level $n-1$ stacks s_k of s and t_k of t. Let s_k be of distance m_1 from $[\varepsilon]^{n-1}$ and t_k be of distance m_2 from $[\varepsilon]^{n-1}$. Applying Claim 1 we can conclude that m_1 differs from m_2 by one. If $m_1 = m_2 + 1$ (respectively $m_2 = m_1 + 1$) then applying the proposition we can conclude that t_k (respectively s_k) is contained in the unique shortest path from $[\varepsilon]^{n-1}$ to s_k (t_k). This contradicts that both s and t have the same distance from $[\varepsilon]^n$.

It remains to show that $i(s) \neq j(t)$. We first show that both $i, j \neq \mathrm{pop}_n^=$. Assume that w.l.o.g. $i = \mathrm{pop}_n^=$ is applicable to s. Then $s_k = s_{k-1}$. Since $i \neq (i')^{-1}$, i.e. $i' \neq \mathrm{push}_n$, we know that $s' = [s_{k-1}, s_{k-1}, s_{k-1}, \ldots, s_1]$

or $s' = [s'_k, s_{k-1}, \ldots, s_1]$ for some level $n-1$ stack s'_k. In both cases $i(s) = [s_{k-1}, \ldots, s_1]$ is a substack of s' and hence by Lemma 3.13 of distance less than $m-1$ from $[\varepsilon]^n$. Thus the distance of $i^{-1}(i(s)) = s$ is less than m from $[\varepsilon]^n$ which contradicts the assumption that s is of distance m from $[\varepsilon]^n$.

Now assume that $i(s) = j(t)$ and that none of the instructions i, j is $\mathrm{pop}_n^=$. Then we know k and l differ at most by one. We first consider the case that k and l differ by one and assume w.l.o.g. $l = k+1$. Then i has to be push_n and $j \in \mathrm{Instr}_{n-1}^=$. Thus $s = [s_k, \ldots, s_1] = [t_{l-1}, \ldots, t_1]$ which is a substack of t. Hence by Lemma 3.13 the stack s is of distance less than m from $[\varepsilon]^n$ which again contradicts the assumption that s is of distance m from $[\varepsilon]^n$.

Now let $k = l$. If $s_p \neq t_p$ for some $1 \leq p < k$ then $i(s) \neq j(t)$ for all $i, j \in \mathrm{Instr}_n^=$. So assume that $s_p = t_p$ for $1 \leq p < k$ and $s_k \neq t_k$. Since $i, j \neq \mathrm{pop}_n^=$ the only instruction of level n remaining is push_n. If both $i, j = \mathrm{push}_n$ then $i(s) \neq j(s)$ because $s_k \neq t_k$. If only one of the instructions i, j is push_n and the other one is an instruction of level less than n, say $j \in \mathrm{Instr}_{n-1}^=$, then $i(s)$ contains one more stack of level $n-1$ than $j(t)$ and hence $i(s) \neq j(t)$.

Thus we know that under the assumption that $i(s) = j(t)$ and $k = l$ the instructions $i, j \in \mathrm{Instr}_{n-1}^=$ and hence it suffices to consider the top level $n-1$ stacks s_k of s and t_k of t. By Lemma 3.13 we know that s_k is of distance $m_1 < m$ from $[\varepsilon]^{n-1}$ and t_k is of distance $m_2 < m$ from $[\varepsilon]^{n-1}$. Since $i(s) = j(t)$, $i, j \in \mathrm{Instr}_{n-1}^=$ and therefore $i(s_k) = j(t_k)$ we know by Claim 1 that m_1 and m_2 differ at most by two. Thus we have to consider three cases:

Case 1. If $m_1 = m_2$ the induction hypothesis applies contradicting the assumption that $i(s) = j(t)$.

Case 2. If w.l.o.g. $m_2 = m_1 + 1$ then $i(s_k)$ is by Claim 1 (applied with $m_1 + 1 < m$) of distance $m_1 + 1 = m_2$ or distance $m_1 - 1$ from $[\varepsilon]^{n-1}$ and $j(t_k)$ is of distance $m_2 + 1 > m_1 + 1$ or $m_2 - 1 = m_1$ from $[\varepsilon]^{n-1}$. That is $i(s_k)$ and $j(s_k)$ do not have the same distance from $[\varepsilon]^{n-1}$ and thus $i(s_k) \neq j(t_k)$ which implies $i(s) \neq j(t)$.

Case 3. If w.l.o.g. $m_2 = m_1 + 2$ then we know that $i(s_k) = j(t_k)$ is of distance $m_1 + 1$ from $[\varepsilon]^{n-1}$ Since $[s_{k-1}, s_{k-1}, \ldots .s_1] = [t_{k-1}, t_{k-1}, \ldots, t_1]$ and this stack occurs on any shortest path from $[\varepsilon]^n$ to $[s_k, s_{k-1}, \ldots .s_1]$ respectively $[t_k, t_{k-1}, \ldots, t_1]$ we know that the distance of s_k from s_{k-1} is the same as the distance of t_k from $t_{k-1} = s_{k-1}$, say d. Let $n_1 < m$ be the distance of s_{k-1} from $[\varepsilon]^{n-1}$. If $n_1 \leq m_1$ then by the induction hypothesis on the existence of unique shortest generating paths for stacks of distance less than m the stack s_{k-1} occurs on both unique shortest paths from $[\varepsilon]^{n-1}$ to s_k and t_k and hence $m_1 - d = n_1$ and $m_2 - d = n_1$ contradicting the assumption that $m_2 = m_1 + 2$. If $n_1 \geq m_2$ the again by the induction hypothesis both t_k and s_k occur on the unique shortest path from $[\varepsilon]^{n-1}$ to s_{k-1} which contains only a single stack of distance d from s_{k-1} and hence $t_k = s_k$. If $n_1 = m_1 + 1$ then $j(t_k) = s_{k-1} = t_{k-1}$ and thus $j(t) = [t_{k-1}, t_{k-1}, \ldots t_1]$ which implies that $j = (j')^{-1}$.

This finishes the proof of Claim 2 and thus also the proof of Proposition 3.14. □

Note that there are no unique shortest paths if we allow to use standard pop_k instructions, see Figure 3.7 for an example. This example also shows that a stack may have many predecessors of varying distance from $[\varepsilon]^n$.

$$\begin{bmatrix} [abcd] \\ [cd] \end{bmatrix} \xrightarrow{\mathrm{pop}_1} \begin{bmatrix} [bcd] \\ [cd] \end{bmatrix} \xrightarrow{\mathrm{pop}_1} \begin{bmatrix} [cd] \\ [cd] \end{bmatrix}$$

$$\begin{bmatrix} [abcd] \\ [cd] \end{bmatrix} \xrightarrow{\mathrm{pop}_2} [[cd]] \xrightarrow{\mathrm{push}_2} \begin{bmatrix} [cd] \\ [cd] \end{bmatrix}$$

Figure 3.7: Multiple shortest paths with pop_2

From Proposition 3.14 can conclude that there exist unique shortest paths between any two stacks s and t. If $[\varepsilon]^n = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \ldots \xrightarrow{i_{k-1}} s_k = s$ and $[\varepsilon]^n = t_1 \xrightarrow{j_1} t_2 \xrightarrow{j_2} \ldots \xrightarrow{j_{l-1}} t_l = t$ are the unique shortest paths from $[\varepsilon]^n$ to s respectively to t and m is chosen maximal such that $i_p = j_p$ for every $p \leq m$ then the shortest path from s to t is given by

$$s = s_k \xrightarrow{i_{k-1}^{-1}} \ldots \xrightarrow{i_{m+1}^{-1}} (s_m = t_m) \xrightarrow{j_m} \ldots \xrightarrow{j_l} t_l = t.$$

We call s_m the *shortest common prefix* of s and t.

The previous results now allow us to show that on every path from a stack s to itself every instruction has to be reversed, i.e. the only non-trivial paths from s to s consist of possibly nested loops. We formalize this with the notion of reversal paths.

Let $s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \dots s_m \xrightarrow{i_m} s_{m+1}$ a sequence of stacks obtained from s_1 by applying $i_1, \dots, i_m$. We call this sequence a *reversal path* if either

(a) $i_m = i_1^{-1}$ and $m = 2$ or $s_2 \xrightarrow{i_2} \dots \xrightarrow{i_m} s_{m+1}$ is a reversal path or

(b) there exists a $1 < k < m$ such that $s_1 \xrightarrow{i_1} \dots \xrightarrow{i_{k-1}} s_k$ and $s_k \xrightarrow{i_k} \dots \xrightarrow{i_m} s_{m+1}$ are both reversal paths.

Lemma 3.15 *If $s = s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \dots \xrightarrow{i_m} s_{m+1} = s$ then this sequence is a reversal path.*

Proof. We proceed by induction on the length m of the sequence. For $m = 0$ no such sequence exists. Let $m > 0$. If there exists $1 \leq k < l \leq m$ with $s_k = s_l$ we know that by induction hypothesis that both $s_1 \xrightarrow{i_1} \dots \xrightarrow{i_k} s_k = s_l \xrightarrow{i_l} \dots \xrightarrow{i_m} s_{m+1}$ as well as $s_k \xrightarrow{i_k} \dots \xrightarrow{i_{l-1}} s_l$ are reversal paths and hence $s_1 \xrightarrow{i_1} \dots \xrightarrow{i_m} s_{m+1}$ is a reversal path too.

On the other hand we know by Claim 1 stated in the proof of Proposition 3.14 that the distance of s_k from $[\varepsilon]^n$ differs by one from the distance of s_{k+1} from $[\varepsilon]^n$. So if $s_k \neq s_l$ for $1 \leq k < l \leq m$ and hence $i_{k+1} \neq i_k^{-1}$ there exists a $1 \leq k \leq m$ and a d such that s_k has distance d from $[\varepsilon]^n$ and both s_{k-1} and s_{k+1} have distance $d+1$ from $[\varepsilon]^n$ or both have distance $d-1$ from $[\varepsilon]^n$. Since $s_{k+1} \neq s_{k-1}$ the case that both are of distance $d-1$ from $[\varepsilon]^n$ is ruled out by the uniqueness of shortest generating paths. Thus again by Lemma 3.14 (applied several times) we obtain that $s_1 \neq s_{m+1}$ which contradicts the assumption. □

The next lemma shows that every path from s to t differs from the shortest path from s to t only by trivial deviations, i.e. all deviations are reversal paths.

Lemma 3.16 *Let s and t be stacks of level n and $s = s_1 \xrightarrow{i_1} \dots \xrightarrow{i_m} s_{m+1} = t$ be the unique shortest path between s and t. Let $s = t_1 \xrightarrow{j_1} \dots \xrightarrow{j_n} t_{n+1} = t$ be another path between s and t. Then there exists a subsequence $(l_k)_{1 \leq k \leq m}$ such that $j_{l_k} = i_k$ and $t_{l_k} \xrightarrow{j_{l_k}} \dots \xrightarrow{j_{l_{k+1}-1}} t_{l_{k+1}}$ is a reversal path for every $1 \leq k \leq m-1$.*

Proof. By definition $n \geq m$. If $n = m$ then we know by Proposition 3.14 that the two paths coincide, hence it remains to consider the case $n > m$.

Consider the path $s = t_1 \xrightarrow{j_1} \dots \xrightarrow{j_n} t_{n+1} = t$. If this path contains a stack twice, say $t_p = t_q$ for $1 \leq p < q \leq n+1$ we know by Lemma 3.15 that the path from t_p to t_q is a reversal path and hence it suffices to consider $t_1 \xrightarrow{j_1} \dots \xrightarrow{j_{p-1}} t_p \xrightarrow{j_q} t_{q+1} \xrightarrow{j_{q+1}} \dots \xrightarrow{j_n} t_{n+1}$. We can continue to cut reversal paths in this manner until for the resulting path either $n = m$ or $t_p \neq t_q$ for all $1 \leq p < q \leq n$. In the second case we obtain that $j_{p+1} \neq j_p^{-1}$ for $1 \leq p \leq n$. Since $n > m$ we know that the distance of the stacks t_p from $[\varepsilon]^n$ is neither strictly increasing nor strictly decreasing, hence similar to the argument presented in the proof of Lemma 3.15, there exist a $1 < p < n+1$ such that t_{p-1} and t_{p+1} have the same distance d from $[\varepsilon]^n$ and t_p is of distance $d-1$ from $[\varepsilon]^n$. Since all stacks on the path from s to t are different this implies that t_p is indeed the longest common prefix of s and t. Proposition 3.14 then implies that the distance of the stacks $t_p, t_{p_1}, \dots, t_1 = s$ as well as $t_p, t_{p+1}, \dots t_n = t$ from $[\varepsilon]^n$ is strictly increasing. This however contradicts $n > m$. □

We can now define an encoding of a higher-order stack from which we can extract whether the two top level k stacks coincide. The encoding will record for every $2 \leq k \leq n$ the last change to the top stack of level k which is still persistent. This *last persistent change* is the effect of the application of the last instruction in the unique shortest sequence from $[\varepsilon]^k$ to the current top level k stack. Lemma 3.16 ensures that we can update this information correctly.

We will use the following additional stack symbols for $2 \leq k \leq n$, $2 \leq j < k$ and $\gamma \in \Gamma$, on the right hand side we denote their intended meanings:

$(k, +\gamma)$	the last persistent change of level k was to add γ ,
$(k, -\gamma)$	the last persistent change of level k was to remove γ ,
$(k, +j)$	the last persistent change of level k was a push_j ,
$(k, -j)$	the last persistent change of level k was a $\mathrm{pop}_j^=$,
(k, ε)	the last persistent change of level k was push_k .

In the encoding every stack of level one will be headed by an $(n-1)$-tuple $(n, *)(n-1, *) \dots (2, *)$. The empty stack of level n is encoded by $[(n, \varepsilon) \dots (2, \varepsilon)]^n$.

To describe the updates on a stack when an instruction is applied we need some more notation. Let s be a stack of level n. We denote by $s[l]$ the top stack of level l of s. Let $s[1] := [m_n \dots m_2 \gamma_r \dots \gamma_1]$ with $m_k = (k, *)$ for $2 \leq k \leq n$ and $\gamma_k \in \Gamma$ for $1 \leq k \leq r$ be the top level one stack of s. We call $m_n \dots m_2$ the *head* of $s[1]$ and $\gamma_k \dots \gamma_1$ the *tail* of $s[1]$ and denote them by $\mathrm{head}(s)$ respectively $\mathrm{tail}(s)$. By $s\{m_k := (k, *)\}$ we denote the stack which is obtained from s by replacing m_k with $(k, *)$ in the head of $s[1]$. Similarly $s\{\mathrm{tail} := \beta_r \dots \beta_1\}$ denotes the stack which is obtained by replacing in s the tail of $s[1]$ with $\beta_r \dots \beta_1$.

The algorithm which updates the encoding of a stack when an instruction is applied is given in Figure 3.8. The input consists of three parts, the stack s, the level l on which the encoding still has to be updated and the instruction i for which the update is requested.

The encoding of a stack s of level n is defined via its shortest generating sequence $[\varepsilon]^n = s_1 \xrightarrow{i_1} \dots \xrightarrow{i_m} s_m = s$. It is the stack $c(s)$ computed by

$$\mathrm{update}_n(\dots \mathrm{update}_n(\mathrm{update}_n([(n,\varepsilon)\dots(2,\varepsilon)]^n, n, i_1), n, i_2)\dots, n, i_m).$$

Figure 3.9 shows an example how the encoding of a level 3 stack evolves. The labels at the arrows depict the instruction and the level with which the algorithm is called on the stack at the source of the arrow. The call of $\mathrm{update}([(3,\varepsilon)(2,\varepsilon)]^3, 3, \mathrm{push}_1^a)$ first results in the right hand side of the first line. The top level 2 stack is copied and the mark $(3,\varepsilon)$ of the copy is replaced by $(3,+a)$. Then update is called for level 2 on this stack, the top level 1 stack is copied and the mark $(2,\varepsilon)$ is replaced by $(2,+a)$. Finally a is added to the tail of the top level 1 stack. This stack of the right hand side of line 2 is the result of $\mathrm{update}([(3,\varepsilon)(2,\varepsilon)]^3, 3, \mathrm{push}_1^a)$. The first stack of line 3 shows the final result of the update algorithm applied to the previous stack for $i = \mathrm{push}_1^b$, and the right hand side of line 3 shows the result after applying push_3. Note that in this case there is no recursive call to the update algorithm. The last line finally shows the evolution of the encoding for $i = \mathrm{pop}_1$. First the level 2 stack is copied and the mark $(3,\varepsilon)$ is replaced by $(3,-b)$, then the level 1 stack is popped, keeping the mark $(3,-b)$ on the top level 1 stack.

Lemma 3.17 *The algorithm depicted in Figure 3.8 is correct, i.e. for every instruction $i \in \mathrm{Instr}_n^=$ and every stack s of level n we have $c(i(s)) = \mathrm{update}(c(s), n, i)$. In particular, $\mathrm{update}(c(s), n, i)$ returns an encoding of a stack if, and only if, i can be applied to s.*

```
**Algorithm** update($s, l, i$):

let $m'_n, \dots, m'_{l+1}$ be the first $n-l$ symbols of $s[1]$
case
  $i = \mathrm{push}_1^\beta$ :
     if $l = 1$ then return $s\{\mathrm{tail} := \beta\,\mathrm{tail}(s)\}$
     else if $m_l = (l, -\beta)$ then
               return $(\mathrm{pop}_l(s))\{m_n \dots m_{l+1} := m'_n \dots m'_{l+1}\}$
     else $s' := (\mathrm{push}_l(s))\{m_l := (l, +\beta)\}$
          return update($s', l-1, i$)
  $i = \mathrm{pop}_1^=$ :
     $\gamma := \mathrm{top}(\mathrm{tail}(s))$
     if $l = 1$ then return $s\{\mathrm{tail} := \mathrm{pop}_1(\mathrm{tail}(s))\}$
     else if $m_l = (l, +\gamma)$ then
               return $(\mathrm{pop}_l(s))\{m_n \dots m_{l+1} := m'_n \dots m'_{l+1}\}$
     else $s' := (\mathrm{push}_l(s))\{m_l := (l, -\gamma)\}$
          return update($s', l-1, i$)
  $i = \mathrm{push}_k$ :
     if $k = l$ then return $(\mathrm{push}_k(s))\{m_k := (k, \varepsilon)\}$
     else if $m_l = (l, -k)$ then
               return $(\mathrm{pop}_l(s))\{m_n \dots m_{l+1} := m'_n \dots m'_{l+1}\}$
     else $s' := (\mathrm{push}_l(s))\{m_l := (l, +k)\}$
          return update($s', l-1, i$)
  $i = \mathrm{pop}_k^=$ :
    if $k \neq l$ then
       if $m_l = (l, +k)$ then
            return $(\mathrm{pop}_l(s))\{m_n \dots m_{l+1} := m'_n \dots m'_{l+1}\}$
       else $s' := (\mathrm{push}_l(s))\{m_l := (l, -k)\}$
            return update($s', l-1, i$)
    else if $m_k = (k, \varepsilon)$ then
            return $(\mathrm{pop}_k(s))\{m_n \dots m_{l+1} := m'_n \dots m'_{l+1}\}$
end case
```

Figure 3.8: The algorithm update(s, l, i)

$$\left[\left[[(3,\varepsilon)(2,\varepsilon)]\right]\right] \xrightarrow[3]{\mathrm{push}_1^a} \left[\begin{array}{r}\left[[(3,+a)(2,\varepsilon)]\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right]$$

$$\xrightarrow[2]{\mathrm{push}_1^a} \left[\begin{array}{r}\left[\begin{array}{r}[(3,+a)(2,+a)]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right] \xrightarrow[1]{\mathrm{push}_1^a} \left[\begin{array}{r}\left[\begin{array}{r}[(3,+a)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right]$$

$$\overset{\mathrm{push}_1^b}{\rightsquigarrow} \left[\begin{array}{r}\left[\begin{array}{r}[(3,+b)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+a)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right] \xrightarrow{\mathrm{push}_3} \left[\begin{array}{r}\left[\begin{array}{r}[(3,\varepsilon)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+b)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+a)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right]$$

$$\xrightarrow[3]{\mathrm{pop}_1} \left[\begin{array}{r}\left[\begin{array}{r}[(3,-b)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,\varepsilon)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+b)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+a)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right] \xrightarrow[2]{\mathrm{pop}_1} \left[\begin{array}{r}\left[\begin{array}{r}[(3,-b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,\varepsilon)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+b)(2,+b)ba]\\ [(3,+b)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[\begin{array}{r}[(3,+a)(2,+a)a]\\ [(3,+a)(2,\varepsilon)]\end{array}\right]\\ \left[[(3,\varepsilon)(2,\varepsilon)]\right]\end{array}\right]$$

Figure 3.9: Update sequence of an encoding of a level 3 stack

Proof. Let s be a stack of level n and $s[1] = [\gamma_r \dots \gamma_1]$ be its top level one stack. It suffices to show the invariant of the algorithm stated in the following claim:

Claim: $c(s)[1] = m_n \dots m_2 \gamma_r \dots \gamma_1$ and m_l records the effect of the last persistent instruction on level l, i.e. the effect of the last instruction of the unique shortest sequence of instructions which generates $s[l]$ from $[\varepsilon]^l$ for every $2 \leq l \leq n$.

From this it immediately follows that

$$\mathrm{update}(\mathrm{update}(c(s), n, i), n, i^{-1}) = c(s)$$

and in particular $\mathrm{update}(c(s), n, \mathrm{pop}_l^{=})$ returns an encoded stack iff the two top level l stacks of s coincide. Hence by Lemma 3.16 we can conclude that the algorithm returns for every sequence of instructions which generates s from $[\varepsilon]^n$ the correct result $c(s)$.

Proof of the claim: The claim is shown by a nested induction on the distance m of s from $[\varepsilon]^n$ (outer induction) and the level l of the stack considered (inner induction).

For $m = 0$ we have $s = [\varepsilon]^n$ and $c(s) = [(n, \varepsilon) \dots (2, \varepsilon)]$ which is obviously correct. So assume that the claim holds for all stacks of distance $m' \leq m$ from $[\varepsilon]^l$ and every level $1 \leq l \leq n$.

Let s be a stack of distance m from $[\varepsilon]^n$ and $i \in \mathrm{Instr}_n^{=}$. We now proceed by induction on the level $l = n, \dots, 1$ of the substack considered.

Let $n \geq l \geq 1$ and assume that $m_n \dots m_{l+1}$ record the effect of the last instruction of the unique shortest sequence of instructions which generates $i(s)[l]$. We have to consider the four cases distinguished in the algorithm:

Case 1. Let $i = \mathrm{push}_1^{\beta}$. First suppose that $l > 1$. If $m_l = (l, -\beta)$ the last persistent change to s on level l was to remove β. In particular the mark $(l, -\beta)$ was added after the algorithm executed a push_l instruction on a stack t. By the induction hypothesis (applied for m and l) $t[l]$ satisfies the claim for $l \geq l' \geq 1$. By executing the instruction pop_l followed by a replacement of the first $n - l$ marks the algorithm ensures the properties of the claim.

If $m_l \neq (l, -\beta)$ we know that i is not inverse to the last persistent change to s on level l. Therefore by Claim 1 stated in the proof of Proposition 3.14 $i(s[l])$ is of distance $m + 1$ from $[\varepsilon]^l$ and the effect of i is a persistent change on level l. Executing a push_l instruction and replacing the current label m_l by $(l, +\beta)$ thus ensures the correctness of the claim for $m_n, \ldots, m_l$.

Finally, if $l = 1$ then β is added to the top of the stack and by the induction hypothesis for $m + 1$ and $l \geq 2$ the claim is satisfied.

Case 2. Let $i = \mathrm{pop}_1$ and γ be the top symbol of the tail of s. For $l > 1$, similar to Case 1 we have that i is inverse to the instruction of the last persistent change of level l iff $m_l = (l, +\gamma)$. By the same arguments as above the algorithm produces a stack which satisfies the claim for $m + 1$ and every level if $m_l = (l, +\gamma)$, and which satisfies the claim for $m + 1$ up to l otherwise. Note that if the top level 1 stack is empty, by definition $\mathrm{pop}_1(\mathrm{tail}(s))$ does not return a result. Therefore the stack $s\{\mathrm{tail}(s) := \mathrm{pop}_1(\mathrm{tail}(s))\}$ is undefined and the algorithm terminates without an output.

Case 3. Let $i = \mathrm{push}_k$. If $l > k$ and $m_l = (l, -k)$ the last persistent change to the level l stack was to remove the top level k stack. Thus, as in the cases considered before, executing a pop_l instruction and replacing the first $n - l$ marks with the current values produces a stack which satisfies the claim for $m + 1$ and every level l. Analogously, if $l > k$ and $m_l \neq (l, -k)$ executing push_l and replacing m_l with $(l, +k)$ ensures the claim for $m + 1$ for every level $n, \ldots, l$.

If $k = l$ the operation push_k does not change top stacks of any level smaller than l. Thus push_k is not persistent for any of the smaller levels and therefore executing push_k and replacing m_k with (k, ε) ensures the claim for $m + 1$ and every $n \geq l \geq 1$.

Case 4. Let $i = \mathrm{pop}_k^=$. i is inverse to the last persistent change of level l iff $l > k$ and $m_l = (l, +k)$ or $l = k$ and $m_k = (k, \varepsilon)$. Thus in these cases, executing a pop_l instruction and replacing the first $n - l$ marks with the current values produces a stack which satisfies the claim.

If i is not inverse and $l > k$ the mark $(l, -k)$ is set. This is correct if the level k stack can be popped. This is ensured by testing in case $k = l$ whether the last persistent change to the level k stack was a push_k, i.e. if $m_k = (k, \varepsilon)$. If this is not the case the algorithm does not return an encoding of the stack. $\square$

The fact that the algorithm of Figure 3.8 can be implemented as a finite state program now finally allows us to construct for every higher-order pushdown system with equality pop operations an equivalent standard higher-order pushdown system, i.e. a system which generates the same graph.

Proposition 3.18 *If $\mathcal{A}$ is a higher-order pushdown system with equality pop of level n then there exists a higher-order pushdown system $\mathcal{B}$ of level n such that $\mathcal{A}$ and $\mathcal{B}$ generate the same graph.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta)$ be a higher-order pushdown system of level n with equality pop instructions. We will construct a higher-order pushdown system $\mathcal{B}$ of level n with standard pop instructions which generates the same graph as $\mathcal{A}$. $\mathcal{B}$ will work with the encoded stacks and implement the algorithm presented in Figure 3.8 to update the encoding of a stack. This is possible due to the fact that the algorithm is indeed a finite state program: To compute the update of a stack on level l only the marks $m_n, \dots, m_{l+1}$ have to be stored in the finite state control.

The initial stack for $\mathcal{B}$ is $[(n, \varepsilon) \dots (2, \varepsilon)]^n$. To check whether a transition $\delta = (q, a, \alpha, q')$ is applicable, i.e. whether the top symbol of the encoded stack is α, it suffices to copy the top level 1 stack of the encoding, remove $m_n, \dots, m_2$, test for the top symbol α and remove the copy again.

If the test succeeds the automaton then executes the finite state program defined by the algorithm for i. To ensure that the graph generated by $\mathcal{B}$ contains only edges in which the algorithm terminates, i.e. only if i was applicable to s, we have to add transitions which generate ε-loops on all intermediate states which are necessary to implement the algorithm.

Applying Lemma 3.17 we obtain that there is an a-labeled edge from (q, s) to (q', s') in the graph generated by $\mathcal{A}$ iff there is an a-labeled edge from $(q, c(s))$ to $(q', c(s'))$ in the graph generated by $\mathcal{B}$. □

Combining Proposition 3.12 and Proposition 3.18 we obtain

Theorem 3.19 *For every graph G we have that G is generated by a higher-order pushdown system of level n iff G is generated by a higher-order pushdown system with equality pop of level n.* □

3.4.2 Unfoldings of HOPDGs

Recall that the unfolding of a graph G from a vertex $r \in V^G$ is the graph whose vertices are the finite paths $ra_1v_2a_2 \dots a_{n-1}v_n$ of G originating in r, with an a-labeled edge between w and w' iff $w' = w \cdot a \cdot v$ for some $v \in V^G$.

If G is a higher-order pushdown graph of level n, every vertex of G is represented by a configuration (q, s) where q is a state of the pushdown system generating G and s is a higher-order pushdown stack. A path in a pushdown graph is therefore represented by a sequence $(q_1, s_1)a_1(q_2, s_2)a_2 \dots a_{n-1}(q_n, s_n)$. We can code such a sequence by a pair $(q_n, [s_n s'_{n-1} \dots s'_1])$ consisting of a state and a pushdown stack of level $n + 1$. Here s'_i is obtained from s_i by adding q_i and a_i to the top of the stack s_i.

Figure 3.10 shows in the first line a configuration sequence of a higher-order pushdown system of level 2. The second line shows how this path is encoded using a level 3 stack and additional stack symbols for states and letters of the input alphabet.

$$\left(q_0, \left[\,{[\alpha\beta]}\,\right]\right) \xrightarrow[a]{\text{push}_2} \left(q_1, \begin{bmatrix} {[\alpha\beta]} \\ {[\alpha\beta]} \end{bmatrix}\right) \xrightarrow[b]{\text{pop}_2} \left(q_2, \left[\,{[\alpha\beta]}\,\right]\right)$$

$$(q_0, [[[\alpha\beta]]]) \overset{\text{push}_2}{\underset{a}{\rightsquigarrow}} \left(q_1, \begin{bmatrix} \begin{bmatrix} {[\alpha\beta]} \\ {[\alpha\beta]} \end{bmatrix} \\ \begin{bmatrix} {[q_0 a\alpha\beta]} \end{bmatrix} \end{bmatrix}\right) \overset{\text{pop}_2}{\underset{b}{\rightsquigarrow}} \left(q_2, \begin{bmatrix} \begin{bmatrix} {[\alpha\beta]} \end{bmatrix} \\ \begin{bmatrix} {[q_1 b\alpha\beta]} \\ {[\alpha\beta]} \end{bmatrix} \\ \begin{bmatrix} {[q_0 a\alpha\beta]} \end{bmatrix} \end{bmatrix}\right)$$

Figure 3.10: Recording a path on the stack

We will follow this idea of encoding sequences of configurations of higher-order pushdown systems of level n by configurations of a pushdown system of level $n + 1$ in the proof of the following proposition.

Proposition 3.20 *For every* $G \in \mathrm{HOPDG}(n)$ *and* $r \in V^G$ *we have* $\mathrm{Unf}(G, r) \in \mathrm{HOPDG}(n + 1)$. *Furthermore, if G is generated by a higher-order pushdown system with equality pop then* $\mathrm{Unf}(G, r)$ *also is.*

Proof. Let $G \in \mathrm{HOPDG}(n)$ be generated by a higher-order pushdown system $\mathcal{A} = (Q, \Sigma, \Gamma, q_i, \Delta)$. We assume that Q, Σ and Γ are pairwise disjoint. Let $r = (q_0, s_0) \in V^G$.

We will construct, starting from $\mathcal{A}$ and following the ideas presented above, a higher-order pushdown system $\mathcal{B} \in \mathrm{HOPDS}(n+1)$ which generates $\mathrm{Unf}(G, r)$. To achieve this we will ensure that

(a) for every path $(q_1, s_1)a_1(q_2, s_2)a_2 \ldots a_{n-1}(q_n, s_n)$ in G which originates in r there is a configuration of $\mathcal{B}$ which codes this sequence in the way described above.

(b) for all configurations (q_1, s_1) and (q_2, s_2) of $\mathcal{A}$ with an a-labeled edge from (q_1, s_1) to (q_2, s_2) in G, for every configuration $(q_1, [s_1, t_m, \ldots, t_1])$ of $\mathcal{B}$ with top stack s_1 of level n there is be a path in the configuration graph of $\mathcal{B}$ labeled a from $(q_1, [s_1, t_m, \ldots, t_1])$ to $(q_2, [s_2, s_1', t_m, \ldots, t_1])$. Here s_1' denotes the stack obtained from s_1 by storing a and q_1 in its topmost level 1 stack.

These properties can be achieved by adding in the transition relation of $\mathcal{B}$ for every transition (q, a, α, q', i) of $\mathcal{A}$ a sequence of transitions which first adds a and q to the topmost stack, copies the complete level n stack, and then removes q and a from the copy. Formally we add for every transition $(q, a, \alpha, q', i) \in \Delta_{\mathcal{A}}$ with $a \in \Sigma$ the transitions

$$\begin{array}{ccc} (q, a, \alpha, p_1, \mathrm{push}_1^a) & (p_1, \varepsilon, a, p_2, \mathrm{push}_1^q) & (p_2, \varepsilon, q, p_3, \mathrm{push}_{n+1}) \\ (p_3, \varepsilon, q, p_4, \mathrm{pop}_1) & (p_4, \varepsilon, a, p_5, \mathrm{pop}_1) & (p_5, \varepsilon, \alpha, q', i) \end{array}$$

to $\Delta_{\mathcal{B}}$ where $p_1, \ldots, p_5$ are new states for every transition from $\Delta_{\mathcal{A}}$. To handle the ε-transitions of $\mathcal{A}$ we have to add a new symbol $e \notin \Sigma \cup \Gamma \cup Q$ to the stack alphabet of $\mathcal{B}$ and replace the upper left transition in the table above by $(q, \varepsilon, \alpha, p_1, \mathrm{push}_1^e)$.

To generate exactly the unfolding from vertex r by $\mathcal{B}$ we have to add finitely many ε-transitions which enable $\mathcal{B}$ to reach from the initial configuration $(q_i, [\varepsilon]^{n+1})$ the configuration $(q_0, [s_0])$ which codes the vertex $r = (q_0, s)$.

Note that the transitions added above do not contain pop_k instructions for $k \geq 2$. Hence $\mathcal{B}$ will be a higher-order pushdown system with equality pop if $\mathcal{A}$ was such a system. $\square$

3.4.3 Caucal Graphs as HOPDGs

With the preparatory work of the previous sections we can now prove that every graph of level n of the Caucal hierarchy is generated by a higher-order pushdown system of level n.

First we show using the results of Subsection 3.4.1 and the special structure of the higher-order pushdown system generating $\mathrm{Unf}(G, r)$ that there is a system of the same level which generates the extension of $\mathrm{Unf}(G, r)$ with reverse edges. This result is presented in Lemma 3.21.

Then we use this result to show how to construct for a pushdown graph G of level n, a vertex $r \in V^G$ and an inverse rational mapping h a higher-order pushdown system of level $n+1$ which generates $h^{-1}(\mathrm{Unf}(G, r))$.

Lemma 3.21 *For every $G \in \mathrm{HOPDG}(n)$ there is a $\mathcal{A} \in \mathrm{HOPDS}(n+1)$ which generates the extension of $\mathrm{Unf}(G, r)$ by reverse edges.*

Proof. Let G be a higher-order pushdown graph of level n and $r \in V^G$. By Theorem 3.19 we may assume that G is generated by a higher-order pushdown system $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta)$ with equality pop. Hence by Proposition 3.20 there is a higher-order pushdown system $\mathcal{A}'$ with equality pop of level $n+1$ which generates $\mathrm{Unf}(G, r)$.

We will construct from $\mathcal{A}'$ a higher-order pushdown system with equality pop which generates the extension of $\mathrm{Unf}(G, r)$ by reverse edges. Recall from the proof of Proposition 3.20 that for every transition $(q, a, \alpha, q', i) \in \Delta_{\mathcal{A}}$ applicable to stack s of level n the system $\mathcal{A}'$ changes its level $n+1$ stack with top level n stack s by adding q and a to s, copying s (without q and a) and then applying i to s. That is the resulting level $n+1$ stack looks like

$$\begin{bmatrix} i(s) \\ \mathrm{push}_1^q(\mathrm{push}_1^a(s)) \\ \vdots \end{bmatrix}$$

While executing these instructions on the stack $\mathcal{A}'$ changes its state from q to q'. Thus to reverse the transition $\delta = (q, a, \alpha, q', i) \in \Delta_{\mathcal{A}}$ we add transitions $(q', \bar{a}, \beta, p_1, i^{-1})$ for every $\beta \in \Gamma \cup \{\varepsilon\}$ and the following sequence to $\Delta_{\mathcal{B}}$:

$$\begin{array}{rr}
(p_1, \varepsilon, \alpha, p_2, \mathrm{push}_1^a) & (p_2, \varepsilon, a, p_3, \mathrm{push}_1^q) \\
(p_3, \varepsilon, q, p_4, \mathrm{pop}_{n+1}^=) & (p_4, \varepsilon, q, p_5, \mathrm{pop}_1) \\
(p_5, \varepsilon, a, q, \mathrm{pop}_1) &
\end{array}$$

Here, as in the constructions before, new states p_i are chosen for every transition δ. Let $[\delta]$ denote the set of transitions added for δ. We can now show for all configurations (q, s) and (q', s') with $q, q' \in Q$ that

$$(q, s) \xrightarrow{a} (q', s') \text{ via } \delta \iff (q', s') \xrightarrow{\bar{a}} (q, s) \text{ via a sequence in } [\delta].$$

For the direction from left to right note that with the transitions $(q', \bar{a}, \beta, p_1, i^{-1})$ firstly added above the effect of the instruction i is reversed. In states p_1 and p_2 the symbols a and q are added to the stack. Thus the two top level n stacks coincide and the transition from p_3 can be executed. Finally q and a are removed again.

For the converse direction assume that there is a sequence in $[\delta]$ such that $(q', s') \xrightarrow{\bar{a}} (q, s)$. By construction this implies that there transition $\delta' = (q, b, \beta, q', j)$ such that $(q, s) \xrightarrow{b} (q', s')$ via δ', i.e. $s' = j(s)$. The transition from p_1 in $[\delta]$ ensures that $\alpha = \beta$. The transition from p_3 is due to the equality pop instruction only applicable if $a = b$ and $i(s') = i(j(s)) = s$, i.e. only if $i = j^{-1}$. Thus we can conclude that $\delta' = \delta$. $\square$

Note that the equality pop instruction executed in state p_3 cannot be replaced by a standard pop instruction without equality test. Figure 3.11 shows a situation where without an equality test backward edges to both configurations on the left hand side would be added.

$$\left(q, \begin{bmatrix} [ccd] \\ [cd] \end{bmatrix}\right) \xrightarrow[a]{\mathrm{pop}_1} \left(q', \begin{bmatrix} [cd] \\ [cd] \end{bmatrix}\right)$$

$$(q, [\ [cd]\ ]) \xrightarrow[a]{\mathrm{push}_2} \left(q', \begin{bmatrix} [cd] \\ [cd] \end{bmatrix}\right)$$

Figure 3.11: Equality pop operations are necessary

Proposition 3.22 *For every $n \geq 1$, if $G \in \mathrm{Graph}(n)$ then $G \in \mathrm{HOPDS}(n)$.*

Proof. We proceed by induction on the level n of the graph considered. For $n = 1$ we can apply the characterization of $\mathrm{Graph}(1)$ as the class of prefix recognizable graphs and therefore as pushdown graphs [Bar97, Sti, Blu01].

For the induction step it suffices to show that if $G \in \mathrm{HOPDG}(n)$ is a higher-order pushdown graph of level n, $r \in V^G$ and h an inverse rational mapping then there exist a higher-order pushdown system of level $n+1$ which generates $h^{-1}(\mathrm{Unf}(G, r))$.

Let $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta) \in \mathrm{HOPDS}(n+1)$ be the higher-order pushdown system constructed in Lemma 3.21 which generates the extension of $\mathrm{Unf}(G, r)$ by reverse edges. Let $h : \Sigma' \to \mathcal{P}(\Sigma \cup \bar{\Sigma})^*$ be a rational mapping and for $b \in \Sigma'$ let $\mathcal{H}_b = (P_b, \Sigma \cup \bar{\Sigma}, p_0, \Delta_b, \{f\})$ be a finite (non-deterministic) automaton recognizing $h(b)$. W.l.o.g. we assume that the automata $\mathcal{H}_b$ are in the following normal form:

- $\mathcal{H}_b$ has a single initial state p_0 and no transitions back to the initial state.
- $\mathcal{H}_b$ has only a single final state f which is a sink. If $h(b)$ contains the empty word we have to allow a single ε-transition (p_0, ε, f) from the initial to the final state.

Furthermore we assume that all automata $\mathcal{H}_b$ share the same initial state p_0 and the same final state f, but otherwise their state sets are disjoint.

We describe now how to construct from $\mathcal{A}$ and the family $(\mathcal{H}_b)_{b \in \Sigma'}$ an automaton which generates $h^{-1}(\mathrm{Unf}(G, r))$.

Step 1: We add new states to the higher-order pushdown system which will represent the states of the automata $\mathcal{H}_b$. For every $q \in Q$ and every $p \in \bigcup_{b \in \Sigma'} P_b \setminus \{f\}$ create a new state (q, p). Note that only a single state p_0 is added which represents both the initial and final state of every automaton $\mathcal{H}_b$. We add the transitions $(q, \varepsilon, \alpha, (q, p), -)$ for every $q \in Q$, $p \in \bigcup_{b \in \Sigma'} P_b \setminus \{f\}$ and $\alpha \in \Gamma \cup \{\varepsilon\}$.

We obtain a new higher-order pushdown system $\mathcal{A}_1$ whose configuration graph contains as many copies of every configuration as there are states in $\bigcup_{b \in \Sigma'} P_b \setminus \{f\}$. There is a ε-edge from every configuration (q, s) to its copy $((q, p), s)$.

Step 2: We now add transitions to simulate the behavior of the automata $\mathcal{H}_b$ in the configuration graph directly.

Let $b \in \Sigma'$. For every $a \in \Sigma \cup \bar{\Sigma}$, $(p, a, p') \in \Delta_b$ and $(q, a, \alpha, q', i) \in \Delta_{\mathcal{A}}$ we add $((q, p), \tilde{a}, \alpha, (q', \tilde{p}), i)$ where $\tilde{a} = b$ if $p = p_0$ and $\tilde{a} = \varepsilon$ otherwise, and $\tilde{p} = p_0$ if $p' = f$ and $\tilde{p} = p'$ otherwise. For $(q, \varepsilon, \alpha, q', i) \in \Delta_{\mathcal{A}}$ and $p \in \bigcup_{b \in \Lambda} P_b \setminus \{f\}$ we add $((q, p), \varepsilon, \alpha, (q', p), i)$. If $h(b)$ contains the empty word, we also add $((q, p_0), b, \alpha, (q, p_0), -)$ for every $q \in Q$ and every $\alpha \in \Gamma \cup \{\varepsilon\}$.

We obtain a new higher-order pushdown system $\mathcal{A}_2$ which has the property that for all configurations (q, s), (q', s') and every word $u \in (\Sigma \cup \bar{\Sigma})^*$: $u \in L(\mathcal{H}_b)$ and there exists a path labeled by u from (q, s) to (q', s') in $C(\mathcal{A}_1)$ iff there exists a path labeled b from $((q, p_0), s)$ to $((q', p_0), s')$ in $C(\mathcal{A}_2)$.

Step 3: The configuration graph of the higher-order pushdown system $\mathcal{A}_2$ constructed above still contains all the edges that were contained in $\mathcal{A}$. To remove these edges from the generated graph, but to keep the configuration graph connected we replace every transition $(q, a, \alpha, q', i) \in \Delta_{\mathcal{A}}$ by $(q, \varepsilon, \alpha, q', i)$. We obtain a new automaton $\mathcal{A}_3$ whose configuration graph $C(\mathcal{A}_3)$ differs from the configuration graph $C(\mathcal{A}_2)$ only in so far as every edge labeled with a symbol from $(\Sigma \cup \bar{\Sigma})$ is replaced by an ε-labeled edge. Note that in $C(\mathcal{A}_3)$ from every configuration only ε-transitions or only non-ε-transitions are possible.

The graph $C(\mathcal{A}_3)$ still contains paths labeled by $b \in \Sigma'$ starting in a state (q, p_0) and ending in a non-final state, i.e. a state (q', p) with $p \neq p_0$. To ensure that these paths will be removed when taking the ε-closure we add an ε-loop to every state (q', p) with $p \neq p_0$. Let $\mathcal{A}_4$ be the this higher-order pushdown automaton. According to the construction the graph generated by $\mathcal{A}_4$ is $h^{-1}(\mathrm{Unf}(G, r))$ which finishes the proof of the proposition. □

We illustrate constructions in the proof of Proposition above 3.22 with a very simple example. Let $\Sigma' = \{c\}$ and the rational mapping defined by $h(c) = a(a + b)b$. Figure 3.12 shows in the first line an automaton recognizing $h(c)$ where the initial and the final state have the same name. Below a sequence of configurations (q_i, s_i) of $\mathcal{A}$ is depicted. The third line shows the new configurations obtained after Step 1. For clarity a configuration $((q_i, p_j), s_i)$ is denoted just by p_j. The curved arrows show the transitions added in Step 2. Note that since the path from (q_0, s_0) to (q_3, s_3) is labeled by $aab \in h(c)$ there is a path labeled by c from the leftmost p_0 to the rightmost p_0. In Step 3 the labels a and

b are replaced by ε and an ε-loop is added to all configurations p_1 and p_2. This ensures that the path labeled by c starting from p_0 below (q_1, s_1) will be removed in the ε-closure of this graph.

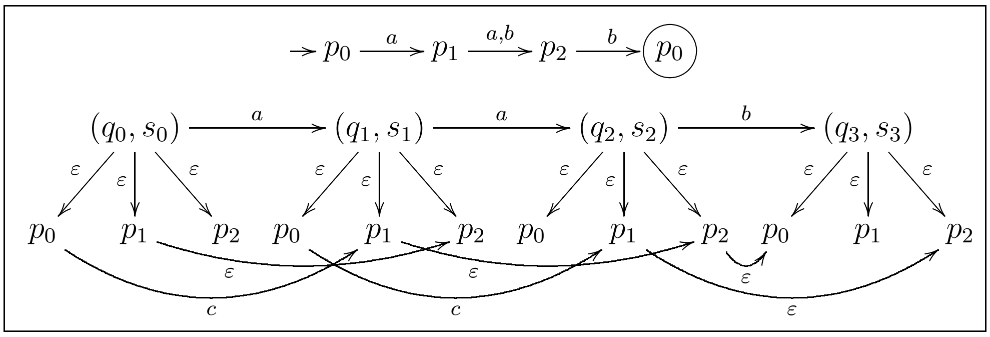


Figure 3.12: Example for the construction in Proposition 3.22

Combining Proposition 3.11 and Proposition 3.22 we obtain one of the main results of this chapter.

Theorem 3.23 *For every $n \geq 1$ we have $G \in \mathrm{HOPDG}(n)$ iff $G \in \mathrm{Graph}(n)$.* □

3.5 Properties of HOPDGs

In this section we present some consequences of characterization of higher-order pushdown graphs as graphs in the Caucal hierarchy.

First we summarize the results of Theorem 3.10, Theorem 3.23 and the proof of Proposition 3.11.

Corollary 3.24 *For $n \geq 1$ the following are equivalent:*

(a) $G \in \mathrm{HOPDG}(n)$.

(b) $G \in \mathrm{Graph}(n)$.

(c) G can be obtained from a finite graph by iteratively applying the treegraph operation and MSO-transductions n times.

(d) G can be defined in Δ^n_m by an MSO-transduction for some $m \geq 1$. □

Since Δ_m^n is definable in Δ_2^n by an MSO-interpretation we can strengthen the last point in the sense that every higher-order pushdown graph of level n can be obtained from the single graph Δ_2^n via an MSO-transduction. For this reason we call Δ_2^n a *generator* for $\mathrm{HOPDG}(n)$.

Combining Theorem 3.23 and the second statement of Theorem 3.10 we obtain a decidability result.

Theorem 3.25 *Every higher-order pushdown graph has a decidable MSO+C theory.* □

The classes $\mathrm{HOPDG}(n)$ inherit from $\mathrm{Graph}(n)$ the closure properties stated in Corollary 3.8.

Corollary 3.26

1. *For every $n \geq 1$ and every $G \in \mathrm{HOPDG}(n)$ we have $\mathrm{Treegraph}(G, \sharp) \in \mathrm{HOPDG}(n+1)$.*
2. *For every $n \geq 1$, every $G \in \mathrm{HOPDG}(n)$ and every MSO-transduction $\mathcal{T}$ we have $\mathcal{T}(G) \in \mathrm{HOPDG}(n)$.* □

Proposition 3.2 allows us to transfer the hierarchy result of J. Engelfriet [Eng91] in terms of non-elementary time complexity classes from languages to the higher-order pushdown graph classes. To state the result we define the functions $\exp_0(n) := n$ and $\exp_k(n) := 2^{\exp_{k-1}(n)}$ for $n \geq 0$ and $k \geq 1$. By $\mathrm{DTIME}(f(n))$ we denote as usual the class of languages which can be accepted by a deterministic Turing machine with time bound $f(n)$.

Theorem 3.27 ([Eng91]) *Let $L \subseteq \Sigma^*$. Then L is accepted by a higher-order pushdown automata of level k iff $L \in \bigcup_{d\geq 1} \mathrm{DTIME}(\exp_{k-1}(dn))$.*

Combining Proposition 3.2 and 3.27 we obtain.

Theorem 3.28 *The classes $\mathrm{HOPDG}(n)$ of higher-order pushdown graphs of level n form a strict hierarchy.* □

By Theorem 3.23 we obtain the same result for the graph classes of the Caucal hierarchy.

Corollary 3.29 *The classes* $\mathrm{Graph}(n)$ *of Caucal graphs of level* n *form a strict hierarchy.* □

The characterization of higher-order pushdown graphs in terms of complexity classes provides a proof method via the complexity of the set of traces to show that a graph G is not a level n higher-order pushdown graph, or that it does not belong to the hierarchy at all. In particular it allows us to provide a graph with a decidable MSO theory outside the hierarchy.

Let $\exp_\omega(0) := \exp_0(1)$ and $\exp_\omega(n) := \exp_n(1)$. The tree $T_{\exp_\omega}$ associated to the function $\exp_\omega$ is depicted in Figure 3.13. It consist of a semi-infinite line of of vertices connected by a-labeled edges, with a path of length $\exp_\omega(n)$ labeled by b originating at the nth vertex of this line. $T_{\exp_\omega}$ serves as an example that not all graphs (even not all trees) with a decidable MSO-theory are higher-order pushdown graphs.

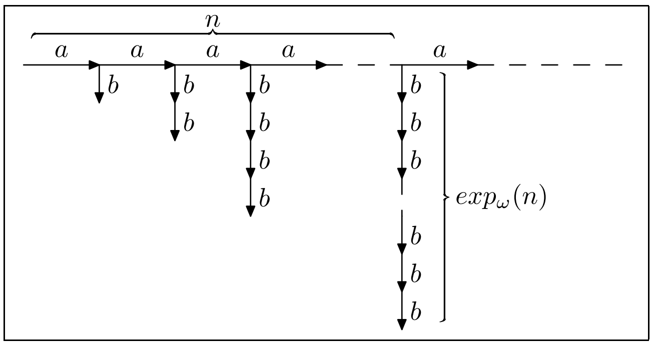


Figure 3.13: The tree T_{exp_ω} associated to exp_ω

Proposition 3.30 *There exits a graph with a decidable MSO-theory which is not a higher-order pushdown graph.*

Proof. We consider the tree $T_{\exp_\omega}$ introduced above. Its set of traces is the language $\{a^n b^{\exp_\omega(n)} \mid n \geq 1\}$ which is not contained in any of the complexity classes $\bigcup_{d\geq 1} \mathrm{DTIME}(\exp_{k-1}(dn))$ for $k \geq 1$. Hence by Theorem 3.27 the tree $T_{\exp_\omega}$ is not generated by a higher-order pushdown system. It remains to show that the MSO-theory of $T_{\exp_\omega}$ is decidable.

Let $(\mathcal{N}, P)$ be the extension of the positive integers $\mathcal{N} = (\omega, S)$ by the predicate $P := \{m \mid m = \exp_\omega(n) \text{ for some } n\}$. Since it is easy to interpret $T_{\exp_\omega}$ in $(\mathcal{N}, P)$ by an MSO-interpretation it suffices to show that that the MSO-theory of $(\mathcal{N}, P)$ is decidable. For this we can apply a result of O. Carton and W. Thomas [CT02]. They define the class $\mathcal{K}$ of profinitely ultimately periodic sequences and show that the MSO-theory of $(\mathcal{N}, P)$ with P defined as $P := \{k_n \mid n \in \omega\}$ is decidable for every sequence $(k_n)_{n\geq 0} \in \mathcal{K}$. The predicate P defined above belongs to $\mathcal{K}$. □

A. Montanari and G. Puppis give an alternative proof of the decidability of the MSO-theory of $T_{\exp_\omega}$ by providing a residually regular factorization of this tree, see [MP04].

We end this chapter by mentioning some immediate questions which show that the class of higher-order pushdown graphs deserves further studies. A more comprehensive summary of the result and questions which arise from them is given in the conclusion of the thesis.

We start with the example tree $T_{\exp_\omega}$ which shows that the class of higher-order pushdown graphs does not encompass all graphs with a decidable monadic second-order theory. This graph is constructed in such a way that it transcends every level of the hierarchy. It thus represents, in a certain sense, a limit of a sequence of graphs in the hierarchy. Can we define a monadic second-order compatible limit operation which captures graphs such as $T_{\exp_\omega}$?

The only method we currently have to separate the levels of the hierarchy uses the complexity results of J. Engelfriet [Eng91]. A. Blumensath recently showed pumping lemma for higher-order pushdown automata [Blu], which is however not yet applicable for this purpose. It would be nice to have a graph theoretic property like the theory of ends developed by Muller and Schupp [MS85] for configuration graphs of pushdown automata to characterize the graphs on the levels of the hierarchy. This last question is closely related to the question whether it is possible to decide for a higher-order pushdown graph to which level it belongs.

Chapter 4

Synchronized Products of Graphs

In the previous chapter we investigated internal and transformational defined classes of graphs with a decidable monadic second-order theory. In both approaches the idea of a hierarchical structuring of the graphs was present, either by the iteration of operations or in the internal data structures of the automata used to described the graphs. We now turn to a different methodology where graphs are composed from components rather than hierarchically structured.

In this chapter we investigate the power of forming synchronized products with respect to decision problems for various logics. As already stated in Subsection 2.3.3 asynchronous products do not preserve the decidability of the MSO-theory of the graphs under consideration. The infinite grid as an asynchronous product of two copies of the positive integers serves as the standard example.

In the next section we explore transitive closure logic over the infinite grid. We show that even if we allow transitive closure subformulas to define only new edge relations, i.e. if we consider transitive closure operators of arity one (even without parameters), the corresponding theory over the infinite grid is in general undecidable. Only if we additionally disallow the nesting of transitive closure operators, the theory is decidable.

This result shows that transitive closure logic is still too expressive to preserve its decidability under asynchronous products. In Section 4.2 we provide a decidability result for the less expressive reachability logic FO(R) and a class of restricted synchronized products which we call finitely synchronized. In this case we are able to show a composition theorem which effectively reduces the evaluation of an FO(R)-formula in the product graph to the evaluation of families of FO(R)-formulas in the components and the evaluation of a Boolean formula which combines their truth values.

In Section 4.3 we finally show that this result is optimal in the following sense: Firstly, if we extend the expressive power of the logic under consideration to allow regular path descriptions, i.e. we consider FO(Reg), then already asynchronous products do not preserve the decidability of the respective theory. Secondly, if we slightly relax the restrictions posed on the synchronization operation, i.e. if we allow semi-finitely synchronized products, the decidability of the reachability logic FO(R) is not preserved anymore.

A preliminary version of the results presented in this chapter has been published in [WT04].

4.1 Transitive Closure Logic over the Infinite Grid

In this section we investigate transitive closure logic over the infinite grid $\mathcal{G}$. We first show how to interpret the first-order theory of addition and multiplication of the positive integers in $\text{FO(TC)}^2_{(1)}$ without parameters over $\mathcal{G}$. This yields that transitive closure logic over the infinite grid is in general undecidable.

The interpretation of the multiplication of integers over $\mathcal{G}$ is accomplished by a transitive closure formula in which the transitive closure operators are nested. In the second part of this section we show that this nesting of operators is indeed necessary to accomplish the undecidability result. If the nesting of transitive closure operators is disallowed, i.e. if we consider only the fragment $\text{FO(TC)}^1_{(1)}$, then the corresponding theory becomes decidable even in the presence of parameters in the transitive closure formulas. The proof of this theorem relies on the locality of first-order logic and the regularity of the infinite grid.

We recall some definitions of Section 2.1. $\text{FO(TC)}_{(k)}$ is the logic obtained by extending first-order logic with formulas of the form

$$\psi(\bar{s},\bar{t},\bar{z}) := [\text{TC}_{\bar{x},\bar{y}}\,\varphi(\bar{x},\bar{y},\bar{z})]\,\bar{s},\bar{t}$$

where $\varphi(\bar{x},\bar{y},\bar{z})$ is again a $\text{FO(TC)}_{(k)}$-formula, $\bar{x},\bar{y}$ and $\bar{z}$ are disjoint tuples of free variables, $\bar{x}$ and $\bar{y}$ are of the same length $1 \le l \le k$, and $\bar{s},\bar{t}$

are tuples of variables of length l. The variables $\bar{z}$ are called parameters for $\psi(\bar{s}, \bar{t}, \bar{z})$.

In the following we will consider transitive closure operators of arity $k = 1$ and $k = 2$ and we will denote the corresponding formulas by

$$\psi(s, t, \bar{z}) := [\mathrm{TC}_{x,y}\, \varphi(x, y, \bar{z})]\, s, t$$

respectively

$$\psi(s_1 s_2, t_1 t_2, \bar{z}) := [\mathrm{TC}_{x_1 x_2, y_1 y_2}\, \varphi(x_1 x_2, y_1 y_2, \bar{z})]\, s_1 s_2, t_1 t_2.$$

The logic $\mathrm{FO(TC)}^2_{(k)}$ is the restriction of $\mathrm{FO(TC)}_{(k)}$ to formulas in which the nesting depth of TC-operators is restricted to two. A transitive closure formula is called *free of parameters* if for any transitive closure subformula $\psi(\bar{s}, \bar{t}, \bar{z})$ as above we have $\bar{z} = \emptyset$, i.e. the only free variables of ψ are $\bar{s}$ and $\bar{t}$.

By $\mathcal{N} := (\omega, S)$ with $S := \{(i, i+1) \mid i \geq 0\}$ we denote the semi-infinite line which we interpret as the positive integers with successor relation. The infinite grid is the graph structure $\mathcal{G} := (\omega \times \omega, S_1, S_2)$ with $S_1 := \{((i,j),(i+1,j)) \mid i,j \geq 0\}$ and $S_2 := \{((i,j),(i,j+1)) \mid i,j \geq 0\}$. We depict $\mathcal{G}$ as a square open upwards and to the right, with horizontal edges S_1 and vertical edges S_2, see Figure 2.1. Note that $\mathcal{G}$ can be viewed as the asynchronous product (defined by the empty synchronization constraint) of two copies of the positive integers $\mathcal{N}_1 = (\omega, S_1)$ and $\mathcal{N}_2 = (\omega, S_2)$ with S_1 and S_2 interpreted as the successor relation $S_1, S_2 := S$.

4.1.1 Undecidability of $\mathbf{FO(TC)}^2_{(1)}$

We show in this subsection how to interpret the first-order theory of addition and multiplication of the positive integers in $\mathrm{FO(TC)}^2_{(1)}$ without parameters over the infinite grid. This allows us to conclude that the decidability of $\mathrm{FO(TC)}^2_{(1)}$ is not preserved under asynchronous products.

Theorem 4.1 *The $\mathrm{FO(TC)}^2_{(1)}$-theory of the infinite grid is undecidable.*

This result already holds for the fragment of $\mathrm{FO(TC)}^2_{(1)}$ which contains only formulas without parameters.

Proof. We define addition and multiplication in $\mathrm{FO(TC)}^2_{(1)}$ over $\mathcal{G}$ without the use of parameters. Since the first-order theory of $(\omega, +, \cdot)$ is undecidable this immediately yields the undecidability result for $\mathrm{FO(TC)}^2_{(1)}$ and the infinite grid.

We show that for every first-order formula $\varphi(x_1,\dots,x_n)$ interpreted over $(\omega,+,\cdot)$ there exists a FO(TC)$^2_{(1)}$ formula $\hat{\varphi}(x_1,\dots,x_n)$ such that

$$(\omega,+,\cdot) \models \varphi[k_1,\dots,k_n] \Leftrightarrow \mathcal{G} \models \hat{\varphi}[(k_1,l_1),\dots,(k_n,l_n)]$$

for some $l_1,\dots,l_n \geq 0$, i.e. we forget about values of the second components of the grid vertices.

To restrict the arity of the transitive closure operators to one we have combine two variables representing positive integers into one variable representing a grid vertex. We use the operations

(i) $\pi_1((x,y)) := (x,0)$,

(ii) swap$((x,y)) := (y,x)$ and

(iii) comb$((u,v),(x,y)) := (u,y)$

on grid vertices which can be defined in FO(TC)$^1_{(1)}$ due to the following equivalences

(i) $\pi_1(x) = y \leftrightarrow y \leq_2 x \wedge \forall z(z \leq_2 y \rightarrow z = y)$

(ii) $\text{swap}(x) = y \leftrightarrow \exists z_1 \exists z_2 \Big(\bigwedge_{i=1,2} [\mathrm{TC}_{x,y}\, \exists w(S_1xw \wedge S_2wy)]0, z_i$
$\wedge\, z_1 \leq_1 x \wedge z_1 \leq_2 y \wedge x \leq_2 z_2 \wedge y \leq_2 z_2 \Big)$

(iii) $\text{comb}(x,y) = z \leftrightarrow [\mathrm{TC}_{x,y}\, S_2xy \vee S_2yx]x,z \wedge [\mathrm{TC}_{x,y}\, S_1xy \vee S_1yx]y,z$

where the constant 0 is defined by $x = 0 \Leftrightarrow \forall y(\neg S_1yx \wedge \neg S_2yx)$ and $x \leq_i y \leftrightarrow x = y \vee [\mathrm{TC}_{x,y}\, S_ixy]xy$ is the linear order induced by S_i for $i = 1,2$. The FO(TC)$^1_{(1)}$-formula stated in (ii) equivalent to swap$(x) = y$ just states that there are two vertices z_1 and z_2 on the diagonal such that the area enclosed by x,y,z_1,z_2 forms a square.

The definition of addition is now straightforward. Note that over the positive integers with successor relation we have

$$a + b = c \leftrightarrow \mathcal{N} \models [\mathrm{TC}_{x_1x_2,y_1y_2}\, Sx_1y_1 \wedge Sx_2y_2]0\, a, b\, c.$$

With the operations above we now can easily translate this formula into a formula to be interpreted over the infinite grid. Let

$$\varphi_+(x,y,z) := \exists v \exists w(v = \text{swap}(\pi_1(x)) \wedge w = \text{comb}(y,z)$$
$$\wedge\, [\mathrm{TC}_{x,y}\, \exists z(S_1(x,z) \wedge S_2(z,y))]v,w).$$

Then obviously $a + b = c$ iff $\mathcal{G} \models \varphi_+(x, y, z)[(a, l_1), (b, l_2), (c, l_3)]$ holds for some $l_1, l_2, l_3 \geq 0$. Also $\varphi_+(x, y, z)$ is indeed a FO(TC)$^1_{(1)}$-formula since the nesting of TC-operators which appears in the subformula $v = \text{swap}(\pi_1(x))$ can be avoided by replacing this part by $\exists v'(v' = \pi_1(x) \wedge v = \text{swap}(v'))$.

To define multiplication note that

$$x \cdot y = \frac{(x + y)^2 - x^2 - y^2}{2}.$$

Hence it suffices to define $x \mapsto x^2$. We use the fact that $x^2 = \sum_{i=0}^{x-1} 2i + 1$. The formula

$$\psi(x, y) = [\text{TC}_{x_1x_2,y_1y_2}\, y_2 = x_2 + (x_2 - x_1) + 2 \wedge y_1 = x_2]0\,1, xy$$

defines all pairs of square numbers

$$(x, y) = \big(\sum_{i=0}^{k-2} 2i + 1, \sum_{i=0}^{k-1} 2i + 1\big) \text{ for } k \geq 2.$$

as it is shown in Figure 4.1. Hence $\mathcal{N} \models \psi[a, b]$ iff $a = (k - 1)^2$ and $b - a = 2k - 1$ for some $k \geq 2$.

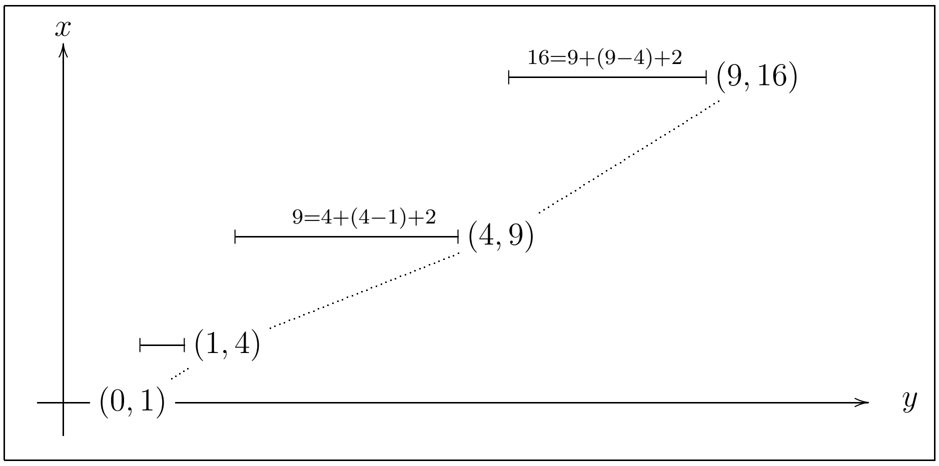


Figure 4.1: Pairs of square numbers defined by $\psi(x, y)$

Let

$$\chi(x, y) = \exists z_1\Big(\psi(z_1, y) \wedge \frac{y - z_1 + 1}{2} = x\Big).$$

Then $\mathcal{N} \models \chi[a, b]$ iff $b = a^2$. As before this formula can be translated into a formula to be interpreted over the infinite grid, with the only nesting of transitive closure quantifiers introduced by the use of addition (and subtraction) inside a TC-operator in the formula $\psi(x, y)$ above. □

Since FO(TC)$_{(1)}$ over $\mathcal{N}$ can be interpreted in the monadic second-order theory over $\mathcal{N}$ and this theory is decidable, we can conclude that the decidability of FO(TC)$^2_{(1)}$ is not preserved under the formation of asynchronous products.

Corollary 4.2 *Asynchronous products do not preserve the decidability of FO(TC)$_{(1)}$.* □

4.1.2 Decidability of FO(TC)$^1_{(1)}$

In this subsection we show that the nesting of transitive closure operators of arity two as it appears in the proof of Theorem 4.1 is necessary to obtain the undecidability result for FO(TC)$_{(1)}$ over the infinite grid $\mathcal{G}$.

We show that the FO(TC)$^1_{(1)}$-theory of $\mathcal{G}$ can be reduced to Presburger arithmetic [Pre30, Pre91], the first-order theory of the positive integers with addition $\mathcal{N}_+ := (\omega, +, 0)$, which is well known to be decidable.

The reduction relies on a normal form for first-order logic over graphs of bounded degree which can be deduced from Hanf's locality theorem [Han65], see also [EF95, Tho97]. To state the locality theorem we have to introduce some notation first.

Let G be a graph and $v, w \in V^G$ be vertices of G. By $\mathrm{dist}(v, w)$ we denote the length of a shortest path from v to w in G if such a path exists. Thereby we allow to traverse the edges of the graph in either direction. The *r-sphere* around v is the set $r\text{-sph}(v) := \{w \in V^G \mid \mathrm{dist}(v, w) \leq r\}$ of vertices of distance $\leq r$ from v.

Let $v_1, \ldots, v_n \in V$ be vertices of G. The *isomorphism type* of $S := \bigcup_{1\leq i\leq n} r\text{-sph}(v_i)$ is the set of first-order formulas $\varphi(x_1, \ldots, x_n)$ such that $(G \restriction S) \models \varphi[v_1, \ldots, v_n]$. In general, an isomorphism type τ for such a union of r-spheres is a maximal consistent set of formulas with at most n free variables for some n.

If a graph G is of bounded degree then the number of vertices contained in an r-sphere around a vertex v is finite. In this case every isomorphism type τ of a union $S := \bigcup_{1\leq i\leq n} r\text{-sph}(v_i)$ of r-spheres can be characterized by a single first-order formula $\varphi_\tau(x_1, \ldots, x_n)$. Let $w_1, \ldots, w_l$

be an enumeration of S with $w_i = v_i$ for $1 \leq i \leq n$ and

$$\text{Lit}(S) := \{\alpha(x_1, \ldots, x_l) \mid (G \restriction S) \models \alpha[w_1, \ldots, w_l],\ \alpha \text{ a literal}\}$$

be the set of literals (atomic or negations of atomic formulas) over variables $x_1, \ldots x_l$ which are true in $G \restriction S$. Let

$$\varphi_\tau(x_1, \ldots, x_n) := \exists x_{n+1} \ldots x_l \Big(\bigwedge \text{Lit}(S) \wedge \forall x_{l+1} \bigvee_{1 \leq i \leq l} x_{l+1} = x_i \Big).$$

Then we have for every union $S' := \bigcup_{1 \leq i \leq n} r\text{-sph}(u_i)$ of n r-spheres with $(G \restriction S') \models \varphi_\tau[u_1, \ldots, u_n]$ that

$$(G \restriction S) \models \psi[v_1, \ldots, v_n] \Leftrightarrow (G \restriction S') \models \psi[u_1, \ldots, u_n]$$

for every $\psi(x_1, \ldots, x_n) \in \tau$. In this case we say that τ can be *realized* in G. Since for spheres of radius r and graphs of degree $\leq d$ the number of elements of S is bounded by nd^r there are only finitely many such isomorphism types.

Let $T(d, r)$ be the set of isomorphism types of r-spheres which can be realized in a graph of degree at most d. Hanf's theorem states that the validity of a first-order sentence in graphs of bounded degree depends only on how often (up to a certain threshold) every isomorphism type as defined above can be realized in the graph.

Theorem 4.3 (Hanf [Han65, EF95]) *For every $d \geq 0$ and for every first-order sentence φ there exist $r, q \geq 0$ such that for all graphs G, H of degree $\leq d$ we have: If every isomorphism type $\tau \in T(d, r)$ is realized by both G and H more than q times or both G and H have the same number $\leq q$ of elements which realize τ, then $G \models \varphi \Leftrightarrow H \models \varphi$.*

From Hanf's Theorem we can extract a normal form for first-order formulas over graphs of bounded degree (see also [Tho97]): Every first-order formula is equivalent to a Boolean combination of statements "isomorphism type τ is realized more than j times".

In the proof of the following theorem we will apply this normal form to the first-order kernels of the transitive closure formulas in $\text{FO(TC)}^1_{(1)}$.

Theorem 4.4 *The $\text{FO(TC)}^1_{(1)}$-theory of the infinite grid is decidable.*

Proof. We reduce the FO(TC)$^1_{(1)}$-theory of the infinite grid $\mathcal{G}$ to Presburger arithmetic in the following sense: For every FO(TC)$^1_{(1)}$-formula $\varphi(x_1, \dots, x_n)$ there is a Presburger formula $\tilde{\varphi}(x_{11}, x_{12}, \dots, x_{n1}, x_{n2})$ such that for all $k_1, l_1, \dots, k_n, l_n \geq 0$ we have

$$\mathcal{G} \models \varphi[(k_1, l_1), \dots, (k_n, l_n)] \Leftrightarrow \mathcal{N}_+ \models \tilde{\varphi}[k_1, l_1, \dots, k_n, l_n]. \qquad (4.1)$$

We proceed by induction on the structure of the FO(TC)$^1_{(1)}$-formulas. Since the successor relations S_1 and S_2 are immediately definable from the constant 0 and $+$, it suffices to consider transitive closure formulas

$$\varphi(x_1, \dots, x_n) = [\mathrm{TC}_{x,y}\, \psi(x, y, x_3, \dots, x_n)]x_1, x_2$$

where $\psi(x, y, x_3, \dots, x_n)$ is a first-order formula and the variables $x_3, \dots, x_n$ serve as parameters for the transitive closure operator of φ.

In a first step we rewrite the first-order kernel $\psi(x, y, x_3, \dots, x_n)$ in disjunctive normal form. Then we apply Hanf's Theorem. There exists a suitable r such that ψ is equivalent to a Boolean combination of statements "isomorphism type τ is realized more than j times".

For the sake of convenience we assume that the origin $(0, 0)$ of the grid is included in the set of parameters. Then, due to the regular structure of the infinite grid, every isomorphism type τ realizable outside $\bigcup_{1 \leq i \leq n} r\text{-sph}(c_i)$ for some tuple $c_1, \dots, c_n$ of grid vertices occurs an infinite number of times. In particular, counting statements such as "isomorphism type τ is realized more than j times" outside $\bigcup_{1 \leq i \leq n} r\text{-sph}(c_i)$ are superfluous here. We obtain that $\psi(x, y, x_3, \dots, x_n)$ is equivalent to a (finite) disjunction of formulas $\varphi_\tau(x, y, x_3, \dots, x_n)$ which describe isomorphism types τ of $\bigcup_{1 \leq i \leq n} r\text{-sph}(c_i)$ for certain tuples $c_1, \dots, c_n$ of grid vertices. Let T be the finite set of all isomorphism types τ of $\bigcup_{1 \leq i \leq n} r\text{-sph}(c_i)$ realizable in the infinite grid.

Due to the special structure of the grid, for every $\tau \in T$ the formula $\varphi_\tau(x_1, \dots, x_n)$ describing an isomorphism type from T can be expressed by conditions on the vertices $x_1, \dots, x_n$ which fix their distances up to the radius r from the left margin as well as the bottom margin, and their relative distances up to $2r$.

It is convenient to express $\varphi_\tau(x_1, \dots, x_n)$ in terms of the $2n$ components of the vertices. We obtain a formula $\tilde{\varphi}_\tau(x_{11}, x_{12}, \dots, x_{n1}, x_{n2})$ which is interpreted over $\mathcal{N}_+$ and equivalent to φ in the sense of (4.1) above. It is a conjunction of statements

- $x_{ih} = k$ for $0 \leq k \leq r$ or $x_{ih} > r$
- $(x_{i1}, x_{i2}) = (x_{j1}, x_{j2}) + (k, l)$ for $-2r \leq k, l \leq 2r$
- $\mathrm{dist}((x_{i1}, x_{i2}), (x_{j1}, x_{j2})) > 2r$

where $1 \leq i, j \leq n$ and $h \in \{1, 2\}$. Note that $\mathrm{dist}((x_{i1}, x_{i2}), (x_{j1}, x_{j2})) > 2r$ can be rewritten as a Presburger formula.

We now have to evaluate formulas of the form

$$\Big[\mathrm{TC}_{(x_{11},x_{12}),(x_{21},x_{22})} \bigvee_{\tau \in T'} \tilde{\varphi}_\tau(x_{11}, x_{12}, \ldots, x_{n1}, x_{n2})\Big](s,t),(u,v) \tag{4.2}$$

for some $T' \subseteq T$.

In a first step we note that it is possible to add disjuncts to (4.2) such that vertices tied to occur in a $2r$-sphere around a parameter (x_{i1}, x_{i2}) for $i > 2$ only need to appear as start vertex or as end vertex of any path described by (4.2). Hence vertices tied to parameters can be handled without the use of TC, by an appropriate modification of the formula.

Let I be an initial segment of the grid encompassing the $2r$-spheres around parameters (x_{i1}, x_{i2}) for $i > 2$. Outside this initial segment, in a second step, it suffices to consider formulas (4.2) in which only type formulas $\tilde{\varphi}_\tau$ which contain

$$x_{11} = k_1 \wedge x_{12} > r \quad \text{or} \quad x_{11} > r \wedge x_{12} = k_2 \quad \text{or} \quad x_{11} > r \wedge x_{12} > r$$

and

$$x_{21} = l_1 \wedge x_{22} > r \quad \text{or} \quad x_{21} > r \wedge x_{22} = l_2 \quad \text{or} \quad x_{21} > r \quad \wedge x_{22} > r$$

and

$$\mathrm{dist}((x_{11}, x_{12}), (x_{21}, x_{22})) > 2r \quad \text{or} \quad (x_{11}, x_{12}) = (x_{21}, x_{22}) + (k, l)$$

for $k_1, k_2, l_1, l_2 \leq r$ and $-2r \leq k, l \leq 2r$ appear.

It is now possible to apply a finite saturation process for the transitive closure operation to obtain a formula

$$\Big[\mathrm{TC}_{(x_{11},x_{12}),(x_{21},x_{22})} \bigvee_{1 \leq j \leq m} \tilde{\varphi}_j(x_{11}, x_{12}, \ldots, x_{n1}, x_{n2})\Big](s,t),(u,v) \tag{4.3}$$

which is equivalent to (4.2) and where TC and $\bigvee$ commute, i.e.

$$\mathcal{G} \models \Big[\mathrm{TC}_{(x_{11},x_{12}),(x_{21},x_{22})} \bigvee_{1\le j\le m} \tilde{\varphi}_j\Big](s,t),(u,v) \Leftrightarrow \\ \mathcal{G} \models \bigvee_{1\le j\le m} \Big[\mathrm{TC}_{(x_{11},x_{12}),(x_{21},x_{22})}\, \tilde{\varphi}_j\Big](s,t),(u,v).$$

The subformulas $\tilde{\varphi}_j$ in (4.3) have the same format as the subformulas $\tilde{\varphi}_\tau$ in (4.2) except that the center of the excluded $2r$-sphere around (x_{11}, x_{12}) may be shifted by a bounded distance from (x_{11}, x_{12}) or be missing, or $\tilde{\varphi}_\tau$ defines the complete relation outside I and the border stripes of width r. Thus it remains to consider two cases.
Case 1. If $\tilde{\varphi}_j$ contains a conjunct excluding some $2r$-sphere then the relation defined by $[\mathrm{TC}_{(x_{11},x_{12}),(x_{21},x_{22})}\, \tilde{\varphi}_j](s,t),(u,v)$ is cofinite (w.r.t. the grid excluding the initial segment I and border stripes of width r, or a fixed line in one of the border stripes) and hence definable without the use of a transitive closure operator.
Case 2. If $\tilde{\varphi}_j$ fixes relations of the form

$$(x_{21}, x_{22}) = (x_{11}, x_{12}) + (k_i, l_i) \tag{4.4}$$

for $i = 1, \dots, N$ and $-2r \le k_i, l_i \le 2r$. The formula

$$[\mathrm{TC}_{(x_{11},x_{12}),(x_{21},x_{22})}\, \tilde{\varphi}_j](s,t),(u,v)$$

expresses that there is a path from (s,t) to (u,v) consisting of steps of the form (4.4). The set of vertices (u,v) reachable in this way from (s,t) can be represented as the union of paths in the finite initial segment I of the grid and finitely many sets of the form

$$\{(u,v) \mid (u,v) = (s',t') + y_1(k_1,l_1) + \ldots + y_N(k_N,l_N)\}.$$

Here $y_i \ge 0$, the (s',t') range over boundary vertices of I, and the (k_i, l_i) are from (4.4). It follows that the relation defined by (4.2) is definable in Presburger arithmetic. □

4.2 A Composition Theorem for FO(R)

In this section we consider the logic FO(R) which extends first-order logic by reachability predicates which can express that there is a path from a vertex v to a vertex w in a graph labeled solely by symbols of a specified subset of edge labels.

We introduce a restricted class of synchronized products which we call finitely synchronized and show that the decidability of the FO(R)-theory is preserved under these products. We follow a compositional approach and show that evaluation of an FO(R)-formula in a finitely synchronized product can be effectively reduced to the evaluation of families of FO(R)-formulas in the component graphs.

This positive result is complemented by undecidability results presented in Section 4.3 where we show that for slight variations of the logic under consideration or the power of the synchronization operations the decidability of the corresponding theories is not preserved.

We recall the definition of a synchronization constraint and of synchronized products from Section 2.1. For $1 \leq i \leq n$ let $G_i := (V_i, (E^i_a)_{a \in \Sigma_i})$ be a Σ_i-labeled graph. We assume that Σ_i is partitioned into a set Σ^l_i of *local* labels and a set Σ^s_i of *synchronizing* labels. A *synchronization constraint* is a set $C \subseteq \times_{1 \leq i \leq n} \tilde{\Sigma}^s_i$ where $\tilde{\Sigma}^s_i := \Sigma^s_i \cup \{\varepsilon\}$.

The *synchronized product* of $(G_i)_{1 \leq i \leq n}$ defined by C is the graph G with vertex set $V := \times_{1 \leq i \leq n} V_i$, asynchronous edges with labels $a \in \bigcup_{1 \leq i \leq n} \Sigma^l_i$ defined by $E^G_a \bar{v}\bar{w}$ if $E^i_a v_i w_i$ and $v_j = w_j$ for $j \neq i$, and synchronized edges with labels $\bar{c} \in C$ defined by $E^G_{\bar{c}} \bar{v}\bar{w}$ if $E^i_{c_i} v_i w_i$ for every $1 \leq i \leq n$.

For a set of synchronization constraints $C \subseteq \times_{1 \leq i \leq n} \tilde{\Sigma}^s_i$ let

$$\mathrm{dom}(C) := \{i \mid \exists\, (c_1, \dots, c_n) \in C : \; c_i \neq \varepsilon\}$$

be the set of indices of the graph components which may participate in a synchronization. Recall that for a set $X \subseteq \{1, \dots, n\}$ and a tuple $\bar{v} = (v_1, \dots, v_n)$ the subtuple $(v_i)_{i \in X}$ is denoted by $\bar{v}\restriction X$. To define when a product G of $(G_i)_{1 \leq i \leq n}$ with respect to C is finitely synchronized we introduce an equivalence relation on the set $V := \times_{1 \leq i \leq n} V_i$ of vertices of G. For $\bar{u}, \bar{v} \in V$ and $\bar{c} in C$ we define

$$\bar{u} \sim_{\bar{c}} \bar{v} :\Leftrightarrow \bar{u}\restriction\mathrm{dom}(\bar{c}) = \bar{v}\restriction\mathrm{dom}(\bar{c}),$$

i.e. $\bar{u} \sim_{\bar{c}} \bar{v}$ if $\bar{u}$ and $\bar{v}$ agree on the synchronizing components. The synchronized product G is called *finitely synchronized* if $\mathrm{index}(\sim_{\bar{c}})$ is finite for every $\bar{c} \in C$.

Note that G is finitely synchronized by C iff for every $\bar{c} \in C$ either for every $1 \leq i \leq n$ with $c_i \neq \varepsilon$ the set of vertices $V_i(\bar{c}) := \{v \in V_i \mid \exists w\ (v, w) \in E_{c_i}^{G_i}\}$ of G_i from which a c_i-labeled edge originates is finite, or there exists a $1 \leq i \leq n$ such that $c_i \neq \varepsilon$ and G_i does not contain a c_i-labeled edge, i.e. $V_i(\bar{c}) = \emptyset$. In the second case there will be no $\bar{c}$-labeled edge in the product graph. Thus in the remainder of this section we may w.l.o.g assume that $V_i(\bar{c}) \neq \emptyset$ for every $1 \leq i \leq n$ and every $\bar{c} \in C$ (otherwise just consider $C \setminus \{\bar{c}\}$).

With this assumption we can conclude that for a finitely synchronized product, for every $1 \leq i \leq n$ and every $\bar{c} \in C$ the set $V_i(\bar{c})$ is finite. Thus also the sets $V_i(C) := \bigcup_{\bar{c} \in C} V(\bar{c})$ of vertices of the component graphs G_i which may participate in a synchronization are finite.

Examples of fintely synchronized products can be obtained from the systems depicted in Figure 4.2. The edges labeled by s respectively f denote starting respectively finishing transitions of the processes. We assume that in every graph there are only finitely many vertices from which an s-labeled or an f-labeled edge originates. In this case every synchronized product of P_1, P_2, P_3 with respect to a constraint $C \subseteq \{s, f\}^3$ will be finitely synchronized. These include for example the constraint set $C = \{(s, s, s), (f, f, f)\}$ or $C' = \{(s, \varepsilon, \varepsilon), (f, s, \varepsilon), (\varepsilon, f, s)\}$. The set C models a simultaneous start and finishing of the three processes while C' models that process $i + 1$ may only be started if process i finishes at the same time.

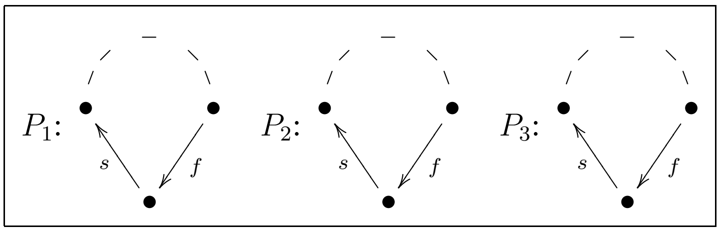


Figure 4.2: Family of graphs forming a finitely synchronized product

The definition of finitely synchronized products of this thesis extends the one which was given in [WT04]. There finitely synchronized products involved only finitely many individual synchronizing transitions, which disallowed the label ε to appear in the i-th component of a constraint $\bar{c} \in C$ as soon as the component graph G_i was infinite. The restriction to components i with $c_i \neq \varepsilon$ in the definition of $\sim_{\bar{c}}$ eliminates this restriction.

Next we present the main result of this section.

Theorem 4.5 *Let $(G_i)_{1\leq i\leq n}$ be a family of graphs with decidable FO(R)-theories and C a set of synchronization constraints.*

(a) If the product G of $(G_i)_{1\leq i\leq n}$ with respect to C is finitely synchronized, then the FO(R)-theory of G is decidable.

(b) If additionally the size of the sets $V_i(C)$ for $1 \leq i \leq n$ is computable, then for every FO(R)-formula φ one can effectively construct finite sets $\Psi_1, \ldots, \Psi_n$ of $FO(R)$-formulas and a Boolean formula α over predicates $p(\psi^i_j)$ $(1 \leq i \leq n,\ 1 \leq j \leq |\Psi_i|)$ such that

$$G \models \varphi[\bar{v}_1, \ldots, \bar{v}_m] \Leftrightarrow I(\bar{v}_1, \ldots, \bar{v}_m) \models \alpha \tag{4.5}$$

where $I(\bar{v}_1, \ldots, \bar{v}_m)$ is the Boolean interpretation defined by

$$I(\bar{v}_1, \ldots, \bar{v}_m)(p(\psi^i_j)) = \begin{cases} \textit{true} & \textit{if } (G_i, v^i_1, \ldots, v^i_m) \models \psi^i_j \\ \textit{false} & \textit{otherwise.} \end{cases}$$

The proof of Theorem 4.5 proceeds by induction on the structure of the formula φ. Before that we illustrate how the composition method can be applied to reduce evaluation of reachability predicates Reach_Γ in $\mathcal{G}$ to the evaluation of families of FO(R)-formulas in the component graphs an a Boolean formula α as above.

Let $(G_i)_{1\leq i\leq n}$ be a family of graphs whose signatures $\Sigma_i := \Sigma^l_i \cup \Sigma^s_i$ are partitioned into local labels Σ^l_i and synchronizing labels Σ^s_i. Let $C \subseteq \times_{1\leq i\leq n}\tilde{\Sigma}^s_i$ be a synchronization constraint such that the product G of $(G_i)_{1\leq i\leq n}$ is finitely synchronized with respect to C.

First we consider reachability predicates Reach_Γ with $\Gamma \subseteq \Sigma^l$, i.e. Γ contains only local transition labels. In this case, for every $1 \leq i \leq n$, the set Ψ_i of formulas to be evaluated in component i only contains the single formula $\psi_i(x,y) := \mathrm{Reach}_{\Gamma\cap\Sigma^l_i}(x,y)$ and $\alpha := \bigwedge_{1\leq i\leq n} p(\psi_i)$. Then obviously (4.5) holds for all vertices $\bar{v}_1, \bar{v}_2 \in V$.

Next we consider the case that Γ contains a single synchronizing label $\bar{c}$ and that this label occurs only once on a path between two vertices. Let

$$\psi_i(x,y) := \exists z_i \Big(\mathrm{Reach}_{\Gamma\cap\Sigma_i^l}(x,z_i) \wedge \exists w \big(E_{c_i} z_i w \wedge \mathrm{Reach}_{\Gamma\cap\Sigma_i^l}(w,y) \big) \Big)$$

for every i with $c_i \neq \varepsilon$. This formula expresses that in component G_i there is a path from vertex x to vertex y which contains an edge labeled c_i once. The vertices assigned to z_i form a subtuple of a vertex of the product G from which a $\bar{c}$-labeled edge originates. Let

$$\psi_i(x,y) := \mathrm{Reach}_{\Gamma\cap\Sigma_i^l}(x,y)$$

for every i with $c_i = \varepsilon$ and $\alpha := \bigwedge_{1\leq i\leq n} p(\psi_i)$. Then $I(\bar{v},\bar{w}) \models \alpha$ iff there exists a path in G from $\bar{v}$ to $\bar{w}$ which contains an edge labeled $\bar{c}$ exactly once.

More generally, for every $m \geq 1$ and every $1 \leq i \leq n$ we can construct a formula $\psi^i_{(\bar{c},m)}(x,y)$ such that $G_i \models \psi^i_{(\bar{c},m)}[v_i,w_i]$ iff there is a path from v_i to w_i on which label c_i occurs exactly m times. Hence $I(\bar{v},\bar{w}) \models \bigvee_{m\geq 0} \bigwedge_{1\leq i\leq n} p(\psi^i_{(\bar{c},m)})$ iff there is an m and a path in G from $\bar{v}$ to $\bar{w}$ which contains an edge labeled $\bar{c}$ exactly m-times. It is easy to see that for every vertex $\bar{w}$ reachable from $\bar{v}$ via a Γ-labeled path there exists such a path in which every $\sim_{\bar{c}}$-equivalence class is passed at most once. Thus we obtain that the number of disjuncts in the formula above can be bounded by $k := \mathrm{index}(\sim_{\bar{c}})$, i.e.

$$I(\bar{v},\bar{w}) \models \bigvee_{m\geq 0} \bigwedge_{1\leq i\leq n} p(\psi^i_{(\bar{c},m)}) \Leftrightarrow I(\bar{v},\bar{w}) \models \bigvee_{0\leq m\leq k} \bigwedge_{1\leq i\leq n} p(\psi^i_{(\bar{c},m)})$$

and the sets Ψ_i can be chosen to be finite.

The case that Γ contains more than one synchronizing label can be handled similarly. We just have to bound the number of synchronized edges which have to be passed on a path. For this we introduce a path normal form where local edges are passed as soon as possible.

Let $\pi := \bar{v}_1 a_1 \bar{v}_2 a_2 \ldots a_m \bar{v}_{m+1}$ with $\bar{v}_j \in V$ and $a_j \in \Sigma^l \cup C$ be a path through G. By $\pi[j]$ we denote the suffix of π starting at vertex $\bar{v}_j$. Let

$$\mathrm{dom}(\pi) := \{ i \mid \exists a_j \in C : \ i \in \mathrm{dom}(a_j) \}$$

be set of indices of components affected by a synchronization on π.

The path $\pi := \bar{v}_1 a_1 \bar{v}_2 a_2 \dots a_m \bar{v}_{m+1}$ is in *normal form* if for all $1 \leq j < k \leq m$ of successive synchronizing transitions $a_j, a_k \in C$, i.e. with $a_{j+1}, \dots, a_{k-1} \in \Sigma^l$, we already have $a_{j+1}, \dots, a_{k-1} \in \bigcup_{i \in \mathrm{dom}(a_j)} \Sigma_i^l$, and if a_k is the last synchronizing label from C on π then $a_{k+1}, \dots, a_m \in \bigcup_{i \in \mathrm{dom}(a_k)} \Sigma_i^l$.

Figure 4.3 shows an example of a path and its normal form. Labels a, b are local, label a for the third component, label b for the first component. The two synchronizing labels $\bar{c}_1, \bar{c}_2$ are of the form $\bar{c}_1 := (*, *, \varepsilon)$ and $\bar{c}_2 := (\varepsilon, *, *)$ respectively. The upper path is not in normal form since $\mathrm{dom}(a) \cap \mathrm{dom}(\bar{c}_1) = \emptyset$ and $\mathrm{dom}(b) \cap \mathrm{dom}(\bar{c}_2) = \emptyset$. The lower path is in normal form since the local edges are traversed as soon as possible.

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \xrightarrow{\bar{c}_1} \begin{pmatrix} v_1' \\ v_2' \\ v_3 \end{pmatrix} \xrightarrow{a} \begin{pmatrix} v_1' \\ v_2' \\ v_3' \end{pmatrix} \xrightarrow{\bar{c}_2} \begin{pmatrix} v_1' \\ v_2'' \\ v_3'' \end{pmatrix} \xrightarrow{b} \begin{pmatrix} v_1'' \\ v_2'' \\ v_3'' \end{pmatrix} \xrightarrow{\bar{c}_1} \begin{pmatrix} v_1''' \\ v_2''' \\ v_3'' \end{pmatrix}$$

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \xrightarrow{a} \begin{pmatrix} v_1 \\ v_2 \\ v_3' \end{pmatrix} \xrightarrow{\bar{c}_1} \begin{pmatrix} v_1' \\ v_2' \\ v_3' \end{pmatrix} \xrightarrow{b} \begin{pmatrix} v_1'' \\ v_2' \\ v_3' \end{pmatrix} \xrightarrow{\bar{c}_2} \begin{pmatrix} v_1'' \\ v_2'' \\ v_3'' \end{pmatrix} \xrightarrow{\bar{c}_1} \begin{pmatrix} v_1''' \\ v_2''' \\ v_3'' \end{pmatrix}$$

Figure 4.3: A path and its normal form

By the definition of a synchronized product we know that edges with disjoint domains commute with each other, i.e. if $\bar{u}a\bar{v}b\bar{w}$ is a path in G and $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$ there exists a $\bar{v}'$ such that $\bar{u}b\bar{v}'a\bar{w}$ is also a path in G. Thus if a path between two vertices of G exists there is also one in normal form.

The special structure of a path in normal form now allows us to analyze the structure of the vertices on the path from which a synchronized edge is taken. Let $\pi := \bar{v}_1 a_1 \bar{v}_2 a_2 \dots a_m \bar{v}_{m+1}$ be in normal form. Let $a_j \in C$. Then $\bar{v}_j \restriction \mathrm{dom}(\pi[j]) \in \times_{i \in \mathrm{dom}(\pi[j])} V_i(C)$ and for every $j \leq k \leq m+1$ we have $\bar{v}_k \restriction (\{1, \dots, n\} \setminus \mathrm{dom}(\pi[j])) = \bar{v}_j \restriction (\{1, \dots, n\} \setminus \mathrm{dom}(\pi[j]))$. That is, if a component i is affected by a synchronization later on the path then the i-th component of $\bar{v}_j$ is already contained in $V_i(C)$. In Figure 4.3 this can be seen at the second vertex from the left where the path in normal form depicted on the lower line has this property while the corresponding vertex of the upper line does not satisfy it. Furthermore we have that if component i is not affected by a synchronization later on the path then the i-th component does not change anymore.

It is now easy to see that if $\pi := \bar{v}_1 a_1 \bar{v}_2 a_2 \dots a_m \bar{v}_{m+1}$ is a path through G in normal form and $\bar{v}_j \restriction \mathrm{dom}(\pi[j]) = \bar{v}_k \restriction \mathrm{dom}(\pi[j])$ for some $j < k$ then $\bar{v}_j = \bar{v}_k$ and hence

$$\pi' := \bar{v}_1 a_1 \bar{v}_2 a_2 \dots a_{j-1} a_j \bar{v}_j a_k \bar{v}_{k+1} a_{k+1} \dots a_m \bar{v}_{m+1}$$

is also a path through G and thus the number of synchronizing edges on any path through G can be bounded by $\Pi_{1 \leq i \leq n}(|V_i(C)| + 1)$.

Now we can turn to the proof of the main result of this section.

Proof of Theorem 4.5. We show by induction that for every FO(R)-formula $\varphi(x_1, \dots, x_m)$ over Σ there are finite sets Ψ_i of Σ_i-formulas and a Boolean formula α over predicates $p_i(\psi^i_j)$ $(1 \leq i \leq n,\ 1 \leq j \leq |\Psi_i|)$ such that the equivalence stated in (4.5) holds for all $\bar{v}_1, \dots, \bar{v}_m \in V$. Under the premise of Part (b) all constructions presented below are effective.

We start with the atomic formulas. For $x = y$ let $\psi_i := (x = y)$ for every $1 \leq i \leq n$, for $E_a xy$ with $a \in \Sigma^l_i$ let $\psi_i := E_a xy$ and $\psi_j := (x = y)$ for $i \neq j$, and for $E_{\bar{c}} xy$ with $\bar{c} \in C$ let $\psi_i := E_{c_i} xy$ for every $1 \leq i \leq n$. For this last case recall that $E_\varepsilon xy$ is equivalent to $x = y$. For every formula above let $\alpha := \bigwedge_{1 \leq i \leq n} p_i(\psi_i)$. Obviously (4.5) holds in all cases, so the remaining atomic formulas we have to take care of are of the form $\mathrm{Reach}_\Gamma(x, y)$ for $\Gamma \subseteq \Sigma$.

Let $C' := \Gamma \cap C$. W.l.o.g. we may assume that Γ comprises all local edge labels, i.e. that $\Sigma^l \subseteq \Gamma$. Otherwise in the following every occurrence of Σ^l_i has to be replaced by $\Sigma^l_i \cap \Gamma$. Let $k := \Pi_{1 \leq i \leq n}|V_i(C') + 1|$. For every $i \in \mathrm{dom}(C')$, for every $1 \leq m \leq k$ and every mapping $\sigma : \{1, \dots, m\} \to C'$ we define a formula

$$\begin{aligned}\psi^i_{(m,\sigma)}(x, y) := \exists z_1 \dots \exists z_{m+1} \Big(& \mathrm{Reach}_{\Sigma^l_i}(x, z_1) \wedge y = z_{m+1} \\ & \wedge \bigwedge_{1 \leq j \leq m} \exists w \big(E_{\sigma(j)_i} z_j w \wedge \mathrm{Reach}_{\Sigma^l_i}(w, z_{j+1}) \big) \Big)\end{aligned}$$

where $\sigma(j)_i = c_i$ if $\sigma(j) = (c_1, \dots, c_n) \in C'$. The formula $\psi^i_{(m,\sigma)}$ expresses that on the path from x to y in component i exactly m vertices $z_1, \dots, z_m$ are passed such that from each z_j a synchronizing edge whose label is determined by $\sigma(j)$ is traversed. For $i \notin \mathrm{dom}(C')$ we define

$$\psi^i_{(m,\sigma)}(x, y) := \mathrm{Reach}_{\Sigma^l_i}(x, y)$$

independent of m and σ. Let

$$\alpha := \bigvee_{1 \leq m \leq k} \bigvee_{\sigma:\{1,\ldots,m\} \to C'} \bigwedge_{1 \leq i \leq n} p(\psi^i_{(m,\sigma)}).$$

By the arguments given before the proof we know that $G \models \mathrm{Reach}_\Gamma[\bar{v}, \bar{w}]$ iff there is a path from $\bar{v}$ to $\bar{w}$ in G on which a synchronized transition is passed at most k times. Thus (4.5) holds for all atomic formulas.

Formulas composed by Boolean connectives and existential quantification are now easy to handle.

The case of Boolean connectives may be solved in the standard way. Let $\varphi_1(\bar{x})$ and $\varphi_2(\bar{y})$ be FO(R)-formulas and α_1, $(\Psi^1_i)_{1 \leq i \leq n}$ as well as α_2, $(\Psi^2_i)_{1 \leq i \leq n}$ be given by the induction hypothesis. Then, for $\neg\varphi_1(\bar{x})$ we can choose the same $(\Psi^1_i)_{1 \leq i \leq n}$ and the Boolean formula to be $\neg\alpha_1$, and for $\varphi_1(\bar{x}) \vee \varphi_2(\bar{y})$ we choose $\Psi_i := \Psi^1_i \cup \Psi^2_i$ and $\alpha = \alpha_1 \vee \alpha_2$.

To finish the proof let $\varphi(x_1, \ldots, x_n) := \exists x_{n+1} \varphi_1(x_1, \ldots, x_{n+1})$. Let Ψ^1_i and α_1 be the formulas computed for $\varphi_1(x_1, \ldots, x_{n+1})$. Let $\mathcal{I}$ be the set of all satisfying assignments for α_1. For every $I \in \mathcal{I}$ let $I_i := \{j \mid I(p(\psi^i_j) = \text{true}\}$. The sets Ψ_i for $1 \leq i \leq n$ are constructed by adding for every $I \in \mathcal{I}$ the formula

$$\psi^i_I(x_1, \ldots, x_n) := \exists x_{n+1} \Big(\bigwedge_{j \in I_i} \psi^i_j \wedge \bigwedge_{j \notin I_i} \neg\psi^i_j \Big).$$

Then we can define $\alpha := \bigvee_{I \in \mathcal{I}} \bigwedge_{1 \leq i \leq n} p(\psi^i_I)$. □

Recall that FO(Reg) extends FO(R) by allowing reachability predicates of the form Reach_r where r is a regular expression over a set of edge labels Σ.

The predicates Reach_r where r is a regular expression built from sets Γ^*_i with $\Gamma_i \subseteq \Sigma$ using $\cdot$ and $+$ are definable in FO(R). Let FO(Reg′) denote the restriction of FO(Reg) to formulas in which all regular expressions in the reachability predicates are of this form. We immediately obtain the following Corollary.

Corollary 4.6 *If G is a finitely synchronized product of a family $(G_i)_{1 \leq i \leq n}$ with decidable FO(R)-theories, then the FO(Reg′)-theory of G is also decidable.*

□

In Section 4.3 we show that if reachability predicates with regular expressions of the form $(\Gamma_1 \cdot \Gamma_2)^*$ are allowed, the decidability of the corresponding theory is lost.

Theorem 4.5 reduces the evaluation of an FO(R)-formula φ in a product graph to the evaluation of families Ψ_i of formulas in the component graphs. Note that the reduction is nonuniform in the sense that the sets Ψ_i not only depend on the formula φ to be evaluated but also on properties of the graph family.

Under the premise that a bound on the number of vertices of every component graph which may participate in a synchronization can be computed, this reduction is effective. Thus it is possible to combine existing model checking algorithms for FO(R) for the component graphs into a model checking algorithm for FO(R) for the product. In particular there is no need to compute a finite representation of the product graph itself.

The complexity of the model checking algorithm which results from this approach certainly depends on the complexity of the model checking algorithms for the components. Since, given an interpretation of its predicates, a Boolean formula can be evaluated in linear time and linear space and since the Boolean formulas constructed above only depend linearly on the number of the formulas which have to be evaluated in the component graphs, this number of formulas and their size are reasonable parameters to estimate the complexity of the model checking algorithm for the product.

We present such a complexity analysis now. For all atomic formulas except the reachability predicates only a single atomic formula has to be evaluated in every component. For a reachability predicate Reach_Γ the number of formulas to be evaluated in the components depends on the number $k := \Pi_{1 \leq i \leq n}(|V_i(C)| + 1)$ of vertices which may participate in a synchronization. Since there are $|C|^m$ many mappings $\sigma : \{1, \ldots, m\} \to C$, the number of formulas $\Psi^i_{(m,\sigma)}$ in ψ_i can be bounded by $\sum_{0 \leq m \leq k} |C|^m \in O(|C|^k)$ and their length depends only linearly on k.

To evaluate a formula $\neg\varphi$ in the product we have to evaluate the same formulas in the components as for φ. To evaluate $\varphi \vee \psi$ we have to evaluate all formulas which are necessary to evaluate φ respectively ψ which yields only a linear increase in the number of formulas to be evaluated while their length stays the same.

The main contribution to the complexity of the model checking algorithm for the product stems from the treatment of quantifiers. The consideration of all satisfying assignments in this step leads to an exponential increase of the number of formulas which have to be evaluated, and their maximal length increases by factor m if m formulas were present in a set Ψ_i before.

Thus given a formula φ of quantifier rank q, $O(\exp_q(c))$ many formulas of length up to $O(c^{2^q})$ have to be evaluated in the components. Here c is a constant which depends exponentially on the number $k := \Pi_{1\leq i\leq n}(|V_i(C)|+1)$ of synchronizing vertices in the product graph.

Overall this yields a non-elementary increase of the formula complexity of the model checking algorithm for finitely synchronized products in comparison to the complexity of the model checking algorithms for the components.

4.3 Undecidability Results

In this section we show that the decidability result for FO(R) and finitely synchronized products is optimal in the following sense. Firstly, if the logic under consideration allows the formalization of regular path descriptions then already asynchronous (and hence finitely synchronized) products do not preserve the decidability of the corresponding theory anymore. Secondly, if we allow the synchronization of graphs to be semi-finite as defined below, then also the decidability of FO(R) is lost.

The undecidable problems we use for our reductions are the reachability problem for universal Turing machines and a disguise of it for pushdown automata with two stacks. We have to introduce some additional notation, see [HU79] for more details.

A *deterministic Turing machine* is a tuple $M = (Q, \Gamma, q_0, q_f, \delta)$ where Q is a finite set of states, Γ is an alphabet containing a designated blank symbol $\llcorner\!\lrcorner$, q_0 is the initial state, q_f is the halting state, and $\delta : Q\times\Gamma \to Q\times \Gamma\times\{L, R\}$ is the transition function. We assume that the Turing machine is normalized, i.e. that every state is reachable from the initial state, the only sink is the final state q_f and there are no incoming transitions to q_0. A *configuration* of M is a sequence $a_1, \ldots a_k, q, b_l, b_{l-1} \ldots b_1$ where $a_i, b_i \in \Gamma$, $q \in Q$ and b_l denotes the symbol currently read by the head of the machine.

Turing machines can be coded as words over $\{0,1\}$. A *universal Turing machine* is a Turing machine which is started on an input $v\$w\#$ where v codes a Turing machine M and w codes a configuration of M. The universal Turing machine simulates the behavior of M on w and reaches a halting state q_f iff M reaches a halting state from the configuration w.

The reachability problem for a universal Turing machine is the following:

Input:	A code v of a Turing machine M, a code w of a configuration of M.
Problem:	Does the universal Turing machine reach a halting state?

This problem obviously generalizes the halting problem for Turing machines and is therefore undecidable.

A *pushdown automaton with two stacks* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, q_f)$ where Q is a finite set of states, q_0 is the initial state, f is the final state, and $\Delta \subseteq Q \times \Sigma \times (\Gamma \cup \{\varepsilon\})^2 \times (\Gamma^*)^2 \times Q$. We abbreviate pushdown automata with two stacks by 2-PDA. Configurations and the configuration graph $\mathcal{C}(\mathcal{A})$ are defined analogously to the ones for pushdown automata in Subsection 2.2.1. Again we assume that the 2-PDA is normalized in the same way as the Turing machine above is.

It is easy to simulate a Turing machine M with a 2-PDA because every configuration $a_1, \ldots a_k, q, b_l, b_{l-1} \ldots b_1$ of M can be recorded in the two stacks of the 2-PDA. The content $a_k, \ldots, a_1$ of the work tape left to the head of M is stored in the first stack with a_k on top and the state q of M and the contents $b_l, \ldots, b_1$ of the work tape underneath and right to the head of M in the second stack with q, b_l on top. Since transitions of M modify a configuration only locally at positions which are stored on the top of the two stacks, the 2-PDA can simulate the Turing machine M.

We thus immediately obtain that there is also a universal 2-PDA $\mathcal{A}$ which simulates the behavior of a universal Turing machine. Started on an input consisting of a code v of a Turing machine M and a code w of a configuration of M it first copies $v\$w\#$ into the first stack with the last symbol of w on top and then moves $v\$w\#$ into the second stack thereby reversing the order such that the state and the first symbol of

v are now on top. These stacks code the initial configuration of the universal Turing machine.

For this 2-PDA the following reachability problem is again undecidable.

Input:	A code v of a Turing machine M, a code w of a configuration of M.
Problem:	Does the 2-PDA reach a halting configuration after processing $v\$w\#$?

We now turn to the undecidability proofs.

Theorem 4.7 *Asynchronous products do not preserve the decidability of the FO(Reg)-theory.*

Proof. We reduce the reachability problem for an universal pushdown automaton with two stacks to the model-checking problem for FO(Reg) over asynchronous products of graphs with decidable FO(Reg)-theories.

Let $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, q_f)$ be a normalized universal 2-PDA as described before. To be able to define the configuration of $\mathcal{A}$ when the simulation of the universal Turing machine starts we assume that the state set Q is partitioned into $Q := Q_1 \cup Q_2$ where the states in Q_1 are solely used to create the initial stack content as described above. We furthermore assume that there is a single transition which leads from Q_1 to a state $q_0^2 \in Q_2$ which is labeled by the unique symbol $\#$.

To reduce this reachability problem for $\mathcal{A}$ to the model-checking problem for FO(Reg) over asynchronous products of graphs with decidable FO(Reg)-theory we split $\mathcal{A}$ into two component pushdown automata

$$\mathcal{A}_1 = (Q, \Sigma \times \Delta, \Gamma, q_0, \Delta_1, q_f)$$
$$\mathcal{A}_2 = (Q, \bar{\Sigma} \times \bar{\Delta}, \Gamma, q_0, \Delta_2, \bar{q}_f)$$

where for every $\delta = (q, a, \gamma_1, \gamma_2, \gamma_3, \gamma_4, p) \in \Delta$ the following transitions are added:

$$(q, (a, \delta), \gamma_1, \gamma_3, p) \text{ to } \Delta_1,$$
$$(q, (\bar{a}, \bar{\delta}), \gamma_2, \gamma_4, p) \text{ to } \Delta_2.$$

The graphs generated by $\mathcal{A}_1$ and $\mathcal{A}_2$ have a decidable MSO-theory and therefore also their FO(Reg)-theory is decidable. Let $\mathcal{B}$ their asynchronous product.

Let r be the regular expression $r = \left(\bigvee_{\substack{\delta\in\Delta\\ a\in\Sigma}}(a,\delta)(\bar{a},\bar{\delta})\right)^*$ which states that a transition of $\mathcal{A}_1$ is followed by the corresponding transition of $\mathcal{A}_2$.

By construction we obtain that

$$\mathcal{B} \models \mathrm{Reach}_r(x,y)[((q,[u]),(q,[v])),((q',[u']),(q'',[v']))]$$

iff $q' = q''$ and $\mathcal{A}$ can reach from configuration $(q,[u],[v])$ the configuration $(q',[u'],[v'])$.

It is now easy to construct for every input $v\$w\#$ to the reachability problem a first-order formula $\varphi_{v\$w\#}(x,y)$ such that

$$\mathcal{B} \models \varphi_{v\$w\#}(x,y)[((q_0,[\varepsilon]),(q_0,[\varepsilon])),((q,[u_1]),(q,[u_2]))]$$

iff $u_1 = \varepsilon$, $u_2 = v\$w\#$ and $q = q_0^2$. Then we obtain that

$$\begin{aligned}\mathcal{B} \models \exists z_1 \exists z_2 \exists z_3 \Big(&\varphi_{v\$w\#}(((q_0,[\varepsilon]),(q_0,[\varepsilon])),z_1) \wedge \mathrm{Reach}_r(z_1,z_2)\\ &\wedge \bigvee_{\substack{\delta\in\Delta\\ a\in\Sigma}} \left(E_{(a,\delta)}z_2z_3 \wedge E_{(\bar{a},\bar{\delta})}z_3((q_f,[u]),(q_f,[v]))\right)\Big)\end{aligned}$$

iff $\mathcal{A}$ reaches a halting configuration after processing $v\$w\#$. Note that since $\mathcal{A}$ is normalized we can ensure that the initial configuration $((q_0,[\varepsilon]),(q_0,[\varepsilon]))$ and all final configurations $((q_f,[u]),(q_f,[v]))$ are first-order definable. $\square$

Next we introduce semi-finitely synchronized products which slightly generalize the finitely synchronized products by allowing that in at most one component graph infinitely many vertices participate in a synchronization.

Let $(G_i)_{1\leq i\leq n}$ be a family of graphs and C be a set of synchronization constraints. Recall that for $1 \leq i \leq n$ and $\bar{c} = (c_1,\ldots,c_n) \in C$ the set $V_i(\bar{c}) := \{v \in V_i \mid \exists w\ (v,w) \in E_{c_i}^{G_i}\}$ denotes the set of vertices in G_i which may participate in a synchronization with $\bar{c}$, and that $V_i(C) := \bigcup_{\bar{c}\in C} V_i(\bar{c})$. Let G be the synchronized product of $(G_i)_{1\leq i\leq n}$ with respect to C. We call G *semi-finitely synchronized* if $V_i(C)$ is finite for all but one $i \in \{1,\ldots,n\}$.

To show that semi-finite synchronization does not preserve the decidability of the FO(R)-theory we directly reduce the reachability problem for a universal Turing machine to the model checking problem for FO(R) over a (fully) synchronized product of a finite graph and an infinite graph. The infinite graph is generated by a ground tree rewriting systems which have been defined in Subsection 2.2.4. Ground tree rewriting graphs enjoy a decidable FO(R)-theory [DT90].

For the reduction we will define a ground tree rewriting system $\mathcal{R}$ such that the graph of $\mathcal{R}$ will encode all computations of the universal Turing machine M, but which will also contain paths which do not encode valid computations, i.e. paths which contain a least one pair of successive vertices which do not encode successive configurations of M. We will use the synchronization with a finite graph to eliminate computations which are not valid.

The construction of the ground tree rewriting graphs graph encompassing computations of M follows ideas of [Löd03].

Theorem 4.8 *Semi-finite synchronization does not preserve the decidability of the FO(R)-theory.*

Proof. We reduce the reachability problem for a universal Turing machine to the model-checking problem for FO(R) over a semi-finitely synchronized product of graphs with decidable FO(R)-theories.

Let $M = (Q, \Gamma, q_0, q_f, \delta)$ be a universal deterministic Turing machine. We assume that $q_0 \neq q_f$, $Q \cap \Gamma = \emptyset$, $X \notin Q \cup \Gamma$ and encode a configuration $a_1, \ldots, a_k, q, b_l, b_{l-1}, \ldots b_1$ of M by a tree

$$\begin{tikzcd}
& \bullet \arrow[dl, dash] \arrow[dr, dash] & \\
X \arrow[d, dash] & & X \arrow[d, dash] \\
a_1 \arrow[d, dotted, dash] & & b_1 \arrow[d, dotted, dash] \\
a_k & & b_l \arrow[d, dash] \\
& & q
\end{tikzcd}$$

Every transition of the Turing machine will be simulated by the rewriting system in two steps, by first rewriting the right branch of the configuration tree, and then rewriting the left branch. The labels of the rewriting rules will indicate which letter from Γ has to be added $(+)$ or removed $(-)$ from the left branch of the configuration tree, and $\top$

respectively $\bot$ indicate whether the halting state has been reached or not.

We define a ground tree rewriting system $\mathcal{R} = (A, \Sigma, R, t_0)$ where $A_2 = \{\bullet\}$, $A_1 = \Gamma \cup \{X\}$, $A_0 = A_1 \cup Q$, $\Sigma = \{+,-\} \times (\Gamma \cup \bar{\Gamma}) \times \{\bot, \top\}$ and

$$t_0 := \begin{tikzcd}[row sep=small, column sep=small] & \bullet \arrow[dl, dash] \arrow[dr, dash] & \\ X & & X \arrow[d, dash] \\ & & q_0 \end{tikzcd}.$$

The set of rewriting rules R is defined by adding for $\delta(q, b) = (p, c, L)$ and every $a \in \Gamma$ the rules

$$\begin{array}{c} b \\ \shortmid \\ q \end{array} \xrightarrow{(-,a,*)} \begin{array}{c} c \\ \shortmid \\ a \\ \shortmid \\ p \end{array} \text{ and } \begin{array}{c} X \\ \shortmid \\ q \end{array} \xrightarrow{(-,a,*)} \begin{array}{c} X \\ \shortmid \\ c \\ \shortmid \\ a \\ \shortmid \\ p \end{array} \text{ if } b = \textvisiblespace,$$

and for $\delta(q, b) = (p, c, R)$ with $p \neq q_f$ and every $a \in \Gamma$ the rules

$$\begin{array}{c} a \\ \shortmid \\ b \\ \shortmid \\ q \end{array} \xrightarrow{(+,c,*)} \begin{array}{c} a \\ \shortmid \\ p \end{array} \text{ and } \begin{array}{c} X \\ \shortmid \\ q \end{array} \xrightarrow{(+,c,*)} \begin{array}{c} X \\ \shortmid \\ p \end{array} \text{ if } b = \textvisiblespace$$

where $* = \top$ if $p = q_f$ and $* = \bot$ otherwise. Note that these rules can only be applied to the right branch of a configuration tree. For the left branch we add for every $a, c \in \Gamma$ and $* \in \{\bot, \top\}$ the rules

$$a \xrightarrow{(-,\bar{a},*)} \varepsilon \text{ and } X \xrightarrow{(-,\bar{a},*)} X \text{ if } a = \textvisiblespace,$$

as well as

$$a \xrightarrow{(+,\bar{c},*)} \begin{array}{c} a \\ \shortmid \\ c \end{array}$$

and

$$X \xrightarrow{(+,\bar{c},*)} \begin{array}{c} X \\ \shortmid \\ c \end{array} \text{ if } c \neq \textvisiblespace \text{ and } X \xrightarrow{(+,\bar{c},*)} X \text{ if } c = \textvisiblespace.$$

By construction, a path through the graph G generated by $\mathcal{R}$ corresponds to a valid computation of M started on the empty tape iff every transition with label $(+, a, *)$ respectively $(-, a, *)$ is followed by its counterpart labeled $(+, \bar{a}, *)$ respectively $(-, \bar{a}, *)$. Let H be the star graph with $|\Sigma|+1$ many vertices where the center vertex v_c has for every $(\$, a, *) \in \{+,-\} \times \Gamma \times \{\bot, \top\}$ a single outgoing edge with this label to

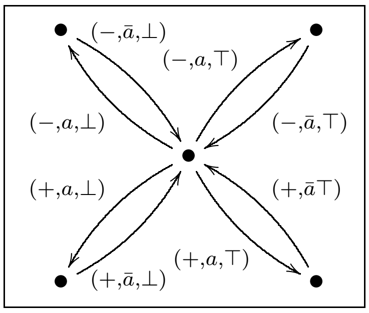


Figure 4.4: Star graph ensuring valid computations

a vertex w and the single corresponding incoming edge from w labeled $(\$, \bar{a}, *)$. Figure 4.4 shows the vertices and edges added for every $a \in \Gamma$.

If we define the synchronization constraint $C := \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$, the synchronized product of G and H with respect to C will contain exactly the valid computations of M.

To decide whether M halts on an input $v\$w\#$ we assume that M first scans its input and then returns to the beginning of it to start the simulation. Then similarly to the proof of Theorem 4.7 we can construct a first-order formula $\varphi_{v\$w\#}(x, y)$ which ensures that the input to M was $v\$w\#$.

Thus we obtain that to decide whether M reaches a halting configuration when started on input $v\$w\#$ we have to check the truth of the formula

$$\exists x \exists y \exists z \big(\varphi_{v\$w\#}((t_0, v_c), x) \wedge \mathrm{Reach}_\Sigma(x, y) \wedge \bigwedge_{\sigma \in \{+,-\} \times \bar{\Gamma} \times \{\top\}} E_\sigma yz\big)$$

in the semi-finitely synchronized product of G and H. Note that since we assumed that M is normalized the vertex (t_0, v_c) consisting of the initial tree t_0 and the center vertex v_c of the star graph is the only vertex in the synchronized product without an incoming edge. It is thus first-order definable. □

Chapter 5

Conclusion

In this thesis we investigated several classes of finitely representable infinite graphs and decision problems for them. In particular we were interested in identifying classes of graphs for which the model checking problem for a logic $\mathcal{L}$ in which reachability can be expressed is decidable.

We investigated three approaches to finitely represent graphs:

- the internal approach where graphs are presented by generating devices.
- the transformational approach where graphs are represented by a sequence of compatible operations applied to an initial finite (or finitely representable) graph.
- the compositional approach where graphs are represented by a family of finite (or finitely representable) graphs and compatible composition operations.

In Chapter 3 we studied the hierarchy of graphs generated by higher-order pushdown systems. We characterized the higher-order pushdown graphs by showing that for every $n > 0$ the following statements are equivalent.

(1) A graph G is generated by a higher-order pushdown system of level n.

(2) G can be obtained from a finite graph by an n-fold iterative application of unfolding operations and inverse rational mappings.

(3) G can be obtained from a finite graph by an n-fold iterative application of tree-graph operations and monadic second-order transductions.

In particular, by the equivalence of (1) and (2) we established the equality of the class of higher-order pushdown graphs and the class of graphs in the Caucal hierarchy and thus gave an internal representation of the latter graphs originally defined in a transformational way.

The equivalence of (2) and (3) was proved by showing that every level of the graph hierarchy as defined in (2) is closed under monadic second-order transductions and that the application of a treegraph operation to a graph of level n leads to a graph of level $n+1$.

The most complicated part was to show equivalence of (1) and (2). We made use of these closure properties and a variant of higher-order pushdown systems we introduced. This variant is equipped with a built-in test on the equality of the two top level k stacks for every $k \geq 1$.The main technical difficulty in the proof of the equivalence of (1) and (2) then was to establish that higher-order pushdown systems and higher-order pushdown systems with equality-pop generate the same classes of graphs.

The equality of the graph classes allowed to transfer results obtained in the different settings in either direction. For example, in one direction we concluded that the monadic second-order theory of higher-order pushdown graphs is decidable, and using the other direction we obtained that the Caucal hierarchy of graphs is strict level by level.

In Chapter 4 we investigated synchronized products as composition operations. We first considered the extension FO(TC)$_{(1)}$ of first-order logic with transitive closure operators of arity one as a classical logic in expressive power between first-order logic and monadic second-order logic.

We showed that FO(TC)$^2_{(1)}$ where the nesting of transitive closure operators is restricted to depth at most two is undecidable over the infinite grid while for FO(TC)$^1_{(1)}$ without nesting the corresponding theory is decidable. The first result was established by a reduction from the arithmetic of the positive integers, while the second result was obtained by a reduction to Presburger arithmetic. The undecidability result in particular implies that the formation of an asynchronous product is not FO(TC)-compatible.

For this reason we considered the less expressive logic FO(R) in Section 4.2. We introduced a semantic restriction of the power of the synchronization operations and showed that the decidability of FO(R) is preserved for finitely synchronized products. This result was established by providing a composition theorem which reduces the evaluation of a formula in the product to the evaluation of finitely many formulas in the components. We showed that this result is optimal in the following sense: If either the expressive power of the logic is increased to allow regular reachability constraints or the constraint posed on the synchronization operations is slightly liberalized, then the decidability of the corresponding logic is not preserved anymore.

Open Questions and Further Research

The class of higher-order pushdown graphs deserves further studies. While we have a clear picture of the graphs of level 1 of the hierarchy, almost nothing is known for higher levels. Muller and Schupp gave a graph theoretic characterization of the configuration graphs of pushdown automata by considering their "ends".

Question *Is there a theory of ends of configuration graphs of higher-order pushdown systems?*

Such a graph theoretic property could possibly help to develop methods to separate the classes of the higher-order pushdown graphs. Currently we rely on the complexity theoretic approach by J. Engelfriet [Eng91]. Recently there has been some progress in this direction. In [Blu] A. Blumensath showed a pumping lemma for higher-order pushdown automata which is however not yet applicable for this purpose.

The question how the graph classes can be separated is tightly related to the problem of determining the lowest level to which a higher-order pushdown graph belongs.

Question *Is the level of a higher-order pushdown graph computable?*

We know from the example $T_{\exp_\omega}$ presented at the end of Section 3.5 that not every graph with a decidable monadic second-order theory is a higher-order pushdown graph. $T_{\exp_\omega}$ however can be seen as limit of a sequence of graphs which transcends every level of the hierarchy.

Question *Is there a suitable notion of a limit of a sequence of higher-order pushdown graphs such that limit graphs still enjoy a decidable monadic second-order theory?*

The precise complexity of the model checking problem for higher-order pushdown graphs is unknown, but at least known to be prohibitively high. For pushdown graphs there are efficient algorithms to compute the set of reachable states from a given configuration, see e.g. [BEM97]. These algorithms rely on the fact that such reachability sets are regular and thus can be finitely represented. A notion of higher-order regularity should be developed to enable the transfer of these techniques to higher levels.

Question *Are there subclasses of higher-order pushdown graphs and/or sublogics of MSO for which the model checking problem can be solved in reasonable time?*

A first result in this direction was recently obtained in [BM04].

In Chapter 4 we thoroughly investigated the limitations of compatible synchronized product operations. Other methods to define synchronized products should now be investigated, including a synchronized product which incorporates the special cases of synchronization of many identical components, as it appears in the setting of parameterized systems.

More generally one should investigate composition operations for graphs with respect to their compatibility with certain logics in order to extend the methods which can be used for modeling computational systems.

Question *Which graph operations are compatible with a logic that allows the formalization of reachability?*

Acknowledgement

There are many people who contributed to this thesis. First I thank my advisor Wolfgang Thomas for his great support. He raised the questions which initiated both the research of higher-order pushdown systems and synchronized products of graphs. I am also indebted to my coauthor Arnaud Carayol. We joined forces after we realized that we have been working on the same topic for some time. Without his knowledge Section 3.2 would definitely look different. Didier Caucal and Wolfgang Thomas made my collaboration with Arnaud possible by financing mutual research visits. I also thank Christof Löding with whom I shared an office during all these years. He was always interested in problems which appeared and many times he opened ways to their solution. He read a first version of this thesis and pointed to many errors, mistakes and inaccuracies. I also thank all other (sometimes former colleagues) for providing a pleasant and stimulating research atmosphere in our group, especially Philipp Rohde, Dietmar Berwanger, Thierry Cachat, Stephan Kreutzer and Antje Nowack. Finally I thank my wife Tina for all her inconspicuous support and especially her endurance with my bad mood when a proof went wrong.

Bibliography

[Aho69] A.V. Aho. Nested stack automata. *Journal of the Association for Computing Machinery*, 16:383–406, 1969.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Arn94] A. Arnold. *Finite Transition Systems*. Prentice Hall, 1994.

[Arn02] A. Arnold. Nivat's processes and their synchronization. *Theoretical Computer Science*, 281:31–36, 2002.

[Bar97] K. Barthelmann. On equational simple graphs. Technical Report 9/97, Universität Mainz, 1997.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[BG00] A. Blumensath and E. Grädel. Automatic structures. In *Proceedings of the 15th IEEE Symposium on Logic in Computer Science*, pages 51–62. IEEE Computer Society Press, 2000.

[BK] A. Blumensath and S. Kreutzer. An extension of Muchnik's theorem. Submitted.

[Blu] A. Blumensath. A pumping lemma for higher-order pushdown automata. Unpublished.

[Blu99] A. Blumensath. Automatic structures. Diploma Thesis, RWTH Aachen, 1999.

[Blu01] A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report AIB-2001-06, RWTH Aachen, 2001.

[Blu03] A. Blumensath. *Structures of Bounded Partition Width*. PhD thesis, RWTH Aachen, 2003.

[BM04] A. Bouajjani and A. Meyer. Symbolic reachability analysis of higher-order context-free processes. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 135–147. Springer, 2004.

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

[Cac03] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

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

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

[CC03] A. Carayol and T. Colcombet. On equivalent representations of infinite structures. In *Proceedings of the 30th International Colloquim on Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 599–610. Springer, 2003.

[CE81] E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer, 1981.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.

[CK02] B. Courcelle and T. Knapik. The evaluation of first-order substitution is monadic second-order compatible. *Theoretical Computer Science*, 281:177–206, 2002.

[Col02] T. Colcombet. On families of graphs having a decidable first order theory with reachability. In *Proceedings of the 29th International Conference on Automata, Languages, and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 98–109, 2002.

[Col04] T. Colcombet. *Représentations et propriétés de structures infinies*. PhD thesis, Université de Rennes 1, 2004.

[Cou90a] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, pages 194–242. Elsevier Science Publisher, 1990.

[Cou90b] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[Cou94] B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126:53–75, 1994.

[CT02] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. *Information and Computation*, 176:51–65, 2002.

[CW98] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.

[CW03] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

[Dam82] W. Damm. The IO and OI hierarchies. *Theoretical Computer Science*, 20:95–208, 1982.

[DG86] W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71:1–32, 1986.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proceedings of the fifth IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990.

[EF95] H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.

[EGNR98] Y.L. Ershov, S.S. Goncharov, A. Nerode, and J.B. Remmel. *Handbook of Recursive Mathematics*. North-Holland, 1998.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

[Eng91] J. Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95:21–75, 1991.

[FS93] C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108:45–82, 1993.

[FV59] S. Feferman and R.L. Vaught. The first-order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

[Gre70] S.A. Greibach. Full AFLs and nested iterated substitution. *Information and Control*, 16:7–35, 19070.

[Gro96] M. Grohe. Arity hierarchies. *Annals of Pure and Applied Logic*, 82:103–163, 1996.

[Gur85] Y. Gurevich. Monadic second-order theories. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, pages 479–506. Springer, 1985.

[Han65] W. Hanf. Model-theoretic methods in the study of elementary logic. In *Proceedings of the Symposium on the Theory of Models*, pages 132–145. North Holland, 1965.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *ACM Communications*, 21:666–677, 1978.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[KNU02] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pusdown trees are easy. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

[KP99] T. Knapik and É. Payet. Synchronized product of linear bounded machines. In *Proceedings of the 12th International Symposium on Fundamentals of Computation Theory*, volume 1684 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 1999.

[Löd02] C. Löding. Model-checking infinite systems generated by ground tree rewriting. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2002.

[Löd03] C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.

[Mak04] J.A. Makowsky. Algorithmic aspects of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126:159–213, 2004.

[Mas76] A.N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite State Systems*. PhD thesis, TU München, 1998.

[Mil85] R. Milner. Lectures on a calculus for communicating systems. In *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 197–220, 1985.

[Mor00] C. Morvan. On rational graphs. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures*, volume 1784 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2000.

[MP04] A. Montanari and G. Puppis. Decidability of MSO theories of tree structures. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*, pages 434–446. Springer, 2004.

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

[Pel01] D.A. Peled. *Software Reliability Methods*. Springer, 2001.

[Pre30] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa 1929*, pages 92–101, 1930.

[Pre91] M. Presburger. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *History and Philosophy of Logic*, 12:225–233, 1991. English translation of [Pre30].

[Rab] A. Rabinovich. Composition theorem for generalized sum. Submitted.

[Rab01] A. Rabinovich. On the compositional method and its limitations. Technical Report EDI-INF-RR-0035, University of Edinburgh, 2001.

[Rab05] A. Rabinovich. On compositionality and its limitations. To appear in ACM Transactions on Computational Logic, 2005.

[Sem84] A. Semenov. Decidability of monadic theories. In *Proceedings of the 11th International Symposium on Mathematical Foundations of Computer Science*, volume 176 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 1984.

[She75] S. Shelah. The monadic theory of orders. *Annals of Mathematics*, pages 379–419, 1975.

[She96] S. Shelah. On the very weak $0-1$ law for random graphs with orders. *Journal of Logic and Computation*, 6:137–159, 1996.

[Sti] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Submitted.

[Tho97] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

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

[Tho03] W. Thomas. Constructing infinite graphs with a decidable MSO-theory. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003.

[Wal02] I. Walukiewicz. Monadic second order logic on tree-like structures. *Theoretical Computer Science*, 275:311–346, 2002.

[WT04] S. Wöhrle and W. Thomas. Model checking synchronized products of infinite transition systems. In *Proceedings of the 19th Annual Symposium on Logic in Computer Science*, pages 2–11. IEEE Computer Society, 2004.

Index

Lebenslauf

Zur Person

Name:	Stefan Wöhrle
geboren:	16. Februar 1973 in Offenburg
Familienstand:	verheiratet, ein Kind

Bildungsgang

1979–1983	Wilhelm-Hausenstein-Grundschule Hornberg
1983–1992	Schwarzwald-Gymnasium Triberg, Abitur
1993–1996 und 1997–2001	Studium der Mathematik mit Nebenfach Informatik Albert-Ludwigs-Universität Freiburg
1996–1997	Studium an der University of Washington, Seattle, Department of Mathematics Austauschprogramm der Albert-Ludwigs-Universität Freiburg, Fulbright-Stipendiat
Januar 2001	Diplom in Mathematik, Titel der Arbeit: *Lokalität in der Logik und ihre algorithmischen Anwendungen*
seit Februar 2001	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Informatik VII "Logik und Theorie diskreter Systeme" (Prof. Dr. Wolfgang Thomas) Rheinisch-Westfälische Technische Hochschule Aachen