Lecture Notes in Computer Science 1354
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Budapest
Hong Kong
London
Milan
Paris
Santa Clara
Singapore
Tokyo

Olaf Burkart

Automatic Verification of Sequential Infinite-State Processes

Springer

Author

Olaf Burkart
Lehrstuhl Informatik V, Universität Dortmund
Baroper Straße 301, D-44221 Dortmund
E-mail: burkart@ls.5.informatik.uni-dortmund.de

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Burkart, Olaf:
Automatic verification of sequential infinite-state processes / Olaf Burkart. - Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Santa Clara ; Singapore ; Tokyo : Springer, 1997
(Lecture notes in computer science ; Vol. 1354)
ISBN 3-540-63982-9

CR Subject Classification (1991): F.3.1, D.2.4, D.1.3, D.2.1

ISSN 0302-9743
ISBN 3-540-63982-9 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10661329 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

Software design and program development is widely recognised as a highly creative task. The reason for this understanding is that industrial-strength software has some inherent conceptual complexity which can, as in the case of concurrent, reactive systems, easily exceed the human intellectual capacity. Finding ways to master this complexity is therefore one of the major challenges in computer science today.

So far, a common approach in software engineering has been to apply during the design phase a variety of structured techniques like top-down design, decomposition, and abstraction in order to cope with the complexity of large software systems. Only after the design is completed does intensive testing in the implementation phase ensure reliability, usually understood as absence of program errors. However, this approach neglects the fact that central aspects of software design and program development have a strong formal character that in principle admits tool support for the construction of reliable and correct computer systems. A crucial precondition for the success of such a computer-aided effort is, in fact, the availability of methods and techniques which perform the required formal reasoning.

This monograph aims to provide the theoretical foundations needed for the verification of reactive, sequential infinite-state systems. In particular, we will develop two new algorithms that allow us to automatically verify important aspects, like safety or liveness properties, of the given infinite-state system. As we deal with infinite-state spaces, many theoretical topics are involved, including process algebras, fixpoint theory, modal logics, and model checking. To stress the importance of a sound foundation for the developed verification methods, we put particular emphasis on the presentation of the formal framework which we hope will be of use also for future extensions.

This monograph is a revised version of my doctoral dissertation which was submitted to the Faculty of Mathematics and Natural Sciences of the Rheinisch-Westfälische Technische Hochschule Aachen and accepted in July 1995.

I would like to thank my supervisor Bernhard Steffen who introduced me to the subject and provided many inspiring discussions that greatly influenced the contents of my thesis. I also want to thank him and his wife Tiziana for their kind hospitality during my several visits to Passau.

I also thank Klaus Indermark for his constant support as well as his enthusiasm during my employment at the Department of Computer Science at Aachen.

Thanks are also due to Didier Caucal, who after a discussion in Passau initiated my work on the bisimulation equivalence problem by pointing me to some of his previous research. I also want to thank him for his many useful comments that influenced much of the topic of Chapter 5, as well as for his kind support during my stay at IRISA in Rennes where we continued the work on the bisimulation problem for context-free processes.

I am also indebted to Colin Stirling who influenced implicitly, and after we met personally in Edinburgh also explicitly, the presentation of the model checking algorithm and its associated theory.

Finally, I am also grateful both to an anonymous referee and to Markus Schweighofer, who gave a number of hints on a draft version that helped to improve the final presentation.

Last but not least, warm thanks to my wife Simone for taking good care of me by providing constant emotional and practical support.

Dortmund, October 1997 *Olaf Burkart*

Foreword

After almost twenty years of concurrency theory, we face a wide spectrum of formalisms – process algebras, models for concurrency, and application specific languages – which all come with their specific expressive power and conceptual complexity. There is currently no sign that this tendency to diversity will end. In response to this state of the art, various tools have been developed, each addressing very specific scenarios on the basis of tailored methodologies. They provide keys to the practical use of all the described theory, as they have the potential to constitute an interface between theory and practice on a purely phenomenological level.

The typical development of such tools, which I am convinced provide a new momentum to concurrency theory and even to formal methods in general, goes through three sometimes overlapping phases: a *conceptual phase*, where the underlying decidability issues are studied, usually by pure mathematical reasoning, a *complexity-oriented realization phase*, where appropriate data structures and algorithms are designed and implemented, and first case studies are performed, and a *'civilisation' phase*, where people look at application scenarios and profiles, at appropriate interfacing to industrial environments and user communities, and where the investigation of the practical behaviour of algorithms on concrete applications gains importance over the usual worst case reasoning.

Currently, although the first 'civilised' formal-method based tools have become reality, the main effort is still invested in the first two phases, which still provide a huge potential for investigation.

The contributions of this monograph belong to the first and second phase. New concepts and algorithms are provided, for both model checking and equivalence checking, which drastically extend the scope of automatic verification, and which, nevertheless, have the potential to give directions for efficient implementation. In fact, the results of the underlying dissertation include the first effective model checking algorithm for infinite state systems, namely the class of context-free and pushdown processes, which can be regarded as procedural extensions of finite automata. The underlying second-order semantics is rather elegant and surprisingly efficient: context-free processes can be model checked essentially in time proportional to the size of the argument process. Only the size of the property to be checked is critical.

Moreover, as it has turned out in the meantime, second-order semantics allow comparatively simple extensions to formulae of higher alternation depth and to further generalised process calculi.

In addition, the first bisimulation checking algorithm of elementary complexity is presented, which covers all context-free processes. This algorithm, although extremely intricate and computationally expensive, paves the way towards efficient implementation: for the first time it allows an estimation of the worst case complexity of this class of processes. One may well expect that, as already experienced in the case of normed context-free processes, which over the years have been shown to admit polynomial bisimulation checking, the complexity result provided in this thesis will be drastically improved in the near future, making context-free bisimulation checking a tool of practical relevance.

All these contributions are based on an impressive collection of new algebraic theorems, which are interesting already on their own: they provide in fact a strong intuition about infinite state systems, properties of parallel composition, and differences between normed and unnormed processes, as well as between context-free and pushdown processes in a branching-sensitive scenario. Thus this monograph also provides a comprehensive overview of the foundations of infinite state verification for the considered classes of processes, and reveals essential differences between the new branching-sensitive theory and 'classical' automata and formal language theory.

Summarising, the reader will find elegant and deep theory as well as comprehensible algorithms, which I am convinced will be the key for a better understanding of process theory. In fact, I believe that corresponding implementations will enhance this understanding, and tuned versions will enter modern environments for concurrent system design in the near future in the general course of formal methods integration. This general trend has recently been observed for finite state model checking, which, like the now omnipresent type checking, quickly conquered the industrial hardware design arena. I am convinced that this trend will continue for other fully automatic verification techniques like the ones presented in this thesis. Thus this monograph provides a wealth of valuable information, both for pure theoreticians interested in algebraic theories and for tool builders who are open to conquering new ground in their desire to construct practically relevant tools.

Dortmund, November 1997 *Bernhard Steffen*

Contents

1. Introduction

Distributed, concurrent, and reactive systems, are playing an increasingly important role in computer science, both in theory and in practice. Software systems with these characteristics, among them communication protocols, database systems, and operating systems, tend to exhibit a rich set of behaviours, are inherently complex and therefore complicated to build. Consequently, a great deal of work has been done investigating formal techniques supporting the correct construction of concurrent systems, as well as their verification. The latter is often of particular interest, since concurrent systems are being used in e.g. safety-critical control systems and arise in the design of digital circuits. Verification can then be applied to guarantee that central aspects of the intended system behaviour, like e.g. safety or liveness properties, are correct with respect to their specifications. In contrast to pure testing, this methodology has the important benefit that it directly supports early error detection by revealing logical errors already in the design phase, as well as that it increases considerably the confidence into the correctness of the produced software.

Not surprisingly, the verification of programs has already a long history in computer science. Since Floyd [Flo67] it is known that sequential programs can be verified by considering their partial correctness, specified in terms of pre-/postcondition annotations, together with their termination. In this concept the input/output behaviour of sequential programs constitutes a predominant factor. Subsequently, Hoare [Hoa69] has recast Floyd's method and introduced a logical framework which admits to prove in a structured and compositional way that a sequential program meets its specification.

Reactive systems, on the other hand, are typically *nonterminating*, as they maintain an ongoing interaction with the environment. Hence, verification methods which rely on the existence of a final state are, usually, not applicable and must be replaced by radically different approaches. The fundamental concept of *invariance* which considers what remains true throughout the execution of a program is then the appropriate notion to use for reactive system. By investigating formalisms which support the specification of invariance properties the automated verification of concurrent, reactive systems has made great progress over the last decade (cf. [Lam94]) and a

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 1-7, 1997

number of tools exploiting the various paradigms for verification exist today (cf. [Mad92, CGL94]).

In this monograph we shall concentrate on two powerful techniques for the verification of reactive systems (or processes): *model checking* and *equivalence checking*. Both methods have successfully been applied, in particular, to finite-state systems since the finitary nature of these systems guarantees immediately the existence of effective verification procedures. Under these circumstances analysis and verification tools typically explore completely the reachable state space of the system at hand in order to prove it automatically correct. However, it is a fact of life that realistic systems are often infinite-state excluding them, in general, from any automated verification. Consequently, a great deal of research effort has been put into identifying restricted, but still expressive, classes of infinite-state spaces that possess a decidable verification problem.

This monograph aims to show that large classes of *sequential infinite-state processes* are amenable to automated verification by means of model checking, as well as equivalence checking.

1.1 Sequential Processes

From its beginning, one of the main objectives of concurrency theory has been the study of suitable models for concurrent and distributed systems. Notably the algebraic approach has proved valuable, and a number of powerful process calculi such as CSP [Hoa85], CCS [Mil89], or ACP [BW90] have emerged. All these calculi have in common that they consist of an algebraic syntax specifying the operators for constructing larger processes from smaller ones, some structural operational semantics defining the behaviour of processes in terms of labelled transition graphs, and one or more behavioural equivalences, like e.g. bisimulation or the coarser language equivalence, which reflect some notion of observation.

The purpose behind the various process calculi that have been proposed has been to capture, and thus to support the understanding, of such central aspects of concurrency, as e.g. parallelism, nondeterminism, or communication. However, the computational power of these calculi immediately delivers undecidability results for a broad range of properties. Therefore much attention has been devoted to the study of less powerful process models and their related verification problems.

In this monograph we shall particularly be interested in several classes of processes where sequential composition plays a major role. The least expressive class we would like to mention is the class of *regular processes* as introduced by Milner [Mil84] which contains all processes recursively defined in terms of the operations of nondeterministic choice '+' and prefixing '$a.$'. Prefixing is a restricted form of sequential composition since a process $a.P$ can be viewed as consisting of two processes: the one which terminates after

execution of the a-action, and the process denoted by P. The expressiveness of regular processes is necessarily limited since they may describe only finite-state systems, and hence, correspond to finite automata. Interpreted with respect to trace (language) semantics regular processes thus characterise the well-known class of regular languages.

Beyond the finite-state case, the next natural class of interest is the class of *context-free processes* modelled by Basic Process Algebra (BPA) [BW90], a subalgebra of ACP. Here prefixing is replaced by general sequential composition $P \cdot Q$ with the interpretation that the process denoted by Q may start reacting only after P has successfully terminated. This generalisation allows context-free processes to model, for instance, infinite counters or unbounded stacks, entities which can be found in many realistic systems. However, the enhancement to the expressive power gives also immediately rise to some undecidable properties since there is an obvious correspondence between transition graphs of context-free processes and leftmost derivations of context-free grammars. Consequently, the (completed) trace sets of context-free processes coincide with the class of context-free languages which is known to possess a number of undecidable properties itself.

One may now be led to wonder what the correspondence will look like for transition graphs of pushdown automata, called *pushdown processes* in the area of concurrency theory. Clearly, trace semantics is then of minor interest as context-free grammars and pushdown automata are equally expressive concerning the languages they characterise. However, it turns out that in the finer setting of bisimulation semantics the class of pushdown processes strictly includes the class of context-free processes [CM90], and therefore builds a class of its own interest.

In this monograph we shall explore in detail the class of pushdown processes by means of *Pushdown Process Algebra* (PDPA) [BS94], a process calculus we propose as the algebraic framework for this family of processes. Its main innovation is a generalised sequential composition operator incorporating an additional control component. We shall develop some algebraic properties for pushdown processes and will show that they are the smallest extension of context-free processes up to relabelling that is closed under synchronised parallel composition with finite-state processes.

1.2 Model Checking

One of the most promising approaches to the verification of concurrent systems is *model checking*. In this approach, one uses formulas of a temporal logic to specify the desired properties of a system and a decision procedure then determines automatically whether the start state of the system in question satisfies the given formulas. Although theoretically model checking is always applicable to finite-state systems, since an exhaustive traversal through the reachable state space of the system at hand can effectively provide enough

information to solve the verification problem, there exist practical limits in terms of the system size which can be handled. Unfortunately, systems made up of several concurrent components can easily reach these limits as the size of their global state space is, usually, exponential in the number of components. Despite this state explosion problem, for finite-state systems a number of automatic verification tools supporting a variety of model checking algorithms for different temporal logics have successfully been developed over the last decade [Mad92, CGL94].

Whereas model checking for finite-state systems is already well-established (cf. e.g. [EL86, CES86, Lar88, SW89, Win89, Cle90, CS92]), the theory for infinite-state spaces is a current research topic. Bradfield and Stirling [Bra91, BS92a], for instance, observed that tableaux-based model checking covers general infinite-state systems. However, their method is due to its generality not always effective. Therefore much research [CHS92, CHM93, HS93, BER94, Esp94, EK95] has aimed at identifying restricted classes of infinite-state processes which still allow *automatic* verification.

One of the central results so far is the paper of Muller and Schupp [MS85] who proved that the theory of monadic second-order logic (MSOL) is decidable for the class of pushdown transition graphs. As a consequence, the model checking problem on pushdown transition graphs is decidable for the full modal μ-calculus, a powerful temporal logic which can be interpreted in MSOL. However, their decision procedure is nonelementary and thus not applicable to practical problems.

We contribute to this field of research by developing an iterative model checking algorithm that decides the *alternation-free* part of the modal μ-calculus for pushdown processes [BS92b, BS94] in exponential time. The point of our algorithm is to consider a *second-order* variant of the standard iterative model checking techniques which operates on the Fisher–Ladner closure [FL79] of a given formula. It determines *property transformers* for the *fragments* of the pushdown automaton, which describe the set of subformulas that are valid at the start state of a fragment relative to the set of subformulas that are valid at its end states. Here the number of end states of a fragment, which actually coincides with the number of states of the finite control of the underlying pushdown automaton, determines the arity of the corresponding property transformer. Our new equation system-based algorithm elegantly realizes the corresponding computation. After the determination of these property transformers, the model checking problem can easily be decided. We simply check whether the formula under consideration is a member of the set of subformulas that results from applying the property transformer associated with the initial fragment of the pushdown automaton to the set of subformulas that are valid at the end states.

Having settled the model checking problem for pushdown processes we shall prove that the set of states of a pushdown transition graph satisfying a formula of the μ-calculus is always regular. This result is a simple consequence

of the compositionality of second-order semantics and relies on an automata-based construction.

1.3 Equivalence Checking

Equivalence checking is a powerful alternative to model checking for the verification of concurrent processes. It is based on the central notion of observational equivalence which captures formally when two processes are said to exhibit the same behaviour (cf. [Gla90]). Equivalence checking assumes that the system `SYS`, as well as the specification `SPEC`, are given as process descriptions, usually, however, at different levels of abstraction. The system `SYS` is then proved correct with respect to the specification `SPEC` by simply showing $\texttt{SYS} \equiv \texttt{SPEC}$ for the equivalence $\equiv$ of interest. The observation that for finite-state systems all reasonable equivalences are known to be decidable led to the development of a number of software tools supporting the automatic verification of such systems [Mad92, CGL94]. However, when considering infinite-state spaces completely different methods are required, as the standard equivalence checking algorithms which are mainly based on exhaustive decision procedures fail to work, and decidability becomes an important issue.

For example, from classical language theory it is known that language equivalence is decidable for regular systems, whereas the problem becomes undecidable if one moves further up in the Chomsky-hierarchy to the class of context-free languages. In the "finer" process algebraic setting the situation is, however, different, as e.g. *bisimilarity* is decidable for *normed* context-free processes (BPA processes) [BBK87a], which are context-free processes that can terminate in finitely many steps at any point of the execution. This exceptional property of the bisimulation equivalence led to an intense investigation (cf. [Cau90, Gro91, HS91, HT94, HM94]), which resulted in a polynomial time decision procedure for normed context-free processes [HJM94].

When trying to generalise these results to the unnormed case where also nonterminating processes are allowed it turns out that again completely new techniques are required, as the decomposition properties for the normed case fail to hold. Nevertheless, considering *bisimulation bases* B characterising the bisimulation equivalence as the least congruence w.r.t. sequential composition containing B, Christensen, Hüttel and Stirling proved in [CHS92] that bisimulation is decidable also for unnormed context-free processes. The existence of such a finite relation B can be exploited for a decision algorithm based on two semi-decision procedures: one for enumerating all possible relations B which may contain the pair (α, β), and one for the enumeration of all non-bisimilar pairs of processes.

In this monograph we shall improve on the result of Christensen, Hüttel and Stirling by showing how to compute recursively a bisimulation base B, and exploit it for the construction of an elementary bisimulation decision

procedure for arbitrary context-free processes [BCS95]. The key idea behind the construction of B is the determination of a new bound for the number of transitions needed to separate two normed non-bisimilar processes along the lines of [Cau89]. This bound allows the construction of an "initial" base, which subsequently must be refined by means of a fixpoint iteration similar to the one used in [HJM94] to obtain B. The decision algorithm is then completed by a straightforward branching algorithm.

1.4 Organisation of This Book

In this book we address the analysis and verification of infinite-state systems by means of syntactical, logical and semantical methods. This classification is also reflected in the organisation of this book which is in detail as follows.

In *Chapter 2* we lay the background for our work. We summarise the basic definitions and properties concerning ordered sets, lattice theory and fixpoint theorems which will be needed when developing our model checker. Moreover, we briefly introduce some notions from the theory of rewrite systems which are important for our bisimulation decision procedure for arbitrary context-free processes. Subsequently, we give the central facts about context-free languages allowing to compare the setting of formal language theory with the framework of process theory for context-free structures. Next we shall introduce labelled transition graphs as our computational model and bisimulation as the behavioural equivalence we are interested in. The chapter closes by introducing the process calculus BPA modelling the class of context-free processes.

In *Chapter 3* we extend the class of context-free processes to the class of pushdown processes, thereby proposing the process calculus *Pushdown Process Algebra* (PDPA). We shall discuss equational laws for this process calculus, and present a normal form theorem which allows to represent pushdown processes by means of recursive PDPA equations obeying some restricted format. Furthermore, we establish that pushdown processes are the smallest generalisation of context-free processes up to relabelling that is closed under parallel composition with finite-state systems. We close this chapter by presenting related work exploiting different formalisms for the description of pushdown processes.

In *Chapter 4* we present a model-checker that decides the alternation-free modal μ-calculus for context-free processes, as well as for pushdown processes. We shall define the modal μ-calculus used as our logical language for specifying system properties. We then extend the ordinary semantics of μ-formulas to the assertion-based semantics which turns out to be more suitable when considering decompositions of sequential processes. The dual point of view will lead us from the assertion-based semantics to the second-order semantics constituting the foundation of our equation based model checking algorithm which is subsequently presented. Finally, we shall give an automata-based

construction which proves the regularity of μ-formula semantics with respect to pushdown transition graphs.

In *Chapter 5* we develop an elementary algorithm for deciding bisimulation equivalence for arbitrary context-free processes. We shall introduce the notion of separability dealing with the complements of finite bisimulation approximations. We then present a branching algorithm for deciding bisimilarity of normed BPA processes which improves a similar tableaux-based proof system given by Hüttel and Stirling [HS91, Hüt91]. Hence, we exploit the branching algorithm for the development of a bound on the number of transitions needed to separate two non-bisimilar normed context-free processes thereby extending work of Caucal [Cau89]. Finally, we shall give our bisimulation base construction algorithm which can be used to obtain a bisimulation decision procedure for arbitrary context-free processes.

In *Chapter 6* we summarise the results and give perspectives for further research. In particular, we outline relevant results which have been obtained since the submission of my Ph.D. thesis in summer 1995.

2. Background

2.1 Introduction

In this chapter we present the background material needed in the rest of this monograph. We first review some basic definitions and results from lattice theory with a particular emphasis on fixpoint theorems, as well as from the theory of rewriting. We also recall some facts about formal language theory and consider in more detail the class of context-free languages. Then we introduce *labelled transition graphs* as the underlying models of concurrent systems and *bisimulation equivalence* as the notion of behavioural equivalence we are interested in. Finally, we define the class of *context-free processes* using the process calculus BPA.

2.2 Fixpoint Theory

The theory of fixpoints is applied in many areas of computer science, perhaps most prominently in the theory of denotational semantics of sequential programs. In this monograph, however, fixpoint theory will provide the mathematical basis for a different framework: the semantics of temporal logics which include fixpoint operators. Formulas of these logics will be interpreted with respect to a given model with the intended meaning that a formula will denote the set of states in the model where the formula holds. The semantics of the two fixpoint operators are then naturally defined as the appropriate fixpoints of a particular function associated with the formula at hand.

In this section we summarise some of the basic definitions and properties of ordered sets, and give a brief introduction into lattice theory. The interested reader can find a more detailed elaboration of the field in e.g. [DP90].

2.2.1 Ordered Sets

Definition 2.2.1. *A* partial order *on a set M is a binary relation $\sqsubseteq$ on M such that, for all $x, y, z \in M$,*

1. $x \sqsubseteq x$
2. $x \sqsubseteq y$ *and* $y \sqsubseteq x$ *imply* $x = y$

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 9-31, 1997

3. $x \sqsubseteq y$ *and* $y \sqsubseteq z$ *imply* $x \sqsubseteq z$

If $\sqsubseteq$ *is a partial order on* M*, we call* M *a* partially ordered set *(or* poset *for short).*

Definition 2.2.2. *A poset* M *is called a* chain *(or* totally ordered set*) if, for all* $x, y \in M$*, either* $x \sqsubseteq y$ *or* $y \sqsubseteq x$*.*

When considering maps between ordered sets in particular those which preserve the underlying structure will be of special interest.

Definition 2.2.3. *Let* $(M, \sqsubseteq_M)$ *and* $(N, \sqsubseteq_N)$ *be posets. A mapping* $f : M \longrightarrow N$ *is said to be* monotone *(or* order-preserving*) if, for all* $x, y \in M$*,*

$$x \sqsubseteq_M y \quad \Rightarrow \quad f(x) \sqsubseteq_N f(y)$$

An important point to observe is that structure preserving maps are closed under functional composition.

Lemma 2.2.1. *Let* $f : M \longrightarrow N$ *and* $g : N \longrightarrow O$ *be monotone mappings. Then the composite mapping* $g \circ f : M \longrightarrow O$*,* $(g \circ f)(x) =_{\mathrm{df}} g(f(x))$ *is also monotone.*

As usual, we will use the notation $f(X)$ for $\{ f(x) \mid x \in X \}$ in the sequel.

Definition 2.2.4. *Let* M *be a poset and let* $X \subseteq M$*. Then* $x \in X$ *is said to be the* greatest *element of* X *if we have* $y \sqsubseteq x$ *for all* $y \in X$*. The* least *element of* X *is defined dually.*

This definition implies that the greatest (least) element of X is unique, if it exists.

Definition 2.2.5. *Let* M *be a poset. The greatest element of* M*, if it exists, is called the* top *element and denoted by* $\top$*. Similarly, the least element of* M *is called the* bottom *element and written as* $\bot$*.*

The following constructions of new posets from existing ones will be useful.

Definition 2.2.6. *Let* $M_1, \ldots, M_n$ *be posets. The cartesian product* $M = M_1 \times \ldots \times M_n$ *can be partially ordered by the* coordinatewise order *defined by*

$$(x_1, \ldots, x_n) \sqsubseteq_M (y_1, \ldots, y_n) \quad \textit{iff} \quad x_i \sqsubseteq_{M_i} y_i, \quad \textit{for all } 1 \leq i \leq n$$

Given a poset M we use the abbreviation M^n for the n-fold product $M \times \ldots \times M$.

Definition 2.2.7. *Let* M *be any set and* N *be a poset. The set of all mappings from* M *to* N*, denoted by* $(M \longrightarrow N)$*, can be partially ordered as follows*

$$f \sqsubseteq g \quad \textit{iff} \quad f(x) \sqsubseteq_N g(x), \textit{ for all } x \in M.$$

If M *itself is a poset, we may build the set of all monotone mappings from* M *to* N*, denoted by* $\langle M \longrightarrow N \rangle$*, equipped with the above given ordering.*

Definition 2.2.8. *Let M be a poset and let $X \subseteq M$.*

- *An element $u \in M$ is called an* upper bound *of X if $x \sqsubseteq u$ for all $x \in X$. A* lower bound *of X is defined dually. If b is an upper (lower) bound of X we will also write $X \sqsubseteq b$ ($b \sqsubseteq X$).*
- *An element $x \in M$ is called the* least upper bound *(or* supremum*) of X, denoted by $\mathtt{sup}\ X$, if x is an upper bound of X and it is the least element of all upper bounds of X. The* greatest lower bound *(or* infimum*) of X is defined dually and denoted by $\mathtt{inf}\ X$.*

Note that the least upper bounds and greatest lower bounds are unique, if they exist. We will use the following notations, provided the suprema and infima occurring exist.

$$\begin{array}{lll} x \sqcap y & =_{\mathrm{df}} & \mathtt{inf}\ \{ x, y \} \quad \text{for the "meet" ,} \\ x \sqcup y & =_{\mathrm{df}} & \mathtt{sup}\ \{ x, y \} \quad \text{for the "join" ,} \\ \sqcap X & =_{\mathrm{df}} & \mathtt{inf}\ X, \quad \text{and} \\ \sqcup X & =_{\mathrm{df}} & \mathtt{sup}\ X. \end{array}$$

We shall now concentrate on posets where the above operations are always defined.

Definition 2.2.9. *Let L be a nonempty poset.*

- *If any two elements have a supremum and an infimum, L is called a* lattice.
- *If the supremum and the infimum exist for all $X \subseteq M$, then L is called a* complete *lattice.*

Now it can be shown that every complete lattice has a top and a bottom element. Moreover, it is known that every finite lattice is complete.

Important examples of complete lattices are the powerset of any set M, as well as the set of all mappings between two complete lattices.

1. The powerset 2^M of any set M is a complete lattice where join and meet are given by

$$\begin{array}{lll} \sqcup \{ X_i \subseteq M \mid i \in I \} & =_{\mathrm{df}} & \bigcup_{i \in I} X_i \\ \sqcap \{ X_i \subseteq M \mid i \in I \} & =_{\mathrm{df}} & \bigcap_{i \in I} X_i \end{array}$$

2. Let M be an arbitrary set, and N be a complete lattice. Then the set $(M \longrightarrow N)$ of all mappings from M and N is also a complete lattice with the usual pointwise ordering, and join and meet are given by

$$\begin{array}{lll} \sqcup \{ f_i : M \longrightarrow N \mid i \in I \}(x) & =_{\mathrm{df}} & \sqcup \{ f_i(x) \mid i \in I \} \\ \sqcap \{ f_i : M \longrightarrow N \mid i \in I \}(x) & =_{\mathrm{df}} & \sqcap \{ f_i(x) \mid i \in I \} \end{array}$$

Ordered sets are often used to model approximations. Therefore, subsets which tend towards a limit are particularly interesting.

Definition 2.2.10. *Let X be a nonempty subset of a poset M. X is called* directed *if, for every finite subset $Y \subseteq X$, there exists some $x \in X$ such that $Y \sqsubseteq x$.*

Definition 2.2.11. *A poset M is called a* complete partially ordered set *(CPO) if*

1. *M has a bottom element $\bot$, and*
2. *$\sqcup\, D$ exists for each directed subset D of M.*

Examples of CPO's are given by complete lattices which clearly satisfy the conditions given in the definition.

In particular, when considering maps on infinite domains the property of structure preservation is often not enough. In these cases, a finer notion which takes into account the behaviour of mappings at infinity is needed.

Definition 2.2.12. *Let M, N be CPO's. A mapping $f : M \longrightarrow N$ is called* continuous *if for each directed subset of M,*

$$f(\sqcup\, D) = \sqcup\, f(D)$$

It can be proved that continuous mappings are closed under functional composition, and that continuity is a stronger notion for mappings than monotonicity. However, for mappings on finite domains both notions coincide.

2.2.2 Fixpoint Theorems

In theoretical computer science, solving equations can frequently be expressed as searching for a value x of an appropriate domain M such that x is a fixpoint of some function $f : M \longrightarrow M$, i.e. $f(x) = x$. Particularly, when M carries an order the additional structure plays an important role as it may be used to guide the search for solutions. In this case solutions are often obtained by computing successive approximations, and the overall computation process then aims in each step at increasing the information contents of these partial solutions until a complete solution, provided it exists and can be computed in a finite amount of time, is found. The theoretical foundations of this approach are formulated in a number of fixpoint theorems which explore the existence of fixpoints for different kinds of orders and different properties of the function f under consideration. In the following we give the definitions we need in this monograph in order to develop our results.

Definition 2.2.13. *Let M be a poset and let $f : M \longrightarrow M$ be a mapping. We say that $x \in M$ is a* fixpoint *of f if $f(x) = x$. Moreover, we call x a* pre-fixpoint*, respectively a* post-fixpoint*, if $x \sqsubseteq f(x)$, respectively $f(x) \sqsubseteq x$. Note that the set of all fixpoints of f, denoted by $Fix(f)$, may also be equipped with the order induced by M. The least (greatest) element of $Fix(f)$, when it exists, is then denoted by μf (νf).*

When computing fixpoints the iterative application of functions plays a major role. Therefore, we define the n-fold composition of a function $f : M \longrightarrow M$ inductively by $f^0 =_{\mathrm{df}} id_M$ and $f^n =_{\mathrm{df}} f \circ f^{n-1}$, for $n \geq 1$, where id_M denotes the identity function on M.

Fixpoint theory is now based on the so-called *fixpoint theorems*. Here, we will only present the versions for continuous functions defined on CPO's, and for monotone functions defined on complete lattices. For a more detailed survey, in particular about the history of various fixpoint theorems, the interested reader is referred to [LNS82].

Theorem 2.2.1. *Let M be a CPO and $f : M \longrightarrow M$ be a continuous mapping. Then*

$$\mu f = \sqcup_{n \geq 0}\ f^n(\bot), \qquad \textit{and} \qquad \nu f = \sqcap_{n \geq 0}\ f^n(\top).$$

The importance of this theorem lies in the fact that on finite domains where continuity coincides with monotonicity it allows to compute effectively the appropriate fixpoint by simply taking either the bottom or the top element, respectively, and then iteratively applying f. Monotonicity of f and finiteness of the underlying domain will then guarantee that this computation eventually terminates.

Theorem 2.2.2 (Knaster, Tarski). *Let L be a complete lattice and $f : L \longrightarrow L$ a monotone mapping. Then*

$$\mu f = \sqcap\{\, x \in L \mid f(x) \sqsubseteq x \,\}, \qquad \textit{and} \qquad \nu f = \sqcup\{\, x \in L \mid x \sqsubseteq f(x) \,\}.$$

Moreover, we have that $Fix(f)$ is a complete lattice.

The characterisation of the least and greatest fixpoint as the meet over all post-fixpoints, respectively the join over all pre-fixpoints, as given by Knaster and Tarski in the previous theorem allows immediately to use the following lemma in proofs where we have to show that a certain value is indeed the appropriate fixpoint we wanted to compute.

Lemma 2.2.2. *Let L be a complete lattice and $f : L \longrightarrow L$ a monotone mapping. Then we have, for all $x \in L$,*

$$\begin{array}{lll} f(x) \sqsubseteq x & \textit{implies} & \mu f \sqsubseteq x, \ \textit{and} \\ x \sqsubseteq f(x) & \textit{implies} & x \sqsubseteq \nu f \end{array}$$

2.3 Relations and Rewrite Systems

The development of our algorithm for deciding bisimulation equivalence of arbitrary context-free processes (c.f. Chapter 5) relies on the computability of a certain bisimulation base B which characterises bisimulation as the least congruence which contains B. In this section we introduce the notions concerning relations and the theory of rewriting we need in order to establish this result.

2.3.1 Relations

Let M be a set. A subset $R \subseteq M \times M$ is called a (homogeneous) *relation* on M. The *domain* $dom(R)$ of a relation R is the set $\{ x \mid (x,y) \in R \}$, whereas the *image* $im(R)$ of a relation R is the set $\{ y \mid (x,y) \in R \}$.

Let R_1 and R_2 be two binary relations on $M \times M$. The *relational product* of R_1 and R_2, denoted by $R_1 \circ R_2$, is defined as

$$R_1 \circ R_2 =_{\mathrm{df}} \{ (x,z) \mid \exists\, y \in M.\ (x,y) \in R_1 \ \wedge\ (y,z) \in R_2 \}$$

The *n-fold product* of R is defined inductively as $R^0 =_{\mathrm{df}} \{ (x,x) \mid x \in M \}$ and $R^{n+1} =_{\mathrm{df}} R \circ R^n$, for $n \geq 1$. The *transitive closure* of R, denoted by R^+, is the relation

$$R^+ =_{\mathrm{df}} \bigcup_{n \geq 1} R^n,$$

whereas the *reflexive, transitive closure* of R is defined by

$$R^* =_{\mathrm{df}} \bigcup_{n \geq 0} R^n.$$

Moreover, the *inverse* of R is defined by $R^{-1} =_{\mathrm{df}} \{ (y,x) \mid (x,y) \in R \}$ and the *symmetric closure* of R is the relation $R \cup R^{-1}$. Finally, the *reflexive, symmetric, transitive closure* of R is defined by $(R \cup R^{-1})^*$.

A relation $R \subseteq M \times M$ is said to be an *equivalence relation* if

- R is *reflexive*, i.e. $(x,x) \in R$ for all $x \in R$,
- R is *symmetric*, i.e. $(x,y) \in R$ implies $(y,x) \in R$ for all $x, y \in M$, and
- R is *transitive*, i.e. $(x,y) \in R$ and $(y,z) \in R$ implies $(x,z) \in R$ for all $x, y, z \in M$.

2.3.2 Rewrite Systems

Let M be a nonempty set and $\to \subseteq M \times M$ be a binary relation on M. Then $(M, \to)$ is called a *rewrite system*.

As usually, we write $x \to y$ for $(x,y) \in \to$. We will also use the following standard notations: $\not\to$ for the complement of $\to$, $\to^n$ for the n-fold product, $\to^+$ for the transitive closure, $\to^*$ for the reflexive, transitive closure, $\leftarrow$ for the inverse relation, $\leftrightarrow$ for the symmetric closure, and finally, $\leftrightarrow^*$ for the reflexive, symmetric, and transitive closure.

Much attention has been devoted to the *word problem* for rewrite systems, i.e. the question whether we can decide $x \leftrightarrow^* y$ for a given rewrite system $(M, \to)$ and $x, y \in M$. Although it turned out that this problem is, in general, undecidable there exist for restricted classes of rewrite systems often sophisticated decision procedures. In this monograph we focus on rewrite systems which have the property that each element of M can in a finite number of steps be rewritten into a unique normalform, as this property will guarantee decidability of the word problem.

Definition 2.3.1. *Let $(M, \to)$ be a rewrite system.*

- *$x \in M$ is called* irreducible *if $x \not\to y$ for any $y \in M$,*
- *$y \in M$ is said to be a* normalform *of $x \in M$ if $x \to^* y$ and y is irreducible,*
- *$\to$ is called* terminating *if there does not exist an infinite sequence of rewrite steps $x_0 \to x_1 \to \ldots$, and*
- *$\to$ is called* confluent *if, for all $x, y_1, y_2 \in M$*

$$x \overset{*}{\to} y_1 \text{ and } x \overset{*}{\to} y_2 \quad \text{implies} \quad \exists\, z \in M \text{ such that } y_1 \overset{*}{\to} z \text{ and } y_2 \overset{*}{\to} z.$$

Proposition 2.3.1. *If $(M, \to)$ is terminating and confluent, then every $x \in M$ has a unique normalform, denoted by $x\!\downarrow$. Moreover, we have*

$$x \overset{*}{\leftrightarrow} y \quad \text{iff} \quad x\!\downarrow = y\!\downarrow$$

The last proposition provides now, at least if $(M, \to)$ is terminating and confluent, an effective procedure to solve the word problem for two words x and y as follows.

1. Compute the normalforms of x and y. Note that this process must terminate since $(M, \to)$ is terminating, as well as that it must produce uniquely defined results since $(M, \to)$ is also confluent.
2. Compare for syntactical identity.

Finally, we mention that M is frequently the set of words over some alphabet Σ. A binary relation $R \subseteq \Sigma^* \times \Sigma^*$ then induces a rewrite relation $\to_R$ as follows.

$$\underset{R}{\to} =_{\text{df}} \{\, (xuy, xvy) \mid (u, v) \in R \text{ and } x, y \in \Sigma^* \,\}$$

In this case, we call (Σ^*, R) a *word rewrite system*. For a word rewrite system (Σ^*, R) the reflexive, symmetric, transitive closure $\leftrightarrow_R^*$ is always a *congruence relation*, i.e. it is an equivalence relation which additionally satisfies

$$u \overset{*}{\underset{R}{\leftrightarrow}} v \quad \text{implies} \quad xuy \overset{*}{\underset{R}{\leftrightarrow}} xvy, \text{ for all } x, y \in \Sigma^*.$$

2.4 Context-Free Languages

In this section we introduce context-free grammars and pushdown automata as description mechanisms for context-free languages. Finally, we give some decidability results concerning this class of formal languages. Detailed expositions of the related theory can be found e.g. in the classical books [Har78] and [HU79].

Definition 2.4.1 (Context-Free Grammar).
A context-free grammar *is a quadruple $\mathcal{G} = (V, \Sigma, P, S)$, where V is a finite set of* nonterminals*, Σ is a finite set of* terminals*, P is a finite set of* productions*, each having the form $A \to \alpha$, where A is a nonterminal and α is*

a word over $V \cup \Sigma$, and S is a distinguished nonterminal, called the start symbol.

For each context-free grammar $\mathcal{G} = (V, \Sigma, P, S)$ the *derivation relation* $\Rightarrow_{\mathcal{G}}$ is defined by

$$\alpha A \beta \underset{\mathcal{G}}{\Rightarrow} \alpha\gamma\beta \quad \text{iff} \quad A \to \gamma \text{ is a production of } P \text{ and } \alpha, \beta \in (V \cup \Sigma)^*.$$

The special case when $\alpha = \epsilon$ is called a *leftmost derivation*. Writing $\Rightarrow^*_{\mathcal{G}}$ for the reflexive and transitive closure of $\Rightarrow_{\mathcal{G}}$, the *language generated by* $\mathcal{G}$, denoted by $\mathcal{L}(\mathcal{G})$, is defined as

$$\mathcal{L}(\mathcal{G}) =_{\text{df}} \{\, w \in \Sigma^* \mid S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w \,\}.$$

We call $L \subseteq \Sigma^*$ a *context-free language*, if $L = \mathcal{L}(\mathcal{G})$ for some context-free grammar $\mathcal{G}$. A useful simplification of context-free grammars which does not reduce the generative power is the reduction to Greibach Normal Form.

Definition 2.4.2 (Greibach Normal Form).
A context-free grammar $\mathcal{G} = (V, \Sigma, P, S)$ is said to be in Greibach Normal Form *(GNF) if each production of P is of the form $A \to a\alpha$, where $A \in V$, $a \in \Sigma$ and $\alpha \in V^*$, i.e. each right-hand side starts with a terminal and is followed by a sequence of nonterminals.*

Now context-free grammars in GNF are as expressive as general context-free grammars since it can be proved that for every context-free language L which does not contain ϵ there exists a context-free grammar in GNF that generates L. An alternative, machine based characterisation of context-free languages is given by the notion of a *pushdown automaton*.

Definition 2.4.3 (Pushdown Automaton).
A (nondeterministic) pushdown automaton[1] *(PDA) is a 6-tuple*

$$\mathcal{A} = (Q, \Gamma, \Sigma, \vartheta, q_1, Z_1),$$

where Q is a finite set of (control) states*, Γ is a finite set of* stack symbols*, Σ is a finite set of* terminals*, $q_1 \in Q$ is the* initial state*, $Z_1 \in \Gamma$ is the* initial stack symbol *and ϑ, the* transition function*, is a mapping from $Q \times (\Sigma \cup \{\,\epsilon\,\}) \times \Gamma$ to the finite subsets of $Q \times \Gamma^*$.*

The operational behaviour of a pushdown automaton $\mathcal{A}$ is formalised by the binary relation $\vdash_{\mathcal{A}}$ over *configurations* of $\mathcal{A}$ which capture the current internal state of the automaton together with the remaining input during a computation. We define for $a \in \Sigma, w \in \Sigma^*, q \in Q, Z \in \Gamma$ and $\gamma \in \Gamma^*$

$$(q, Z\gamma, aw) \underset{\mathcal{A}}{\vdash} \begin{cases} (q', \beta\gamma, w), & \text{if } (q', \beta) \in \vartheta(q, a, Z) \\ (q', \beta\gamma, aw), & \text{if } (q', \beta) \in \vartheta(q, \epsilon, Z) \end{cases}$$

[1] Since we will only consider acceptance by empty stack, we omit the otherwise needed set of final states.

If $\vdash_{\mathcal{A}}^{*}$ denotes the reflexive and transitive closure of $\vdash_{\mathcal{A}}$, the *language accepted (by empty stack) by* $\mathcal{A}$, written as $\mathcal{L}(\mathcal{A})$, is defined by

$$\mathcal{L}(\mathcal{A}) =_{\mathrm{df}} \{\, w \in \Sigma^* \mid (q_1, Z_1, w) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q, \epsilon, \epsilon) \text{ for some state } q \,\}.$$

Now it is a well-known fact that the class of languages accepted by PDA's coincides with the class of context-free languages. Moreover, it can be shown that the inclusion and the equivalence problem for context-free languages are undecidable, i.e. there does not exist any algorithm which decides for two arbitrary context-free grammars $\mathcal{G}_1$ and $\mathcal{G}_2$ whether $\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$, or $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$, respectively. However, the situation looks somewhat different in the subclass of "simple" languages.

Definition 2.4.4 (Simple Grammar).
A context-free grammar $\mathcal{G} = (V, \Sigma, P, S)$ *in GNF is said to be a* simple grammar *if for all* $A \in V, a \in \Sigma$ *and* $\alpha, \beta \in V^*$ *we have:*

$$A \to a\alpha \text{ and } A \to a\beta \quad \text{implies} \quad \alpha = \beta$$

A language $L \subseteq \Sigma^*$ *is then called* simple, *if* $L = \mathcal{L}(\mathcal{G})$ *for some simple grammar* $\mathcal{G}$.

Simple grammars have, in contrast to arbitrary context-free grammars, the pleasant property that they are deterministic in the sense that for any combination of a nonterminal to be expanded and a terminal to be generated there is only a single production which can be applied. Although this restriction eliminates any nondeterminism it turns out that for this subclass of context-free languages the inclusion problem is still undecidable [Fri76]. However, as worked out by Korenjak and Hopcroft [KH66] the equivalence problem now becomes solvable. Remarkably, the best algorithm known today for testing language equivalence of simple grammars has been obtained as a particular instance of an algorithm for checking bisimulation equivalence of normed context-free processes developed by Hirshfeld, Jerrum, and Moller [HJM94].

Finally, we mention the class of *deterministic* context-free languages which are accepted by *deterministic* pushdown automata with final state set. The question whether equivalence is decidable within this class of languages which lies strictly between the class of simple languages and the one of context-free languages is still an out-standing problem.

2.5 Processes and Labelled Transition Graphs

One of the main objectives of concurrency theory is to develop mathematical frameworks which allow to model the behaviour and interaction of objects. The hope is that a clear mathematical description of the complex phenomena

which may occur when objects interact will ease the understanding of and the reasoning about the overall behaviour of systems. In this setting a *process* stands for the mathematical abstraction of the interactions between a system and its environment.

A common approach in concurrency theory is to base the intended mathematical theory on a language or calculus which allows to describe processes as terms of a particular algebra, and then to define in a second step the operational semantics of the denoted process in terms of *labelled transition graphs*. These labelled transition graphs model the underlying semantics by using the fundamental concepts of *state* and *change of state*.

Definition 2.5.1 (Labelled Transition Graph).
A labelled transition graph *is a triple* $(\mathcal{S}, Act, \rightarrow)$ *consisting of*

- *a set of* states $\mathcal{S}$*,*
- *a set of* actions *Act, and*
- *a* transition relation $\rightarrow \subseteq \mathcal{S} \times Act \times \mathcal{S}$*.*

Intuitively, a labelled transition graph encodes the operational behaviour of a reactive system. The set $\mathcal{S}$ represents the set of states the system may enter, Act the set of actions the system may perform, and $\rightarrow$ the state transitions that may result upon execution of the actions. With this interpretation the abstract concept of a process can more concretely be identified with a state of a labelled transition graph.

In the remainder of this monograph we will use the standard notations concerning labelled transition graphs as follows. If $(\mathcal{S}, Act, \rightarrow)$ is a labelled transition graph, we shall write $s \stackrel{a}{\rightarrow} s'$ for $(s, a, s') \in \rightarrow$. As usual, we extend the transition relation by reflexivity and transitivity to allow $s \stackrel{w}{\rightarrow} s'$ for $w \in Act^*$. If no transition evolves from s, denoted by $s \not\rightarrow$, s is said to be *terminating*. If $s \stackrel{w}{\rightarrow} s'$ we call s' a *successor* of s. A labelled transition graph is called *finitely branching* if for each state s the set $\{ s \stackrel{a}{\rightarrow} s' \mid a \in Act, s' \in \mathcal{S} \}$ is finite, and is called *finite-state* (or *regular*) if the set of states and the set of actions are finite. To emphasise that a labelled transition graph $\mathcal{T}$ has a distinguished initial state s, we will call $\mathcal{T}$ a *rooted* labelled transition graph with *root* s.

In this monograph we shall particularly be interested in processes which are sequentially composed. Intuitively, such processes consist of several *sequential components* each with a distinguished entry point, the *start state*, and possibly multiple *end states* representing successful termination.

In general, sequential components P_1 and P_2 may be connected by fusing an end state of P_1 with the start state of P_2. This fusion reflects the intended behaviour that the second component may start reacting only after the first one has successfully terminated and reached the end state where P_2 was connected. In terms of labelled transition graphs this notion is made precise as follows.

Definition 2.5.2 (Sequential Labelled Transition Graph (SLTG)).
A sequential labelled transition graph

$$\mathcal{T} = (\mathcal{S}, Act, \rightarrow, s_0, (s_1, \ldots, s_n))$$

of arity n *is a labelled transition graph where*

– s_0 *is a distinguished state, the* start state*, and*
– $s_1, \ldots, s_n$ *are* end states *which are terminating, i.e.* $s_i \not\rightarrow$*, for* $1 \leq i \leq n$.

For ease of presentation, we will assume that start states of SLTG's possess no in-going transitions. This is no severe restriction since an SLTG $\mathcal{T}$ where some transitions leading to the start state exist may be transformed into an SLTG $\mathcal{T}'$ complying with the assumption by introducing a new start state s equipped with the same out-going transitions as the original one. This way we obtain the sequential labelled transition graph

$$\mathcal{T}' = (\mathcal{S} \cup \{ s \}, Act, \rightarrow \cup \{ s \xrightarrow{a} s' \mid s_0 \xrightarrow{a} s' \text{ for some } s' \in \mathcal{S} \}, s, (s_1, \ldots, s_n))$$

which clearly obeys the restriction. Concerning any reasonable behavioural equivalence there should be no difference between $\mathcal{T}$ and $\mathcal{T}'$.

Furthermore, incorporating the possibility of termination in one of *multiple* end states will allow us later on to use sequential labelled transition graphs for the modelling of transition graphs of pushdown automata. In this case termination will correspond to having reached the empty stack, while the finite control of the pushdown automaton determines the end state reached. Motivated by this behaviour of a pushdown automaton we define the *sequential composition of SLTG's* in the following particular way.

Definition 2.5.3 (Sequential Composition).
The sequential composition *of* $n+1$ *sequential labelled transition graphs*

$$\mathcal{T}_i = (\mathcal{S}_i, Act, \rightarrow_i, s_{i0}, (s_{i1}, \ldots, s_{in})), 0 \leq i \leq n,$$

denoted by $\mathcal{T}_0; (\mathcal{T}_1, \ldots, \mathcal{T}_n)$*, is defined as the sequential labelled transition graph obtained by fusing, for* $1 \leq i \leq n$*, the* i*-th end state of* $\mathcal{T}_0$ *with the start state of* $\mathcal{T}_i$ *resulting in the state* s_i*, as well as fusing, for* $1 \leq i \leq n$*, the* i*-th end states of* $\mathcal{T}_1, \ldots, \mathcal{T}_n$ *resulting in a state* e_i.

There are two things to note about this choice of definition for sequential composition: first that all SLTG's involved must have the same arity, and second that end states of $\mathcal{T}_1, \ldots, \mathcal{T}_n$ which occur at the same position are identified. Both properties together will ensure that SLTG's of a fixed arity n are closed under sequential composition.

To demonstrate this concept we illustrate in Figure 2.1 the sequential composition of labelled transition graphs with arity 2.

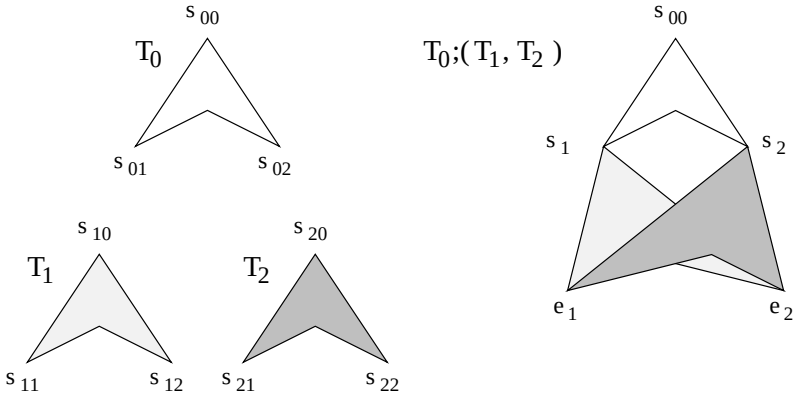

Fig. 2.1. Sequential composition of labelled transition graphs with arity 2.

2.5.1 Behavioural Equivalences

In order to capture when two processes are said to exhibit the same behaviour an abundance of equivalences have been proposed in the literature. Most of them have been classified by van Glabbeek [Gla90] into a hierarchy of behavioural equivalences ordered according to how many identifications between processes they make. The finest equivalence in this hierarchy is *bisimulation equivalence* proposed by Milner and Park [Mil80, Par81]. In particular, after the introduction of Milner's Calculus of Communicating Systems (CCS) [Mil89] bisimulation evolved as an important theoretical notion, and now forms the foundation of a whole theory.

Definition 2.5.4 (Bisimulation Equivalence).
A binary relation R between processes is a bisimulation *if whenever $(p, q) \in R$ then, for each $a \in Act$,*

1. $p \xrightarrow{a} p'$ *implies* $\exists\, q'.\ q \xrightarrow{a} q' \ \wedge\ (p', q') \in R$
2. $q \xrightarrow{a} q'$ *implies* $\exists\, p'.\ p \xrightarrow{a} p' \ \wedge\ (p', q') \in R$

Two processes p and q are said to be *bisimulation equivalent* or *bisimilar*, written $p \sim q$, if $(p, q) \in R$ for some bisimulation R. In [Mil89] it is shown that $\sim$ is the largest bisimulation and that it is an equivalence relation. Henceforth, $[p]_{\sim_{\mathcal{P}}}$ will denote the bisimulation equivalence class of some process p with respect to a set of processes $\mathcal{P}$.

Bisimulation may be further generalised to the notion of *bisimulation up to* which turns out to be useful in proofs for bisimilarity (cf. [Mil89]). Note that $\sim R \sim$ will denote the composition of the three binary relations $\sim$, R, and $\sim$.

Definition 2.5.5 (Bisimulation Up To).
A binary relation R between processes is a bisimulation up to $\sim$ *if whenever* $(p, q) \in R$ *then, for each* $a \in Act$,

1. $p \xrightarrow{a} p'$ *implies* $\exists\, q'.\ q \xrightarrow{a} q' \ \wedge\ (p', q') \in \sim R \sim$
2. $q \xrightarrow{a} q'$ *implies* $\exists\, p'.\ p \xrightarrow{a} p' \ \wedge\ (p', q') \in \sim R \sim$

The importance of bisimulations up to $\sim$ follows from the fact that $R \subseteq \sim$ whenever R is a bisimulation up to $\sim$. Thus in order to show that two processes p and q are bisimilar it suffices to prove that the pair (p, q) is contained in some bisimulation up to $\sim$.

Compared to bisimulation equivalence which takes into account for instance the notion of deadlock, livelock or causality classical *language equivalence* is rather coarse since it ignores the branching structure of processes. Nevertheless, in the special case of normed and deterministic processes which will be introduced in the next subsection we may apply results and proof techniques concerning language equivalence known from formal language theory.

Definition 2.5.6 (Language Equivalence).
Let $(\mathcal{S}, Act, \rightarrow)$ *be a labelled transition graph. The* language *generated by a process* $p \in \mathcal{S}$*, denoted* $\mathcal{L}(p)$*, is defined as*

$$\mathcal{L}(p) =_{\mathrm{df}} \{\, w \in Act^* \mid p \xrightarrow{w} p' \not\rightarrow \ \text{for some } p' \,\}$$

Two processes p and q are said to be language equivalent *iff* $\mathcal{L}(p) = \mathcal{L}(q)$.

As well as one can safely eliminate useless productions in grammars without changing the generated language, also in this setting only terminating processes may contribute to the generation of languages. With each class of processes $\mathcal{K}$ we associate now the class of languages $\mathcal{L}(\mathcal{K})$ defined by $\{\, \mathcal{L}(p) \mid p \in \mathcal{K} \,\}$.

2.5.2 Normedness and Determinism

Although processes often represent continuously reacting systems it is sometimes also desirable that they possess an option to terminate. The notion of *norm* is then a measure for how fast this can happen.

Definition 2.5.7 (Norm).
The norm *of a process p, written as $||p||$, is the length of the shortest transition sequence from p to a terminating state. If a process p cannot reach a terminating state its norm is defined to be infinite. A process is said to be* normed *if its norm is finite, and is called* strongly normed *if any successor is normed. Finally, a transition $p \xrightarrow{a} p'$ of a normed process p is called a* norm-reducing transition *if* $||p'|| = ||p|| - 1$.

Note that bisimulation equivalence always respects the norm, i.e. we have

$$p \sim q \quad \text{implies} \quad ||p|| = ||q||.$$

A different property associated with processes concerns the notion of *determinism*. In general, labelled transition graphs are *nondeterministic*, i.e. processes can have more than one successor labelled with the same action a. This feature is crucial since in process calculi parallelism is often reduced to nondeterministic interleaving of actions. However, from a theoretical point of view it is quite natural to explore also the subclass of *deterministic* labelled transition graphs, as defined by Milner [Mil89].

Definition 2.5.8 (Deterministic Labelled Transition Graphs). *A labelled transition graph* $(\mathcal{S}, Act, \rightarrow)$ *is* deterministic *(up to bisimilarity) iff, for all states* p *in* $\mathcal{S}$*, we have that* $p \stackrel{a}{\rightarrow} p'$ *and* $p \stackrel{a}{\rightarrow} p''$ *implies* $p' \sim p''$*. As usual, a process* p *is called* deterministic *if it belongs to a deterministic labelled transition graph.*

Now it is easy to show that language equivalence and bisimulation equivalence coincide for deterministic labelled transition graphs where every vertex is normed. One direction of the proof follows immediately from the fact that bisimulation is contained in language equivalence, while the other is straightforwardly shown by proving that

$$\{\, (p,q) \mid \mathcal{L}(p) = \mathcal{L}(q) \text{ and } p, q \text{ strongly normed and deterministic} \,\}$$

is a bisimulation. Hence, any bisimulation decision procedure for a class of processes $\mathcal{K}$ is also an algorithm for deciding language equivalence on the class of languages $\mathcal{L}(\mathcal{K}_{det})$ where $\mathcal{K}_{det}$ denotes the deterministic fragment of $\mathcal{K}$.

2.6 Context-Free Processes

Since the beginning algebraic frameworks have played an important role in the study of concurrent communicating processes. Of paramount influence has been the *Calculus of Communicating Processes* (CCS) introduced by Milner [Mil80, Mil89], but also other calculi have contributed significantly to the better understanding of the complex phenomena which may occur in concurrent systems. Here we mention only the *Algebra of Communicating Processes* (ACP) which was first presented by Bergstra and Klop in [BK84]. ACP is an equational specification which has in its signature a constant δ for deadlock, as well as operators for nondeterministic choice, sequential composition, asynchronous parallel composition (called merge), synchronous parallel composition (called communication merge), and encapsulation.

Although the combination of these operators admits to model concisely very complex real-world systems, it hides at the same time the underlying

properties of single operators to a certain degree. Consequently, a number of subcalculi have been considered, focusing only on a few operators. In the case of ACP, the most prominent ones are maybe the *Process Algebra* (PA) [BW90] which has only the operators for nondeterministic choice, sequential composition, and merge, together with its two subcalculi *Basic Parallel Processes* (BPP) [Chr93] where sequential composition is replaced by prefixing, and *Basic Process Algebra* (BPA) [BBK87a] where the merge is omitted.

In this section we introduce the latter calculus, i.e. the theory *Basic Process Algebra* and its extensions BPA_δ, BPA_ε, and $\mathrm{BPA}_{\delta\varepsilon}$. Although we will mainly deal with "pure" BPA specifications in the remainder of this monograph we chose to introduce also its extensions in greater detail since they may serve as a good motivation for our definition of Pushdown Process Algebra (PDPA) given in the next chapter.

2.6.1 Syntax and Semantics

BPA can be interpreted as an *equational specification* $\mathrm{BPA} = (\Sigma_{\mathrm{BPA}}, E_{\mathrm{BPA}})$ where

- Σ_{BPA} is a *signature* containing the binary operators $+$ and $\cdot$, as well as a set of constants Act, and
- E_{BPA} is a set of *equations* (or *axioms*), called the BPA laws, given in Table 2.1.

$A1:$	$E_1 + E_2$	$=$	$E_2 + E_1$
$A2:$	$(E_1 + E_2) + E_3$	$=$	$E_1 + (E_2 + E_3)$
$A3:$	$E_1 + E_1$	$=$	E_1
$A4:$	$(E_1 + E_2) \cdot E_3$	$=$	$E_1 \cdot E_3 + E_2 \cdot E_3$
$A5:$	$(E_1 \cdot E_2) \cdot E_3$	$=$	$E_1 \cdot (E_2 \cdot E_3)$

Table 2.1. The BPA laws.

The operator $+$ is interpreted as nondeterministic choice while $\cdot$ denotes sequential composition – henceforth we usually omit the $\cdot$. Moreover, we use the convention that $\cdot$ binds stronger than $+$. Intuitively, the BPA laws express the commutativity, associativity and idempotence of $+$, right distributivity of $\cdot$ over $+$, and associativity of $\cdot$.

As it is standard in process calculi the operational semantics of BPA expressions is defined in terms of *action relations* which are often given in the style of *structural operational semantics*, introduced by Plotkin [Plo81]. In this monograph we will follow the presentation given in [BW90].

To start with, let $\xrightarrow{a}$, for each $a \in Act$, be a binary relation over BPA terms where $t \xrightarrow{a} t'$ denotes that t can perform an a-action and thereby turn into t'. Moreover, we introduce, for each $a \in Act$, the unary predicate $\xrightarrow{a} \surd$ over BPA terms. The intention of $t \xrightarrow{a} \surd$ is that t terminates after execution of an a-action. Accordingly, the symbol $\surd$ denotes successful termination[2]. The action relations for BPA are given in Table 2.2.

$$a \xrightarrow{a} \surd \qquad \frac{E \xrightarrow{a} \surd}{EF \xrightarrow{a} F} \qquad \frac{E \xrightarrow{a} E'}{EF \xrightarrow{a} E'F}$$

$$\frac{E \xrightarrow{a} \surd}{E + F \xrightarrow{a} \surd} \qquad \frac{F \xrightarrow{a} \surd}{E + F \xrightarrow{a} \surd} \qquad \frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'}$$

Table 2.2. Action relations for BPA.

Thus far, successful termination can only be dealt with *implicitly* in the theory BPA. However, it is also possible to add successful termination *explicitly* to BPA yielding the proper extension BPA_ε. In fact, we will go one step further by introducing the distinction between *successful* and *unsuccessful* termination which will play an important role when considering sequential composition in greater detail. The idea is that the second component of a process E_1E_2 can only start execution, if E_1 has *successfully* terminated. Otherwise, in case of an *unsuccessful* termination of E_1, we also say that E_1 has *deadlocked* and thus does not allow E_2 to proceed.

The two different kinds of termination are modelled in our theory by introducing the new constants ε for the process that may only successfully terminate (also known as the *empty process*) and δ for *deadlock*, respectively.

We obtain the theory $\mathrm{BPA}_{\delta\varepsilon} = (\Sigma_{\mathrm{BPA}_{\delta\varepsilon}}, E_{\mathrm{BPA}_{\delta\varepsilon}})$, a nontrivial extension of BPA, where

- $\Sigma_{\mathrm{BPA}_{\delta\varepsilon}}$ has the binary operators $+$, $\cdot$, a set Act of constants and the special constants δ and ε with $\delta, \varepsilon \notin Act$, and
- $E_{\mathrm{BPA}_{\delta\varepsilon}}$ consists of the equations given for BPA in Table 2.1 and the new axioms given in Table 2.3.

In order to give the action relations for $\mathrm{BPA}_{\delta\varepsilon}$ the termination predicate $\xrightarrow{a} \surd$ will be split into the transition $\xrightarrow{a} \varepsilon$ and the successful termination $\varepsilon \downarrow$ where the unary predicate $\downarrow$ indicates that the process has an option to terminate[3]. In contrast, the introduction of deadlock into BPA does not

[2] To simplify the presentation, instead of $\surd$ often the empty word ϵ is used to represent successful termination with the convention that $\epsilon E = E\epsilon = E$.

[3] This notion of termination option will further be refined when introducing the process calculus PDPA in the next chapter.

$$
\begin{array}{lrcl}
A6: & E+\delta & = & E \\
A7: & \delta E & = & \delta \\
A8: & E\varepsilon & = & E \\
A9: & \varepsilon E & = & E
\end{array}
$$

Table 2.3. Deadlock and the empty process.

require a change in the action relations since it may not perform any action, and moreover, blocks even processes following in a sequential composition as stated by the axiom $A7$. The action relations of $\mathrm{BPA}_{\delta\varepsilon}$ are now given in Table 2.4.

$$
\varepsilon\downarrow \qquad \frac{E\downarrow}{(E+F)\downarrow} \qquad \frac{F\downarrow}{(E+F)\downarrow} \qquad \frac{E\downarrow, F\downarrow}{(EF)\downarrow}
$$

$$
a \xrightarrow{a} \varepsilon \qquad \frac{E\downarrow, F \xrightarrow{a} F'}{EF \xrightarrow{a} F'} \qquad \frac{E \xrightarrow{a} E'}{EF \xrightarrow{a} E'F}
$$

$$
\frac{E \xrightarrow{a} E'}{E+F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E+F \xrightarrow{a} F'}
$$

Table 2.4. Action relations for $\mathrm{BPA}_{\delta\varepsilon}$.

By themselves $\mathrm{BPA}_{\delta\varepsilon}$ expressions do not provide much expressive power since they may describe only finite processes. To remedy this lack of expressiveness, we introduce *recursion* to $\mathrm{BPA}_{\delta\varepsilon}$ by means of *recursive equations* $X = t(X)$ where X is a process variable and $t(X)$ is a $\mathrm{BPA}_{\delta\varepsilon}$ term with free variable $\{ X \}$. *$\mathrm{BPA}_{\delta\varepsilon}$ specifications* are then a straight-forward generalisation[4].

Definition 2.6.1. *A* $\mathrm{BPA}_{\delta\varepsilon}$ specification $\mathcal{C}$ *is a quadruple* $(V, Act, \mathcal{E}, X_1)$ *consisting of*

- *a finite set of* process variables *or* nonterminals $V = \{ X_1, \ldots, X_n \}$,
- *a finite set of* actions *or* terminals Act,
- *a finite set of recursive process equations* $\mathcal{E} = \{ X_i =_{\mathrm{df}} E_i \mid 1 \leq i \leq n \}$, *where each* E_i *is a* $BPA_{\delta\varepsilon}$ *expression with free variables in* V,
- *a variable* $X_1 \in V$, *called the* root.

[4] A $\mathrm{BPA}_{\delta\varepsilon}$ specification is the process theory analogon to a context-free grammar in formal language theory.

If V, and Act are clear from the context we will occasionally present a specification merely by its equations where we adopt the convention that the first variable represents the root.

The semantics of a specification will later be defined as the set of all processes in the underlying model which satisfy the given equations. One possible way to interpret the existence of multiple solutions is then to consider the given specification as being only partial. For example, the trivial equation $X = X$ of which every process is a solution does not specify anything at all. Of more interest are therefore specifications which determine *uniquely* a process. A useful, syntactical criterion that will guarantee this property is *guardedness*.

Definition 2.6.2 (Guardedness).
We call an occurrence of a variable X in a $BPA_{\delta\varepsilon}$ term t guarded *if t has a subterm $a.t'$ such that a is an atomic action and t' contains the occurrence of X. A $BPA_{\delta\varepsilon}$ term t is guarded if every variable occurrence is guarded, and a $BPA_{\delta\varepsilon}$ specification is guarded if each E_i is guarded, for $1 \leq i \leq n$.*

This definition of guardedness may further be relaxed by calling a term *semantically guarded* if it can be rewritten using the $\mathrm{BPA}_{\delta\varepsilon}$ axioms to a (syntactically) guarded term. Analogously, a $\mathrm{BPA}_{\delta\varepsilon}$ specification $\mathcal{C}$ is said to be *semantically guarded* if it is possible to rewrite $\mathcal{C}$ to a guarded $\mathrm{BPA}_{\delta\varepsilon}$ specification using the axioms and by (iterative) replacement of variables by their defining right-hand sides.
Finally, recursion is dealt with in the action relations for $\mathrm{BPA}_{\delta\varepsilon}$ (and $\mathrm{BPA}_{\varepsilon}$) by introducing the following rules.

$$\frac{E\downarrow}{X\downarrow} \quad X = E \in \mathcal{E} \qquad\qquad \frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \quad X = E \in \mathcal{E}$$

Besides extending BPA with both kinds of termination simultaneously, it is also possible to consider the single extensions $\mathrm{BPA}_{\varepsilon}$ and BPA_{δ} on their own. Of greater interest for us will be BPA_{δ} which is defined by the BPA laws given in Table 2.1 and the axioms $A6, A7$ of Table 2.3. The action rules for BPA_{δ} coincide with those of BPA, and thus are already given in Table 2.2. Moreover, they are extended to handle also recursion by the rules

$$\frac{E \xrightarrow{a} \surd}{X \xrightarrow{a} \surd} \quad X = E \in \mathcal{E} \qquad\qquad \frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \quad X = E \in \mathcal{E}$$

Since all the definitions given for $\mathrm{BPA}_{\delta\varepsilon}$ can straight-forwardly be adapted to the weaker settings of BPA, $\mathrm{BPA}_{\varepsilon}$, and BPA_{δ} we will henceforth use the adapted notions without giving further definitions.

BPA, or one of its extensions, may now be interpreted in a number of models. Of particular interest are, however, models of which BPA is a *complete axiomatisation* since then an equation between closed terms is valid in the model iff it is derivable from BPA. In this monograph we will focus on the so called *graph model* $\mathbb{G}/\sim$ for BPA, as well as BPA_δ. $\mathbb{G}/\sim$ is as proved in [BW90] a complete model for BPA and consists of the set of finitely branching rooted transition graphs factorised by bisimulation.

Definition 2.6.3 (Operational Semantics).
The operational semantics *of a guarded BPA specification* $(V, Act, \mathcal{E}, X_1)$ *is given by the labelled transition graph* $(V^*, Act, \rightarrow)$ *rooted at* X_1 *where the transition relation* $\rightarrow$ *is given by the action relations for BPA on* $V^* \times V^*$. *The process defined by* X_1 *is then also called* context-free.

It can be shown that in this *graph model* any guarded BPA specification has a unique solution up to bisimulation equivalence [BK84]. Furthermore, for ease of presentation, it is useful that we can restrict our attention to BPA specifications in a certain normal form.

Definition 2.6.4 (Greibach Normal Form).
A BPA specification $(V, Act, \mathcal{E}, X_1)$ *is said to be in* Greibach Normal Form *(GNF) if all defining equations are of the form*

$$X_i =_{\mathrm{df}} \sum_{j=1}^{m_i} a_{ij}\alpha_{ij}, \quad \text{for } 1 \leq i \leq n$$

where $\alpha_{ij} \in V^*$. *If, moreover, each variable sequence* α_{ij} *has length of at most* K *the BPA specification is said to be in* K-Greibach Normal Form *(*K*-GNF)*[5].

Note that BPA specifications in GNF have the nice property that a transition step in the operational semantics corresponds directly to a leftmost derivation step in the related grammar. As the following Normal Form Theorem [BBK87b, Hüt91] shows we may henceforth assume wlog. that BPA specifications are always given in GNF, since any guarded BPA specification which is not in GNF may effectively be transformed up to bisimilarity into a BPA specification satisfying this condition.

Proposition 2.6.1. *Any guarded BPA specification* $\mathcal{C} = (V, Act, \mathcal{E}, X)$ *can effectively be transformed into a BPA specification* $\mathcal{C}' = (V', Act, \mathcal{E}', X')$ *in* 2*-Greibach Normal Form, such that* $X \sim X'$.

By means of this proposition and the correspondence between BPA specifications in GNF and context-free grammars in GNF it is now possible to

[5] Note that some authors call this restricted format $K+1$-Greibach Normal Form as they include the action a when considering the length of a right-hand side expression.

establish that languages generated by BPA processes are always context-free. Hence, we may conclude from the undecidability of language equivalence for context-free languages mentioned in Section 2.4 that language equivalence is also undecidable for BPA specifications. In fast, all behavioural equivalences in the van Glabbeek linear-time/branching hierarchy [Gla90] except bisimulation equivalence are undecidable for BPA [GH94]. By reduction from the problem of language inclusion for simple grammars, which was shown undecidable by Friedman [Fri76], the same holds for the corresponding preorders.

We close this section by remarking that the BPA laws are easily shown to be sound with respect to bisimilarity [BK88].

Proposition 2.6.2. *For any BPA expressions E_1, E_2 and E_3 we have that*

$$\begin{aligned} E_1 + E_2 &\sim E_2 + E_1, \\ (E_1 + E_2) + E_3 &\sim E_1 + (E_2 + E_3), \\ E_1 + E_1 &\sim E_1, \\ (E_1 + E_2)E_3 &\sim E_1E_3 + E_2E_3, \text{ and} \\ (E_1E_2)E_3 &\sim E_1(E_2E_3). \end{aligned}$$

Given a BPA specification $\mathcal{C} = (V, Act, \mathcal{E}, X)$, we shall use $X, Y, \ldots$ to range over variables in V and Greek letters $\alpha, \beta, \ldots$ to range over elements in V^*. For technical convenience, we use ϵ to denote the empty variable sequence with the convention $\epsilon\alpha = \alpha\epsilon = \alpha$. Moreover, the function $|\,.\,|$ gives the length of a sequence. Finally, we shall often give a BPA specification by simply presenting $\mathcal{E}$ with the convention that the first variable represents the root.

2.6.2 Normedness

In the sequel we shall distinguish normed and unnormed BPA specifications. As usual, a BPA specification is said to be *normed* if all variables are normed and it is called *unnormed* otherwise. In fact, the norm of variables and therefore the normedness of BPA specifications can easily be determined by a slight variant of Dijkstra's shortest path algorithm. Moreover, we want to point out that in normed BPA specifications all processes defined by some variable are even *strongly normed*, while variables in unnormed BPA specifications define not only unnormed, but also normed processes.

Although the class of processes defined by normed BPA specifications does not fully include the class of regular processes related research has proved to be valuable since it may serve as a starting point for studying general BPA processes. In the remainder of this section we shall state some well-known properties of normed BPA processes. In particular, normedness of BPA processes allows to take advantage of some important cancellation rules which build the basis of all bisimulation decision procedures known for this class of processes.

To start with, we observe that the norm is additive wrt. sequential composition, i.e. $||\alpha\beta|| = ||\alpha|| + ||\beta||$, and that bisimulation equivalence is a congruence relation wrt. sequential composition, i.e. we have

$$\alpha\beta \sim \alpha'\beta' \quad \text{whenever} \quad \alpha \sim \alpha' \text{ and } \beta \sim \beta'.$$

Lemma 2.6.1 (Cancellation rules for normed BPA).
Let α, β and γ be normed. Then

1. *$\gamma\alpha \sim \gamma\beta$ implies $\alpha \sim \beta$ and*
2. *$\alpha\gamma \sim \beta\gamma$ implies $\alpha \sim \beta$.*

Note however that both implications of the cancellation lemma are invalid for unnormed processes as demonstrated by the following examples:

1. Let $X = a + aX + aY$ and $Y = bY$ then we have $XY \sim XXY$ but $Y \not\sim XY$.
2. Let $X = a$ and $Y = aY$ then we have $XY \sim XXY$ but $X \not\sim XX$.

Both counterexamples are shown for illustration in Figure 2.2.

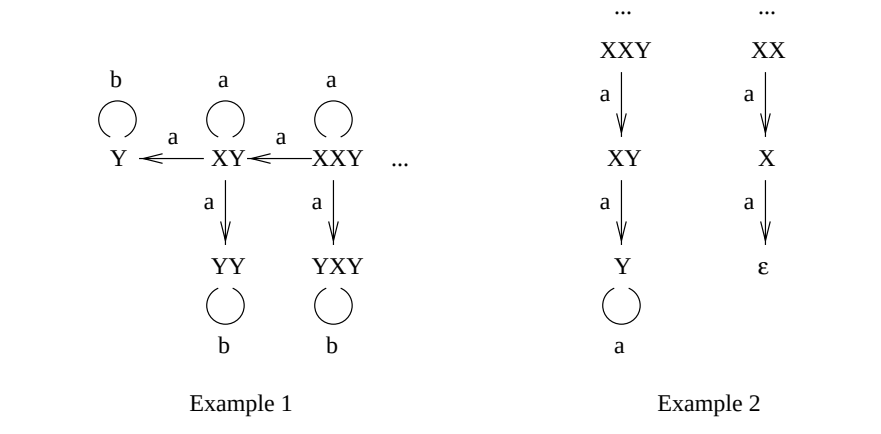

Fig. 2.2. Examples that cancellation does not hold for unnormed BPA processes.

A crucial property of *normed* BPA processes which is based on the previous cancellation rules is now stated in the following splitting lemma given first by Caucal [Cau90].

Lemma 2.6.2 (Splitting rule for normed BPA).
Let $X\alpha, Y\beta \in V^+$ be normed such that $||X|| \leq ||Y||$. Then

$$X\alpha \sim Y\beta \quad \textit{iff} \quad X\gamma \sim Y \text{ and } \alpha \sim \gamma\beta \text{ for some } \gamma.$$

By means of this splitting lemma the problem of proving $X\alpha \sim Y\beta$ can thus be reduced to the "smaller" subproblems of proving $X\gamma \sim Y$ and $\alpha \sim \gamma\beta$. In fact, all recently obtained algorithms for deciding bisimulation equivalence for normed BPA processes [Cau90, Gro91, HS91, HT94, HM94] rely on this reduction.

Finally, we mention that in the presence of unnormed processes we only have the following trivial right-cancellation rule.

Lemma 2.6.3 (Right-cancellation rule for unnormed BPA). *If X is unnormed, then $\alpha X\beta \sim \alpha X$.*

2.6.3 Self-bisimulations

In order to prove bisimilarity of two processes p and q it suffices to exhibit a bisimulation relating p and q. However, since this task is sometimes difficult to accomplish another proof technique introduced by Caucal [Cau90] is often useful.

Definition 2.6.5. *Given a binary relation R between processes we define $p \cong_R q$ if, for each $a \in Act$,*

1. *$p \xrightarrow{a} p'$ implies $\exists\, q'.\ q \xrightarrow{a} q' \ \wedge\ p' \leftrightarrow_R^* q'$, and*
2. *$q \xrightarrow{a} q'$ implies $\exists\, p'.\ p \xrightarrow{a} p' \ \wedge\ p' \leftrightarrow_R^* q'$.*

A binary relation R between processes is then said to be a self-bisimulation *if $R \subseteq \cong_R$.*

The importance of self-bisimulations is now revealed by the following lemma which was first proved in [Cau90].

Lemma 2.6.4. *If R is a self-bisimulation then $\leftrightarrow_R^* \subseteq \sim$.*

Intuitively, the lemma states that solving the word problem for a self-bisimulation R and two processes p and q is already sufficient to establish bisimilarity of p and q. Interestingly, the proof of Caucal even shows that $\leftrightarrow_R^*$ itself is a bisimulation whenever R is a self-bisimulation. We therefore have also the following equivalence.

Corollary 2.6.1. *R is a self-bisimulation iff $\leftrightarrow_R^*$ is a bisimulation.*

Finally, we close this section by proving the additional properties that $\cong_R$ is transitive, as well as a right-congruence with respect to sequential composition. Both properties are needed when developing our branching algorithm for deciding bisimilarity of normed context-free processes in Section 5.4.

Lemma 2.6.5. *Let R be a binary relation between processes. Then we have*

1. *$p_1 \cong_R p_2$ and $p_2 \cong_R p_3$ implies $p_1 \cong_R p_3$, and*

2. $p_1 \cong_R p_2$ *implies* $p_1 q \cong_R p_2 q$.

Proof. To prove the first part of the lemma let $p_1 \cong_R p_2$ and $p_2 \cong_R p_3$. Assume that $p_1 \xrightarrow{a} p_1'$ for some action a and some process p_1'. Due to $p_1 \cong_R p_2$ we know that $p_2 \xrightarrow{a} p_2'$ for some p_2' such that $p_1' \leftrightarrow_R^* p_2'$. Moreover, from $p_2 \cong_R p_3$ we deduce the existence of some process p_3' which satisfies $p_3 \xrightarrow{a} p_3'$ and $p_2' \leftrightarrow_R^* p_3'$. Now the transitivity of $\leftrightarrow_R^*$ yields $p_1' \leftrightarrow_R^* p_3'$. The other direction that $p_3 \xrightarrow{a} p_3'$ implies $p_1 \xrightarrow{a} p_1'$ and $p_1' \leftrightarrow_R^* p_3'$ for some p_1' is shown in a similar way.

For the proof of the second part observe that p_1 and p_2 must either be both the empty process or we have $p_1, p_2 \neq \varepsilon$. Since in the first case the lemma obviously holds we assume $p_1, p_2 \neq \varepsilon$, and $p_1 q \xrightarrow{a} p_1' q$ for some a, p_1'. As this transition is then due to $p_1 \xrightarrow{a} p_1'$ we conclude from $p_1 \cong_R p_2$ that $p_2 \xrightarrow{a} p_2'$ and $p_1' \leftrightarrow_R^* p_2'$ for some p_2'. This yields $p_2 q \xrightarrow{a} p_2' q$ and by the congruence property of $\leftrightarrow_R^*$ also $p_1' q \leftrightarrow_R^* p_2' q$. As the other case for $p_2 q \xrightarrow{a} p_2' q$ is shown analogously this completes the proof. □

3. Pushdown Processes

3.1 Introduction

From "classical" language theory it is well-known that context-free grammars, as well as pushdown automata both characterise the class of context-free languages. In the setting of bisimulation semantics, however, the situation looks different as Caucal and Monfort have shown in [CM90] that the class of labelled transition graphs generated by pushdown automata is strictly larger than the class of labelled transition graphs generated by context-free grammars. This result obviously raises the questions

- what are the new structural properties of pushdown processes in contrast to those of context-free processes, as well as
- which of the known decision procedures for context-free processes can be extended to pushdown processes.

In this chapter we contribute to this line of research by introducing the theory *Pushdown Process Algebra* (PDPA) which we believe is a convenient formalism for modelling the operational behaviour of pushdown automata when considered as processes. Using the algebraic approach has the benefit of being based on a sound mathematical foundation, as well as allowing us to build upon results already known for similar process algebras.

We proceed by exploring the question of how the essential difference between context-free processes and pushdown processes can be explained. A possible explanation is given by our result that the class of *pushdown processes* is *closed under parallel composition* with finite state systems. This follows from a new expansion theorem, whose implied 'representation explosion' is no worse than for finite state systems. Moreover, it turns out that pushdown processes are also the *smallest* extension of context-free processes up to renaming of labels which allows parallel composition with finite state processes. This representation property is shown by proving that every pushdown process is bisimilar to a relabelled parallel composition of a context-free process with some finite process. Overall, these results indicate that pushdown processes are an appropriate generalisation of context-free processes for dealing with some notion of parallelism.

In Section 3.2 we define the syntax and semantics of Pushdown Process Algebra (PDPA), a generalisation of BPA to the framework of pushdown au-

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 33-65, 1997

tomata, while in Section 3.3 we compare the expressive power of pushdown processes with that of context-free processes. Section 3.4 then presents the PDPA laws which evolve directly from the corresponding BPA laws, and in Section 3.5 we develop a procedure which transforms every pushdown process specification effectively into a Greibach-like normal form. In Section 3.6 the closure result under parallel composition with finite state processes is proved, while in Section 3.7 we establish a decomposition theorem for pushdown processes. Finally, in Section 3.8 we survey other proposed formalisms describing the class of pushdown transition graphs.

3.2 Syntax and Semantics

Pushdown automata are known from formal language theory as devices accepting some context-free language. One possible way to define the language accepted is as the set of inputs for which the pushdown automaton empties its stack by means of some sequence of moves. In process theory, however, we are more interested in the branching structure of the transition graph associated with the pushdown automaton. Each internal configuration of the automaton consisting of a state q and a stack content γ is interpreted as a process, while a "sequence of moves" corresponds to a transition sequence in the transition graph. Moreover, reaching a configuration with empty stack component represents successful termination. Consequently, when considering sequential compositions of pushdown transition graphs (or pushdown processes) we have to take into account the final state the pushdown automaton has entered when the stack is empty (cf. Figure 3.1).

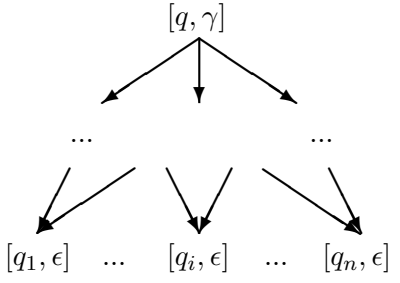

Fig. 3.1. Shape of a pushdown transition graph for $[q, \gamma]$.

In order to reflect this interpretation our process algebra will have two sorts of processes, as well as two kinds of sequential composition. The first sort shall contain processes of *arity* 1 which may only terminate by reaching the

unique process ε, while the second sort will consist of all processes which may terminate by reaching one of the n different configurations $[q_i, \epsilon]$, $1 \leq i \leq n$, where n is the number of states Q in the finite control of the automaton[1]. Processes of the latter kind will have arity n.

To model pushdown processes we introduce now the theory *Pushdown Process Algebra* (PDPA).

Definition 3.2.1 (Pushdown Process Algebra).
Pushdown Process Algebra *(PDPA) is parameterised by a set of* atomic actions *Act, a set of (control)* states $Q = \{ q_1, \ldots, q_n \}$, *and a set of* stack symbols $\mathcal{Z}$. PDPA process expressions *are then given by the abstract syntax*

$$E ::= \varepsilon \mid a \mid [q, \gamma] \mid E_1 + E_2 \mid E_1.E_2 \mid E; (E_1, \ldots, E_n)$$

where a ranges over Act, q over Q, and γ over finite words of $\mathcal{Z}$.

Intuitively, ε represents the empty process of the first sort, which can immediately terminate, while expressions of the form $[q, \gamma]$, the *fragments*[2], are the building blocks in this algebra. Later on *simple* fragments, i.e. fragments where the word γ is a single stack symbol, will play the role of nonterminals when defining pushdown processes. As usual, the operator '+' is nondeterministic choice. Note however, that we use two kinds of sequential composition: the unary $E_1.E_2$ and the n-ary composition of processes $E; (E_1, \ldots, E_n)$.

In the remainder of this monograph we assume that ';' has higher precedence than '+' and we use the abbreviation $\bar{E}_n$ for $(E_1, \ldots, E_n)$. In particular, we use $\bar{E}_n.F$ for $(E_1.F, \ldots, E_n.F)$ and $\bar{E}_n; \bar{F}_n$ for $(E_1; \bar{F}_n, \ldots, E_n; \bar{F}_n)$. Moreover, to shorten notation, we shall write $[\bar{q}_n, \gamma]$ for $([q_1, \gamma], \ldots, [q_n, \gamma])$.

The introduction of fragments in combination with the n-ary sequential composition causes now a problem which was not present in the BPA case: when combining processes we have to take care of the *arity* or *type* of an expression. Formally, given a set of *control states* $Q = \{ q_1, \ldots, q_n \}$, the *arity* of a PDPA expression is defined by the equations of Figure 3.1.

A PDPA expression is then said to be *well-typed* if its arity is defined. An immediate consequence of our definition is that *well-typed* PDPA expressions have either arity 1 or n determining the type of sequential composition to use for them.

As in the case of BPA, recursion is added to PDPA by introducing *PDPA specifications* which are (mutually) recursive equations over PDPA expressions together with a distinguished root.

---

[1] Note that for context-free processes this distinction is not necessary since they may be interpreted as state-less pushdown processes. Thus the empty process ε is identified with the empty stack ϵ.

[2] We denote configurations $[q, \gamma]$ by the name *fragment* in order to emphasise that they will represent the labelled transition graph rooted at $[q, \gamma]$ (cf. [BBK87a]).

$$
\begin{array}{lcl}
\mathtt{arity}(\varepsilon) & = & 1 \\
\mathtt{arity}(a) & = & 1 \\
\mathtt{arity}([q,\gamma]) & = & n \\
\mathtt{arity}(E_1 + E_2) & = & \begin{cases} \mathtt{arity}(E_1), & \text{if } \mathtt{arity}(E_1) = \mathtt{arity}(E_2) \\ \text{undefined}, & \text{otherwise} \end{cases} \\
\mathtt{arity}(E_1.E_2) & = & \begin{cases} \mathtt{arity}(E_2), & \text{if } \mathtt{arity}(E_1) = 1 \\ \text{undefined}, & \text{otherwise} \end{cases} \\
\mathtt{arity}(E;(E_1,\ldots,E_n)) & = & \begin{cases} \mathtt{arity}(E_1), & \text{if } \mathtt{arity}(E) = n \text{ and} \\ & \forall\ 1 \le i,j \le n \\ & \mathtt{arity}(E_i) = \mathtt{arity}(E_j) \\ \text{undefined}, & \text{otherwise} \end{cases}
\end{array}
$$

Table 3.1. The arity of PDPA expressions.

Definition 3.2.2 (PDPA Specification).
A PDPA specification *(or* pushdown process specification*) is a quintuple*

$$\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, [q_1, Z_1])$$

consisting of

- *a finite set of* control states $Q = \{ q_1, \ldots, q_n \}$,
- *a finite set of* stack symbols $\mathcal{Z}$,
- *a finite set of* actions Act,
- *a finite set of* recursive process equations

$$\mathcal{E} = \{ [q_i, Z_j] = E_{ij} \mid q_i \in Q, Z_j \in \mathcal{Z} \}$$

  where the E_{ij} are PDPA expressions of arity n over Q, $\mathcal{Z}$, and Act, obeying the restrictions
  - *ε does not occur in E_{ij}, and*
  - *empty fragments $[q,\epsilon]$ may only occur in E_{ij} in subterms of the form $E.[q,\epsilon]$,*

  and
- *a fragment $[q_1, Z_1]$ called the* root.

The restrictions given for right-hand sides of process equations will guarantee that e.g. expressions of the form $\varepsilon + E$, or $[q,\epsilon] + E$ do not occur in a PDPA specification. Nevertheless, one could also consider PDPA specifications constructed without these restrictions leading to PDPA_ε specifications, similar to the distinction of BPA and BPA_ε specifications. But different from BPA specifications where the only terminating process ε is not allowed to occur, although it may be reached by means of the transition rule $a \xrightarrow{a} \varepsilon$, in our framework we must admit occurrences of empty fragments at least in

restricted form. Otherwise there would be no way for the defined process to terminate.

Analogous to BPA specifications, the following syntactical notion of *guardedness* will now ensure that every *guarded* PDPA specification always defines a uniquely determined process.

Definition 3.2.3 (Guardedness).
An occurrence of a fragment $[q, \gamma]$ *in a PDPA term* t *is said to be* guarded *if* t *has a subterm* $a.t'$ *such that 'a' is an atomic action and* t' *contains the occurrence of* $[q, \gamma]$. *We call a PDPA term guarded if every occurrence of a fragment is guarded in* t, *and a PDPA specification is called guarded if each right-hand side term is guarded.*

In the sequel, we will restrict our attention to guarded PDPA specifications.

After we have dealt with the problem of arity we have now to solve a second problem arising from the presence of two different sorts of processes which concerns *termination*. Essentially, a pushdown process can either terminate by reaching ε, or by reaching a fragment $[q_i, \epsilon]$ with empty stack component, respectively. The kind of termination plays a major role when sequentially composed processes are considered and must therefore explicitly be distinguished. This distinction is accomplished by means of the following *termination predicates*.

Definition 3.2.4 (Termination).
The termination *predicates* $\downarrow_0$ *and* $\downarrow_i$, *for* $1 \leq i \leq n$, *are inductively defined by the following rules. The base cases are*

$$\varepsilon \downarrow_0 \quad \text{and} \quad [q_i, \epsilon] \downarrow_i,$$

while composed processes are handled by the rules

$$\begin{array}{lll}
E \downarrow_0 & \text{implies} & (E+F) \downarrow_0 \text{ and } (F+E) \downarrow_0 \\
E \downarrow_i & \text{implies} & (E+F) \downarrow_i \text{ and } (F+E) \downarrow_i \\
E \downarrow_0, F \downarrow_0 & \text{implies} & E.F \downarrow_0 \\
E \downarrow_0, F \downarrow_i & \text{implies} & E.F \downarrow_i \\
E \downarrow_j, F_j \downarrow_0 & \text{implies} & E;\bar{F}_n \downarrow_0, \text{ for } 1 \leq j \leq n \\
E \downarrow_j, F_j \downarrow_i & \text{implies} & E;\bar{F}_n \downarrow_i, \text{ for } 1 \leq j \leq n
\end{array}$$

A simple consequence of this definition is that processes of arity 1 may only $\downarrow_0$-terminate, while processes of arity n may $\downarrow_i$-terminate, for some $1 \leq i \leq n$.

Using the auxiliary termination predicates, the operational semantics of a pushdown process is now given by the following definition.

Definition 3.2.5 (Operational Semantics).
Any PDPA specification $\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, [q_1, Z_1])$ *with* $Q = \{ q_1, \ldots, q_n \}$

defines a labelled transition graph $\mathcal{T} = (Q \times \mathcal{Z}^*, Act, \rightarrow)$ *where the transition relations* $\stackrel{a}{\rightarrow}, a \in Act$ *are given as the least relations satisfying the rules shown in Table 3.2.*

$$a \stackrel{a}{\rightarrow} \varepsilon, a \in Act \qquad \frac{E \stackrel{a}{\rightarrow} E'}{E + F \stackrel{a}{\rightarrow} E'} \qquad \frac{F \stackrel{a}{\rightarrow} F'}{E + F \stackrel{a}{\rightarrow} F'}$$

$$\frac{E \stackrel{a}{\rightarrow} E'}{[q, Z] \stackrel{a}{\rightarrow} E'} \quad [q, Z] = E \in \mathcal{E} \qquad \frac{[q, Z] \stackrel{a}{\rightarrow} E}{[q, Z\gamma] \stackrel{a}{\rightarrow} E; [\bar{q}_n, \gamma]}$$

$$\frac{E \stackrel{a}{\rightarrow} E'}{E.F \stackrel{a}{\rightarrow} E'.F} \qquad \frac{F \stackrel{a}{\rightarrow} F'}{E.F \stackrel{a}{\rightarrow} F'} \quad E \downarrow_0$$

$$\frac{E \stackrel{a}{\rightarrow} E'}{E; \bar{F}_n \stackrel{a}{\rightarrow} E'; \bar{F}_n} \qquad \frac{F_i \stackrel{a}{\rightarrow} F_i'}{E; \bar{F}_n \stackrel{a}{\rightarrow} F_i'} \quad E \downarrow_i,\ 1 \leq i \leq n$$

Table 3.2. Action relations of PDPA.

A state in the labelled transition graph defined by some PDPA specification is called a *pushdown process* (or *PDPA process*). If the PDPA specification $\mathcal{D}$ is clear from the context we shall also call a PDPA expression E a pushdown process meaning the state labelled with E in the labelled transition graph defined by $\mathcal{D}$. It should be noted that pushdown processes are *finitely branching*, i.e. for every E there are only finitely many E' with $E \stackrel{a}{\rightarrow} E'$.

The name *pushdown* process algebra originates from the fact that each classical pushdown automaton without empty moves induces a PDPA specification in the following sense.

Definition 3.2.6. *Let* $\mathcal{A} = (Q, \mathcal{Z}, Act, \vartheta, q_1, Z_1)$ *be a pushdown automaton. The* PDPA specification induced by $\mathcal{A}$ *is defined as*

$$\mathcal{D}_\mathcal{A} = (Q, \mathcal{Z}, Act, \mathcal{E}_\mathcal{A}, [q_1, Z_1])$$

where

$$[q, Z] = \sum_i a_i.[q_i', \beta_i] \ \in \mathcal{E}_\mathcal{A} \quad \textit{iff} \quad (q_i', \beta_i) \in \vartheta(q, a_i, Z).$$

3.3 Expressiveness

Regular processes and BPA (Basic Process Algebra) processes, which are also known as *context-free* processes (cf. e.g. [CHS92]), can both be seen as special instances of PDPA processes. More specifically, a *BPA process* is a PDPA process where Q contains only a single control state q. As in this case

the single state is irrelevant the fragment $[q, \gamma]$ is identified with the word γ and a simple fragment is called a variable. Even more, the n-ary sequential composition collapses into the unary one which is usually written as $E \cdot F$ or simply as EF. Restricting the sequential composition further to *prefixing* $a \cdot X$, also written as $a.X$, gives the class of *regular processes*. Note that the stack symbols Z_i play the role of the nonterminals of a context-free or regular grammar in these cases.

It is well known that each regular process possesses only a finite number of equivalence classes wrt. bisimulation – a regular process is therefore also called a *finite-state system* – whereas BPA processes, and consequently also pushdown processes, have in general an infinite number of equivalence classes. Moreover, Caucal and Monfort have shown in [CM90] that pushdown processes are strictly more expressive with respect to bisimulation semantics than context-free processes. The authors prove this remarkable result by considering the *multiplicity* of transition graphs.

First, a vertex s is called a *multiple start vertex* for vertices t_1 and t_2 where $t_1 = t_2$ is allowed if there exist two disjoint paths $s \stackrel{w_1}{\rightarrow} t_1$ and $s \stackrel{w_2}{\rightarrow} t_2$, i.e. both paths have except the start and possibly the end vertices no vertices in common. A graph is then said to have *finite multiplicity* if any pair of vertices t_1 and t_2 have only a finite number of start vertices.

Now it turns out that BPA processes have always finite multiplicity, whereas there exist pushdown processes which have infinite multiplicity. An example is given by the pushdown automaton $\mathcal{A}$ of Table 3.3.

$$\mathcal{A} = (Q, \mathcal{Z}, Act, \vartheta, q_1, Z) \quad \text{where}$$

$$\begin{array}{lcl} Q & = & \{\, q_1, q_2, q_3, q_4 \,\} \\ Act & = & \{\, a, b, c, d \,\} \\ \mathcal{Z} & = & \{\, Z \,\} \text{ and} \end{array}$$

$$\begin{array}{lcl} \vartheta(q_1, a, Z) & = & \{\, (q_1, ZZ) \,\} \\ \vartheta(q_1, b, Z) & = & \{\, (q_2, \epsilon) \,\} \\ \vartheta(q_2, d, Z) & = & \{\, (q_2, \epsilon) \,\} \\ \vartheta(q_1, c, Z) & = & \{\, (q_3, \epsilon) \,\} \\ \vartheta(q_3, d, Z) & = & \{\, (q_4, Z) \,\} \\ \vartheta(q_4, d, Z) & = & \{\, (q_3, \epsilon) \,\} \end{array}$$

Table 3.3. A pushdown automaton.

As can be seen from its associated transition graph shown in Figure 3.2 the states $q_2\epsilon$ and $q_3\epsilon$ have an infinite multiplicity with start vertices $q_1 Z^i$, for $i \geq 1$. Hence, this proves that there cannot exist a BPA process which is bisimilar to the labelled transition graph of the pushdown automaton $\mathcal{A}$.

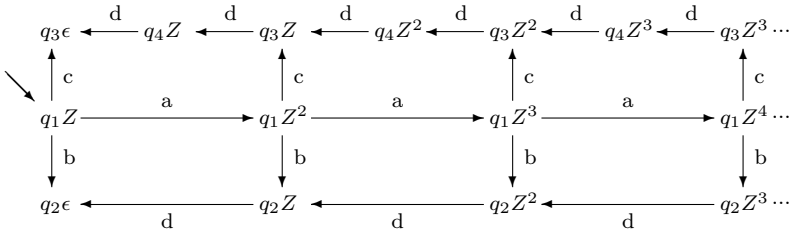

Fig. 3.2. The labelled transition graph generated by the PDA $\mathcal{A}$.

3.4 PDPA Laws

In this section we review the usual BPA_ε laws [BW90] in the context of pushdown processes. These generalised BPA_ε laws, accordingly called *PDPA laws*, express the algebraic properties of the PDPA operators, and are given in Table 3.4[3].

Intuitively, the first three laws express the commutativity, associativity and idempotence of nondeterministic choice. $A4_1$ is the usual BPA law A4, while $A4_n$ is the corresponding generalisation for pushdown processes. These two laws state the right distributivity of sequential composition over '+'. Moreover, the associativity of sequential composition follows from $A5_{1,1}, \ldots, A5_{n,n}$. Of special interest is the law $A6$ which shows how to decompose nonsimple fragments. This law will play a major role in our model-checking algorithm presented in Chapter 4. Finally, the laws $A7$ and $A8$, which are identical to the appropriate BPA_ε laws, state that ε is the neutral element for unary sequential composition. $A9$ and $A10$ are generalisations of these two laws capturing the structure of fragments: $[q_i, \epsilon]$ behaves like a left neutral element, which additionally selects the process to continue with, and $[\bar{q}_n, \epsilon]$ like a right neutral element for n-ary sequential composition.

As for the BPA_ε laws, the PDPA laws are easily shown to be sound with respect to bisimilarity.

Proposition 3.4.1 (Soundness of the PDPA Laws).
All the PDPA laws are valid up to $\sim$.

Proof. Each law is shown to be sound with respect to bisimilarity by proving that a certain corresponding binary relation over PDPA expressions is in fact a bisimulation. In the following we present all the required bisimulation relations, together with a correctness proof for the most difficult cases $A5_{n,n}$

[3] Note again that the class of pushdown processes with arity 1 coincides with the class of BPA processes, since the n-ary sequential composition ';' then collapses into the unary one. Consequently, in this case the axioms $A4_1$ and $A4_n$, as well as $A5_{1,1}, \ldots, A5_{n,n}$ collapse too.

$A1:$	$E_1 + E_2$	$=$	$E_2 + E_1$
$A2:$	$(E_1 + E_2) + E_3$	$=$	$E_1 + (E_2 + E_3)$
$A3:$	$E_1 + E_1$	$=$	E_1
$A4_1:$	$(E_1 + E_2).F$	$=$	$E_1.F + E_2.F$
$A4_n:$	$(E_1 + E_2); \bar{F}_n$	$=$	$E_1; \bar{F}_n + E_2; \bar{F}_n$
$A5_{1,1}:$	$(E.F).G$	$=$	$E.(F.G)$
$A5_{1,n}:$	$(E.F); \bar{G}_n$	$=$	$E.(F; \bar{G}_n)$
$A5_{n,1}:$	$(E; \bar{F}_n).G$	$=$	$E; (\bar{F}_n.G)$
$A5_{n,n}:$	$(E; \bar{F}_n); \bar{G}_n$	$=$	$E; (\bar{F}_n; \bar{G}_n)$
$A6:$	$[q, Z\gamma]$	$=$	$[q, Z]; [\bar{q}_n, \gamma]$
$A7:$	$\varepsilon.E$	$=$	E
$A8:$	$E.\varepsilon$	$=$	E
$A9:$	$[q_i, \epsilon]; \bar{F}_n$	$=$	F_i
$A10:$	$E; [\bar{q}_n, \epsilon]$	$=$	E

Table 3.4. The PDPA laws.

and $A6$. To simplify the notation let $\mathcal{P}$ be the set of all PDPA expressions. Accordingly, $\mathcal{P}^n$ will denote the n-fold cartesian product of $\mathcal{P}$. Moreover, we will write $\mathcal{P}_{id}$ for the identity relation on $\mathcal{P}$, i.e. $\mathcal{P}_{id} =_{\text{df}} \{ (p, p) \mid p \in \mathcal{P} \}$. For consistency, we assume that all PDPA expressions occurring in the proof are well-typed.

- $A1 : E_1 + E_2 \sim E_2 + E_1$ as
  $R1 = \{ (E_1 + E_2, E_2 + E_1) \mid E_1, E_2 \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A2 : (E_1 + E_2) + E_3 \sim E_1 + (E_2 + E_3)$ as
  $R2 = \{ ((E_1 + E_2) + E_3, E_1 + (E_2 + E_3)) \mid E_1, E_2, E_3 \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A3 : E_1 + E_1 \sim E_1$ as
  $R3 = \{ (E_1 + E_1, E_1) \mid E_1 \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A4_1 : (E_1 + E_2).F \sim E_1.F + E_2.F$ as
  $R4_1 = \{ ((E_1 + E_2).F, E_1.F + E_2.F) \mid E_1, E_2, F \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A4_n : (E_1 + E_2); \bar{F}_n \sim E_1; \bar{F}_n + E_2; \bar{F}_n$ as
  $R4_n = \{ ((E_1 + E_2); \bar{F}_n, E_1; \bar{F}_n + E_2; \bar{F}_n) \mid E_1, E_2 \in \mathcal{P}, \bar{F}_n \in \mathcal{P}^n \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A5_{1,1} : (E.F).G \sim E.(F.G)$ as
  $R5_{1,1} = \{ ((E.F).G, E.(F.G)) \mid E, F, G \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A5_{1,n} : (E.F); \bar{G}_n \sim E.(F; \bar{G}_n)$ as
  $R5_{1,n} = \{ ((E.F); \bar{G}_n, E.(F; \bar{G}_n)) \mid E, F \in \mathcal{P}, \bar{G}_n \in \mathcal{P}^n \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A5_{n,1} : (E; \bar{F}_n).G \sim E; (\bar{F}_n.G)$ as

$R5_{n,1} = \{\, ((E;\bar{F}_n).G, E;(\bar{F}_n.G)) \mid E, G \in \mathcal{P}, \bar{F}_n \in \mathcal{P}^n \,\} \cup \mathcal{P}_{id}$ is a bisimulation.

– $A5_{n,n} : (E;\bar{F}_n);\bar{G}_n \sim E;(\bar{F}_n;\bar{G}_n)$ as
$R5_{n,n} = \{\, ((E;\bar{F}_n);\bar{G}_n, E;(\bar{F}_n;\bar{G}_n)) \mid E \in \mathcal{P}, \bar{F}_n, \bar{G}_n \in \mathcal{P}^n \,\} \cup \mathcal{P}_{id}$ is a bisimulation.
First suppose that there exists a transition

$$(E;\bar{F}_n);\bar{G}_n \xrightarrow{a} (E';\bar{F}_n);\bar{G}_n$$

due to some transition $E;\bar{F}_n \xrightarrow{a} E';\bar{F}_n$ which in turn must be due to some transition $E \xrightarrow{a} E'$. Then we also have

$$E;(\bar{F}_n;\bar{G}_n) \xrightarrow{a} E';(\bar{F}_n;\bar{G}_n)$$

and by definition

$$(\,(E';\bar{F}_n);\bar{G}_n,\ E';(\bar{F}_n;\bar{G}_n)\,) \in R5_{n,n}.$$

The reverse direction for this case is shown in a similar way. Now consider a transition of the form

$$(E;\bar{F}_n);\bar{G}_n \xrightarrow{a} F_i';\bar{G}_n$$

due to $E \downarrow_i$, $F_i \xrightarrow{a} F_i'$, and thus $E;\bar{F}_n \xrightarrow{a} F_i'$. This yields

$$E;(\bar{F}_n;\bar{G}_n) \xrightarrow{a} F_i';\bar{G}_n$$

due to $F_i;\bar{G}_n \xrightarrow{a} F_i';\bar{G}_n$ and obviously

$$(F_i';\bar{G}_n, F_i';\bar{G}_n) \in R5_{n,n}.$$

Again the reverse direction follows by symmetric arguments. Finally, we may have a transition

$$(E;\bar{F}_n);\bar{G}_n \xrightarrow{a} G_j'$$

since $E \downarrow_i, F_i \downarrow_j$ and $G_j \xrightarrow{a} G_j'$. But then

$$E;(\bar{F}_n;\bar{G}_n) \xrightarrow{a} G_j'$$

follows from $F_i;\bar{G}_n \xrightarrow{a} G_j'$ and we have $(G_j', G_j') \in R5_{n,n}$. Observing that the reverse direction is shown symmetrically completes the proof.

– $A6 : [q, Z\gamma] \sim [q, Z];[\bar{q}_n, \gamma]$ as
$R6 = \{\, ([q, Z\gamma], [q, Z];[\bar{q}_n, \gamma]) \mid q, q_i \in Q, Z\gamma \in \Gamma^+ \,\} \cup \mathcal{P}_{id}$ is a bisimulation.

In order to prove this case assume that there exists a transition

$$[q, Z\gamma] \xrightarrow{a} E;[\bar{q}_n, \gamma]$$

due to a transition $[q, Z] \xrightarrow{a} E$. Then we also have

$$[q, Z];[\bar{q}_n, \gamma] \xrightarrow{a} E;[\bar{q}_n, \gamma]$$

and clearly $(E;[\bar{q}_n, \gamma], E;[\bar{q}_n, \gamma]) \in R6$, as desired. If on the other hand

$$[q, Z]; [\bar{q}_n, \gamma] \xrightarrow{a} E; [\bar{q}_n, \gamma]$$

due to some transition $[q, Z] \xrightarrow{a} E$, we obtain also

$$[q, Z\gamma] \xrightarrow{a} E; [\bar{q}_n, \gamma]$$

and clearly $(E; [\bar{q}_n, \gamma], E; [\bar{q}_n, \gamma]) \in R6$ to complete also this part of the proof.

- $A7 : \varepsilon.E \sim E$ as
  $R7 = \{ (\varepsilon.E, E) \mid E \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A8 : E.\varepsilon \sim E$ as
  $R8 = \{ (E.\varepsilon, E) \mid E \in \mathcal{P} \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A9 : [q_i, \epsilon]; \bar{F}_n \sim F_i$ as
  $R9 = \{ ([q_i, \epsilon]; \bar{F}_n, F_i) \mid q_i \in Q, \bar{F}_n \in \mathcal{P}^n \} \cup \mathcal{P}_{id}$ is a bisimulation.
- $A10 : E; [\bar{q}_n, \epsilon] \sim E$ as
  $R10 = \{ (E; [\bar{q}_n, \epsilon], E) \mid E \in \mathcal{P}, q_i \in Q \} \cup \mathcal{P}_{id}$ is a bisimulation.

□

3.5 Pushdown Normal Form

For any computing devices, like e.g. Turing machines, grammars or automata, it is, in general, interesting to know whether there exist certain normal forms which allow to study the object under consideration merely for a restricted range of instances without compromising expressiveness. Usually, proofs benefit most from the existence of normal forms as they can often significantly be simplified if the device at hand can be assumed to be of a specific form.

In formal language theory, for example, Chomsky Normal Form, as well as Greibach Normal Form, both constitute important normal forms for context-free grammars. Furthermore, for pushdown automata a normal form is known which admits to restrict the attention to pushdown automata pushing at most 2 symbols onto the stack when reading an input symbol[4].

By applying these ideas to the setting of process theory Baeten, Bergstra, and Klop have shown in [BBK87a] that any guarded BPA specification can effectively be represented in *Greibach Normal Form* (GNF), i.e. by equations of the form $X = \sum_i a_i.\alpha_i$. This result is further strengthened in [Hüt91] by proving that the length of the string α_i can be bound by 2. Recursive specifications with this property are said to be in *2-GNF*. Here we present the analogous result for PDPA specifications that each pushdown process up to bisimilarity is induced by some pushdown automaton, i.e. a pushdown process in *Pushdown Normal Form*. In Section 3.7 we improve on this result, by showing that the pushdown automaton can additionally be assumed to push at most 2 symbols onto the stack during a transition step.

[4] In fact, there are even stronger normal forms known for pushdown automata (cf. [HU79]) which are, however, of no interest here.

Definition 3.5.1 (Pushdown Normal Form (PDNF)).
A PDPA specification $\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, [q_1, Z_1])$ *is said to be in* Pushdown Normal Form *(PDNF) if all equations of* $\mathcal{E}$ *are of the form*

$$[q, Z] = \sum_i a_i.[q_i', \beta_i]$$

If the length of each word β_i *is bounded by some* k*, the PDPA specification is said to be in* k-PDNF*.*

Theorem 3.5.1. *If* $\mathcal{D}$ *is a guarded PDPA specification, we can effectively find a specification* $\mathcal{D}'$ *in PDNF such that* $\mathcal{D} \sim \mathcal{D}'$*, i.e. that the roots of* $\mathcal{D}$ *and* $\mathcal{D}'$ *are bisimilar.*

Proof. Let $\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, [q_1, Z_1])$ be a guarded PDPA specification. Our procedure for rewriting $\mathcal{E}$ into PDNF consists of a sequence of transformations, which are similar to the ones used for rewriting BPA equations into Greibach Normal Form (cf. [Hüt91]). Since during the rewrite procedure we will temporarily generate equations with *ordinary* variables on the left-hand side, we will use *generalised variables* $\mathcal{X}$, to denote both ordinary variables and simple fragments, respectively.

Step 1: We apply the right-distributive laws $A4_1$ and $A4_n$ from left-to-right as far as possible. Then, for all atomic actions a, we replace all internal occurrences of a by X_a, while keeping the resulting specification guarded, and add the equation $X_a = a$ to $\mathcal{E}$.

Step 2: We successively remove all outermost unresolved sums, all unresolved sequential compositions and all nonsimple fragments: An *unresolved sum* is a sum $F + G$ which occurs in an expression of the form

$$E.(F + G) \quad \text{or} \quad E; (E_1, \ldots, F + G, \ldots, E_n),$$

while an *unresolved sequential composition* is either a sequential composition $F.G$ occurring in an expression

$$E.F.G \quad \text{or} \quad E; (E_1, \ldots, F.G, \ldots, E_n),$$

or a sequential composition $F; \bar{G}_n$ occurring in an expression

$$E; (E_1, \ldots, F; \bar{G}_n, \ldots, E_n).$$

We now repeat the following loop until all unresolved expressions and nonsimple fragments are eliminated.

- Replace each outermost unresolved sum $F + G$ by a new variable X_{F+G} and add the equation $X_{F+G} = F + G$ to $\mathcal{E}$.
- Replace each outermost unresolved sequential composition α by a new variable X_α and add the equation $X_\alpha = \alpha$ to $\mathcal{E}$.
- Replace each nonsimple fragment $[q, Z\gamma]$ by a new variable $X_{[q,Z\gamma]}$ and add the equation $X_{[q,Z\gamma]} = [q, Z]; [\bar{q}_n, \gamma]$ to $\mathcal{E}$.

After eliminating all unresolved expressions, each equation is of the form

$$\mathcal{X} = \sum_i a_i.\alpha_i + \sum_j \beta_j$$

where $\mathcal{X}$ is a generalised variable, and α_i, β_j are sequences of sequentially composed generalised variables.

Step 3: In the following, we say that an equation belongs to *stratum* i, if it is added in the i-th loop of the previous step. Thus all of the original equations are members of stratum 0. This classification is important for the elimination of the unguarded equations, which may have been introduced in the previous transformation step:

– For each successive stratum i, for each equation in stratum i, and for each unguarded summand $\mathcal{X}_j.\beta_j$ (or $\mathcal{X}_j;\beta_j$), replace $\mathcal{X}_j$ by its definition $\sum_i a_i.\alpha_i$ and subsequently apply the right-distributive laws $A4_1$ and $A4_n$ to obtain the guarded summand $\sum_i a_i.\alpha_i.\beta_j$ (or $\sum_i a_i.\alpha_i;\beta_j$).

The correctness and termination of this procedure relies on the fact that all definitions of variables introduced in stratum i use only variables in strata $< i$ and moreover, that all equations of stratum 0 are already guarded.

Step 4: Now we replace all ordinary variables by appropriate simple fragments:

– Replace each ordinary variable X in an expression $E.X$ by $[q_1, X]$ and the equation $X = E'$ from $\mathcal{E}$ by $[q_1, X] = E'$.
– Replace each ordinary variable X in an expression

$$E;(\mathcal{X}_1, \ldots, \mathcal{X}_{i-1}, X, \mathcal{X}_{i+1}, \ldots, \mathcal{X}_n)$$

by $[q_i, X]$ and each equation $X = E'$ from $\mathcal{E}$ by the n equations $[q_i, X] = E'$, for $1 \leq i \leq n$.

Step 5: The last transformation step normalises the fragments and delivers the intended normal form.

1. Replace each fragment $[q_k, Z]$ in an expression

$$E;(\mathcal{F}_1, \ldots, \mathcal{F}_{i-1}, [q_k, Z], \mathcal{F}_{i+1}, \ldots, \mathcal{F}_n)$$

with $k \neq i$ by $[q_i, Y_{[q_k,Z]}]$ and add the equation $[q_i, Y_{[q_k,Z]}] = E'$ to $\mathcal{E}$ whenever $[q_k, Z] = E'$ is already an equation of $\mathcal{E}$.

2. Replace each expression $([q_1, Z_1], \ldots, [q_n, Z_n])$ containing $Z_i \neq Z_j$, i.e. not all Z_i are identical, by $([q_1, Y_{Z_1 \ldots Z_n}], \ldots, [q_n, Y_{Z_1 \ldots Z_n}])$, and add the equations $[q_i, Y_{Z_1 \ldots Z_n}] = E_i$ to $\mathcal{E}$ whenever $[q_i, Z_i] = E_i$ is already contained in $\mathcal{E}$.

3. Replace each n-ary sequential composition $[q, \alpha]; [\bar{q}_n, \beta]$ by $[q, \alpha\beta]$.

Since all steps of the algorithm either simply apply PDPA laws or introduce new variables that rename expressions, bisimilarity is preserved, which completes the proof. □

Example

We illustrate the transformation into Pushdown Normal Form by means of the following PDPA specification where the right-hand side of $[q_1, Z_1]$ is not in normal form. To ease the readability, expressions that have changed during a transformation step are indicated by underlining.

$$\begin{aligned}
[q_1, Z_1] &= (a + b.a).[q_1, \epsilon] + a.[q_2, Z_1]; ([q_2, Z_1], [q_2, Z_1] + [q_3, Z_1], [q_3, Z_1]) \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1]
\end{aligned}$$

Step 1: *Application of the PDPA law $A4_1$, and elimination of internal actions.*

$$\begin{aligned}
[q_1, Z_1] &= \underline{a.[q_1, \epsilon] + b.X_a.[q_1, \epsilon]} + \\
&\quad a.[q_2, Z_1]; ([q_2, Z_1], [q_2, Z_1] + [q_3, Z_1], [q_3, Z_1]) \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
\underline{X_a} &= \underline{a}
\end{aligned}$$

Step 2: *Elimination of unresolved sequential compositions and unresolved sums.*

$$\begin{aligned}
[q_1, Z_1] &= a.[q_1, \epsilon] + b.\underline{X_1} + a.[q_2, Z_1]; ([q_2, Z_1], \underline{X_2}, [q_3, Z_1]) \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
X_a &= a \\
\underline{X_1} &= \underline{X_a.[q_1, \epsilon]} \\
\underline{X_2} &= \underline{[q_2, Z_1] + [q_3, Z_1]}
\end{aligned}$$

Step 3: *Guardedness transformation via strata.*
Note that the equations of X_1 and X_2 belong to stratum 1, while all others are members of stratum 0. To ease the presentation we omit henceforth the equation of X_a since the variable no longer occurs in the right-hand side of some equation.

$$\begin{aligned}
[q_1, Z_1] &= a.[q_1, \epsilon] + b.X_1 + a.[q_2, Z_1]; ([q_2, Z_1], X_2, [q_3, Z_1]) \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
X_1 &= \underline{a}.[q_1, \epsilon] \\
X_2 &= \underline{a.[q_2, \epsilon]} + \underline{b.[q_1, Z_1]}
\end{aligned}$$

Step 4: *Replacement of ordinary variables.*
Since $[q_2, X_1]$ and $[q_1, X_2]$ are not referenced we omit the associated equations.

$$\begin{aligned}
[q_1, Z_1] &= a.[q_1, \epsilon] + b.\underline{[q_1, X_1]} + a.[q_2, Z_1]; ([q_2, Z_1], \underline{[q_2, X_2]}, [q_3, Z_1]) \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
\underline{[q_1, X_1]} &= a.[q_1, \epsilon] \\
\underline{[q_2, X_2]} &= a.[q_2, \epsilon] + b.[q_1, Z_1]
\end{aligned}$$

Step 5.1: *Normalization of fragments I.*

$$\begin{aligned}
[q_1, Z_1] &= a.[q_1, \epsilon] + b.[q_1, X_1] + a.[q_2, Z_1]; (\underline{[q_1, X_3]}, [q_2, X_2], [q_3, Z_1]) \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
[q_1, X_1] &= a.[q_1, \epsilon] \\
[q_2, X_2] &= a.[q_2, \epsilon] + b.[q_1, Z_1] \\
\underline{[q_1, X_3]} &= \underline{a.[q_2, \epsilon]}
\end{aligned}$$

Step 5.2: *Normalization of fragments II.*
Since the equations for $[q_1, X_3]$ and $[q_2, X_2]$ are no longer needed we omit reference to them.

$$\begin{aligned}
[q_1, Z_1] &= a.[q_1, \epsilon] + b.[q_1, X_1] + a.[q_2, Z_1]; \underline{([q_1, X_4], [q_2, X_4], [q_3, X_4])} \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
[q_1, X_1] &= a.[q_1, \epsilon] \\
\underline{[q_1, X_4]} &= \underline{a.[q_2, \epsilon]} \\
\underline{[q_2, X_4]} &= \underline{a.[q_2, \epsilon] + b.[q_1, Z_1]} \\
\underline{[q_3, X_4]} &= \underline{b.[q_1, Z_1]}
\end{aligned}$$

Step 5.3: *Normalization of fragments III.*

$$\begin{aligned}
[q_1, Z_1] &= a.[q_1, \epsilon] + b.[q_1, X_1] + a.\underline{[q_2, Z_1 X_4]} \\
[q_2, Z_1] &= a.[q_2, \epsilon] \\
[q_3, Z_1] &= b.[q_1, Z_1] \\
[q_1, X_1] &= a.[q_1, \epsilon] \\
[q_1, X_4] &= a.[q_2, \epsilon] \\
[q_2, X_4] &= a.[q_2, \epsilon] + b.[q_1, Z_1]
\end{aligned}$$

Finally, the PDPA specification obtained is in Pushdown Normal Form.

3.6 Parallel Composition

BPA processes arise, as explained in Section 2.6, in a natural way from the more expressive Process Algebra (PA) by omission of the (asynchronous) parallel operator. When considering PDPA processes, parallel composition comes, however, again into play, as the operational behaviour of the pushdown process may be interpreted as the synchronised parallel combination of the finite state control and the stack where certain transitions are forbidden.

In this section we elaborate on this point of view by introducing a binary CSP-like parallel operator $\|$ for PDPA processes (cf. [Hoa85]) and, subsequently, proving that the class of PDPA processes is closed under parallel composition with regular processes. Intuitively, the parallel composition $P \parallel Q$ of two processes requires the synchronisation of the actions common to both component alphabets and allows the interleaving of the others.

Definition 3.6.1 (Synchronous Parallel Composition).
The synchronous parallel composition $P \parallel Q$ *of two processes* P, Q *behaves as follows:*

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} \quad a \notin Act(Q) \qquad \frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'} \quad a \notin Act(P)$$

$$\frac{P \xrightarrow{a} P' \ , \ Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P' \parallel Q'}$$

Using this notion of synchronous parallelism it turns out that the parallel composition of a pushdown process P with a regular process R can again be modelled up to bisimilarity as a pushdown process P'. The construction given in the proof of the following theorem, essentially, incorporates the behaviour of R into the finite state control of P without modifying the stack.

Theorem 3.6.1 (Expansion Theorem).
Given a pushdown process P *and a regular process* R*, we can effectively construct a pushdown process* P' *such that*

$$P' \ \sim \ P \parallel R$$

Proof. Let P be the root of the guarded PDPA specification

$$\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}_{\mathcal{D}}, [q_1, Z_1])$$

with state set $Q = \{ q_1, \ldots, q_n \}$, and R be the root of the guarded regular specification

$$\mathcal{R} = (V, Act, \mathcal{E}_{\mathcal{R}}, X_1)$$

with variables $V = \{ X_1, \ldots, X_m \}$, respectively. For simplicity, we assume that each equation in $\mathcal{E}_{\mathcal{R}}$ is either of the form $X = \sum_{i=1}^{r} a_i X_{j_i}$ with $a_i \in Act$ and $X_{j_i} \in V$, or of the form $X = \varepsilon$.

We construct the PDPA specification

$$\mathcal{D}' = (Q \times V, \mathcal{Z}, Act, \mathcal{E}_{\mathcal{D}'}, [(q_1, X_1), Z_1]),$$

whose root $[(q_1, X_1), Z_1]$ will define P' as then follows:

$$[(q, X), Z] = E_{\mathcal{D}} + E_{\mathcal{R}} + E_{\mathcal{D},\mathcal{R}} \ \in \mathcal{E}_{\mathcal{D}'}, \quad \text{if}$$

$$E_{\mathcal{D}} = \sum a.[(q', X), \beta], \quad \text{where } a.[q', \beta] \text{ is a right-hand side summand of } [q, Z] \text{ in } \mathcal{E}_{\mathcal{D}} \text{ and } a \notin Act(\mathcal{R}),$$

$$E_{\mathcal{R}} = \sum a.[(q, X'), Z], \quad \text{where } a.X' \text{ is a right-hand side summand of } X \text{ in } \mathcal{E}_{\mathcal{R}} \text{ and } a \notin Act(\mathcal{D}),$$

$$E_{\mathcal{D},\mathcal{R}} = \sum a.[(q', X'), \beta], \quad \text{where } a.[q', \beta] \text{ is a right-hand side summand of } [q, Z] \text{ in } \mathcal{E}_{\mathcal{D}} \text{ and } a.X' \text{ is a right-hand side summand of } X \text{ in } \mathcal{E}_{\mathcal{R}}.$$

In order to prove $P' \sim P \parallel R$, we will now show that

$$S = \{\, ([(q, X), \gamma], [q, \gamma] \parallel X) \mid q \in Q, \gamma \in \mathcal{Z}^*, X \in V \,\}$$

is a bisimulation up to $\sim$. Thus let $([(q, X), \gamma], [q, \gamma] \parallel X) \in S$.

Let us first consider the case where the stack is empty, and assume

$$[(q, X), \epsilon] \xrightarrow{a} E$$

Since $[q, \epsilon]$ cannot perform any action we deduce that $E = [(q, X'), \epsilon]$ for some X', that $a.X'$ is a right-hand side summand of X in $\mathcal{E}_{\mathcal{R}}$ and that $a \notin Act(\mathcal{D})$. Hence there exists a matching transition

$$[q, \epsilon] \parallel X \xrightarrow{a} [q, \epsilon] \parallel X'$$

and, moreover, we have $([(q, X'), \epsilon], [q, \epsilon] \parallel X') \in S$ by definition. For the converse direction suppose that

$$[q, \epsilon] \parallel X \xrightarrow{a} E$$

This immediately implies that $E = [q, \epsilon] \parallel X'$ for some X', that $a.X'$ must be a right-hand side summand of X in $\mathcal{E}_{\mathcal{R}}$ and that $a \notin Act(\mathcal{D})$. Thus we obtain

$$[(q, X), \epsilon] \xrightarrow{a} [(q, X'), \epsilon]$$

and again by definition $([(q, X'), \epsilon], [q, \epsilon] \parallel X') \in S$, as desired.

Now assume we have an expression with nonempty stack component and some transition of the form

$$[(q, X), Z\gamma] \xrightarrow{a} E.$$

We will only consider the case where a required synchronisation occurs. The other cases are simpler analogues. In this case, by definition of the transition relation we must have

$$[(q, X), Z] \xrightarrow{a} [(q', X'), \beta] \quad \text{and} \quad E = [(q', X'), \beta]; [\bar{q}_n \times \bar{X}_m, \gamma]$$

where $[\bar{q}_n \times \bar{X}_m, \gamma]$ abbreviates $([(q_1, X_1), \gamma], \ldots, [(q_n, X_m), \gamma])$. Thus by construction of $\mathcal{E}_{\mathcal{D}'}$ we know that $a.[q', \beta]$ is a right-hand side summand of $[q, Z]$ in $\mathcal{E}_{\mathcal{D}}$, while $a.X'$ is a right-hand side summand of X in $\mathcal{E}_{\mathcal{R}}$. Therefore we also have

$$[q, Z\gamma] \parallel X \xrightarrow{a} [q', \beta]; [\bar{q}_n, \gamma] \parallel X'$$

and clearly

$$\begin{aligned} [(q', X'), \beta]; [\bar{q}_n \times \bar{X}_m, \gamma] &\sim [(q', X'), \beta\gamma] \; S \; [q', \beta\gamma] \parallel X' \\ &\sim [q', \beta]; [\bar{q}_n, \gamma] \parallel X'. \end{aligned}$$

As the converse direction is shown by symmetric arguments this concludes the proof. □

Corollary 3.6.1. *If the PDPA specification $\mathcal{D}$ is in k-PDNF the resulting expanded PDPA specification $\mathcal{D}'$ constructed in the previous proof is also in k-PDNF.*

Proof. Since the construction of the expanded PDPA specification only deals with integrating the regular process structure into the control component of the given PDPA specification, it is easy to see that the length of the words in the stack component does not change. □

Finally, we mention that an automata theoretic variant of this result, which also applies to macro processes, can be found in [PI93].

3.6.1 Example

We illustrate the construction given in the proof of Theorem 3.6.1 by means of the process management system `PMS` consisting of the parallel composition of the "process handler" `PH` and the "controller" `C`. The specifications of the parallel components `PH` and `C` are defined as follows

$$\begin{aligned} \texttt{PH} &= \{ \quad X_0 = \texttt{create}.X_1, \quad X_1 = \texttt{create}.X_1X_1 + \texttt{term} \quad \} \\ \texttt{C} \;\; &= \{ \quad Y_0 \;= \texttt{create}.Y_1, \quad Y_1 \;= \texttt{create}.Y_1 + \texttt{shutdown}.Y_2, \\ &\qquad Y_2 \;= \varepsilon \quad \} \end{aligned}$$

whereas their associated labelled transition graphs are given in Figure 3.3.

Here the process handler `PH` can be interpreted as a system that allows the creation of processes via the action `create` and the termination of active processes via the action `term`. The process management system offers its

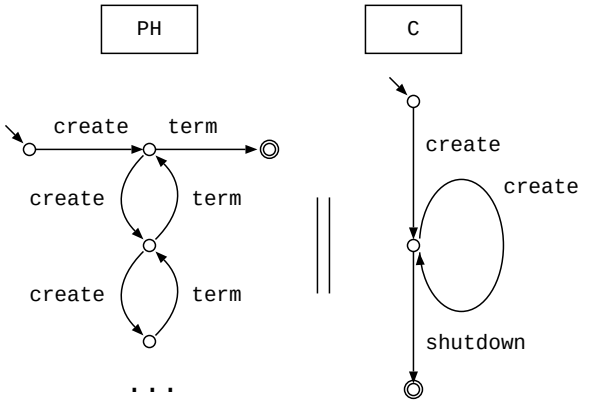

Fig. 3.3. The parallel components of the process management system.

service until it is 'shut-down' by the controller C via the action shutdown, which causes the process management system to deadlock after having terminated all active processes. By the construction of Theorem 3.6.1 we obtain the process management system $\texttt{PMS} =_{\text{df}} \texttt{PH} \parallel \texttt{C}$ as follows:

$$\begin{aligned}
\texttt{PMS} = \{ \quad [Y_0, X_0] = \;& \texttt{create}.[Y_1, X_1], \\
[Y_1, X_1] = \;& \texttt{create}.[Y_1, X_1 X_1] + \texttt{term}.[Y_1, \epsilon] + \\
& \texttt{shutdown}.[Y_2, X_1], \\
[Y_2, X_1] = \;& \texttt{term}.[Y_2, \epsilon] \quad \}
\end{aligned}$$

The construction also yields the equations

$$\begin{aligned}
[Y_0, X_1] &= \texttt{create}.[Y_1, X_1 X_1] + \texttt{term}.[Y_0, \epsilon] \\
[Y_1, X_0] &= \texttt{create}.[Y_1, X_1] + \texttt{shutdown}.[Y_2, X_0]
\end{aligned}$$

which are, however, useless since the fragments $[Y_0, X_1]$ and $[Y_1, X_0]$ do not occur on a right-hand side in PMS.
Finally, the labelled transition graph of the overall system which results from the specification PMS given above is shown in Figure 3.4.

3.7 Parallel Decomposition and 2-PDNF

When considering classes of formal languages, as well as classes of processes, representation theorems are of particular interest since they may provide additional insights in the underlying structure. For example, in formal language theory the Chomsky–Schützenberger Theorem (cf. [Har78]) states that for

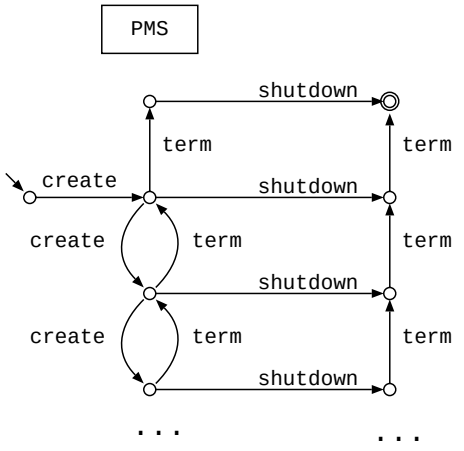

Fig. 3.4. The expanded process management system.

each context-free language L there exists a semi-Dyck language D, a regular language R, and a homomorphism φ such that

$$L = \varphi(D \cap R).$$

In this section we present a similar characterisation theorem for pushdown processes, which intuitively states that each pushdown process $\mathcal{D}$ can be represented as the relabelled synchronous parallel composition of a context-free process $\mathcal{C}$ and a regular process $\mathcal{R}$, i.e.

$$\mathcal{D} \sim (\mathcal{C} \parallel \mathcal{R})[\varphi].$$

Besides making clear the difference between context-free processes and pushdown processes, the representation theorem also allows to obtain some structural properties of pushdown specifications. For example, using the well-known 2-GNF for BPA processes the representation theorem implies that every pushdown process can effectively be rewritten in such a way that each fragment $[q, \gamma]$ appearing on a right-hand side has length at most 2.

We start our exposition by introducing formally what it means when a process is being relabelled.

Definition 3.7.1 (Relabelling).
The behaviour of a relabelled process $P[\varphi]$ with relabelling function $\varphi : Act \longrightarrow Act'$ is determined by the transition rule:

$$\frac{P \xrightarrow{a} P'}{P[\varphi] \xrightarrow{\varphi(a)} P'[\varphi]}$$

By means of relabelling and synchronous parallel composition we are now able to state our representation theorem for pushdown processes.

Theorem 3.7.1 (Decomposition Theorem).
For any guarded PDPA specification $\mathcal{D}$ there exist effectively a guarded BPA specification $\mathcal{C}$, a guarded regular specification $\mathcal{R}$, and a relabelling $[\varphi]$ such that

$$\mathcal{D} \quad \sim \quad (\mathcal{C} \parallel \mathcal{R})\ [\varphi]$$

Proof. Let $\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}_{\mathcal{D}}, [q_1, Z_1])$ be a guarded PDPA specification in PDNF with state set $Q = \{ q_1, \ldots, q_n \}$. Then we define the regular process specification

$$\mathcal{R} =_{\mathrm{df}} (V_Q, Act', \mathcal{E}_{\mathcal{R}}, X_{q_1})$$

with variable set $V_Q =_{\mathrm{df}} \{ X_{q_1}, \ldots, X_{q_n} \}$ and action set $Act' =_{\mathrm{df}} Act \times Q \times Q$ by

$$X_q = \sum_{i,j_i} a_{j_i}^{(q,q'_{j_i})}.X_{q'_{j_i}} \in \mathcal{E}_{\mathcal{R}} \quad \text{iff} \quad [q, Z_i] = \sum_{j_i} a_{j_i}.[q'_{j_i}, \beta_{j_i}] \in \mathcal{E}_{\mathcal{D}}$$

while the BPA specification

$$\mathcal{C} =_{\mathrm{df}} (\mathcal{Z}, Act', \mathcal{E}_{\mathcal{C}}, Z_1)$$

is defined by

$$Z = \sum_{i,j_i} a_{j_i}^{(q_i,q'_{j_i})}.\beta_{j_i} \in \mathcal{E}_{\mathcal{C}} \quad \text{iff} \quad [q_i, Z] = \sum_{j_i} a_{j_i}.[q'_{j_i}, \beta_{j_i}] \in \mathcal{E}_{\mathcal{D}}.$$

Obviously, $\mathcal{R}$, as well as $\mathcal{C}$, are both guarded specifications.

Finally, the relabelling function φ is defined by simply dropping the index (q, q') from any action $a^{(q,q')}$, i.e.

$$\varphi(a^{(q,q')}) =_{\mathrm{df}} a$$

The theorem is now proved by showing that

$$S = \{ ([q, \gamma], (X_q \parallel \gamma)[\varphi]) \mid q \in Q, \gamma \in \mathcal{Z}^* \}$$

is a bisimulation up to $\sim$. First observe that the parallel composition of $\mathcal{C}$ and $\mathcal{R}$ is always forced to synchronise since both process specifications are defined over the same set of actions Act'. Hence, processes of the form $[q, \epsilon]$ and $X_q \parallel \epsilon$ cannot perform any action. Now suppose that we have a transition

$$[q, Z\gamma] \xrightarrow{a} [q', \beta]; [\bar{q}_n, \gamma]$$

due to some transition $[q, Z] \xrightarrow{a} [q', \beta]$. Then by construction of $\mathcal{E}_{\mathcal{R}}$ and $\mathcal{E}_{\mathcal{C}}$ we have

$$X_q \xrightarrow{a^{(q,q')}} X_{q'} \quad \text{and} \quad Z \xrightarrow{a^{(q,q')}} \beta,$$

respectively. Thus also the parallel composition may perform the transition

$$(X_q \parallel Z\gamma)[\varphi] \xrightarrow{a} (X_{q'} \parallel \beta\gamma)[\varphi]$$

and we have clearly

$$[q', \beta]; [\bar{q}_n, \gamma] \sim [q', \beta\gamma] \; S \; (X_{q'} \parallel \beta\gamma)[\varphi].$$

To prove the converse direction assume that there exists a transition

$$(X_q \parallel Z\gamma)[\varphi] \xrightarrow{a} (X_{q'} \parallel \beta\gamma)[\varphi]$$

Since this must be a synchronised transition we know that

$$X_q \overset{a^{(q,q')}}{\rightarrow} X_{q'} \quad \text{and} \quad Z \overset{a^{(q,q')}}{\rightarrow} \beta.$$

Thus there exists a right-hand side summand $a.[q', \beta]$ of $[q, Z]$ in $\mathcal{E}_{\mathcal{D}}$. This yields

$$[q, Z\gamma] \xrightarrow{a} [q', \beta]; [\bar{q}_n, \gamma],$$

and again we obtain

$$[q', \beta]; [\bar{q}_n, \gamma] \sim [q', \beta\gamma] \; S \; (X_{q'} \parallel \beta\gamma)[\varphi]$$

which completes the proof. □

The decomposition theorem thus shows that pushdown processes form the smallest extension of context-free processes up to relabelling being closed under parallel composition with regular processes. As a corollary we obtain that each pushdown process can be rewritten into 2-PDNF, which is essential for the presentation of our algorithm.

Corollary 3.7.1 (2-PDNF).
For any guarded PDPA specification $\mathcal{D}$, we can effectively find a guarded PDPA specification $\mathcal{D}'$ in 2-PDNF such that $\mathcal{D} \sim \mathcal{D}'$.

Proof. Given a guarded PDPA specification $\mathcal{D}$, apply Theorem 3.7.1 to obtain the guarded BPA specification $\mathcal{C}$, the guarded regular process specification $\mathcal{R}$, and the relabelling φ such that $\mathcal{D} \sim (\mathcal{C} \parallel \mathcal{R})[\varphi]$. Then rewrite $\mathcal{C}$ to a bisimilar BPA specification $\mathcal{C}'$ in 2-GNF and apply Theorem 3.6.1 to $\mathcal{C}' \parallel \mathcal{R}$. Finally, rename all actions according to φ.

Note that bisimulation is a congruence relation with respect to fully synchronous parallel composition and relabelling. Thus bisimilarity is preserved while replacing $\mathcal{C}$ by $\mathcal{C}'$. Moreover, Corollary 3.6.1 guarantees that the resulting PDPA specification is in 2-PDNF. □

3.8 Related work

Besides the Pushdown Process Algebra proposed in this monograph, there also exist other equally expressive frameworks modelling the class of pushdown processes. These approaches use a variety of formalisms exploiting graph decomposition properties, particular rewrite techniques, pushdown automata based methods, deterministic graph grammars and logical characterisations for the description of pushdown processes. Summarising, we have the following theorem.

Theorem 3.8.1. *The classes of labelled transition graphs generated by the following formalisms coincide.*

- *Pushdown Process Algebra [BS94].*
- *Context-free graphs [MS85].*
- *Prefix transition graphs [Cau92].*
- *Pushdown transition graphs [CM90].*
- *Equational graphs of finite degree[Cou90].*
- *MSOL definable graphs of finite width and of finite degree[Cou89].*

In this section we present successively the different formalisms, and show how they model pushdown processes by means of a fixed example. For this purpose we take the pushdown specification $\mathcal{D}$, and its associated transition graph $\mathcal{T}_{\mathcal{D}}$, both given in Figure 3.5. The root of $\mathcal{T}_{\mathcal{D}}$ corresponding to $[q_1, Z_1]$ is hereby indicated by the arrow without source.

Let $\mathcal{D} = (\{ q_1, q_2 \}, \{ Z_1, Z_2 \}, \{ a, b, c, d \}, \mathcal{E}, [q_1, Z_1])$ be the PDPA specification where

$$\mathcal{E} = \left\{ \begin{array}{lcl} [q_1, Z_1] & = & a.[q_1, Z_2 Z_1] + c.[q_2, Z_1] \\ [q_1, Z_2] & = & a.[q_1, Z_2 Z_2] + b.[q_1, \epsilon] + c.[q_2, Z_2] \\ [q_2, Z_2] & = & d.[q_2, \epsilon] \end{array} \right\}$$

$\mathcal{T}_{\mathcal{D}} =$

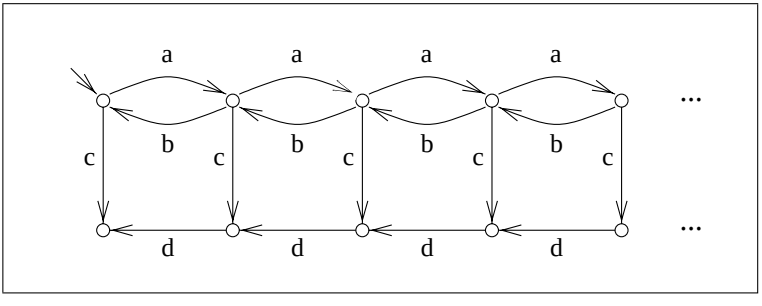

Fig. 3.5. An example pushdown process.

Finally, we close this section by presenting an extension of BPA proposed by Baeten and Bergstra [BB91] which incorporates similar ideas as PDPA, but yields a different class of processes.

3.8.1 Context-Free Graphs

In a landmark paper Muller and Schupp [MS85] introduced the class of *context-free graphs*[5] which is of special interest in the theory of ends since each context-free graph is "finitely behaved at infinity". As a main result, the authors show that a graph is context-free iff it is the transition graph of some pushdown automaton. More remarkably, they relate the theory of monadic second-order logic (MSOL) on context-free graphs to certain tiling problems on these graphs which are further reducible to the emptiness problem on some associated infinite trees. Since by Rabin's theorem [Rab69] the latter problem is known to be solvable this establishes that the monadic second-order theory of context-free graphs is decidable. Subsequently, we briefly recall the basic definitions concerning the class of context-free graphs, and show that it contains the labelled transition graph $\mathcal{T}_{\mathcal{D}}$.

A labelled graph $G = (V_G, \Sigma, E_G)$ is a triple where V_G is a set of *vertices*, Σ is an *alphabet*, and $E_G \subseteq V_G \times \Sigma \times V_G$ is a set of *edges*. We call a labelled graph G *finitely generated* if it has the following three properties:

1. the label alphabet Σ of G is finite,
2. G is a connected graph with a distinguished vertex v_0, called the root of G, and
3. G has uniformly bounded out-degree b, i.e. each vertex of G has at most b outgoing edges.

Now let G be a finitely generated graph with root v_0. For any vertex v of G, we write $|v|$ to denote the length of a shortest path from v_0 to v, also called the *distance* of v from v_0. Moreover, let $V_G^{(n)}$ be the set of all vertices of G which have distance of less than n from v_0. Then by $G^{(n)}$ we mean the subgraph of G consisting of $V_G^{(n)}$ together with all incident edges. Thus, for example, $G^{(0)}$ is empty and $G^{(1)}$ consists of v_0 and incident edges.

Figure 3.6 illustrates the previous definitions for our example graph $\mathcal{T}_{\mathcal{D}}$. The vertices on a dashed line labelled with n have distance n from the root. Accordingly, $G^{(n)}$ denotes the subgraph to the left of distance line n.

Since G has uniformly bounded out-degree, $G \setminus G^{(n)}$ consists of finitely many connected components. If C is such a component, a *frontier point* of C is a vertex u of C such that $|u| = n$. If v is a vertex of G with $|v| = n$, then we shall use $G(v)$ to denote the component of $G \setminus G^{(n)}$ which contains v. For

[5] In the light of later developments the term "context-free graph" has turned out to be an unfortunate choice as it may lead to confusion with the classification of context-free processes and pushdown processes.

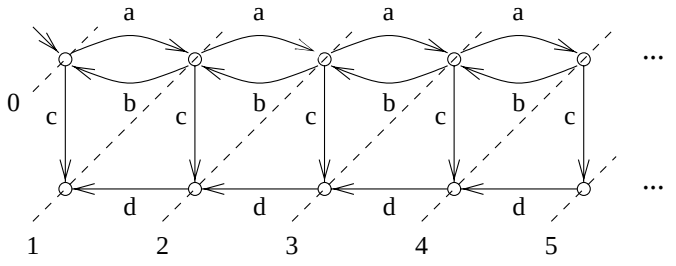

Fig. 3.6. The distances of vertices from the root in $\mathcal{T}_{\mathcal{D}}$.

each vertex v, the set of frontier points of $G(v)$, denoted by $Fr(v)$, is finite due to the finite branching of G.

Definition 3.8.1. *Let u and v be vertices of G. An* end-isomorphism *between the two subgraphs $G(u)$ and $G(v)$ is a mapping $\psi : G(u) \longrightarrow G(v)$ such that*

– *ψ is a label-preserving graph isomorphism, and*
– *ψ maps $Fr(u)$ onto $Fr(v)$.*

Finally, we are in a position to define the notion of a context-free graph.

Definition 3.8.2 (Context-Free Graph).
A graph G is said to be context-free *if G is a finitely generated graph such that*

$$\{\, G(v) \mid v \text{ is a vertex of } G \,\}$$

has only finitely many isomorphism classes under end-isomorphisms.

The labelled transition graph of Figure 3.5 is obviously finitely generated. Moreover, it has only two isomorphism classes under end-isomorphisms. The first is the whole graph itself, while the second one is given in Figure 3.7 where we have colored the frontier points black. Hence this shows that $\mathcal{T}_{\mathcal{D}}$ is a context-free graph.

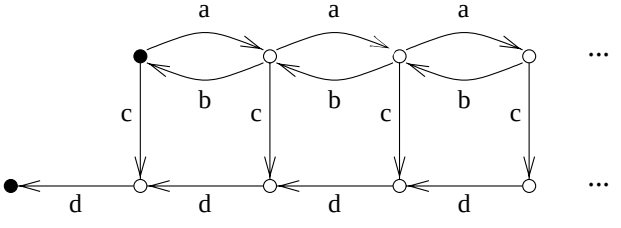

Fig. 3.7. The nontrivial isomorphism class of $\mathcal{T}_{\mathcal{D}}$ under end-isomorphisms.

3.8.2 Prefix Transition Graphs

A different formalism has been investigated by Caucal in [Cau92]. The author considers labelled rewrite systems, i.e. word rewrite systems where each pair contained in the rewrite relation is labelled with some terminal letter from an alphabet Σ. Formally, we have

Definition 3.8.3. *A* labelled rewrite system $\mathcal{R}$ *is a triple* $(X^*, \Sigma, \{\overset{a}{\hookrightarrow}\}_{a\in\Sigma})$ *where*

- X *is a set of nonterminals,*
- Σ *is a finite alphabet, and*
- $\{\overset{a}{\hookrightarrow}\}_{a\in\Sigma}$ *is a family of finite rewrite relations on* $X^* \times X^*$.

As usually, we write $\alpha \overset{a}{\hookrightarrow} \beta$, for $(\alpha, \beta) \in \overset{a}{\hookrightarrow}$, and call $\alpha \overset{a}{\hookrightarrow} \beta$ a rule of $\mathcal{R}$.

Instead of general rewriting, Caucal considers a particular variant, called *prefix rewriting*, where rewrite steps may only occur at the beginning of the word to be rewritten.

Let $\mathcal{R} = (X^*, \Sigma, \{\overset{a}{\hookrightarrow}\}_{a\in\Sigma})$ be a labelled rewrite system. A *prefix rewrite step* with respect to $\mathcal{R}$ is a labelled transition $\alpha\gamma \overset{a}{\to} \beta\gamma$ where $\alpha \overset{a}{\hookrightarrow} \beta$ is a rule of $\mathcal{R}$. We shall extend this definition by reflexivity and transitivity to allow $\alpha \overset{w}{\to} \beta$, for $w \in \Sigma^*$.

Definition 3.8.4 (Prefix Transition Graph).
The prefix transition graph $G(\mathcal{R}, r)$ *of a labelled rewrite system* $\mathcal{R}$ *and an axiom* $r \in X^*$ *has then as vertices the set* V *of words* $\alpha \in X^*$ *such that* $r \overset{w}{\to} \alpha$ *for some* $w \in \Sigma^*$, *and as edges* $\alpha \overset{a}{\to} \beta$ *whenever* $\alpha, \beta \in V$ *and* $\alpha \overset{a}{\to} \beta$ *is a prefix rewrite step.*

The labelled transition graph of Figure 3.5 is now obtained as the prefix transition graph $G(\mathcal{R}, q_1)$ of the labelled rewrite system

$$\mathcal{R} = (\{q_1, q_2, X\}, \{a, b, c, d\}, \mathcal{E}) \quad \text{where} \quad \mathcal{E} = \left\{ \begin{array}{rcl} q_1 & \overset{a}{\hookrightarrow} & q_1X \\ q_1X & \overset{b}{\hookrightarrow} & q_1 \\ q_1 & \overset{c}{\hookrightarrow} & q_2 \\ q_2X & \overset{d}{\hookrightarrow} & q_2 \end{array} \right\}$$

3.8.3 Pushdown Transition Graphs

As we mentioned earlier, from an operational point of view pushdown automata may also be interpreted as labelled rewrite systems [CM90]. Given a pushdown automaton $\mathcal{A} = (Q, \mathcal{Z}, \Sigma, \vartheta, q_1, Z_1)$, we consider the labelled rewrite system

$$R_{\mathcal{A}} =_{\text{df}} ((Q \cup \mathcal{Z})^*, \Sigma, \{\overset{a}{\to}\}_{a\in\Sigma}),$$

called a *pushdown rewrite system*, where $qZ \xrightarrow{a} q'\gamma$ is a rule of $R_{\mathcal{A}}$ whenever $(q', \gamma) \in \vartheta(q, a, Z)$. Each such pushdown rewrite system defines a transition graph by means of general rewrite steps[6] beginning with an axiom r.

Definition 3.8.5 (Pushdown Transition Graph).
Let $\mathcal{A} = (Q, \mathcal{Z}, \Sigma, \vartheta, q_1, Z_1)$ be a pushdown automaton. The pushdown transition graph $G(R_{\mathcal{A}}, q_1 Z_1)$ *of $\mathcal{A}$ is the transition graph of the pushdown rewrite system $R_{\mathcal{A}}$ with respect to the axiom $q_1 Z_1$.*

Obviously, the transition graph $G(R_{\mathcal{A}}, q_1 Z_1)$ of the pushdown automaton

$$\mathcal{A} = (\{ q_1, q_2 \}, \{ Z \}, \{ a, b, c, d \}, \vartheta, q_1, Z_1)$$

where

$$\begin{array}{lcl} \vartheta(q_1, a, Z) & = & \{ (q_1, ZZ) \} \\ \vartheta(q_1, b, Z) & = & \{ (q_1, \epsilon) \} \\ \vartheta(q_1, c, Z) & = & \{ (q_2, \epsilon) \} \\ \vartheta(q_2, d, Z) & = & \{ (q_2, \epsilon) \} \end{array}$$

is identical with $\mathcal{T}_{\mathcal{D}}$.

It may seem that labelled rewrite systems as given in Definition 3.8.3 are more expressive than pushdown rewrite systems since the former are defined by rules over arbitrary words and thus do not have any notion of state or stack symbols. It is therefore remarkable that the class of prefix transition graphs does coincide with the class of pushdown transition graphs [Cau92].

3.8.4 Equational graphs

A somewhat more complicated approach exploits the class of graphs generated by deterministic hypergraph grammars [Cou90]. In the sequel, we present the basic concepts of this framework, and briefly give the necessary definitions.

Hyperedges generalise the concept of directed edges as arcs from a source to a target by interpreting edges as a sequence of k vertices. The number k is then called the *type* of the hyperedge, and accordingly a hyperedge of type 2 is an ordinary edge. Hyperedges may also be labelled. However, the alphabet of hyperedge labels must then be a *ranked alphabet* B, i.e. it must be given with a type mapping $\tau : B \longrightarrow \mathbb{N}$, and the type of a hyperedge must coincide with the type of its label.

Definition 3.8.6 (Hypergraph).
A hypergraph over B of type n *is a five tuple*

$$H = (V_H, E_H, lab_H, vert_H, src_H)$$

where

[6] In fact, the distinction between prefix rewrite steps and general rewrite steps is in this case not necessary, as the special form of the rewrite rules guarantees that a rewrite step may only modify a prefix of the current word.

– V_H *is the set of* vertices,
– E_H *is the set of* hyperedges,
– $lab_H : E_H \longrightarrow B$ *defines the label of a hyperedge,*
– $vert_H : E_H \longrightarrow V_H^*$ *defines the sequence of vertices of a hyperedge, and*
– src_H *is a sequence of* n *vertices of* H.

We impose the condition that the labels of hyperedges are well-typed, i.e. that the length of $vert_H(e)$ *is equal to* $\tau(lab_H(e))$ *for all* e *in* E_H. *An element of* src_H *is called a* source *of* H.

Intuitively, the sources of a hypergraph are its external reference points used when the hypergraph is embedded into a larger context which will be another hypergraph.

As usual, a hypergraph is said to be *finite* if the set of vertices, as well as the set of hyperedges is finite. The class of all hypergraphs over B of type n is denoted by $\mathbf{G}(B)_n$, while $\mathbf{FG}(B)_n$ denotes the class of all finite hypergraphs of type n. Finally, a hypergraph of type n is also called an n-hypergraph.

Hypergraph Expressions. Hypergraphs may also be described algebraically by means of *hypergraph expressions*. To formalise this approach we introduce a many-sorted algebra (cf. [Wec92]) of hypergraphs as follows.

Definition 3.8.7 (Algebra of Hypergraphs).
The many-sorted algebra of hypergraphs $\boldsymbol{G}(B)$ *has* $\mathbb{N}$ *as the set of sorts ,* $\boldsymbol{G}(B)_n$ *as the carrier of sort* n, *and*

$\boldsymbol{0}:$	$0,$	
$\boldsymbol{1}:$	$1,$	
$\boldsymbol{b}:$	$\tau(b),$	*for* $b \in B$
$\oplus_{n,m}:$	$(n,m) \longrightarrow n+m,$	*for* $n,m \in \mathbb{N}$
$\theta_{\delta,n}:$	$n \longrightarrow n,$	*for* $n \in \mathbb{N}$
$\sigma_{\alpha,p,n}:$	$n \longrightarrow p,$	*for* $n,p \in \mathbb{N}$

as an infinite $\mathbb{N}$*-signature* H. $\boldsymbol{0}$ *is interpreted as the empty graph, while* $\boldsymbol{1}$ *represents the* 1*-graph consisting of one vertex that is its unique source. For every* $b \in B$, $\boldsymbol{b}$ *denotes the* $\tau(b)$*-hypergraph* H *consisting of one edge* e, *labelled by* b *and with* $src_H = vert_H(e)$. *Finally, the three operations* $\oplus, \theta_\delta, \sigma_\alpha$ *which are actually operation schemes defining infinitely many operations are interpreted as follows.*

Union: *Let* $G \in \boldsymbol{G}(B)_n$ *and* $G' \in \boldsymbol{G}(B)_m$. *Then* $G \oplus_{n,m} G'$ *denotes the* $n+m$*-hypergraph obtained by disjoint union of* G *and* G' *with the concatenation of* src_G *and* $src_{G'}$, *as the sequence of sources. Note that this operation is not commutative due to the concatenation of sources.*

Fusion: *Let* δ *be an equivalence relation on* $\{1, \ldots, n\}$, *and* $G \in \boldsymbol{G}(B)_n$. *Then* $\theta_{\delta,n}(G)$ *denotes the* n*-hypergraph obtained from* G *by fusing its* i*-th and* j*-th sources for every* (i,j) *in* δ.

Source Selection: *Let* $\alpha : \{1, \ldots, p\} \longrightarrow \{1, \ldots, n\}$ *be a total mapping and* $G \in \boldsymbol{G}(B)_n$. *Then* $\sigma_{\alpha,p,n}(G)$ *is the p-hypergraph consisting of* G *equipped with*

$$(src_G(\alpha(1)), \ldots, src_G(\alpha(p)))$$

as the sequence of sources.

In the sequel, the set of all *hypergraph expressions* of sort n is denoted by $\mathbf{FE}(B)_n$, and we write $val(g)$ for the finite hypergraph defined by an expression g.

To illustrate the use of hypergraph expressions we define the sequential composition of labelled transition graphs given in Section 2.5 in terms of the operations $\oplus, \theta_\delta, \sigma_\alpha$.

Example 3.8.1. Let $G_i \in \mathbf{FG}(Act)_{n+1}$, for $0 \leq i \leq n$, where Act is interpreted as a ranked alphabet with $\tau(a) = 2$, for all $a \in Act$. Moreover, we will interpret $src_{G_i}(1)$ as the start vertex, and $src_{G_i}(2), \ldots, src_{G_i}(n+1)$ as the sequence of end vertices, for each G_i. The sequential composition $G_0 ; (G_1, \ldots, G_n)$ is then defined by the expression

$$\sigma_{\alpha,n+1,(n+1)^2}(\theta_{\delta,(n+1)^2}(\ldots((G_0 \oplus_{n+1,n+1} G_1) \oplus_{2(n+1),n+1} G_2) \\ \ldots \oplus_{n(n+1),n+1} G_n))$$

Here δ denotes the equivalence relation on $\{1, \ldots, (n+1)^2\}$ consisting of the singleton equivalence class $\{1\}$, meaning that the start vertex of G_0 is not joined with any other vertex, the n equivalence classes

$$\{1 + i, i(n+1) + 1\}_{i=1}^n,$$

responsible for joining the i-th end vertex of G_0 with the start vertex of G_i, as well as the n equivalence classes

$$\{j(n+1) + i \mid 1 \leq j \leq n\}_{i=1}^n,$$

responsible for joining the i-th end vertices of $G_1, \ldots, G_n$. Moreover, α denotes the mapping $1 \mapsto 1$, as well as $1 + i \mapsto (n+1) + 1 + i$, for $1 \leq i \leq n$.

A notion we will need in the next subsection when relating graphs definable by a formula with equational graphs concerns the maximal sort occurring in a graph expression.

Definition 3.8.8. *The* width *of an hypergraph expression* g *in* $\boldsymbol{FE}(B)_n$, *denoted by* $wd(g)$, *is the maximal sort of a symbol of* H *occurring in* g. *The* width *of a finite n-hypergraph* G *is then defined as*

$$wd(G) =_{\mathrm{df}} \min\{wd(g) \mid g \in \boldsymbol{FE}(B)_n, val(g) = G\}$$

Hypergraph Grammars. As context-free grammars generate words, *hypergraph grammars* are used to generate (finite) hypergraphs. Both notions of generation may also be extended to infinite structures yielding context-free ω-languages in the first case, and sets of infinite hypergraphs in the second one. Deterministic hypergraph grammars which generate a single hypergraph are of particular interest in this setting since they allow to finitely represent a possibly infinite structure in a concise way. The key concept in this formalism are *equational hypergraphs* which are defined as a component of the canonical solution of a system of hypergraph equations.

Definition 3.8.9. *A* regular system of graph equations *over* $\boldsymbol{G}(B)$ *is a system of the form*

$$S = (x_1 = H_1, \ldots, x_n = H_n)$$

where $X = \{ x_1, \ldots, x_n \}$ *is a ranked alphabet, called the set of* nonterminals *of* S*, and where* H_i*, for* $1 \leq i \leq n$*, is a hypergraph in* $\boldsymbol{FG}(B \cup X)_{\tau(x_i)}$*.*

A solution *of* S *is an* n*-tuple of hypergraphs* $(G_1, \ldots, G_n)$ *with* G_i *in* $\boldsymbol{G}(B)_{\tau(x_i)}$ *such that*

$$G_i = H_i[G_1/x_1, \ldots, G_n/x_n],$$

for all i*, where* $H[G_1/x_1, \ldots, G_n/x_n]$ *denotes the simultaneous replacement of all hyperedges labelled with* x_i *in* H *by the hypergraph* G_i*.*

As it is the case in the theory of recursive applicative program schemes one is interested in the "least" solution of such a system. Courcelle gives a categorical and a graph-expression based approach for finding the appropriate solution which is called the *canonical solution*. First one can associate with each system of equations a category and a functor such that the initial fixpoint of this functor yields the so called *initial* solution of S. Secondly, the system of equations can be solved formally in the domain $\mathbf{E}(B)$ of finite and infinite graph expressions. Each H_i is replaced by a graph expression denoting it and the associated system $\bar{S}$ is solved in the algebra of infinite trees. It can be shown that these two approaches yield the same tuple of graphs.

The class of graphs defined by a regular system of graph equations is strictly larger than the class of pushdown transition graphs. The former includes e.g. graphs with infinite degree. The restriction to finite degree, which can syntactically be determined from the graph equations, yields, however, exactly the class of pushdown transition graphs.

The labelled transition graph $\mathcal{T}_{\mathcal{D}}$, for example, is the canonical solution of the graph equation of Figure 3.8 where the numbers indicate the sources of the nonterminal x_1 and the right-hand side graph.

3.8.5 MSOL Definable Hypergraphs

The last formalism we consider treats graphs as logical structures. In this approach logical formulas express graph properties, and moreover, may be

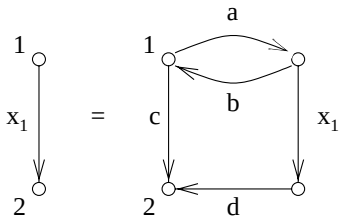

Fig. 3.8. The graph equation characterising $\mathcal{T}_{\mathcal{D}}$.

used to define sets of graphs. Key idea is to relate with a formula the set of those graphs which satisfy the property denoted by the formula under consideration. Interpreted this way, logical languages thus define sets of classes of graphs which can be ordered according to the expressive power of the considered logic. Besides classical logics like first-order or second-order logic particularly the theory of *monadic second-order logic* (MSOL) has attracted a lot of research as it is closely related to the class of equational graphs defined in the previous section (cf. [Cou89]). MSOL extends first-order logic by admitting quantification over monadic second-order predicates. In the sequel, we introduce the monadic second-order theory of graphs and give the connection between this logic and the class of pushdown transition graphs.

Let v denote the sort of *vertices* and e the sort of *edges*, respectively. For every $b \in B$, we define a relational symbol of arity $\tau(b) + 1$ and of type $(e, v, \ldots, v)$ which intuitively expresses that the first argument is a hyperedge equipped with the sequence of vertices given by the remaining arguments. A hypergraph $G = (V_G, E_G, lab_G, vert_G, src_G)$ contained in $\mathbf{G}(B)_n$ may now be interpreted as the logical $\{ v, e \}$-sorted structure

$$\underline{G} =_{\mathrm{df}} (V_G, E_G, (edg_b^G)_{b \in B}, (s_i^G)_{i \in [n]})$$

where edg_b^G has the interpretation

$$edg_b^G(e, v_1, \ldots, v_{\tau(b)}) \quad \text{iff} \quad lab_G(e) = b \ \wedge \ vert_G(e) = (v_1, \ldots, v_{\tau(b)}),$$

and s_i^G denotes a constant of sort v interpreted as $src_G(i)$.

In the following we use *object variables* $x, y, \ldots$ of sort v or e to range over V_G or E_G, respectively, while *set variables* $X, Y, \ldots$ of sort v or e will range over 2^{V_G} or 2^{E_G}. Moreover, the mapping from variables to sorts is given by σ. Now let Var be a $\{ v, e \}$-sorted set of variables and $S = \{ s_1, \ldots, s_k \}$ be a set of vertex constants. As usual, uppercase letters will denote set variables, while object variables and constants are denoted by lowercase letters. The set of *atomic formulas* is then defined by

$$
\begin{array}{lll}
x = x' & \text{for} & x, x' \in Var \cup S \text{ and } \sigma(x) = \sigma(x'), \\
x \in X & \text{for} & x, X \in Var \cup S \text{ and } \sigma(x) = \sigma(X), \\
edg_b(x, v_1, \ldots, v_n) & \text{for} & x, v_1, \ldots, v_n \in Var \cup S, \sigma(x) = e, \\
 & & \text{and } \sigma(v_i) = v, \text{ for } i = 1, \ldots, n, \\
fin(X) & \text{for} & X \in Var \cup S.
\end{array}
$$

Atomic formulas have the usual semantics. In particular, $fin(X)$ holds if X is interpreted as a finite set. Arbitrary formulas of the logic MSOL are then generated from atomic formulas by the Boolean connectives $\neg$, $\vee$, $\wedge$, and the quantifiers $\forall, \exists$ ranging either over object variables, or over set variables, respectively.

Since formulas may be interpreted as predicates on the set of all graphs a graph G is said to be $\mathcal{L}$-definable for some logic $\mathcal{L}$ if G can be characterised up to isomorphism by a formula of $\mathcal{L}$. The main result of this approach is now stated in the following theorem (cf. [Cou90]).

Theorem 3.8.2. *A hypergraph is equational iff it is MSOL-definable and of finite width.*

The class of pushdown transition graphs is thus obtained by restricting MSOL-definable graphs to finite width and to finite degree.

Courcelle [Cou89] remarks, that the construction of an MSOL-formula from a system of graph equations S is intractable since the formula obtained may be of exponential length in the size of S. As the proof of this theorem is rather technical we omit the MSOL characterisation for the graph given in Figure 3.5.

3.8.6 BPA with the state operator

In this final section we present an extension of BPA proposed by Baeten and Bergstra [BB91] that is based on similar ideas as PDPA. They extend BPA by the *state operator* which allows simultaneously to capture side-effects on a finite domain, as well as to rename the visible action resulting from a state transition.

Formally, one has, for each state s of a (finite) state space S, an operator λ_s which is added to the signature of BPA, and an expression $\lambda_s(E)$ then denotes that the process E is in the state s. Furthermore, there are two functions *action* and *effect* describing the visible action and the new state which result from the execution of an action.

$$
\boxed{
\begin{array}{ll}
\text{action}: & Act \times S \longrightarrow Act \\
\text{effect}: & Act \times S \longrightarrow S
\end{array}
}
$$

The behaviour of the state operator is then captured by the axioms SO1, SO2, and SO3, as well as its action rule, which are defined as follows.

Axioms for the state operator

SO1	$\lambda_s(a)$	$=$	$\mathrm{action}(a, s)$
SO2	$\lambda_s(aE)$	$=$	$\mathrm{action}(a, s)\ \lambda_{\mathrm{effect}(a,s)}(E)$
SO3	$\lambda_s(E + F)$	$=$	$\lambda_s(E) + \lambda_s(F)$

Action rule for the state operator

$$\frac{E \xrightarrow{a} E'}{\lambda_s(E) \xrightarrow{\mathrm{action}(a,s)} \lambda_{\mathrm{effect}(a,s)}\ (E')}$$

As can be seen from the action rule the state operator combines two independent concepts: first a finite memory in form of an additional control component and second the renaming of atomic actions. Extending regular processes with these two concepts does, however, not enhance their expressive power, as an application of the state operator to a regular process yields again a regular process. In contrast, the combination of BPA and the state operator leads, as proved by Baeten and Bergstra in [BB91], to a process algebra which is strictly more expressive than pure BPA.

Furthermore, one can show using similar techniques as in the proof of Theorem 3.7.1 that BPA with the state operator is at least as expressible as PDPA. However, it is not known to us whether this inclusion is strict although we conjecture that the answer will be affirmative.

4. Model Checking

4.1 Introduction

In [MS85] Muller and Schupp prove the remarkable result that the theory of monadic second-order logic (MSOL) is decidable for the class of pushdown transition graphs. As a consequence, for each temporal logic which can be interpreted in MSOL, notably the modal μ-calculus, the satisfaction problem is solvable for this class of graphs. A major drawback for practical applications is, however, the nonelementary complexity of the decision procedure.

In this chapter we present a more direct approach by developing an iterative model checker for the class of pushdown processes which decides the alternation-free fragment of the modal μ-calculus. The complexity of the algorithm is exponential in the size of the formula, while only quadratic in the size of the considered pushdown specification.

In Section 4.2 we describe the syntax and semantics of the modal μ-calculus. We then extend in Section 4.3 the ordinary semantics of μ-formulas to the assertion-based semantics and develop the theory of second-order semantics for sequential processes. In Section 4.4 we introduce equational μ-formulas and present our model checking algorithm. Finally, in Section 4.5 we show that the semantics of μ-formulas when interpreted on pushdown processes always describe regular sets of processes.

4.2 The Modal μ-Calculus

Verification by means of model checking assumes that the intended behaviour of the concurrent system at hand is specified by a formula of *temporal logic*. It is widely accepted that this approach is feasible to the specification and verification of concurrent systems, since temporal logic provides a powerful formalism for describing the occurrences of events in time.

When defining temporal logics, they can be classified regarding the underlying nature of time into *linear time* and *branching time* logics. While linear time temporal logics assume that at each moment there is only one possible future, branching time temporal logics consider time as being tree-like. At each moment in time there exists a choice point representing the different

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 67-114, 1997

courses of possible futures. These two points of view concerning the semantics of time are usually reflected by the modal operators of the temporal logic. Thus linear time logics provide operators for reasoning about runs which are sequences of events along a single time path, while modalities of branching time logic quantify over possible futures. For a more general survey about temporal logics we refer the reader to [Eme90, Sti92].

A particularly powerful branching time logic is the propositional modal μ-calculus introduced by Kozen [Koz83]. It combines standard modal logic with least and greatest fixpoint operators which allows to express very complex temporal properties within this formalism. Since it subsumes most of the other propositional modal logics such as PDL [FL79], PDLΔ [Str82], Process Logic [HKP82] and CTL* [EH86], it has attracted a lot of theoretical, as well as practical interest. For example, newer results have established the connection to tree automata by showing that the propositional modal μ-calculus is equally expressive as Rabin automata on infinite trees [Niw86, Niw88, EJ91], and thus also as powerful as monadic second-order logic on these trees. Moreover, satisfiability of modal μ-calculus can be tested in deterministic single exponential time [EJ88]. In this sense, modal μ-calculus is only as "hard" as the much weaker logic PDL. Recently obtained results also show that there exists a finitary sequent-style complete axiomatisation of the modal μ-calculus [Wal93] opening this logic even for theorem provers. Due to its expressiveness and its conciseness the modal μ-calculus can therefore be regarded as the "assembly language" of temporal logics.

In this section we introduce the modal μ-calculus as the specification language of our model checker. For algorithmic simplicity, however, we shall represent formulas also by means of *mutually recursive equational μ-formulas* (cf. [CS91, CS92]). When developing our model checker in Section 4.4 the alternation-free fragment of the modal μ-calculus, i.e. the fragment which does not contain *alternating fixed points* [EL86], will be of special interest. This fragment can also be interpreted as Hennessy-Milner Logic with recursion [Lar88], and is equally expressive as the class of formulas represented by *hierarchically* constructed equational μ-formulas (cf. [CKS92]).

4.2.1 Syntax

Syntactically, the modal μ-calculus is parameterised with respect to a (countable) set of variables Var, and a set of actions Act. In what follows, X will range over Var, and a over Act. Then the syntax of *μ-formulas* simplify our presentation, we only assume the atomic proposition $\mathtt{tt}$ here. However, an extension covering arbitrary *atomic propositions*, which are consistent with the representation of the transition system under consideration, is straightforward.

$$\Phi ::= \mathtt{tt} \mid X \mid \neg\Phi \mid \Phi \vee \Phi \mid \langle a \rangle \Phi \mid \mu X.\Phi$$

The symbols $\neg$ and $\vee$ represents negation and disjunction, respectively, while $\langle a \rangle$ is a *modal operator* parameterised by a. Finally, $\mu X.\Phi$ represents a recursive formula where the fixpoint operator μ binds all free occurrences of X in Φ. In order to ensure the well-definedness of the semantics we additionally impose the syntactic restriction on the body of $\mu X.\Phi$ that any occurrences of X in Φ must occur within the scope of an even number of negations.

To ease notation we introduce the standard abbreviations

$$\begin{aligned} \mathtt{ff} &=_{\mathrm{df}} \neg\mathtt{tt}, \\ \Phi_1 \wedge \Phi_2 &=_{\mathrm{df}} \neg(\neg\Phi_1 \vee \neg\Phi_2), \\ [a]\Phi &=_{\mathrm{df}} \neg\langle a\rangle\neg\Phi, \\ \nu X.\Phi &=_{\mathrm{df}} \neg\mu X.\neg\Phi[\neg X/X] \end{aligned}$$

where $\Phi[\Psi/X]$ denotes the simultaneous replacement of all free occurrences of X in Φ by Ψ[1].

A formula Φ is called *well named* if every fixpoint operator in Φ binds a distinct variable, and free variables are distinct from bound variables. In the remainder of this monograph we will assume that each formula is well named. Furthermore, the set of all μ-formulas will henceforth be denoted by $L\mu$.

The standard subformula relation $\prec$ on μ-formulas is now given by the following definition.

Definition 4.2.1. *The* immediate subformula *relation between μ-formulas, denoted by $\prec$, is defined by*

- *If $\Phi = \neg\Psi$ then $\Psi \prec \Phi$.*
- *If $\Phi = \Psi_1 \vee \Psi_2$ then $\Psi_1 \prec \Phi$ and $\Psi_2 \prec \Phi$.*
- *If $\Phi = \langle a\rangle\Psi$ then $\Psi \prec \Phi$.*
- *If $\Phi = \mu X.\Psi$ then $\Psi \prec \Phi$.*

As usual, the transitive closure of $\prec$ is denoted by $\prec^+$, while $\prec^*$ represents the reflexive, and transitive closure of $\prec$.

Definition 4.2.2 (Subformulas of Φ).
The set of all subformulas of *Φ is defined by*

$$SF(\Phi) =_{\mathrm{df}} \{\, \Psi \mid \Psi \prec^* \Phi \,\}.$$

This set of formulas can further be divided into $SF(\Phi)^+$ and $SF(\Phi)^-$ containing all subformulas of Φ which occur under an even, respectively odd, number of negations.

[1] As usual, the inadvertent binding of free variables of Ψ by fixpoint operators of Φ has to be avoided by appropriately renaming the bound variables of Φ.

Note that due to the syntactic restriction on bodies of fixpoint formulas we have that in each subformula of a closed formula all occurrences of a variable either occur under an even number of negations or they all occur under an odd number of negations, respectively. Hence a subformula of a closed formula Φ which contains a variable can only belong to either $SF(\Phi)^+$ or $SF(\Phi)^-$.

In the remainder, we shall use σ to represent either μ or ν. We call Ψ a *proper* subformula of Φ if $\Psi \prec^+ \Phi$, whereas a subformula of the form $\sigma X.\Psi$ is called a *σ-subformula*. Moreover, we say that Ψ is a *top-level* σ-subformula of Φ if Ψ is a proper σ-subformula of Φ and for every other σ-subformula Θ of Φ we have that Ψ is not a subformula of Θ.

When developing our model-checker in Section 4.4 we will represent μ-formulas by *equational μ-formulas* which are algorithmically more tractable. These formulas in equational form are closely related to particular finite sets of formulas, the so-called *Fischer–Ladner closures* [FL79].

Definition 4.2.3 (Fischer–Ladner Closure).
The Fischer–Ladner closure *of a μ-formula Φ, denoted by $CL(\Phi)$, is defined as the least set of formulas satisfying*

- $\Phi \in CL(\Phi)$
- *If* $\neg\Psi \in CL(\Phi)$ *then* $\Psi \in CL(\Phi)$.
- *If* $\Psi_1 \vee \Psi_2 \in CL(\Phi)$ *then* $\Psi_1 \in CL(\Phi)$ *and* $\Psi_2 \in CL(\Phi)$.
- *If* $\langle a\rangle\Psi \in CL(\Phi)$ *then* $\Psi \in CL(\Phi)$.
- *If* $\mu X.\Psi \in CL(\Phi)$ *then* $\Psi[\mu X.\Psi/X] \in CL(\Phi)$.

Finally, for complexity issues the notion of *size* for formulas is needed.

Definition 4.2.4 (Formula Size).
The size *of a formula Φ, denoted by $|\Phi|$, is inductively defined as*

- *If* $\Phi \in \{\, \mathtt{tt} \,\} \cup Var$ *then* $|\Phi| =_{\mathrm{df}} 1$.
- *If* Φ *is of the form* $\neg\Psi, \langle a\rangle\Psi$ *or* $\mu X.\Psi$ *then* $|\Phi| =_{\mathrm{df}} |\Psi| + 1$.
- *If* Φ *is of the form* $\Psi_1 \vee \Psi_2$ *then* $|\Phi| =_{\mathrm{df}} |\Psi_1| + |\Psi_2| + 1$.

4.2.2 Semantics

The formal semantics of μ-formulas appears in Table 4.1. It is given with respect to a labelled transition graph $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$, and a valuation $\mathcal{V}$ mapping variables to subsets of $\mathcal{S}$, where $\mathcal{V}[\, X \mapsto S \,]$ is the valuation resulting from $\mathcal{V}$ by updating the binding of X to S.

Intuitively, the semantic function maps a formula to the set of states for which the formula is "true". Accordingly, all states satisfy $\mathtt{tt}$, while s satisfies X if s is an element of the set bound to X in $\mathcal{V}$. Negation and disjunction

$$
\begin{array}{rcl}
[\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \mathcal{S} \\
[\![X]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \mathcal{V}(X) \\
[\![\neg\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \mathcal{S} \setminus [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} \\
[\![\Phi_1 \vee \Phi_2]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & [\![\Phi_1]\!]^{\mathcal{T}}_{\mathcal{V}} \cup [\![\Phi_2]\!]^{\mathcal{T}}_{\mathcal{V}} \\
[\![\langle a\rangle\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \{\, s \mid \exists\, s'.\ s \xrightarrow{a} s' \wedge s' \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} \,\} \\
[\![\mu X.\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S]} \subseteq S \,\}
\end{array}
$$

Table 4.1. The semantics of μ-formulas.

are interpreted in the usual fashion: s satisfies $\neg\Phi$ if it does not satisfy Φ, and s satisfies $\Phi_1 \vee \Phi_2$ if it satisfies either Φ_1 or Φ_2. The modal operator $\langle a\rangle$ is interpreted as s satisfies $\langle a\rangle\Phi$ if it has an a-derivative satisfying Φ. The interpretation of $\mu X.\Phi$ is somewhat complicated and depends on a fixpoint characterisation[2] given by Tarski and Knaster [Tar55, Kna28].

Recall that the powerset of any set M together with set inclusion as partial order, union as join, and intersection as meet forms a complete lattice. By the Tarski-Knaster Theorem 2.2.2, any monotone function $f : M \longrightarrow M$ has a least fixpoint, μf, and a greatest fixpoint, νf, given by

$$\mu f = \bigcap \{\, N \subseteq M \mid f(N) \subseteq N \,\}, \text{ and}$$

$$\nu f = \bigcup \{\, N \subseteq M \mid N \subseteq f(N) \,\}.$$

In case of the modal μ-calculus the syntactic restriction on bodies of fixpoint formulas and the semantics of the other logical operators ensures that the function f defined by

$$f(S) =_{\mathrm{df}} [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S]}$$

is monotone for any valuation $\mathcal{V}$ (cf. [Cle90]). Thus it has a least, respectively greatest, fixpoint which is taken as the semantics of $\mu X.\Phi$, respectively $\nu X.\Phi$. If the underlying labelled transition graph is finite-state, every monotone function f over the powerset lattice of states is also continuous. Therefore, the least, respectively greatest, fixpoint of f can iteratively be computed. In particular, we have

$$[\![\mu X.\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} = \bigcup_{i \in \mathbb{N}} f^i(\emptyset) \qquad \text{and} \qquad [\![\nu X.\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} = \bigcap_{i \in \mathbb{N}} f^i(\mathcal{S}). \tag{4.1}$$

Finally, for closed formulas it can be shown that the semantics is independent of the valuation, i.e. that we have $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}_1} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}_2}$, for any valuations $\mathcal{V}_1$ and $\mathcal{V}_2$. In this case we shall occasionally omit reference to the valuation.

[2] A good survey about the history of the various known fixpoint theorems is given by Lassez, Nguyen and Sonenberg in [LNS82].

4.2.3 Continuity

As pointed out by Stirling [Sti92], the iterative characterisation (4.1) of the semantics of fixpoint formulas is no longer valid for infinite-state systems. Consider, for example, the formula

$$\Phi =_{\mathrm{df}} \nu X.[b]Y \wedge [a]X$$

expressing "on every a-path after each b-transition Y is always true". That the semantics of Φ is, in general, not continuous in Y may then be illustrated by means of the labelled transition graph $\mathcal{T}$ given in Figure 4.1. This infinite transition graph is finitely-branching and moreover, even context-free since it is generated by the BPA specification $\mathcal{C} = \{ A = aAB + b, B = b \}$.

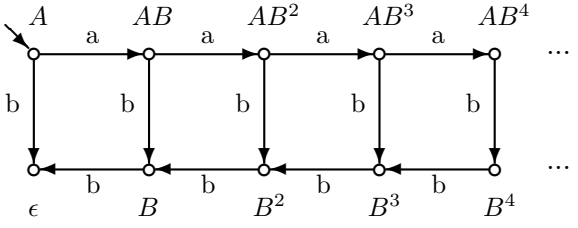

Fig. 4.1. A context-free labelled transition graph.

Now let the semantics of Φ with respect to the valuation of Y be abbreviated by

$$f(S) =_{\mathrm{df}} [\![\nu X.[b]Y \wedge [a]X]\!]^{\mathcal{T}}_{[Y:=S]}$$

and assume that Y holds for $\epsilon, B, B^2, \ldots$. This implies that all processes of $\mathcal{T}$ satisfy Φ, i.e. we have

$$f(\{ B^j \mid j \in \mathbb{N} \}) = \{ B^j \mid j \in \mathbb{N} \} \cup \{ AB^j \mid j \in \mathbb{N} \} = \mathcal{S}.$$

In contrast, assuming that only every *finite* sequence of processes $\epsilon, B, B^2, \ldots, B^j$ satisfies Y yields merely the set $\{ \epsilon, B, B^2, \ldots \}$ as the semantics of Φ, i.e. in this case we obtain

$$\bigcup_{j \in \mathbb{N}} f(\{ B^i \mid 0 \leq i \leq j \}) = \{ B^j \mid j \in \mathbb{N} \}.$$

From this we deduce that the function f is not continuous.
Nevertheless, in the general case we always have the weaker inclusion

$$\bigcup_{i \in \mathbb{N}} f^i(\emptyset) \subseteq [\![\mu Y.\Phi]\!]^{\mathcal{T}}_{\mathcal{V}}$$

for least fixpoints. Equality holds again if we introduce iteration over *transfinite ordinals* a mathematical notion which deals with counting beyond the natural numbers. Accordingly, the least transfinite ordinal is associated with the set of natural numbers and, as usual, denoted by ω. Using iteration over transfinite ordinals the semantics of $\mu Y.\Phi$ with respect to $\mathcal{T}$, for example, may be computed as follows.

$$\begin{aligned} f^j(\emptyset) &= \{ B^i \mid 0 \leq i \leq j \},\ j \in \mathbb{N} \\ f^\omega(\emptyset) &= \{ B^j \mid j \in \mathbb{N} \} \\ f^{\omega+2}(\emptyset) = f^{\omega+1}(\emptyset) &= \{ B^j \mid j \in \mathbb{N} \} \cup \{ AB^j \mid j \in \mathbb{N} \} = \mathcal{S} \end{aligned}$$

Hence in our particular example the least fixpoint of f is already reached by iterating one more step beyond ω. Considering the labelled transition graph $\mathcal{T}$ this fixpoint property is intuitively clear, as the least fixpoint of f corresponds to the closed μ-formula

$$\mu Y.\nu X.[b]Y \wedge [a]X$$

which expresses that "on every $\{ a, b \}$-path only finitely many b-transitions can occur".

4.2.4 Alternation Depth

When analysing the complexity of algorithms dealing with formulas it turns out that the number of nested alternating quantifications occurring in a formula often plays a crucial role. The appropriate notion for μ-formulas is called *alternation depth* and was introduced by Emerson and Lei [EL86]. The definition of alternation depth assumes that the formulas are given in *Positive Normal Form* (PNF), i.e. in negation-free form such that all bound variables are disjoint. This assumption does not impose any restriction, since every μ-formula can be translated into an equivalent formula in PNF by means of the transformation given in the following lemma (cf. [CKS92]).

Lemma 4.2.1 (PNF Transformation).
Let Φ be a closed μ-formula. Then Φ can be translated into an equivalent formula in PNF in $O(|\Phi|)$.

Proof. The translation is done by driving the negations in using the following rewrite rules, and by renaming variables as appropriate.

$$\begin{aligned} \neg\mathtt{tt} &\rightarrow \mathtt{ff} \\ \neg(\neg\Phi) &\rightarrow \Phi \\ \neg(\Phi_1 \vee \Phi_2) &\rightarrow (\neg\Phi_1) \wedge (\neg\Phi_2) \\ \neg(\langle a \rangle \Phi) &\rightarrow [a]\neg\Phi \\ \neg(\mu X.\Phi) &\rightarrow \nu X.\neg(\Phi[\neg X/X]) \end{aligned}$$

The resulting formula is negation-free due to the restriction that bound variables may only occur within the range of an even number of negations. □

The nesting depth of fixpoint operators occurring in a formula may now be formalised as follows.

Definition 4.2.5 (Alternation Depth).
The alternation depth *of a formula Φ in PNF, denoted by $ad(\Phi)$, is inductively defined as:*

1. *If Φ contains proper closed top-level σ-subformulas $\Phi_1 = \sigma X_1.\Psi_1, \ldots, \Phi_n = \sigma X_n.\Psi_n$, then*
   $$ad(\Phi) =_{\mathrm{df}} \max\{\, ad(\Phi[\mathtt{tt}/\Phi_1, \ldots, \mathtt{tt}/\Phi_n]), ad(\Phi_1), \ldots, ad(\Phi_n)\,\}$$
   where $\Phi[\mathtt{tt}/\Phi_1, \ldots, \mathtt{tt}/\Phi_n]$ means the formula obtained from Φ by replacing each Φ_i with $\mathtt{tt}$.
2. *If Φ does not contain any proper closed top-level σ-subformula, then*
   a) *If Φ is of the form $\mathtt{tt}, \mathtt{ff}$, or X then $ad(\Phi) =_{\mathrm{df}} 0$.*
   b) *If Φ is either $\Psi_1 \vee \Psi_2$, or $\Psi_1 \wedge \Psi_2$ then $ad(\Phi) =_{\mathrm{df}} \max\{\, ad(\Psi_1), ad(\Psi_2)\,\}$.*
   c) *If Φ is either $\langle a \rangle \Psi$, or $[a]\Psi$ then $ad(\Phi) =_{\mathrm{df}} ad(\Psi)$.*
   d) *If $\Phi = \sigma X.\Psi$ then*
      $$ad(\Phi) =_{\mathrm{df}} \max\{\, 1, ad(\Psi), 1 + ad(\bar{\sigma} X_1.\Psi_1), \ldots, 1 + ad(\bar{\sigma} X_n.\Psi_n)\,\}$$
      where $\bar{\sigma}$ denotes the dual fixpoint operator of σ, and $\bar{\sigma} X_1.\Psi_1, \ldots, \bar{\sigma} X_n.\Psi_n$ are the top-level $\bar{\sigma}$-subformulas of Ψ.

The alternation depth of a formula Φ which is not in PNF is defined as $ad(\Phi')$ where Φ' is obtained by transforming Φ into PNF. In the remainder of this monograph the set of all μ-formulas with alternation depth k shall be denoted by $L\mu_k$.

4.3 Assertion-Based Semantics

In the standard semantics the validity of a formula is defined with respect to single states. This notion cannot directly be adapted to sequential infinite-state processes: the truth value at the start state might not be the same in different contexts, i.e. for different continuations[3]. However, if the continuations satisfy the same sets of formulas, then so does the start state of the sequential process in these contexts. This observation is the key to the *assertion-based semantics*: It is consistent with usual semantics to view a process as a *property transformer*, i.e. as a function which yields the set of formulas valid at the start state, relative to the assertion that the tuple of sets of formulas to which the transformer is applied are valid at the end states.

[3] Note that we are dealing with forward modalities. Thus the validity of formulas "propagates" backward.

In this section we introduce the *assertion-based* semantics of μ-formulas. It may be interpreted as an extension of the ordinary semantics which allows to control the validity of formulas at certain states relative to *assertions*. Subsequently, the dual point of view is elaborated, where states are associated with the set of formulas that they satisfy under some given assertion. This point of view yields the *second-order* semantics, which is the key to our model checking algorithm.

4.3.1 A Motivating Example

To demonstrate the motivation behind our approach we give a simple example. We consider the μ-formula $\Phi =_{\mathrm{df}} \nu X.\langle a\rangle X$ which, intuitively, expresses the property "there exists an infinite a-path". The closure of the formula Φ is then the set $\{\, \Phi, \langle a\rangle\Phi \,\}$.

When interpreted with respect to the finite state system FS_1 given in Figure 4.2 which has the three states $s1, s2$ and $s3$, we clearly have that all three states satisfy Φ, as well as $\langle a\rangle\Phi$. Interpreting Φ with respect to the second finite state system FS_2 yields similarly that Φ and $\langle a\rangle\Phi$ hold for all states $t1, t2, t3$ and $t4$.

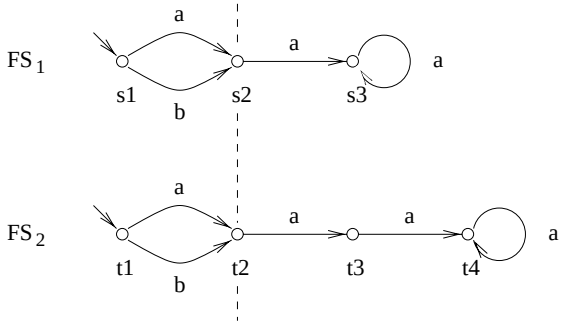

Fig. 4.2. Two simple finite state systems.

Observe, however, that the validity of Φ for $s1$, respectively $t1$, only depends on the validity of Φ for $s2$, respectively $t2$. Given a formula Φ, we may therefore abstract from the behaviour of the system to the right of the dashed line by fixing at the border line the set of formulas $\Psi \in CL(\Phi)$ valid for the process which starts there. Moreover, we may now vary the set of formulas which are assumed to be true for the state on the border line, resulting in a functional dependency between the set of formulas valid for the start state, and the formulas which have been fixed at the border line. The corresponding picture for our finite state systems and the formula Φ is shown in Figure 4.3.

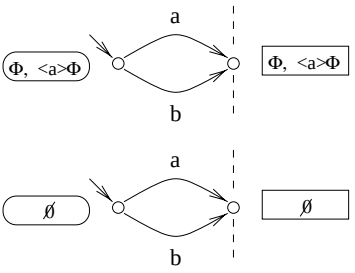

Fig. 4.3. Assertions and the resulting functional dependency.

The set of formulas which have been fixed is drawn in the rectangular box, while the formulas which are as a consequence true for the start state are shown in the rounded box. In particular, only two cases are of interest here: first the system on the right-hand side has an infinite a-path, represented by the fixed set $\{\, \Phi, \langle a \rangle \Phi \,\}$, which implies the existence of an infinite a-path also from the start state, and secondly, the system to the right of the borderline has no infinite a-path, expressed by the set $\emptyset$, which clearly yields that under this condition no infinite a-path can evolve from the start state.

4.3.2 Definition of Assertion-Based Semantics

In the remainder of this chapter we formalise the ideas demonstrated by the previous example by introducing the *assertion-based semantics* which will subsequently lead to the notion of property transformer describing conditionally the behaviour of processes.

Definition 4.3.1 (μ-Assertion).
Let $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$ be a labelled transition graph. A μ-assertion *Ω is a partial mapping $\mathcal{S} \longrightarrow 2^{L\mu}$, i.e. it assigns sets of formulas $\Delta \subseteq L\mu$ to states $s \in \mathcal{S}$. The* assertion set *induced by Ω with respect to a formula Φ is then the set of states defined by $\Omega^{-1}(\Phi) =_{\mathrm{df}} \{\, s \in \mathcal{S} \mid \Phi \in \Omega(s) \,\}$.*

As usual, we denote the *domain* of Ω by $dom(\Omega)$, and the *image* of Ω by $im(\Omega)$. An assertion is said to be *finite* if $dom(\Omega)$ and each $\Delta \in im(\Omega)$ are finite. Moreover, if $dom(\Omega)$ is finite the assignments can explicitly be enumerated as $[s_1 \models \Delta_1, \ldots, s_n \models \Delta_n]$ (or $[s_i \models \Delta_i]_{i=1}^n$ for short) where the s_i are states of $\mathcal{S}$, and the Δ_i are sets of formulas. Accordingly, the empty assertion will be denoted by $[\,]$. For ease of notation, we write $\Omega[s \models \Delta]$ for the assertion which behaves as Ω except that the binding of s is updated to Δ. Finally, we denote by $\Omega \cap \Delta$, for $\Delta \subseteq L\mu$, the μ-assertion

$$(\Omega \cap \Delta)(s) =_{\mathrm{df}} \Omega(s) \cap \Delta,$$

as well as by $\mathcal{V} \setminus dom(\Omega)$ the valuation defined as

$$(\mathcal{V} \setminus dom(\Omega))(X) =_{\mathrm{df}} \mathcal{V}(X) \setminus dom(\Omega).$$

Intuitively, an assertion $\Omega = [s_i \models \Delta_i]_{i=1}^n$ expresses that a state s_i satisfies *exactly* the set of formulas Δ_i. This goal is achieved by considering the usual semantics of a formula Φ for states not affected by Ω, while assuming at the same time that Φ holds for all states $\Omega^{-1}(\Phi)$. Given a set of states S, the effect of an assertion Ω on S is therefore formally defined as[4]

$$\mathfrak{A}_\Omega^\Phi(S) \ =_{\mathrm{df}} S \setminus dom(\Omega) \cup \Omega^{-1}(\Phi)$$

The *assertion-based semantics* (ABS) of a μ-formula Φ is now given in Table 4.2. It is defined with respect to a labelled transition graph $\mathcal{T}$, a valuation $\mathcal{V}$, and an assertion Ω.

$$\begin{array}{rcl}
[\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & =_{\mathrm{df}} & \mathsf{A}^{\mathtt{tt}}_\Omega(\mathcal{S}) \\
[\![X]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & =_{\mathrm{df}} & \mathsf{A}^{X}_\Omega(\mathcal{V}(X)) \\
[\![\neg\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & =_{\mathrm{df}} & \mathsf{A}^{\neg\Phi}_\Omega(\mathcal{S} \setminus [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \\
[\![\Phi' \vee \Phi'']\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & =_{\mathrm{df}} & \mathsf{A}^{\Phi' \vee \Phi''}_\Omega([\![\Phi']\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cup [\![\Phi'']\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \\
[\![\langle a\rangle\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & =_{\mathrm{df}} & \mathsf{A}^{\langle a\rangle\Phi}_\Omega(\{\, s \in \mathcal{S} \mid \exists\, s' \in \mathcal{S}.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \,\}) \\
[\![\mu X.\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & =_{\mathrm{df}} & \mathsf{A}^{\mu X.\Phi}_\Omega(\bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S],\Omega} \subseteq S \,\})
\end{array}$$

Table 4.2. The assertion-based semantics of μ-formulas.

4.3.3 Properties of Assertion-Based Semantics

In the remainder of this section we explore the properties of the assertion-based semantics. They will constitute the foundation for the model-checking algorithm we are going to develop in Section 4.4. We start by comparing the assertion-based semantics with the ordinary semantics for the modal μ-calculus.

As the next lemma shows the assertion-based semantics is a conservative extension of the ordinary semantics since it coincides with the ordinary semantics when the given assertion is empty. Moreover, the lemma expresses that assertions may overwrite the bindings of free variables for states contained in its domain.

Lemma 4.3.1. *Let $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$ be a labelled transition graph, Φ be a formula and $\mathcal{V}$ a valuation. Then we have*

1. $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},[\,]} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}}$, *and*
2. $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}\setminus dom(\Omega),\Omega}$

[4] Throughout we will use the convention that $\setminus$ has higher precedence than $\cup$.

Sketch. In both cases the lemma can be shown by straight-forward structural induction on Φ. For the first case we have to observe that $[\,]^{-1}(\Phi)$ is always the empty set, while the second claim follows directly from the definition of the assertion-based semantics for variables. □

An important observation is that only the part of the assertion Ω which concerns the subformulas of Φ is significant for the assertion-based semantics. This is similar to valuations, where only the bindings of variables occurring free in Φ must be taken into account.

Lemma 4.3.2 (Subformula Property of ABS).
Let $\mathcal{T}$ be a labelled transition graph, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. Then we have

$$[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega\cap SF(\Phi)} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}.$$

Sketch. The important point to observe is that for each $s \in dom(\Omega)$, and each $\Psi \in SF(\Phi)$ we have

$$\Psi \in \Omega(s) \quad \text{iff} \quad \Psi \in (\Omega \cap SF(\Phi))(s).$$

The lemma can then be shown by structural induction on Φ, since the semantics of Φ depends only on the semantics of subformulas Ψ which are elements of $SF(\Phi)$. □

The next result shows that the assertion-based semantics is monotone with respect to valuations, as well as assertions in the following sense.

Lemma 4.3.3 (Monotonicity of ABS).
Let $\mathcal{T}$ be a labelled transition graph, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation.

1. *If every free occurrence of X in Φ is within an even number of negations then the mapping*

$$\phi(S) =_{\mathrm{df}} [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S],\Omega}$$

is monotone on $2^{\mathcal{S}}$.

2. *If Δ^- is a subset of $SF(\Phi)^-$ then the mapping ψ defined by*

$$\psi(\Delta^+) =_{\mathrm{df}} [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s\models\Delta^+\dot\cup\Delta^-]}$$

is monotone between $SF(\Phi)^+$ and $2^{\mathcal{S}}$.

Proof. The first part can be proved following the lines given in [Cle90] for ordinary semantics. A function over $2^{\mathcal{S}}$ is said to be *anti-monotone* if $S_1 \subseteq S_2$ implies $f(S_1) \supseteq f(S_2)$. The proof is then accomplished by showing the slightly stronger result that the function

$$\phi(S) =_{\mathrm{df}} [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S],\Omega}$$

is monotone if X appears under an even number of negations, while anti-monotone if X appears under an odd number of negations, respectively. This follows by straight-forward induction on the structure of Φ from observing that $\vee$ and $\langle a \rangle$, for each action a, are monotone operators, whereas $\neg$ is anti-monotone.

To prove the second part we need again a slightly stronger result. Let

$$\psi : SF(\Phi)^+ \times SF(\Phi)^-, \quad (\Delta^+, \Delta^-) \mapsto [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]}$$

Then ψ is monotone in the first argument, while anti-monotone in the second. This is shown by structural induction on Φ. Since most cases are routine we prove only the case for $\Phi = \neg\Psi$. Here we have

$$\begin{aligned} \psi(\Delta^+, \Delta^-) &= [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]} \\ &= \mathfrak{A}^{\neg\Psi}_{\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]}(\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]}) \end{aligned}$$

Now by induction hypothesis

$$[\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]}$$

is monotone in Δ^-, and anti-monotone in Δ^+. Since $\neg\Psi \in SF(\Phi)^+$ we also have that $\mathfrak{A}^{\neg\Psi}_{\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]}$ is monotone in Δ^+, and independent of Δ^-. Overall, this yields the monotonicity of

$$[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models \Delta^+ \dot{\cup} \Delta^-]}$$

with respect to Δ^+, as well as the anti-monotonicity with respect to Δ^- which proves the lemma for $\Phi = \neg\Psi$. □

As in the case of ordinary semantics we can establish a link between the syntactic notion of *substitution* and the semantic notion of *valuation binding* (cf. [Cle90]). Subsequently, this link will allow fixpoint formulas to be syntactically unrolled without change in the semantics. However, we must impose some *consistency* condition on the given assertion which will be important when considering equational formulas in Section 4.4.1.

Lemma 4.3.4 (Substitution Lemma for ABS).
Let $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$ be a labelled transition graph, Φ and Θ be formulas, X be a variable, Ω be an assertion, and $\mathcal{V}$ be a valuation. If Ω satisfies the consistency condition

$$\Phi[\Theta/X] \in \Delta \quad \textit{iff} \quad \Phi \in \Delta,$$

for all $\Delta \in im(\Omega)$, then we have

$$[\![\Phi[\,\Theta/X\,]]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[\,X \mapsto [\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega}$$

Proof. The proof is done by structural induction on Φ. Here we consider only the most interesting cases $\Phi = X$ and $\Phi = \mu Y.\Psi$. The others are routine.

– If $\Phi = X$ then

$$[\![X[\,\Theta/X\,]]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = [\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = \mathfrak{A}^X_\Omega([\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) = [\![X]\!]^{\mathcal{T}}_{\mathcal{V}[\,X\mapsto[\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega}$$

since we know that for all $s \in dom(\Omega)$ we have

$$s \in [\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \quad \text{iff} \quad \Theta \in \Omega(s) \quad \text{iff} \quad X \in \Omega(s) \quad \text{iff} \quad s \in \Omega^{-1}(X)$$

– If $\Phi = \mu Y.\Psi$ then

$$\begin{array}{ll}
& [\![\mu Y.\Psi[\,\Theta/X\,]]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \\
= & \mathfrak{A}^{\mu Y.\Psi[\,\Theta/X\,]}_\Omega(\bigcap\{\,S \subseteq \mathcal{S} \mid [\![\Psi[\,\Theta/X\,]]\!]^{\mathcal{T}}_{\mathcal{V}[\,Y\mapsto S\,],\Omega} \subseteq S\,\}) \\
= & \text{[by induction hypothesis]} \\
& \mathfrak{A}^{\mu Y.\Psi[\,\Theta/X\,]}_\Omega(\bigcap\{\,S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,Y\mapsto S\,][\,X\mapsto[\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V}[\,Y\mapsto S\,],\Omega}\,],\Omega} \subseteq S\,\}) \\
= & \text{[by def. of subst. we may assume that } Y \text{ is not free in } \Theta] \\
& \mathfrak{A}^{\mu Y.\Psi[\,\Theta/X\,]}_\Omega(\bigcap\{\,S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,Y\mapsto S\,][\,X\mapsto[\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega} \subseteq S\,\}) \\
= & \text{[since } \mu Y.\Psi[\,\Theta/X\,] \in \Omega(s) \text{ iff } \mu Y.\Psi \in \Omega(s),\ \forall\ s \in dom(\Omega)] \\
& \mathfrak{A}^{\mu Y.\Psi}_\Omega(\bigcap\{\,S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,Y\mapsto S\,][\,X\mapsto[\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega} \subseteq S\,\}) \\
= & [\![\mu Y.\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,X\mapsto[\![\Theta]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega}
\end{array}$$

□

Corollary 4.3.1 (Fixpoint Unrolling).
Let $\mathcal{T}$ be a labelled transition graph, $\mu X.\Psi$ be a fixpoint formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. If Ω satisfies the consistency condition

$$\mu X.\Psi \in \Delta \quad \textit{iff} \quad \Psi \in \Delta \quad \textit{iff} \quad \Psi[\,\mu X.\Psi/X\,] \in \Delta,$$

for all $\Delta \in im(\Omega)$, then we have

$$[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = [\![\Psi[\,\mu X.\Psi/X\,]]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}.$$

Proof. Abbreviating $\bigcap\{\,S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,X\mapsto S\,],\Omega} \subseteq S\,\}$ by S_μ, we observe that due to the consistency condition $\mu X.\Psi \in \Delta$ iff $\Psi \in \Delta$, for all $\Delta \in im(\Omega)$, we have

$$[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = S_\mu. \tag{4.2}$$

Since Ω also satisfies $\Psi[\,\mu X.\Psi/X\,]$ iff $\Psi \in \Delta$, for all $\Delta \in im(\Omega)$, we may apply Lemma 4.3.4 to obtain

$$\begin{array}{lll}
& [\![\Psi[\,\mu X.\Psi/X\,]]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & \\
= & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,X\mapsto[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega} & \text{[by Lemma 4.3.4]} \\
= & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\,X\mapsto S_\mu\,],\Omega} & \text{[by equation 4.2]} \\
= & S_\mu & \text{[by fixpoint property of } S_\mu\text{]} \\
= & [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} & \text{[by equation 4.2]}
\end{array}$$

□

Correct assertions which assume exactly the set of formulas valid for a state will play an important role in our theory. In order to make this notion precise we introduce the satisfiability set of a state as follows.

Definition 4.3.2 (Satisfiability Set).
The satisfiability set of s *wrt.* $\Phi, \mathcal{T}, \mathcal{V}$ *and* Ω*, denoted by* $Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$*, is inductively defined as the least set of formulas satisfying the rules given in Table 4.3.*

1. If $\Phi = \mathtt{tt}$	then	$\mathtt{tt} \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ iff $s \in [\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$.
2. If $\Phi = X$	then	$X \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ iff $s \in [\![X]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$.
3. If $\Phi = \neg\Psi$	then	$\neg\Psi \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ iff $s \in [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$,
	and	$Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega} \subseteq Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$.
4. If $\Phi = \Psi' \vee \Psi''$	then	$\Psi' \vee \Psi'' \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ iff $s \in [\![\Psi' \vee \Psi'']\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$,
	and	$Sat(s,\Psi')^{\mathcal{T}}_{\mathcal{V},\Omega} \cup Sat(s,\Psi'')^{\mathcal{T}}_{\mathcal{V},\Omega} \subseteq Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$.
5. If $\Phi = \langle a\rangle\Psi$	then	$\langle a\rangle\Psi \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ iff $s \in [\![\langle a\rangle\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$,
	and	$Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega} \subseteq Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$.
6. If $\Phi = \mu X.\Psi$	then	$\mu X.\Psi \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ iff $s \in [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$,
	and	$Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[\,X \mapsto [\![\mu X.\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\,],\Omega} \subseteq Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$.

Table 4.3. Definition of satisfiability sets.

Intuitively, $Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ is the set of subformulas of Φ which hold for the state s in the transition graph $\mathcal{T}$ with respect to the valuation $\mathcal{V}$ and the assertion Ω.

A direct consequence of the subformula property of the assertion-based semantics as stated in Lemma 4.3.2 is that assertions occurring in the definition of satisfiability sets may also be restricted to the appropriate subformulas without changing the resulting sets of formulas.

Lemma 4.3.5 (Subformula Property of *Sat* Sets).
Let $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$ *be a labelled transition graph,* Φ *be a formula,* Ω *be an assertion, and* $\mathcal{V}$ *be a valuation. Then we have, for any state* $s \in \mathcal{S}$*,*

$$Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega \cap SF(\Phi)} = Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}.$$

Sketch. The lemma is shown by straight-forward induction on the structure of Φ. The proof relies on the subformula property of the assertion-based semantics given in Lemma 4.3.2. □

Moreover, the notion of monotonicity can also be applied to satisfiability sets. Since this notion of monotonicity is, however, somewhat complicated we first give the appropriate definitions.

Definition 4.3.3. *Let Δ^+ denote the restriction of $\Delta \subseteq SF(\Phi)$ to $SF(\Phi)^+$, and analogously $\Delta^- =_{\mathrm{df}} \Delta \cap SF(\Phi)^-$. Then we define the partial order $\sqsubseteq_{sat}$ on $2^{SF(\Phi)}$ by*

$$\Delta_1 \sqsubseteq_{sat} \Delta_2 \quad \textit{iff} \quad \Delta_1^+ \subseteq \Delta_2^+ \textit{ and } \Delta_1^- \supseteq \Delta_2^-.$$

Lemma 4.3.6 (Monotonicity of Satisfiability Sets).
Let s be a state of the labelled transition graph $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. If every free occurrence of X in Φ is within an even number of negations then the mapping which assigns the satisfiability set of s with respect to the interpretation $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto \mathcal{S}],\Omega}$ to $S \subseteq \mathcal{S}$ is monotone, i.e.

$$S_1 \subseteq S_2 \quad \textit{implies} \quad Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega} \sqsubseteq_{sat} Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega}$$

Proof. The lemma is shown by means of the slightly stronger result which states that the mapping

$$S \mapsto Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S],\Omega}$$

is monotone if every free occurrence of X in Φ is within an even number of negations, and is anti-monotone if every free occurrence of X appears negatively in Φ. Since most cases are routine we only consider the case $\Phi = \neg\Psi$ and X occurs positively in Φ.
Let $S_1 \subseteq S_2$. Since every free occurrence of X in Φ is within an even number of negations we know by monotonicity of the assertion-based semantics (Lemma 4.3.3(1)) that

$$s \in [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega} \quad \text{implies} \quad s \in [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega} \tag{4.3}$$

Moreover, since every free occurrence of X in Ψ is within an odd number of negations we obtain by induction hypothesis

$$Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega} \sqsubseteq_{\mathrm{sat}} Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega},$$

and hence

$$(Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega})^+ \subseteq (Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega})^+ \tag{4.4}$$

$$(Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega})^- \subseteq (Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega})^- \tag{4.5}$$

From this we, finally, conclude

$$\begin{array}{ll} & (Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega})^+ \\ = & \text{[by (4.3) and (4.5)]} \\ & \{\, \neg\Psi \mid s \in [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega} \,\} \cup (Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_1],\Omega})^- \\ \subseteq & \{\, \neg\Psi \mid s \in [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega} \,\} \cup (Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega})^- \\ = & (Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V}[X \mapsto S_2],\Omega})^+ \end{array}$$

Analogously, $(Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto\mathcal{S}_2],\Omega})^- \subseteq (Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto\mathcal{S}_1],\Omega})^-$ is deduced by means of (4.4). Overall, this shows, as desired, that

$$Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto\mathcal{S}_1],\Omega} \sqsubseteq_{\text{sat}} Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto\mathcal{S}_2],\Omega}$$

□

The assertion-based semantics can now be shown to be "consistent" in the sense that we always may extend assertions with the satisfiability sets of arbitrary states at arbitrary places.

Lemma 4.3.7 (Extension Lemma).
Let $\mathcal{T} = (\mathcal{S}, Act, \to)$ be a labelled transition graph, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. Then we have, for any set of states $\{ s, s_1, \ldots, s_n \} \subseteq \mathcal{S}$, the following equalities.

1. $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models Sat(s_i,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]_{i=1}^n} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$
2. $Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} = Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models Sat(s_i,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]_{i=1}^n}$

Proof. Both parts of the lemma will be shown by induction on n. To start with, we prove the following claim.
Claim 1: For any single state $s \in \mathcal{S}$, we have

$$[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} = [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$$

Proof. To shorten notation we abbreviate the assertion $\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]$ by Ω_s. If $s \in dom(\Omega)$ we have by definition $Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} = \Omega(s) \cap SF(\Phi)$, and hence $\Omega_s \cap SF(\Phi) = \Omega \cap SF(\Phi)$. So in this case the claim trivially holds by Lemma 4.3.2.

Let us now assume that $s \notin dom(\Omega)$. First note that

$$s \in {\Omega_s}^{-1}(\Phi) \quad \text{iff} \quad \Phi \in Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} \quad \text{iff} \quad s \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \tag{4.6}$$

always holds. In the proof we will then use the fact that $s \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$ may be reduced to membership of s in the semantics of the "unfolding" of Φ whenever $s \notin dom(\Omega)$.

The proof of the claim for the case $s \notin dom(\Omega)$ is now accomplished by structural induction on Φ.

1. $\Phi = \mathtt{tt}$. Then obviously

$$\begin{aligned}[\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} &= \mathfrak{A}^{\mathtt{tt}}_{\Omega_s}(\mathcal{S}) = \mathcal{S} \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(\mathtt{tt}) \\ &= \mathcal{S} \setminus dom(\Omega) \cup \Omega^{-1}(\mathtt{tt}) = \mathfrak{A}^{\mathtt{tt}}_{\Omega}(\mathcal{S}) = [\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}\end{aligned}$$

since by (4.6) $s \in {\Omega_s}^{-1}(\mathtt{tt})$ iff $s \in [\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}$ where the latter always holds.

2. $\Phi = X$. In this case we obtain

$$\begin{aligned}[\![X]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} &= \mathfrak{A}^{X}_{\Omega_s}(\mathcal{V}(X)) = \mathcal{V}(X) \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(X) \\ &= \mathcal{V}(X) \setminus dom(\Omega) \cup \Omega^{-1}(X) = \mathfrak{A}^{X}_{\Omega}(\mathcal{V}(X)) = [\![X]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}.\end{aligned}$$

since

$$s \in \mathcal{V}(X) \quad \text{iff} \quad X \in Sat(s,X)^{\mathcal{T}}_{\mathcal{V},\Omega} \quad \text{iff} \quad s \in {\Omega_s}^{-1}(X)$$

i.e. if s is removed from $\mathcal{V}(X)$ by set subtraction wrt. $dom(\Omega_s)$ it will be added again through ${\Omega_s}^{-1}(X)$.

3. $\Phi = \neg\Psi$. Then we deduce

$$\begin{aligned}
& [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} \\
= \;& \mathfrak{A}^{\neg\Psi}_{\Omega_s}(\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s}) \\
= \;& (\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s}) \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(\neg\Psi) \\
= \;& \text{[by Lemma 4.3.2]} \\
& (\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s\models Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}] \cap SF(\Psi)}) \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(\neg\Psi) \\
= \;& \text{[by definition of } Sat\text{]} \\
& (\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}) \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(\neg\Psi) \\
= \;& \text{[by induction hypothesis]} \\
& (\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(\neg\Psi) \\
= \;& \text{[since } s \in \mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \text{ iff } s \in {\Omega_s}^{-1}(\neg\Psi)\text{]} \\
& (\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \setminus dom(\Omega) \cup \Omega^{-1}(\neg\Psi) \\
= \;& \mathfrak{A}^{\neg\Psi}_{\Omega}(\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \\
= \;& [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}
\end{aligned}$$

4. $\Phi = \Psi_1 \vee \Psi_2$, and 5. $\Phi = \langle a \rangle \Psi$ follow similar lines as case 3.

6. $\Phi = \mu X.\Psi$. Let

$$\begin{aligned}
S_\Omega &=_{\text{df}} \bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S],\Omega} \subseteq S \,\}, \text{ and} \\
S_{\Omega_s} &=_{\text{df}} \bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S],\Omega_s} \subseteq S \,\}.
\end{aligned}$$

As we have

$$\begin{aligned}
[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} &= S_\Omega \setminus dom(\Omega) \cup \Omega^{-1}(\mu X.\Psi), \text{ and} \\
[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} &= S_{\Omega_s} \setminus dom(\Omega_s) \cup {\Omega_s}^{-1}(\mu X.\Psi)
\end{aligned}$$

as well as the property $s \in S_\Omega$ iff $s \in {\Omega_s}^{-1}(\mu X.\Psi)$ it is sufficient to prove that $S_\Omega = S_{\Omega_s}$. To start with, we show $S_{\Omega_s} \subseteq S_\Omega$.

$$
\begin{array}{ll}
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega_s} \\
= & \text{[by definition of } \Omega_s\text{]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega[s\models Sat(s,\mu X.\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= & \text{[by Lemma 4.3.2]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega[s\models Sat(s,\mu X.\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]\cap SF(\Psi)} \\
= & \text{[by definition of } Sat\text{]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}],\Omega}]} \\
= & \text{[by Lemma 4.3.1.2]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega}]} \\
= & \text{[by induction hypothesis]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega} \\
\subseteq & \text{[by definition of } S_\Omega\text{]} \\
 & S_\Omega
\end{array}
$$

From the least fixpoint property of S_{Ω_s} we may now conclude $S_{\Omega_s} \subseteq S_\Omega$. For the other direction observe that the following holds.

$$
\begin{array}{ll}
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega} \\
= & \text{[by induction hypothesis]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega}]} \\
\subseteq & \text{[due to } S_{\Omega_s} \subseteq S_\Omega \text{ and monotonicity]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_\Omega],\Omega}]} \\
= & \text{[by Lemma 4.3.1.2]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[X\mapsto [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}],\Omega}]} \\
= & \text{[by definition of } Sat\text{]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega[s\models Sat(s,\mu X.\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]\cap SF(\Psi)} \\
= & \text{[by Lemma 4.3.2]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega[s\models Sat(s,\mu X.\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= & \text{[by definition of } \Omega_s\text{]} \\
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X\mapsto S_{\Omega_s}],\Omega_s} \\
\subseteq & \text{[by definition of } S_{\Omega_s}\text{]} \\
 & S_{\Omega_s}
\end{array}
$$

The least fixpoint property of S_Ω now yields the remaining inclusion $S_\Omega \subseteq S_{\Omega_s}$.

This completes the proof of Claim 1. □

Next we prove that satisfiability sets as well are not affected when extending the assertion Ω with correct assertions.

Claim 2: For any states $s, s' \in \mathcal{S}$ we have

$$Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} = Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}$$

Proof. The claim is shown by structural induction on Φ. The base cases hold due to the equivalence

$$\begin{aligned}
&\Phi \in Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} \\
&\text{iff } s' \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \\
&\text{iff } s' \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} && \text{[by Claim 1]} \\
&\text{iff } \Phi \in Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}
\end{aligned} \tag{4.7}$$

Now assume Claim 2 holds for all $\Psi \prec^+ \Phi$. The case $\Phi = \neg\Psi$ is then shown by

$$\neg\Psi \in Sat(s', \neg\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega} \quad \text{iff} \quad \neg\Psi \in Sat(s', \neg\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\neg\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}$$

which follows from (4.7), and

$$\begin{aligned}
& Sat(s', \Psi)^{\mathcal{T}}_{\mathcal{V},\Omega} \\
= \ & \text{[by induction hypothesis]} \\
& Sat(s', \Psi)^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= \ & \text{[by definition of } Sat\text{]} \\
& Sat(s', \Psi)^{\mathcal{T}}_{\mathcal{V},\Omega([s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}] \cap SF(\Psi))} \\
= \ & \text{[by Lemma 4.3.5]} \\
& Sat(s', \Psi)^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}
\end{aligned}$$

The remaining cases follow similar lines. This proves Claim 2. □

To ease notation we will abbreviate in the remainder of the proof $Sat(s_i, \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$ by Δ_i.

The second part of the lemma is now shown by induction on n. For $n = 0$ the lemma obviously holds. Now assume the lemma holds for n states $s_1, \ldots, s_n \in \mathcal{S}$, and let $s_{n+1} \in \mathcal{S}$. Then we deduce

$$
\begin{array}{ll}
 & Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} \\
= & \text{[by induction hypothesis]} \\
 & Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}} \\
= & \text{[by Claim 2]} \\
 & Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}[s_{n+1} \models Sat(s_{n+1},\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}}]} \\
= & \text{[again by induction hypothesis]} \\
 & Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}[s_{n+1} \models Sat(s_{n+1},\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= & Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{1}^{n+1}}
\end{array}
$$

Finally, we prove the first part of the lemma also by induction on n. For $n = 0$ the lemma trivially holds. Now assume the lemma holds for n states $s_1, \ldots, s_n \in \mathcal{S}$, and let $s_{n+1} \in \mathcal{S}$. Then we conclude

$$
\begin{array}{ll}
 & [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \\
= & \text{[by induction hypothesis]} \\
 & [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}} \\
= & \text{[by Claim 1]} \\
 & [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}[s_{n+1} \models Sat(s_{n+1},\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}}]} \\
= & \text{[by the second part of the lemma]} \\
 & [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n}[s_{n+1} \models Sat(s_{n+1},\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= & [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{1}^{n+1}}
\end{array}
$$

This completes the proof of the lemma. □

Next we investigate the relationship between *reachability* in the labelled transition graph and the assertion-based semantics.

Definition 4.3.4. *We call* $\mathcal{T}' = (\mathcal{S}', Act, \rightarrow')$ *a* closed component *of the labelled transition graph* $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$ *if*

1. $\mathcal{S}' \subseteq \mathcal{S}$,
2. $\rightarrow' = \rightarrow_{\mid \mathcal{S}'}$, *and*
3. $\mathcal{S}'$ *is closed with respect to out-going transitions, i.e.* $s \in \mathcal{S}'$ *and* $s \xrightarrow{a} s'$ *implies* $s' \in \mathcal{S}'$.

Since the modal μ-calculus is defined in terms of *forward* modalities, it turns out that the semantics of formulas on *closed components* of labelled transition graphs only depends on the behaviour of the states of the closed component at hand.

Lemma 4.3.8 (Closed Component Lemma).
Let $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$ be a labelled transition graph, $\mathcal{T}' = (\mathcal{S}', Act, \rightarrow')$ be a closed component of $\mathcal{T}$, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. If we abbreviate $\Omega_{|\mathcal{S}'}$ by Ω', and $\mathcal{V} \cap \mathcal{S}'$ by $\mathcal{V}'$, respectively, then we have

1. $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' = [\![\Phi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}$
2. $Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} = Sat(s,\Phi)^{\mathcal{T}'}_{\mathcal{V}',\Omega'}$, *for any* $s \in \mathcal{S}'$.

Proof. The first part of the lemma will be shown by structural induction on Φ.

1. $\Phi = \mathtt{tt}$. Then

$$\begin{aligned}[\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' &= \mathfrak{A}^{\mathtt{tt}}_{\Omega}(\mathcal{S}) \cap \mathcal{S}' = \mathfrak{A}^{\mathtt{tt}}_{\Omega'}(\mathcal{S} \cap \mathcal{S}') \\ &= \mathfrak{A}^{\mathtt{tt}}_{\Omega'}(\mathcal{S}') = [\![\mathtt{tt}]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}\end{aligned}$$

2. $\Phi = X$. Then

$$\begin{aligned}[\![X]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' &= \mathfrak{A}^{X}_{\Omega}(\mathcal{V}(X)) \cap \mathcal{S}' = \mathfrak{A}^{X}_{\Omega'}(\mathcal{V}(X) \cap \mathcal{S}') \\ &= \mathfrak{A}^{X}_{\Omega'}(\mathcal{V}'(X)) = [\![X]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}\end{aligned}$$

3. $\Phi = \neg\Psi$. Then

$$\begin{aligned} & [\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \\ = \;& \mathfrak{A}^{\neg\Psi}_{\Omega}(\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \cap \mathcal{S}' \\ = \;& \mathfrak{A}^{\neg\Psi}_{\Omega'}(\mathcal{S}' \setminus ([\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}')) \\ = \;& \text{[by induction hypothesis]} \\ & \mathfrak{A}^{\neg\Psi}_{\Omega'}(\mathcal{S}' \setminus [\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}) \\ = \;& [\![\neg\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}\end{aligned}$$

4. $\Phi = \Psi' \vee \Psi''$. Then

$$\begin{aligned} & [\![\Psi' \vee \Psi'']\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \\ = \;& \mathfrak{A}^{\Psi' \vee \Psi''}_{\Omega}([\![\Psi']\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cup [\![\Psi'']\!]^{\mathcal{T}}_{\mathcal{V},\Omega}) \cap \mathcal{S}' \\ = \;& \mathfrak{A}^{\Psi' \vee \Psi''}_{\Omega'}(([\![\Psi']\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}') \cup ([\![\Psi'']\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}')) \\ = \;& \text{[by induction hypothesis]} \\ & \mathfrak{A}^{\Psi' \vee \Psi''}_{\Omega'}([\![\Psi']\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'} \cup [\![\Psi'']\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}) \\ = \;& [\![\Psi' \vee \Psi'']\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}.\end{aligned}$$

5. $\Phi = \langle a \rangle \Psi$. Then

$$
\begin{array}{ll}
& [\![\langle a\rangle\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \\
= & \mathfrak{A}^{\langle a\rangle\Psi}_{\Omega}(\{\, s \in \mathcal{S} \mid \exists\, s' \in \mathcal{S}.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \,\}) \cap \mathcal{S}' \\
= & \mathfrak{A}^{\langle a\rangle\Psi}_{\Omega'}(\{\, s \in \mathcal{S}' \mid \exists\, s' \in \mathcal{S}.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \,\}) \\
= & [\mathcal{S}' \text{ is closed and } \forall\, s' \in \mathcal{S}' \text{ we know by induction hypothesis} \\
& \qquad s' \in [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \text{ iff } s' \in [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \text{ iff } s' \in [\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}] \\
& \mathfrak{A}^{\langle a\rangle\Psi}_{\Omega'}(\{\, s \in \mathcal{S}' \mid \exists\, s' \in \mathcal{S}'.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'} \,\}) \\
= & [\![\langle a\rangle\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}
\end{array}
$$

6. $\Phi = \mu X.\Psi$.
The proof of $[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' = [\![\mu X.\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}$ will be accomplished by showing both inclusions.

6.1. $[\![\mu X.\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'} \subseteq [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}'$:

Let $\hat{S} \subseteq \mathcal{S}$ satisfy $(*)$ $[\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto \hat{S}],\Omega} \subseteq \hat{S}$, and let $S' =_{\mathrm{df}} \hat{S} \cap \mathcal{S}'$. Then we have by induction hypothesis the inclusion

$$[\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}'[X \mapsto S'],\Omega'} = [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto \hat{S}],\Omega} \cap \mathcal{S}' \subseteq \hat{S} \cap \mathcal{S}' = S'$$

which shows that any set of states $\hat{S}$ satisfying $(*)$ fulfils due to the least fixpoint property also

$$[\![\mu X.\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'} \subseteq \mathfrak{A}^{\mu X.\Psi}_{\Omega'}(S') = \mathfrak{A}^{\mu X.\Psi}_{\Omega}(\hat{S}) \cap \mathcal{S}'$$

Taking in particular $\hat{S} =_{\mathrm{df}} \bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S],\Omega} \subseteq S \,\}$ we, finally, obtain

$$
\begin{array}{ll}
& [\![\mu X.\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'} \\
\subseteq & \mathfrak{A}^{\mu X.\Psi}_{\Omega}(\bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S],\Omega} \subseteq S \,\}) \cap \mathcal{S}' \\
= & [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}'
\end{array}
$$

6.2. $[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \subseteq [\![\mu X.\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}$:

Let $S' \subseteq \mathcal{S}'$ satisfy $(**)$ $[\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}'[X \mapsto S'],\Omega'} \subseteq S'$. In order to extend S' to some appropriate subset $\hat{S} \subseteq \mathcal{S}$ such that $\hat{S}$ satisfies

$$[\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto \hat{S}],\Omega} \subseteq \hat{S}$$

we define the function $\varphi_{S'}$ which maps subsets of $\overline{\mathcal{S}'}$, the complement of $\mathcal{S}'$ with respect to $\mathcal{S}$, to subsets of $\overline{\mathcal{S}'}$ by means of

$$\varphi_{S'}(S) =_{\mathrm{df}} [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto S' \cup S],\Omega} \cap \overline{\mathcal{S}'}$$

Since $\varphi_{S'}$ is monotone on $2^{\overline{\mathcal{S}'}}$, $\varphi_{S'}$ has according to the Tarski-Knaster theorem a least fixpoint, denoted by $\mu\varphi_{S'}$. Now let $\hat{S} =_{\mathrm{df}} S' \cup \mu\varphi_{S'}$. Then we conclude

$$
\begin{array}{ll}
 & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X_1 \to \hat{S}],\Omega} \\
= & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X_1 \to \hat{S}],\Omega} \cap (\mathcal{S}' \dot\cup \overline{\mathcal{S}'}) \\
= & ([\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X_1 \to S' \dot\cup \mu\varphi_{S'}],\Omega} \cap \mathcal{S}') \dot\cup ([\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X_1 \to S' \dot\cup \mu\varphi_{S'}],\Omega} \cap \overline{\mathcal{S}'}) \\
= & \text{[by fixpoint property of } \mu\varphi_{S'}\text{]} \\
 & ([\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[X_1 \to S' \dot\cup \mu\varphi_{S'}],\Omega} \cap \mathcal{S}') \dot\cup \mu\varphi_{S'} \\
= & \text{[by induction hypothesis]} \\
 & [\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}'[X_1 \to S'],\Omega'} \dot\cup \mu\varphi_{S'} \\
\subseteq & \text{[by definition of } S'\text{]} \\
 & S' \dot\cup \mu\varphi_{S'} \\
= & \hat{S}
\end{array}
$$

From this we see that whenever $(**)$ holds for a set of states S' we also have by the least fixpoint property the inclusion

$$[\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \subseteq \mathfrak{A}^{\mu X.\Psi}_{\Omega}(\hat{S}) \cap \mathcal{S}' = \mathfrak{A}^{\mu X.\Psi}_{\Omega'}(\hat{S} \cap \mathcal{S}') = \mathfrak{A}^{\mu X.\Psi}_{\Omega'}(S')$$

In particular for $S' =_{\mathrm{df}} \bigcap \{ S \subseteq \mathcal{S}' \mid [\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}'[X_1 \to S],\Omega'} \subseteq S \}$ we hence obtain

$$
\begin{array}{ll}
 & [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \cap \mathcal{S}' \\
\subseteq & \mathfrak{A}^{\mu X.\Psi}_{\Omega'}(\bigcap \{ S \subseteq \mathcal{S}' \mid [\![\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}'[X_1 \to S],\Omega'} \subseteq S \}) \\
= & [\![\mu X.\Psi]\!]^{\mathcal{T}'}_{\mathcal{V}',\Omega'}
\end{array}
$$

Now the second claim of the lemma follows from the first one by induction on the structure of Φ. □

The satisfaction set of a state can be considered as an assertion that characterises its behaviour. In fact, we can show that pruning the transitions of a state which is assumed to satisfy its satisfaction set does not change the considered assertion-based semantics.

Definition 4.3.5 (Pruning of Transitions).
Let $\mathcal{T} = (\mathcal{S}, Act, \to_{\mathcal{T}})$ be a labelled transition graph, and $S = \{ s_1, \ldots, s_n \} \subseteq \mathcal{S}$ be a set of states. If we prune *the transitions of S in $\mathcal{T}$ we obtain the labelled transition graph $\mathcal{T}_S$ which coincides with $\mathcal{T}$ except that all out-going transitions of the states s_i are deleted, i.e. $\mathcal{T}_S =_{\mathrm{df}} (\mathcal{S}, Act, \to_{\mathcal{T}_S})$ where*

$$\to_{\mathcal{T}_S} =_{\mathrm{df}} \to_{\mathcal{T}} \setminus \{ s_i \xrightarrow{a}_{\mathcal{T}} s' \mid 1 \leq i \leq n, a \in Act, s' \in \mathcal{S} \}.$$

Lemma 4.3.9 (Expressiveness of Assertions).
Let $\mathcal{T} = (\mathcal{S}, Act, \to_{\mathcal{T}})$ be a labelled transition graph, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. Given a set of states $S = \{ s_1, \ldots, s_n \} \subseteq \mathcal{S}$, let Δ_i be the satisfiability set of s_i, i.e. $\Delta_i =_{\mathrm{df}} Sat(s_i, \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}$. Then we have

1. $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} = [\![\Phi]\!]^{\mathcal{T}_S}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^n}$, *and*
2. $Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} = Sat(s', \Phi)^{\mathcal{T}_S}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^n}$, *for any* $s' \in \mathcal{S}$.

Proof. Both parts of the lemma will be shown by induction on n based on the following two claims.

Claim 1: $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} = [\![\Phi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}$, and

Claim 2: $Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} = Sat(s', \Phi)^{\mathcal{T}_s}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]}$

The proof of the first claim will be accomplished by structural induction on Φ. To shorten notation we will abbreviate the assertion $\Omega[s \models Sat(s, \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]$ by Ω_s. The base cases are $\Phi = \mathtt{tt}$, where we have

$$[\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} = \mathfrak{A}^{\mathtt{tt}}_{\Omega_s}(\mathcal{S}) = [\![\mathtt{tt}]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s},$$

and $\Phi = X$, where we know

$$[\![X]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} = \mathfrak{A}^{X}_{\Omega_s}(\mathcal{V}(X)) = [\![X]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s}.$$

Now assume the lemma holds for all $\Psi \prec^+ \Phi$. Applying the induction hypothesis we observe that

$$[\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} = [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s} \tag{4.8}$$

since we have

$$\begin{aligned}
& [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} \\
= \; & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}] \cap SF(\Psi)} && \text{[by Lemma 4.3.2]} \\
= \; & [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega[s \models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} && \text{[by definition of } Sat\text{]} \\
= \; & [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega[s \models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} && \text{[by induction hypothesis]} \\
= \; & [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s} \text{[again by Lemma 4.3.2]}
\end{aligned}$$

For $\Phi = \neg\Psi$ we then conclude by means of equation (4.8)

$$[\![\neg\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} = \mathfrak{A}^{\neg\Psi}_{\Omega_s}(\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s}) = \mathfrak{A}^{\neg\Psi}_{\Omega_s}(\mathcal{S} \setminus [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s}) = [\![\neg\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s}$$

The case $\Phi = \Psi' \vee \Psi''$ follows similar lines since (4.8) also holds for each immediate subformula of Φ. Most interesting is the case $\Phi = \langle a \rangle \Psi$. First remark that obviously

$$S \setminus dom(\Omega_s) = S' \setminus dom(\Omega_s) \quad \text{implies} \quad \mathfrak{A}^{\langle a \rangle \Psi}_{\Omega_s}(S) = \mathfrak{A}^{\langle a \rangle \Psi}_{\Omega_s}(S') \tag{4.9}$$

for any $S, S' \subseteq \mathcal{S}$. Using (4.9) we then conclude that

$$
\begin{aligned}
& [\![\langle a\rangle \Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} \\
= \;& \mathfrak{A}^{\langle a\rangle\Psi}_{\Omega_s}(\{\, t \mid \exists\, t'.\ t \xrightarrow{a}_{\mathcal{T}} t' \text{ and } t' \in [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} \,\}) \\
= \;& \text{[by equations (4.8) and (4.9)]} \\
& \mathfrak{A}^{\langle a\rangle\Psi}_{\Omega_s}(\{\, t \mid \exists\, t'.\ t \xrightarrow{a}_{\mathcal{T}_s} t' \text{ and } t' \in [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s} \,\}) \\
= \;& [\![\langle a\rangle \Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s}
\end{aligned}
$$

In order to prove the last case $\Phi = \mu X.\Psi$ we apply the induction hypothesis to $\mathcal{V}[\, X \mapsto S\,]$ instead of $\mathcal{V}$ yielding like in (4.8)

$$
\begin{aligned}
& [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\, X\mapsto S\,],\Omega_s} \\
= \;& [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V}[\, X\mapsto S\,],\Omega_s[s\models Sat(s,\Psi)^{\mathcal{T}}_{\mathcal{V}[\, X\mapsto S\,],\Omega_s}]} = [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V}[\, X\mapsto S\,],\Omega_s}
\end{aligned}
\tag{4.10}
$$

from which we conclude

$$
\begin{aligned}
& [\![\mu X.\Psi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega_s} \\
= \;& \mathfrak{A}^{\mu X.\Psi}_{\Omega_s}(\bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}}_{\mathcal{V}[\, X\mapsto S\,],\Omega_s} \subseteq S \,\}) \\
= \;& \text{[by equation (4.10)]} \\
& \mathfrak{A}^{\mu X.\Psi}_{\Omega_s}(\bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V}[\, X\mapsto S\,],\Omega_s} \subseteq S \,\}) \\
= \;& [\![\mu X.\Psi]\!]^{\mathcal{T}_s}_{\mathcal{V},\Omega_s}
\end{aligned}
$$

□

The second claim can now be shown straight-forwardly by structural induction on Φ using Claim 1.

Finally, the lemma itself is shown by induction on n simultaneously for both cases. If $n = 0$ both parts are obviously valid. Now assume that the lemma holds for n states $s_1, \ldots, s_n$, and let $s_{n+1} \in \mathcal{S}$. Then we conclude

$$
\begin{aligned}
& [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V},\Omega} \\
= \;& \text{[by induction hypothesis for 1.]} \\
& [\![\Phi]\!]^{\mathcal{T}_{s_1,\ldots,s_n}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^n} \\
= \;& \text{[by Lemma 4.3.7(1)]} \\
& [\![\Phi]\!]^{\mathcal{T}_{s_1,\ldots,s_n}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^n[s_{n+1}\models Sat(s_{n+1},\Phi)^{\mathcal{T}_{s_1,\ldots,s_n}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^n}]} \\
= \;& \text{[by Claim 1]} \\
& [\![\Phi]\!]^{\mathcal{T}_{s_1,\ldots,s_n,s_{n+1}}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^n[s_{n+1}\models Sat(s_{n+1},\Phi)^{\mathcal{T}_{s_1,\ldots,s_n}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^n}]} \\
= \;& \text{[by induction hypothesis for 2.]} \\
& [\![\Phi]\!]^{\mathcal{T}_{s_1,\ldots,s_n,s_{n+1}}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^n[s_{n+1}\models Sat(s_{n+1},\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= \;& [\![\Phi]\!]^{\mathcal{T}_{s_1,\ldots,s_{n+1}}}_{\mathcal{V},\Omega[s_i\models\Delta_i]_{i=1}^{n+1}}
\end{aligned}
$$

as well as

$$
\begin{aligned}
& Sat(s', \Phi)^{\mathcal{T}}_{\mathcal{V},\Omega} \\
= \quad & \text{[by induction hypothesis for 2.]} \\
& Sat(s', \Phi)^{\mathcal{T}_{s_1,\ldots,s_n}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^n} \\
= \quad & \text{[by Claim 2]} \\
& Sat(s', \Phi)^{\mathcal{T}_{s_1,\ldots,s_n,s_{n+1}}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^n [s_{n+1} \models Sat(s_{n+1},\Phi)^{\mathcal{T}_{s_1,\ldots,s_n}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^n}]} \\
= \quad & \text{[by induction hypothesis for 2.]} \\
& Sat(s', \Phi)^{\mathcal{T}_{s_1,\ldots,s_{n+1}}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^n [s_{n+1} \models Sat(s_{n+1},\Phi)^{\mathcal{T}}_{\mathcal{V},\Omega}]} \\
= \quad & Sat(s', \Phi)^{\mathcal{T}_{s_1,\ldots,s_{n+1}}}_{\mathcal{V},\Omega[s_i \models \Delta_i]_{i=1}^{n+1}}
\end{aligned}
$$

This completes the proof. □

The importance of the assertion-based semantics for the verification of sequentially composed processes is now revealed by the following theorem. Intuitively, it expresses that the verification problem for a sequential composition of processes may be decomposed into subproblems for the component processes if we know the set of formulas valid for states where the first component terminates and the second starts to react.

To fix notation, recall that $\mathcal{T}_0; (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ denotes the sequential composition of the sequential labelled transition graphs $\mathcal{T}_i$, for $0 \leq i \leq n$. Henceforth, we will denote by $\mathcal{T}_i'$ the embedding of $\mathcal{T}_i$ in $\mathcal{T}_0; (\mathcal{T}_1, \ldots, \mathcal{T}_n)$. Moreover, if Ω is an assertion defined on $\mathcal{T}_0; (\mathcal{T}_1, \ldots, \mathcal{T}_n)$, we will abbreviate $\Omega_{|\mathcal{T}_i'}$ by Ω_i', while Ω_i denotes the assertion defined on $\mathcal{T}_i$ which yields Ω_i' when embedded in $\mathcal{T}_0; (\mathcal{T}_1, \ldots, \mathcal{T}_n)$. For valuations we use similar conventions.

Theorem 4.3.1 (Compositionality of ABS).
Let $\mathcal{T}_i = (\mathcal{S}_i, Act, \rightarrow_i, s_{i0}, (s_{i1}, \ldots, s_{in}))$, for $0 \leq i \leq n$, be $n+1$ sequential labelled transition graphs, Φ be a formula, Ω be an assertion, and $\mathcal{V}$ be a valuation. If we abbreviate the satisfiability sets of the terminating states of $\mathcal{T}_0$, i.e. $Sat(s_{i0}, \Phi)^{\mathcal{T}_i}_{\mathcal{V}_i,\Omega_i}$, by Δ_i then we have

1. $[\![\Phi]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega} = [\![\Phi]\!]^{\mathcal{T}_0}_{\mathcal{V}_0,\Omega_0[s_{0i} \models \Delta_i]_{i=1}^n} \cup \bigcup_{1 \leq i \leq n} [\![\Phi]\!]^{\mathcal{T}_i}_{\mathcal{V}_i,\Omega_i}$
2. $Sat(s_{00}, \Phi)^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega} = Sat(s_{00}, \Phi)^{\mathcal{T}_0}_{\mathcal{V}_0,\Omega_0[s_{0i} \models \Delta_i]_{i=1}^n}$
3. $Sat(s_i, \Phi)^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega} = Sat(s_i, \Phi)^{\mathcal{T}_i'}_{\mathcal{V}_i',\Omega_i'} = Sat(s_{i0}, \Phi)^{\mathcal{T}_i}_{\mathcal{V}_i,\Omega_i}$, *for* $1 \leq i \leq n$

Proof. The first equality of the third part follows from Lemma 4.3.8 since $\mathcal{T}_i'$ is a closed component of $\mathcal{T}_0; (\mathcal{T}_1, \ldots, \mathcal{T}_n)$. Moreover, $\mathcal{T}_i$ and $\mathcal{T}_i'$ are isomorphic structures which yields the second equality of the third claim.
In order to show the first part let now

$$\Gamma_i =_{\mathrm{df}} Sat(s_i, \Phi)^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega}, \text{ and } \Lambda_i =_{\mathrm{df}} Sat(s_i, \Phi)^{\mathcal{T}_i'}_{\mathcal{V}_i',\Omega_i'}$$

Then we conclude

$$
\begin{array}{ll}
& [\![\Phi]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega} \\
= & \text{[by Lemma 4.3.7]} \\
& [\![\Phi]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega[s_i\models\Gamma_i]_{i=1}^n} \\
= & \text{[by Lemma 4.3.9]} \\
& [\![\Phi]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)\{s_1,\ldots,s_n\}\mapsto}_{\mathcal{V},\Omega[s_i\models\Gamma_i]_{i=1}^n} \\
= & \text{[by Lemma 4.3.8]} \\
& [\![\Phi]\!]^{\mathcal{T}'_0}_{\mathcal{V}'_0,\Omega'_0[s_i\models\Gamma_i]_{i=1}^n} \cup \bigcup_{1\le i\le n} [\![\Phi]\!]^{\mathcal{T}'_i\{s_i\}\mapsto}_{\mathcal{V}'_i,\Omega'_i[s_i\models\Gamma_i]} \\
= & \text{[by part (3)]} \\
& [\![\Phi]\!]^{\mathcal{T}'_0}_{\mathcal{V}'_0,\Omega'_0[s_i\models\Delta_i]_{i=1}^n} \cup \bigcup_{1\le i\le n} [\![\Phi]\!]^{\mathcal{T}'_i\{s_i\}\mapsto}_{\mathcal{V}'_i,\Omega'_i[s_i\models\Lambda_i]} \\
= & \text{[by Lemma 4.3.9]} \\
& [\![\Phi]\!]^{\mathcal{T}'_0}_{\mathcal{V}'_0,\Omega'_0[s_i\models\Delta_i]_{i=1}^n} \cup \bigcup_{1\le i\le n} [\![\Phi]\!]^{\mathcal{T}'_i}_{\mathcal{V}'_i,\Omega'_i} \\
= & \text{[by isomorphism]} \\
& [\![\Phi]\!]^{\mathcal{T}_0}_{\mathcal{V}_0,\Omega_0[s_{0i}\models\Delta_i]_{i=1}^n} \cup \bigcup_{1\le i\le n} [\![\Phi]\!]^{\mathcal{T}_i}_{\mathcal{V}_i,\Omega_i}
\end{array}
$$

Finally, the second part of the theorem is shown by structural induction on Φ. The proof relies on the equivalence

$$s_{00} \in [\![\Phi]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},\Omega} \quad \text{iff} \quad s_{00} \in [\![\Phi]\!]^{\mathcal{T}_0}_{\mathcal{V}_0,\Omega_0[s_{0i}\models\Delta_i]_{i=1}^n}$$

which follows immediately from the first part. □

The last step in the development of our theory is now to apply the compositionality of assertion-based semantics to its dual point of view: satisfiability sets. This will lead us from assertion-based semantics to *second-order semantics*, the foundation for a sound and complete logical decomposition of sequential processes.

Definition 4.3.6 (Property Transformer).
Let $\mathcal{T} = (\mathcal{S}, Act, \to, s_0, (s_1, \ldots, s_n))$ be a sequential labelled transition graph. The property transformer *of s_0 with respect to a closed formula Φ, denoted by $[\![s_0]\!]^{\mathcal{T}}_{\Phi}$, is a mapping from $(2^{SF(\Phi)})^n$ to $2^{SF(\Phi)}$ defined by*

$$[\![s_0]\!]^{\mathcal{T}}_{\Phi}(\bar{\Delta}_n) =_{\mathrm{df}} Sat(s_0,\Phi)^{\mathcal{T}}_{[s_i\models\Delta_i]_{i=1}^n,}.$$

A property transformer thus computes the set of subformulas of Φ valid for s_0 if we assert at the terminating states s_i the set of formulas Δ_i. That the second-order semantics is consistent with the usual semantics of processes in terms of its valid formulas, is stated in the following theorem.

Theorem 4.3.2 (Consistency).
Let $\mathcal{T} = (\mathcal{S}, Act, \rightarrow, s_0, (s_1, \ldots, s_n))$ be a sequential labelled transition graph, and Φ be a closed formula. Moreover, let $\Delta^\varepsilon =_{\mathrm{df}} Sat(\varepsilon, \Phi)$. Then we have

$$\Phi \in [\![s_0]\!]^{\mathcal{T}}_{\Phi}(\bar{\Delta}^{\varepsilon}{}_n) \quad \textit{iff} \quad s_0 \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}}$$

Proof. The theorem is a direct consequence of the slightly stronger property

$$\begin{array}{ll} & \Psi \in [\![s_0]\!]^{\mathcal{T}}_{\Phi}(\bar{\Delta}^{\varepsilon}{}_n) \\ \text{iff} & \Psi \in Sat(s_0, \Phi)^{\mathcal{T}}_{\mathcal{V},[s_i \models \Delta^\varepsilon]^n_{i=1}} \qquad \text{[by Definition 4.3.6]} \\ \text{iff} & \Psi \in Sat(s_0, \Phi)^{\mathcal{T}}_{\mathcal{V},[]} \qquad \text{[by Lemma 4.3.7]} \end{array}$$

for any $\Psi \in SF(\Phi)$, since $s_0 \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}}$ iff $\Phi \in Sat(s_0, \Phi)^{\mathcal{T}}_{\mathcal{V},[]}$. □

As for the assertion-based semantics we are now able to show compositionality for the second-order semantics, corresponding in this case to function composition of property transformers.

Theorem 4.3.3 (Compositionality).
Let $\mathcal{T}_i = (\mathcal{S}_i, Act, \rightarrow_i, s_{i0}, (s_{i1}, \ldots, s_{in}))$, for $0 \leq i \leq n$, be $n+1$ sequential labelled transition graphs. If Φ is a closed formula, we have

$$[\![s_{00}]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\Phi} = [\![s_{00}]\!]^{\mathcal{T}_0}_{\Phi} \circ ([\![s_{10}]\!]^{\mathcal{T}_1}_{\Phi}, \ldots, [\![s_{n0}]\!]^{\mathcal{T}_n}_{\Phi})$$

Proof. We fix an arbitrary n-tuple of formula sets $\bar{\Delta}_n$, and define

$$\Gamma_i =_{\mathrm{df}} Sat(s_i, \Phi)^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{[e_j \models \Delta_j]^n_{j=1},,} \text{ and } \Lambda_i =_{\mathrm{df}} Sat(s_{i0}, \Phi)^{\mathcal{T}_i}_{[s_{ij} \models \Delta_j]^n_{j=1},.}$$

By Theorem 4.3.1.3 we have $\Gamma_i = \Lambda_i$, for $1 \leq i \leq n$. The theorem is then shown by

$$\begin{array}{ll} & [\![s_{00}]\!]^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\Phi}(\bar{\Delta}_n) \\ = & \text{[by Definition 4.3.6]} \\ & Sat(s_{00}, \Phi)^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},[e_i \models \Delta_i]^n_{i=1}} \\ = & \text{[by Lemma 4.3.7]} \\ & Sat(s_{00}, \Phi)^{\mathcal{T}_0;(\mathcal{T}_1,\ldots,\mathcal{T}_n)}_{\mathcal{V},[s_i \models \Gamma_i, e_i \models \Delta_i]^n_{i=1}} \\ = & \text{[by Theorem 4.3.1(2)]} \\ & Sat(s_{00}, \Phi)^{\mathcal{T}'_0}_{\mathcal{V},[s_i \models \Gamma_i]^n_{i=1}} \\ = & \text{[by Theorem 4.3.1(3)]} \\ & Sat(s_{00}, \Phi)^{\mathcal{T}_0}_{\mathcal{V},[s_i \models \Lambda_i]^n_{i=1}} \\ = & \text{[by Definition 4.3.6]} \\ & Sat(s_{00}, \Phi)^{\mathcal{T}_0}_{\mathcal{V},[s_{0i} \models [\![s_{i0}]\!]^{\mathcal{T}_i}_{\Phi}(\bar{\Delta}_n)]^n_{i=1}} \\ = & \text{[by Definition 4.3.6]} \\ & [\![s_{00}]\!]^{\mathcal{T}_0}_{\Phi}([\![s_{10}]\!]^{\mathcal{T}_1}_{\Phi}(\bar{\Delta}_n), \ldots, [\![s_{n0}]\!]^{\mathcal{T}_n}_{\Phi}(\bar{\Delta}_n)) \end{array}$$

□

Theorem 4.3.3 will play a major role in our model-checking algorithm which we are going to develop in the next section. Its importance lies in the fact that it allows to decompose the second-order semantics of composed processes into the behaviour of its constituents thereby providing the foundation for a reduction from model-checking the infinite-state transition graph to model-checking its finite representation.

4.4 Verifying Behavioural Properties

In this section we develop a model checker for pushdown processes which decides the alternation-free modal μ-calculus. The heart of the algorithm consists of determining *property transformers* for each fragment of the PDPA process under consideration which provide a compositional and sound description of the logical behaviour of the pushdown process.

In Section 4.4.1 we introduce *hierarchical equational μ-formulas* as our algorithmic representation of alternation-free μ-formulas. Subsequently, we present our model checking algorithm in Section 4.4.2. The effectiveness of our method is a consequence of the fact that it suffices to deal with the part of a property transformer which concerns the subformulas of the property to be investigated.

4.4.1 Hierarchical Equational μ-Formulas

To ease the development of our model checking algorithm we represent alternation-free μ-formulas by means of *hierarchical equational μ-formulas* (cf. [CS91, CS92]). In this formalism properties can be expressed much more concisely, since it supports the sharing of common subexpressions. Theoretically, this may lead, in the best case, to an exponential reduction in the size of the formula, and practically, algorithms may take advantage of the saving and reuse of intermediate results. In this section we present the syntax and semantics of hierarchical equational μ-formulas, as well as the translation of ordinary μ-formulas into equational form.

Syntax.

Definition 4.4.1 (Hierarchical Equational μ-Formula).
A μ-formula of the form

$$\mathtt{tt},\ \mathtt{ff},\ X,\ X_1 \vee X_2,\ X_1 \wedge X_2,\ \langle a \rangle X,\ \text{or}\ \ [a]X$$

is called a simple formula. *An* equational block σB *is a list of mutually recursive equations* $B = (X_1 = \Phi_1, \ldots, X_n = \Phi_n)$ *where the* X_i *are all distinct variables, each* Φ_i *is a simple formula, and* σ*, called the* parity *of* σB*, is a fixpoint operator, i.e. is either* μ *or* ν*. An equational block* μB *is called a* minimal *block, while an equational block* νB *is also referred to as*

a maximal *block. Moreover, we denote by* $Var(B)$ *the set of all left-hand side variables defined by* B*, and call an equational block* σB closed *if every variable occurring in a right-hand side of some equation in* B *is a member of* $Var(B)$*.*

An equational μ-formula $\mathcal{F} = (\sigma_1 B_1, \ldots, \sigma_m B_m)$ *is now a list of equational blocks* $\sigma_i B_i$ *where the variables defined in all the* B_i*, denoted by* $Var(\mathcal{F})$*, are distinct. As a special case, an equational* μ*-formula* $\mathcal{F} = (\sigma_1 B_1, \ldots, \sigma_m B_m)$ *is said to be* hierarchical *if the existence of a left-hand side variable of a block* B_j *in a right-hand side formula of a block* B_i *implies* $i \leq j$*. An equational* μ*-formula* $\mathcal{F}$ *is, finally, said to be* closed *if every variable occurring in a right-hand side of some equation is a member of* $Var(\mathcal{F})$*.*

Intuitively, an equation $X = \Phi$ represents a least fixpoint if it occurs in a minimal block μB, and represents a greatest fixpoint when occurring in a maximal block νB, respectively. An equational block σB can therefore be interpreted as a *simultaneous* least or greatest fixpoint depending on the parity σ. However, it is worth noticing that the order of equational blocks is semantically significant due to the presence of mutually recursive least and greatest fixpoints.

Semantics. In order to define the semantics of an hierarchical equational μ-formula $\mathcal{F} = (\sigma_1 B_1, \ldots, \sigma_m B_m)$ with respect to a labelled transition graph $\mathcal{T} = (\mathcal{S}, Act, \rightarrow)$, we first define the semantics of an individual block

$$\sigma B = (X_1 = \Phi_1, \ldots, X_n = \Phi_n).$$

Given a fixed valuation $\mathcal{V}$, we may build a function $f^{\mathcal{T}}_{B,\mathcal{V}} : (2^{\mathcal{S}})^n \rightarrow (2^{\mathcal{S}})^n$ representing the "evaluation" of B with respect to $\mathcal{V}$ as follows. Let $\bar{S}_n \in (2^{\mathcal{S}})^n$, and let

$$\mathcal{V}[\, \bar{X}_n \mapsto \bar{S}_n \,] =_{\mathrm{df}} \mathcal{V}[\, X_1 \mapsto S_1 \,] \ldots [\, X_n \mapsto S_n \,]$$

be the valuation that results from $\mathcal{V}$ by updating the binding of X_i to S_i. Then

$$f^{\mathcal{T}}_{B,\mathcal{V}}(\bar{S}_n) =_{\mathrm{df}} (\llbracket \Phi_1 \rrbracket^{\mathcal{T}}_{\mathcal{V}[\, \bar{X}_n \mapsto \bar{S}_n \,]}, \ldots, \llbracket \Phi_n \rrbracket^{\mathcal{T}}_{\mathcal{V}[\, \bar{X}_n \mapsto \bar{S}_n \,]})$$

Now $(2^{\mathcal{S}})^n$ forms a complete lattice, where the ordering, join and meet operations are the pointwise extensions of the set-theoretic inclusion $\subseteq$, union $\cup$ and intersection $\cap$, respectively. Moreover, for any block of equations B and valuation $\mathcal{V}$, $f^{\mathcal{T}}_{B,\mathcal{V}}$ is monotonic with respect to this lattice and therefore, according to the Tarski-Knaster fixpoint theorem, has both a *least* fixed point, $\mu f^{\mathcal{T}}_{B,\mathcal{V}}$, and a *greatest* fixed point, $\nu f^{\mathcal{T}}_{B,\mathcal{V}}$. In general, these may be characterised as follows.

$$\mu f^{\mathcal{T}}_{B,\mathcal{V}} = \bigcap \{\, \bar{S}_n \in (2^{\mathcal{S}})^n \mid f^{\mathcal{T}}_{B,\mathcal{V}}(\bar{S}_n) \subseteq \bar{S}_n \,\}, \text{ and}$$

$$\nu f^{\mathcal{T}}_{B,\mathcal{V}} = \bigcup \{\, \bar{S}_n \in (2^{\mathcal{S}})^n \mid \bar{S}_n \subseteq f^{\mathcal{T}}_{B,\mathcal{V}}(\bar{S}_n) \,\}.$$

Equational blocks μB and νB are now interpreted as the least fixpoint of $f^{\mathcal{T}}_{B,\mathcal{V}}$, respectively the greatest, yielding *valuations* in the following fashion.

$$[\![\mu B]\!]^{\mathcal{T}}_{\mathcal{V}} = \mathcal{V}[\,\bar{X}_n \mapsto \mu f^{\mathcal{T}}_{B,\mathcal{V}}\,], \qquad \text{and} \qquad [\![\nu B]\!]^{\mathcal{T}}_{\mathcal{V}} = \mathcal{V}[\,\bar{X}_n \mapsto \nu f^{\mathcal{T}}_{B,\mathcal{V}}\,].$$

Given $\mathcal{V}$, we finally define the semantics of $\mathcal{F}$ as $[\![\mathcal{F}]\!]_{\mathcal{V}} = \mathcal{V}_1$ where $\mathcal{V}_1$ is obtained by means of the following sequence of valuations:

$$\mathcal{V}_1 = [\![\sigma_1 B_1]\!]_{\mathcal{V}_2},\ \ldots,\ \mathcal{V}_m = [\![\sigma_m B_m]\!]_{\mathcal{V}}.$$

Intuitively, this sequence of valuations corresponds to a bottom-up evaluation of the hierarchy of equational blocks. The base case is given by the last block B_m which does not contain any variables from earlier blocks whenever $\mathcal{F}$ is hierarchical. During the evaluation process we then have the invariant that whenever the semantics of a block B_i is computed we already know the semantics of all its free variables which must by definition belong to blocks B_j with $i < j$.

As for closed μ-formulas, the semantics $[\![\mathcal{F}]\!]_{\mathcal{V}}$ of closed equational μ-formulas $\mathcal{F}$ does not depend on $\mathcal{V}$. Thus when $\mathcal{F}$ is closed, we omit reference to $\mathcal{V}$. Finally, we define the *size* of $\mathcal{F}$, denoted by $|\mathcal{F}|$, as the number of equations contained in $\mathcal{F}$.

The translation of a closed μ-formula Φ in PNF into an equivalent equational μ-formula proceeds by generating successively equations for each subformula of Φ where the parity is determined by the closest surrounding fixpoint operator. Formally, we have

Definition 4.4.2 (Translation of μ-Formulas into Equational Form). *Let* :: *denote the concatenation of sequences. Then the recursive procedure* ***trans*** *which takes as input parameter a parity, a variable, and a μ-formula in PNF and outputs an equational μ-formula is inductively defined by the rules given in Table 4.4. With each closed μ-formula Φ in PNF we associate the equational μ-formula $\mathcal{F}_\Phi =_{\mathrm{df}}$ **trans**(μ, X, Φ) where X is a fresh variable not occurring in Φ.*

The translation of a closed μ-formula Φ into its equivalent equational μ-formula $\mathcal{F}_\Phi$ can be performed in time $O(\,|\Phi|\,)$, and, moreover, we have that the size of the generated formula $\mathcal{F}_\Phi$ is the same as the size of the given formula Φ, i.e. $|\mathcal{F}_\Phi| = |\Phi|$. It is worth noticing that the translation given above yields equational μ-formulas of a particular form, since each generated block consists of exactly one equation. To remedy this algorithmically expensive structure Cleaveland, Klein, and Steffen [CKS92] developed optimisations which aim on reducing the number of blocks without change in the semantics of the root variable. We want also to point out that there exists a close relationship between the equational μ-formula $\mathcal{F}_\Phi$ and the Fischer–Ladner closure of a formula Φ, since each equation in $\mathcal{F}_\Phi$ corresponds to a formula of $CL(\Phi)$.

Example 4.4.1. We illustrate the translation of μ-formulas into equational form by means of the property

$$
\begin{array}{rcl}
\mathbf{trans}(\sigma, Y, \mathtt{tt}) & =_{\mathrm{df}} & \sigma(Y = \mathtt{tt}) \\
\mathbf{trans}(\sigma, Y, \mathtt{ff}) & =_{\mathrm{df}} & \sigma(Y = \mathtt{ff}) \\
\mathbf{trans}(\sigma, Y, X) & =_{\mathrm{df}} & \sigma(Y = X) \\
\mathbf{trans}(\sigma, Y, \Phi_1 \vee \Phi_2) & =_{\mathrm{df}} & \sigma(Y = X_1 \vee X_2) \\
 & & \quad :: \mathbf{trans}(\sigma, X_1, \Phi_1) :: \mathbf{trans}(\sigma, X_2, \Phi_2) \\
 & & \text{where } X_1 \text{ and } X_2 \text{ are fresh variables.} \\
\mathbf{trans}(\sigma, Y, \Phi_1 \wedge \Phi_2) & =_{\mathrm{df}} & \sigma(Y = X_1 \wedge X_2) \\
 & & \quad :: \mathbf{trans}(\sigma, X_1, \Phi_1) :: \mathbf{trans}(\sigma, X_2, \Phi_2) \\
 & & \text{where } X_1 \text{ and } X_2 \text{ are fresh variables.} \\
\mathbf{trans}(\sigma, Y, \mu X.\Phi) & =_{\mathrm{df}} & \sigma(Y = X) :: \mathbf{trans}(\mu, X, \Phi) \\
\mathbf{trans}(\sigma, Y, \nu X.\Phi) & =_{\mathrm{df}} & \sigma(Y = X) :: \mathbf{trans}(\nu, X, \Phi)
\end{array}
$$

Table 4.4. The rules of the procedure **trans**.

$$\Phi =_{\mathrm{df}} \nu X.\langle b\rangle \mathtt{tt} \wedge \langle a\rangle X$$

which intuitively expresses that there exists an infinite a-path on which the action b is always enabled. Application of the procedure **trans** to Φ then yields the equational μ-formula

$$\mathcal{F}'_{\Phi} = \left(\begin{array}{llcll} \mu(& X' & = & X &) \\ \nu(& X & = & X_1 \wedge X_3 &) \\ \nu(& X_1 & = & \langle b\rangle X_2 &) \\ \nu(& X_2 & = & \mathtt{tt} &) \\ \nu(& X_3 & = & \langle a\rangle X'' &) \\ \nu(& X'' & = & X &) \end{array} \right)$$

Here the variable X' can safely be eliminated since it does not occur in a right-hand side of some equation, and moreover, its semantics coincides with the semantics of X. Since now all blocks have the same parity the remaining equations can be joined into a single maximal equational block. Furthermore, after propagation of the right-hand side of X'' the equation $\nu(X'' = X)$ may also be safely discarded. Thus we, finally, obtain the equational μ-formula $\mathcal{F}_{\Phi} = (\nu B)$ which consists of the single equational block

$$\nu B = \nu \left(\begin{array}{lcl} X & = & X_1 \wedge X_3 \\ X_1 & = & \langle b\rangle X_2 \\ X_2 & = & \mathtt{tt} \\ X_3 & = & \langle a\rangle X \end{array} \right)$$

When the equational μ-formula $\mathcal{F}$ is clear from the context we abbreviate $s \in [\![X]\!]_{[\![\mathcal{F}]\!]}$ with $s \models X$. Finally, given a process expression E we write $E \models X$ if the state labelled E in the underlying transition graph satisfies X. Both notations can also be extended to sets of variables.

It is straight-forward to define assertion-based semantics, as well as second-order semantics, also for (hierarchical) equational μ-formulas $\mathcal{F}$. The interpretation of $\mathcal{F}$ relative to some assertion Ω is then obtained by interpreting each right-hand side of an equation in $\mathcal{F}$ relative to Ω. The only difference is that instead of subformulas of a μ-formula Φ assertions then assign subsets of $Var(\mathcal{F}) \cup Rhs(\mathcal{F})$ to processes where $Rhs(\mathcal{F})$ denotes the set of all right-hand sides of $\mathcal{F}$.

When proving the correctness of our model-checking algorithm we will restrict the type of allowed assertions even further to *consistent* assertions. We call a set of formulas Δ *consistent* if we have $X \in \Delta$ iff $\Phi \in \Delta$ whenever $X = \Phi$ is an equation of $\mathcal{F}$. An assertion Ω is then called *consistent* if each $\Delta \in im(\Omega)$ is consistent.

This requirement can be interpreted as a "fixpoint" consistency, and allows to consider only assertions which assigns subsets of $Var(\mathcal{F})$. Furthermore, since assertions valid for the empty process, as well as assertions obtained from property transformer applications to consistent arguments, are always consistent the requirement is no restriction concerning our algorithm.

In the remainder of this monograph we only consider a minimal equational block μB. Maximal equational blocks are dealt with in a dual way and the extension to hierarchical equational μ-formulas is straightforward.

4.4.2 The Model Checking Algorithm

In this section, we present our model checking algorithm for pushdown processes which decides the alternation-free modal μ-calculus. It consists of three steps:

1. Constructing a *recursive function scheme* $\Pi_{\mathcal{D},B}$ with respect to the PDPA specification $\mathcal{D}$ and the minimal equational block μB of interest: The appropriate solution of this scheme, called the *property transformer scheme* (PT-scheme), will yield the property transformers for each fragment of the pushdown process with respect to the subformulas of B.
2. Solving of the PT-scheme $\Pi_{\mathcal{D},B}$ by component-wise computation of a *property transformer* for each fragment of the pushdown process: essentially, we proceed as in the finite-state case (cf. [CKS92]), except for
   - the domain for the iteration, which is second-order here, i.e. it consists of property transforming functions.
   - the handling of fragments. They are dealt with by applying the currently valid approximation of their property transformers.

   The computation of the *component property transformers* (CPT) consists of two phases: the initialisation and the fixpoint computation.
   - The initialisation of the CPT's depends on whether we process a minimal or a maximal block: a CPT is initialised with the maximal transformer in case X is defined in a maximal block and with the minimal

transformer otherwise. This reflects the fact that variables defined by a maximal block are initially assumed to be satisfied by all states and minimal block variables by none.
- The overall fixpoint computation is done by standard fixpoint iteration. The evaluation of the right-hand sides during the fixpoint computation simply consists of functional composition and the application of the meet and join operations of the lattice of all CPT's (cf. [BS92b]).

3. Deciding the model-checking problem: Having the predicate transformers at hand, it can be checked whether the considered formula is a member of the set of formulas that results from applying the property transformer associated with the root fragment of the PDPA specification to the tuple of sets of formulas that are valid for a terminated state (cf. Corollary 4.4.1).

Constructing the PT-Scheme. The first step of the algorithm consists of constructing a system of equations for the property transformers of interest. The construction embodies both, the behavioural structure of the given pushdown process and the semantics of the given formula, respectively. Let

$$\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, q_1, Z_1)$$

be a PDPA specification in 2-PDNF with n control states, i.e. $|Q| = n$. Moreover, let

$$\mu B = \mu(X_1 = \Phi_1, \ldots, X_m = \Phi_m)$$

be a closed minimal equational block with left-hand side variables

$$\mathcal{X} = \{\, X_i \mid 1 \leq i \leq m \,\},$$

where X_1 represents the property to be investigated. Arbitrary hierarchical equational μ-formulas are dealt with by a straightforward hierarchical extension.

To solve the model checking problem we are mainly interested in computing the *semantic* property transformers

$$\begin{aligned} &S_{[q,Z]} : (2^{\mathcal{X}})^n \longrightarrow 2^{\mathcal{X}}, \\ &(\Delta_1, \ldots, \Delta_n) \mapsto [\![[q, Z]]\!]_{\mathcal{X}}^{\mathcal{T}_{\mathcal{D}}} (\Delta_1, \ldots, \Delta_n) \end{aligned}$$

of simple fragments with respect to the variables of B. For the sake of clarity, we split such a property transformer into its *component property transformers* (CPT)

$$\begin{aligned} &S^X_{[q,Z]} : (2^{\mathcal{X}})^n \longrightarrow \mathbb{B}, \\ &(\Delta_1, \ldots, \Delta_n) \mapsto (\lambda \Delta . X \in \Delta) \circ S_{[q,Z]}(\Delta_1, \ldots, \Delta_n) \end{aligned}$$

where $\mathbb{B}$ is the Boolean lattice over $\{0, 1\}$, and $\lambda \Delta . X \in \Delta : 2^{\mathcal{X}} \longrightarrow \mathbb{B}$ denotes the predicate characterising the membership of X in the set of variables Δ

given as an argument. Note that the component property transformers $S^X_{[q,Z]}$ for a given fragment $[q, Z]$ are equivalent to $S_{[q,Z]}$, as

$$S_{[q,Z]} = \bigcup_{X \in \mathcal{X}} \theta_X \circ S^X_{[q,Z]},$$

where $\theta_X(0) = \emptyset$ and $\theta_X(1) = \{X\}$.

Let now $\bot_{(n)}$ denote the least CPT which maps every element of $(2^{\mathcal{X}})^n$ to 0, $\top_{(n)}$ the greatest CPT which maps every element of $(2^{\mathcal{X}})^n$ to 1, and $\chi_{(n)}[X \in j]$ be the predicate characterising whether X is a member of the j-th argument, i.e.

$$\begin{array}{ll} \bot_{(n)} & =_{\mathrm{df}} \lambda \bar{\Delta}_n.0 \\ \top_{(n)} & =_{\mathrm{df}} \lambda \bar{\Delta}_n.1 \\ \chi_{(n)}[X \in j] & =_{\mathrm{df}} \lambda \bar{\Delta}_n.X \in \Delta_j \end{array}$$

Then the PT-scheme $\Pi_{\mathcal{D},B}$ is built by means of the rules given in Figure 4.5.

Property related equations

$t^X_{[q,Z]}$	$= \top_{(n)}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	$\mathtt{tt}$	$\in B$
$t^X_{[q,Z]}$	$= \bot_{(n)}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	$\mathtt{ff}$	$\in B$
$t^X_{[q,Z]}$	$= t^{X'}_{[q,Z]}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	X'	$\in B$
$t^X_{[q,Z]}$	$= t^{X'}_{[q,Z]} \sqcup t^{X''}_{[q,Z]}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	$X' \vee X''$	$\in B$
$t^X_{[q,Z]}$	$= t^{X'}_{[q,Z]} \sqcap t^{X''}_{[q,Z]}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	$X' \wedge X''$	$\in B$
$t^X_{[q,Z]}$	$= \sqcup \{ t^{X'}_{[q',\beta]} \mid [q,Z] \xrightarrow{a} [q',\beta] \}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	$\langle a \rangle X'$	$\in B$
$t^X_{[q,Z]}$	$= \sqcap \{ t^{X'}_{[q',\beta]} \mid [q,Z] \xrightarrow{a} [q',\beta] \}$	$\in \Pi_{\mathcal{D},B}$,	if	X	$=$	$[a]X'$	$\in B$

Equations related to assertions

$t^X_{[q_i,\epsilon]}$	$= \chi_{(n)}[X \in i]$	$\in \Pi_{\mathcal{D},B}$

Decomposition equations

$t^X_{[q,Z_1Z_2]} = t^X_{[q,Z_1]} \circ (t_{[q_1,Z_2]}, \ldots, t_{[q_n,Z_2]})$	$\in \Pi_{\mathcal{D},B}$,	if $t^X_{[q,Z_1Z_2]}$ occurs in some property related equation

Table 4.5. The rules for constructing $\Pi_{\mathcal{D},B}$.

According to the given construction rules the obtained PT-scheme consists of three parts:

1. a set of PT-equations related to the property represented by the minimal equational block μB,

2. a set of PT-equations handling the assertions at the end states of fragments, and
3. a set of PT-equations defining the property transformers of nonsimple fragments by applying the PDPA law $A6$.

The equations constituting the second and third part of the PT-scheme can be seen as evaluation rules for the appropriate property transformers, while the fixpoint computation will mainly deal with the first part. Therefore, we assume in the remainder that the right-hand sides of assertion related equations, as well as decomposition equations, have been substituted for the appropriate left-hand sides occurring in the first part of the PT-scheme. The size of the obtained PT-scheme is then given by the product of the sizes of the PDPA specification $\mathcal{D}$, and the minimal equational block μB.

The Fixpoint Iteration. Due to the fact that we consider a minimal equational block μB, we have to compute the *minimal* fixpoint of the constructed PT-scheme. This is done using a standard fixpoint algorithm as outlined in Figure 4.4. The central procedure **compfix** updates the component property transformers $t^X_{[q,Z]}$ until consistency is reached. It uses the auxiliary functions `rhs`, which delivers the right-hand side term of a given PT-variable, and `eval`, which evaluates a PT-term with respect to a given valuation. After the fixpoint computation the algorithm completes by applying the computed property transformer associated with the root fragment $[q_1, Z_1]$ to the n-tuple $(\Delta^\varepsilon, \ldots, \Delta^\varepsilon)$ where each component $\Delta^\varepsilon =_{\mathrm{df}} Sat(\varepsilon, \mathcal{X})$ is the set of variables of B valid for the empty process. The overall model checking algorithm is then summarised by the procedure **solve** given in Figure 4.4.

Correctness. The procedure **solve** consists of the construction of $\Pi_{\mathcal{D},B}$, the fixpoint computation accomplished by a call of **compfix** and a final computation of $T^{X_1}_{[q_1,Z_1]}(\Delta^\varepsilon_n)$. It always terminates, since the number of simple fragments is finite and the procedure **compfix** is monotonic on the finite lattice $(2^{\mathcal{X}})^n \longrightarrow \mathbb{B}$. Moreover, upon termination of **compfix**, a single application of $T^{X_1}_{[q_1,Z_1]}$ suffices to decide whether the formula represented by X_1 is valid for the overall system, i.e. at the start state of the process defined by the PDPA specification. This is a consequence of the following stronger property.

Theorem 4.4.1 (Correctness of the Property Transformers). *Let $\Pi_{\mathcal{D},B}$ be the PT-scheme obtained from a PDPA specification $\mathcal{D}$ and a closed minimal equational block μB. Moreover, let the* algorithmic *property transformer $T^X_{[q,Z]}$ be the value of $t^X_{[q,Z]}$ after termination of the procedure **compfix**. Then we have, for any tuple of consistent assertions $\bar{\Delta}_n$,*

$$\forall\, X \in \mathcal{X}, q \in Q, Z \in \mathcal{Z}. \quad T^X_{[q,Z]}(\bar{\Delta}_n) \;=\; S^X_{[q,Z]}(\bar{\Delta}_n)$$

Proof. Let us first observe that with each family of property transformers

$$\bar{F} =_{\mathrm{df}} \{\, F^X_{[q,Z]} : (2^{\mathcal{X}})^n \longrightarrow \mathbb{B} \,\}_{X \in \mathcal{X}, q \in Q, Z \in \mathcal{Z}},$$

procedure compfix
BEGIN
/* Initialisation */
Let $\bar{t}$ be the vector of PT-variables which occur on a left-hand side in an equation of $\Pi_{\mathcal{D},B}$.
$\mathcal{V}.old = \emptyset, \quad \mathcal{V}.new = [\,\bar{t} \mapsto \bar{\perp}_{(n)}\,];$

/* Fixpoint Computation */
WHILE $\mathcal{V}.new \neq \mathcal{V}.old$ DO
$\mathcal{V}.old := \mathcal{V}.new;$
$\mathcal{V}.new = [\,\bar{t} \mapsto \mathtt{eval}(\mathtt{rhs}(\bar{t}), \mathcal{V}.old)\,];$
OD
END

procedure solve$(\mathcal{D}, \mu B)$
BEGIN
Construction of the PT-scheme $\Pi_{\mathcal{D},B}$.
compfix;
Let $T^{X_1}_{[q_1,Z_1]}$ be the value of $t^{X_1}_{[q_1,Z_1]}$ after termination of the fixpoint computation, and let $\Delta^{\varepsilon} =_{\mathrm{df}} Sat(\varepsilon, \mathcal{X})$ be the set of variables of B which are valid for the empty process ε.
RETURN $T^{X_1}_{[q_1,Z_1]}(\Delta^{\varepsilon}_n)$;
END

Fig. 4.4. The algorithm for solving the model checking problem.

and each n-tuple of sets of variables $\bar{\Delta}_n$ we may associate the m sets of pushdown processes

$$P^{X_j}(\bar{F}, \bar{\Delta}_n) =_{\mathrm{df}} \{\,[q,\gamma] \mid F^{X_j}_{[q,\gamma]}(\bar{\Delta}_n) = true\,\},\ 1 \leq j \leq m.$$

These sets can be interpreted as an approximation of the semantics of X_j with respect to the assertion that $\bar{\Delta}_n$ holds for the terminating states in terms of the given family of property transformers $\bar{F}$.

If we consider in particular the family of *semantic* property transformers we thus obtain, for a given n-tuple of sets of variables $\bar{\Delta}_n$, the set of pushdown configurations which satisfy X_j under the assertion that each $[q_i, \epsilon]$, for $1 \leq i \leq n$, satisfies Δ_i. I.e. if we abbreviate the assertion $[[q_i,\epsilon] \models \Delta_i]^n_{i=1}$ by $\bar{\Omega}_n$, and denote by $\mathcal{V}_{\mu B,\bar{\Omega}_n}$ the assertion-based semantics of μB with respect to the assertion $\bar{\Omega}_n$ we have

$$[q,\gamma] \in P^{X_j}(\bar{S}, \bar{\Delta}_n) \quad \text{iff} \quad S^{X_j}_{[q,\gamma]}(\bar{\Delta}_n) = true \quad \text{iff} \quad [q,\gamma] \in [\![X_j]\!]_{\mathcal{V}_{\mu B,\bar{\Omega}_n}}$$

The second property of P^{X_j} we will need for the proof is its consistency with respect to the assertion $\bar{\Omega}_n$ whenever $\bar{\Omega}_n$ is consistent. By this we mean the equivalence

$$\mathfrak{A}^X_{\bar{\Omega}_n}(P^X(\bar{F},\bar{\Delta}_n)) = P^X(\bar{F},\bar{\Delta}_n) \tag{4.11}$$

which follows from

$$[q_i,\epsilon] \in \bar{\Omega}_n^{-1}(X) \;\text{ iff }\; X \in \Delta_i \;\text{ iff }\; [q_i,\epsilon] \in P^X(\bar{F},\bar{\Delta}_n),\ \text{for all } 1 \le i \le n,$$

as well as the equivalence

$$\mathfrak{A}^{\Phi}_{\bar{\Omega}_n}(P^X(\bar{F},\bar{\Delta}_n)) = P^X(\bar{F},\bar{\Delta}_n) \tag{4.12}$$

which holds whenever $X = \Phi$ is an equation of B, and is a consequence of the consistency of $\bar{\Omega}_n$.
Henceforth $(P^{X_1}(\bar{F},\bar{\Delta}_n),\ldots,P^{X_m}(\bar{F},\bar{\Delta}_n))$ is abbreviated by $P^{\bar{X}_m}(\bar{F},\bar{\Delta}_n)$.

In order to prove the theorem it now suffices to show the following two claims:

Claim 1: $T^X_{[q,Z]} \sqsubseteq S^X_{[q,Z]}$, for all $X \in \mathcal{X}, q \in Q, Z \in \mathcal{Z}$ and

Claim 2: $P^{\bar{X}_m}(\bar{T},\bar{\Delta}_n) \sqsupseteq_m P^{\bar{X}_m}(\bar{S},\bar{\Delta}_n)$ for all consistent $\bar{\Delta}_n$,

since the second claim implies $T^X_{[q,Z]} \sqsupseteq S^X_{[q,Z]}$, for all $X \in \mathcal{X}, q \in Q, Z \in \mathcal{Z}$. Let now

$$\mathit{\Pi}_{\mathcal{D},B} = \mu\ (t^X_{[q,Z]} = \Psi_{(X,q,Z)})_{X\in\mathcal{X},q\in Q,Z\in\mathcal{Z}}$$

be the PT-scheme obtained from $\mathcal{D}$ and μB by the construction given in Figure 4.5.

Proof of Claim 1: We show that the family of semantic property transformers is a fixpoint of the PT-scheme, i.e. that $[\![\bar{\Psi}]\!]_{[\bar{t}:=\bar{S}]} = \bar{S}$, which proves $T^X_{[q,Z]} \sqsubseteq S^X_{[q,Z]}$ since $\bar{T}$ is the least fixpoint of $\mathit{\Pi}_{\mathcal{D},B}$. Fix now some $\bar{\Delta}_n$, and let $\mathcal{V}$ abbreviate the assertion-based semantics of μB with respect to the assertion $[q_i,\epsilon] \models \Delta_i$, for $1 \le i \le n$. Moreover, let $t^X_{[q,Z]} = \Psi_{(X,q,Z)}$ be an equation of $\mathit{\Pi}_{\mathcal{D},B}$, due to $X = \Phi$ being an equation of B. The proof then proceeds by case analysis of Φ.

Case 1: $\Phi = \mathtt{tt}$.
By construction of $\mathit{\Pi}_{\mathcal{D},B}$ we have $\Psi_{(X,q,Z)} = \top_{(n)}$. However, since we know that $[q,Z] \in [\![\mathtt{tt}]\!]_{\mathcal{V}}$ for any $[q,Z]$, we also see that $S^X_{[q,Z]}(\bar{\Delta}_n)$ always yields true, and thus $S^X_{[q,Z]} = \top_{(n)}$.

Case 2: $\Phi = \mathtt{ff}$.
By similar arguments as in the previous case we obtain that $S^X_{[q,Z]}(\bar{\Delta}_n)$ is always false, and hence $S^X_{[q,Z]} = \bot_{(n)}$.

Case 3: $\Phi = X'$.
In this case $\Psi_{(X,q,Z)} = t^{X'}_{[q,Z]}$, and we thus have to show $S^X_{[q,Z]} = S^{X'}_{[q,Z]}$. But this follows immediately from

$$[q,Z] \in [\![X]\!]_{\mathcal{V}} \quad \text{ iff } \quad [q,Z] \in [\![X']\!]_{\mathcal{V}}$$

as $X = X'$ was the equation of B under consideration.

Case 4: $\Phi = X' \vee X''$.

Then $\Psi_{(X,q,Z)}$ is of the form $t^{X'}_{[q,Z]} \sqcup t^{X''}_{[q,Z]}$. Hence we have to prove

$$S^X_{[q,Z]} = S^{X'}_{[q,Z]} \sqcup S^{X''}_{[q,Z]}$$

which can be seen as follows:

$$\begin{array}{ll} & S^X_{[q,Z]}(\bar{\Delta}_n) = true \\ \text{iff} & [q,Z] \in [\![X]\!]_{\mathcal{V}} \\ \text{iff} & [q,Z] \in [\![X' \vee X'']\!]_{\mathcal{V}} \\ \text{iff} & [q,Z] \in [\![X']\!]_{\mathcal{V}} \text{ or } [q,Z] \in [\![X'']\!]_{\mathcal{V}} \\ \text{iff} & S^{X'}_{[q,Z]}(\bar{\Delta}_n) = true \text{ or } S^{X''}_{[q,Z]}(\bar{\Delta}_n) = true \\ \text{iff} & (S^{X'}_{[q,Z]} \sqcup S^{X''}_{[q,Z]})(\bar{\Delta}_n) = true \end{array}$$

Case 5: $\Phi = X' \wedge X''$. Analogous to $X = X' \vee X''$.

Case 6: $\Phi = \langle a \rangle X'$.

Here we have $\Psi_{(X,q,Z)} = \sqcup\{\, t^{X'}_{[q',\beta]} \mid [q,Z] \xrightarrow{a} [q',\beta] \,\}$. It thus remains to show

$$S^X_{[q,Z]} = \sqcup\{\, S^{X'}_{[q',\beta]} \mid [q,Z] \xrightarrow{a} [q',\beta] \,\}$$

This is proved by

$$\begin{array}{ll} & S^X_{[q,Z]}(\bar{\Delta}_n) = true \\ \text{iff} & [q,Z] \in [\![X]\!]_{\mathcal{V}} \\ \text{iff} & [q,Z] \in [\![\langle a \rangle X']\!]_{\mathcal{V}} \\ \text{iff} & \exists_{[q',\beta]}\ [q,Z] \xrightarrow{a} [q',\beta] \text{ and } [q',\beta] \in [\![X']\!]_{\mathcal{V}} \\ \text{iff} & \exists_{[q',\beta]}\ [q,Z] \xrightarrow{a} [q',\beta] \text{ and } S^{X'}_{[q',\beta]}(\bar{\Delta}_n) = true \\ \text{iff} & (\sqcup\{\, S^{X'}_{[q',\beta]} \mid [q,Z] \xrightarrow{a} [q',\beta] \,\})(\bar{\Delta}_n) = true \end{array}$$

Case 7: $\Phi = [a]X'$. Analogous to $X = \langle a \rangle X'$.

Proof of Claim 2: We fix again some n-tuple $\bar{\Delta}_n$ and abbreviate the assertion $[[q_i,\epsilon] \models \Delta_i]_{i=1}^n$ where we assert at each of the end states $[q_i,\epsilon]$ the set of variables Δ_i by $\bar{\Omega}_n$. We show then that $P^{\bar{X}_m}(\bar{T},\bar{\Delta}_n)$ is a fixpoint of the equational block B when B is interpreted wrt. $\mathcal{D}$ under the assertion $\bar{\Omega}_n$, i.e. that

$$[\![\bar{\Phi}_m]\!]_{[\,\bar{X}_m \mapsto P^{\bar{X}_m}(\bar{T},\bar{\Delta}_n)\,],\bar{\Omega}_n} = P^{\bar{X}_m}(\bar{T},\bar{\Delta}_n),$$

This fixpoint property immediately implies our claim

$$P^{\bar{X}_m}(\bar{S},\bar{\Delta}_n) \sqsubseteq_m P^{\bar{X}_m}(\bar{T},\bar{\Delta}_n)$$

since $P^{\bar{X}_m}(\bar{S},\bar{\Delta}_n)$ is the least fixpoint of B when we assert $\bar{\Omega}_n$. Let now $X = \Phi$ be some equation of B. The proof is then accomplished again by case analysis of Φ. To ease notation we abbreviate the valuation $[\,\bar{X}_m \mapsto P^{\bar{X}_m}(\bar{T},\bar{\Delta}_n)\,]$ by $\mathcal{V}$.

Case 1: $\Phi = \mathtt{tt}$:

$$
\begin{aligned}
& [\![\mathtt{tt}]\!]_{\mathcal{V},\bar{\Omega}_n} \\
= \;& \mathfrak{A}^{\mathtt{tt}}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid q \in Q, \gamma \in \mathcal{Z}^*\,\}) \\
= \;& [\text{by construction of } \Pi_{\mathcal{D},B} \text{ we have} \\
& \qquad T^X_{[q,Z]} = \top_{(n)} \text{ for all } q \in Q, Z \in \mathcal{Z}] \\
& \mathfrak{A}^{\mathtt{tt}}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid T^X_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true}\,\} \\
= \;& \mathfrak{A}^{\mathtt{tt}}_{\bar{\Omega}_n}(P^X(\bar{T}, \bar{\Delta}_n)) \\
= \;& [\text{by equation (4.12)}] \\
& P^X(\bar{T}, \bar{\Delta}_n)
\end{aligned}
$$

Case 2: $\Phi = \mathtt{ff}$. This case is shown similar to the previous case.

Case 3: $\Phi = X'$:

$$
\begin{aligned}
& [\![X']\!]_{\mathcal{V},\bar{\Omega}_n} \\
= \;& \mathfrak{A}^{X'}_{\bar{\Omega}_n}(P^{X'}(\bar{T}, \bar{\Delta}_n)) \\
= \;& \mathfrak{A}^{X'}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid T^{X'}_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true}\,\}) \\
= \;& [\text{by construction of } \Pi_{\mathcal{D},B} \text{ we have} \\
& \qquad T^X_{[q,Z]} = T^{X'}_{[q,Z]} \text{ for all } q \in Q, Z \in \mathcal{Z}) \\
& \mathfrak{A}^{X'}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid T^X_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true}\,\}) \\
= \;& \mathfrak{A}^{X'}_{\bar{\Omega}_n}(P^X(\bar{T}, \bar{\Delta}_n)) \\
= \;& [\text{by equation (4.12)}] \\
& P^X(\bar{T}, \bar{\Delta}_n)
\end{aligned}
$$

Case 4: $\Phi = X' \vee X''$:

$$
\begin{aligned}
& [\![X' \vee X'']\!]_{\mathcal{V},\bar{\Omega}_n} \\
= \;& \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}([\![X']\!]_{\mathcal{V},\bar{\Omega}_n} \cup [\![X'']\!]_{\mathcal{V},\bar{\Omega}_n}) \\
= \;& \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}(\mathfrak{A}^{X'}_{\bar{\Omega}_n}(P^{X'}(\bar{T}, \bar{\Delta}_n)) \cup \mathfrak{A}^{X''}_{\bar{\Omega}_n}(P^{X''}(\bar{T}, \bar{\Delta}_n))) \\
= \;& [\text{by equation (4.11)}] \\
& \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}(P^{X'}(\bar{T}, \bar{\Delta}_n) \cup P^{X''}(\bar{T}, \bar{\Delta}_n)) \\
= \;& \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid T^{X'}_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true}\,\} \cup \\
& \qquad\qquad \{\,[q,\gamma] \mid T^{X''}_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true}\,\}) \\
= \;& \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid (T^{X'}_{[q,\gamma]} \sqcup T^{X''}_{[q,\gamma]})(\bar{\Delta}_n) = \mathit{true}\,\}) \\
= \;& [\text{by construction of } \Pi_{\mathcal{D},B} \text{ we have} \\
& \qquad T^X_{[q,Z]} = T^{X'}_{[q,Z]} \sqcup T^{X''}_{[q,Z]} \text{ for all } q \in Q, Z \in \mathcal{Z}] \\
& \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}(\{\,[q,\gamma] \mid T^X_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true}\,\}
\end{aligned}
$$

$$
\begin{array}{ll}
= & \mathfrak{A}^{X' \vee X''}_{\bar{\Omega}_n}(P^X(\bar{T}, \bar{\Delta}_n)) \\
= & \text{[by equation (4.12)]} \\
& P^X(\bar{T}, \bar{\Delta}_n)
\end{array}
$$

Case 5: $\Phi = X' \wedge X''$. Analogous to $X = X' \vee X''$.

Case 6: $\Phi = \langle a \rangle X'$:

$$
\begin{array}{ll}
& [\![\langle a \rangle X']\!]_{\mathcal{V}, \bar{\Omega}_n} \\
= & \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(\{\, [q,\gamma] \mid \exists_{[q',\beta]}\ [q,\gamma] \xrightarrow{a} [q',\beta] \text{ and } [q',\beta] \in [\![X']\!]_{\mathcal{V}, \bar{\Omega}_n} \,\}) \\
= & \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(\{\, [q,\gamma] \mid \exists_{[q',\beta]}\ [q,\gamma] \xrightarrow{a} [q',\beta] \text{ and } \\
& \qquad\qquad [q',\beta] \in \mathfrak{A}^{X'}_{\bar{\Omega}_n}(P^{X'}(\bar{T}, \bar{\Delta}_n)) \,\}) \\
= & \text{[by equation (4.11)]} \\
& \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(\{\, [q,\gamma] \mid \exists_{[q',\beta]}\ [q,\gamma] \xrightarrow{a} [q',\beta] \text{ and } [q',\beta] \in P^{X'}(\bar{T}, \bar{\Delta}_n) \,\}) \\
= & \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(\{\, [q,\gamma] \mid \exists_{[q',\beta]}\ [q,\gamma] \xrightarrow{a} [q',\beta] \text{ and } T^{X'}_{[q',\beta]}(\bar{\Delta}_n) = \mathit{true} \,\}) \\
= & \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(\{\, [q,\gamma] \mid (\sqcup\{\, T^{X'}_{[q',\beta]} \mid \exists_{[q',\beta]}\ [q,\gamma] \xrightarrow{a} [q',\beta] \,\})(\bar{\Delta}_n) = \mathit{true} \,\}) \\
= & \text{[by construction of } \Pi_{\mathcal{D},B} \text{ we have} \\
& \qquad T^X_{[q,Z]} = \sqcup\{\, T^{X'}_{[q',\beta]} \mid \exists_{[q',\beta]}\ [q,Z] \xrightarrow{a} [q',\beta] \,\} \text{ for all } q \in Q, Z \in \mathcal{Z}] \\
& \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(\{\, [q,\gamma] \mid T^X_{[q,\gamma]}(\bar{\Delta}_n) = \mathit{true} \,\}) \\
= & \mathfrak{A}^{\langle a \rangle X'}_{\bar{\Omega}_n}(P^X(\bar{T}, \bar{\Delta}_n)) \\
= & \text{[by equation (4.12)]} \\
& P^X(\bar{T}, \bar{\Delta}_n)
\end{array}
$$

Case 7: $\Phi = [a]X'$. Analogous to $X = \langle a \rangle X'$.

The correctness of the algorithm is now an easy corollary.

Corollary 4.4.1 (Correctness).
The algorithm computes the set of all variables of B which are valid at the start state of the process defined by $\mathcal{D}$.

$$[q_1, Z_1] \models X \quad \Longleftrightarrow \quad X \in T_{[q_1,Z_1]}(\Delta^\varepsilon_n)$$

Proof. Since Δ^ε is a consistent assertion we have:

$$
\begin{array}{lll}
& [q_1, Z_1] \models X & \\
\text{iff} & X \in S_{[q_1,Z_1]}\ (\Delta^\varepsilon_n) & \text{[by Theorem 4.3.2]} \\
\text{iff} & X \in T_{[q_1,Z_1]}\ (\Delta^\varepsilon_n) & \text{[by Theorem 4.4.1]}
\end{array}
$$

Complexity. Let us now consider the complexity of our algorithm. It turns out that the worst case time complexity of the model checking algorithm is quadratic in the size of the PDPA specification and exponential in the size of the formula as well as in the arity.

Theorem 4.4.2 (Complexity).
Let $\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, q_1, Z_1)$ be a PDPA specification in 2-PDNF and μB be a closed minimal equational block. If we measure the size of $\mathcal{D}$ as the number of summands in all right-hand side expressions then the worst-case time complexity of ***solve*** *is*

$$O(\,|Q| * |\mathcal{D}|^2 * |B|^2 * 4^{|B|*|Q|}\,)$$

Proof. Let us first consider the time complexity for computing new approximate values for property transformers defined by a *decomposition equation*

$$t^X_{[q,Z_1Z_2]} = t^X_{[q,Z_1]} \circ (t_{[q_1,Z_2]}, \ldots, t_{[q_n,Z_2]}).$$

In such a PT-equation the right-hand side can be evaluated by computing the new CPT argument-wise for each of the $(2^{|B|})^{|Q|}$ arguments $\bar{\Delta}_n \in (2^{\mathcal{X}})^n$. Fixing some input $\bar{\Delta}_n$, the argument vector $t_{[q_i,Z_2]}(\bar{\Delta}_n)_{i=1}^n$ and therefore the necessary functional composition can then be computed in $O(\,|B| * |Q|\,)$ and the evaluation of the at most $|\mathcal{D}|$ decomposition equations gives a time complexity of

$$O(\,|Q| * |\mathcal{D}| * |B| * 2^{|B|*|Q|}\,).$$

The next step of the algorithm determines a new approximation for the property transformers occurring in a *property related equation*. The most expensive right-hand side computations result from PT-equations related to an equation $X = \Phi$ where Φ contains a modality. Now let X be such a variable and $\bar{t}^X = \bar{\Psi}^X$ be the vector of PT-equations associated with X. Again we compute the new approximations for the CPT's argument-wise. First note that the number of CPT's occurring in $\bar{\Psi}^X$ is bounded by $|\mathcal{D}|$ and that a join (respectively meet) of two CPT's for a fixed argument can be performed in $O(\,1\,)$. Thus the computation of all joins (respectively meets) occurring in $\bar{\Psi}^X$ can be done in $O(\,|\mathcal{D}|\,)$. Since $\Pi_{\mathcal{D},B}$ can be partitioned into $|B|$ vectors of equations $\bar{t}^X = \bar{\Psi}^X$, this gives us an overall time complexity of

$$O(\,|\mathcal{D}| * |B| * 2^{|B|*|Q|}\,)$$

for the second step in the loop.

Since each execution of the loop improves the approximation of at least one CPT (unless we have reached the fixed-point) and since the maximal chain length arising during the fixpoint computation of each of the at most $|\mathcal{D}| * |B|$ CPT's is less than $2^{|B|*|Q|}$, the overall worst-case time complexity for computing the fixpoint can be estimated by

$$\begin{aligned} &O(\,(|Q| * |\mathcal{D}| * |B| * 2^{|B|*|Q|} + |\mathcal{D}| * |B| * 2^{|B|*|Q|}) * |\mathcal{D}| * |B| * 2^{|B|*|Q|}\,) \\ &= \\ &O(\,|Q| * |\mathcal{D}|^2 * |B|^2 * 4^{|B|*|Q|}\,). \end{aligned}$$

A closer analysis of the algorithm reveals that only CPT's for variables which count as a *weight* are of interest, where the *weight of* B, written as $w(B)$, is

the number of distinct variables that occur after a modality on the right-hand side in some equation of B. Thus we can reduce the factor $|B|$ to $w(B)$, which is in general much smaller.

4.4.3 A Working Example

In this section we illustrate the presented model checking algorithm by means of an example. We take as the system to be model checked the context-free process from Figure 4.1 which is generated by the context-free process specification

$$\mathcal{C} = \{\, A = aAB + b,\ B = b \,\},$$

while the property we want to verify is given by the equational μ-formula $\mathcal{F}_\Phi$ of Example 4.4.1.

The first step of the model checking algorithm consists of constructing by means of the rules given in Table 4.5 the equational scheme for the property transformers. The construction embodies the structure of the BPA specification $\mathcal{C}$, as well as the structure of the equational formula $\mathcal{F}_\Phi$. In Figure 4.5 we present the PT-scheme Π obtained by this construction, together with $\mathcal{C}$ and $\mathcal{F}_\Phi$.

$$\mathcal{C} = \left\{ \begin{array}{lcl} A & = & aAB + b \\ B & = & b \end{array} \right\} \qquad \mathcal{F}_\Phi = \nu \left(\begin{array}{lcl} X & = & X_1 \wedge X_3 \\ X_1 & = & \langle b \rangle X_2 \\ X_2 & = & \mathtt{tt} \\ X_3 & = & \langle a \rangle X \end{array} \right)$$

$$\Pi = \nu \left(\begin{array}{lcl} t_A^X & = & t_A^{X_1} \sqcap t_A^{X_3} \\ t_B^X & = & t_B^{X_1} \sqcap t_B^{X_3} \\ t_A^{X_1} & = & t_\epsilon^{X_2} \\ t_B^{X_1} & = & t_\epsilon^{X_2} \\ t_A^{X_2} & = & \top \\ t_B^{X_2} & = & \top \\ t_A^{X_3} & = & t_{AB}^X \\ t_B^{X_3} & = & \bot \end{array} \right)$$

Fig. 4.5. The BPA specification $\mathcal{C}$, the equational μ-formula $\mathcal{F}_\Phi$, and the PT-scheme Π.

By definition, the component property transformers which shall constitute the elements of our domain during the fixpoint computation are functions from $2^{\mathcal{X}}$ to $\mathbb{B}$ where $\mathcal{X} = \{\, X, X_1, X_2, X_3 \,\}$. To ease notation we will denote the function which tests the membership of X_i simply by X_i itself.

The computation which determines the greatest fixpoint of the PT-scheme Π is then shown in Table 4.6. In the first step every property transformer is initialised with $\top$, as we are processing a maximal equational block. Subsequently, in each iteration loop the values of the property transformers are updated according to the specified operations and the values obtained in the previous iteration. Since the values in the fourth and third iteration coincide we, finally, have reached the fixpoint.

			init	1	2	3	4
t_A^X	=	$t_A^{X_1} \sqcap t_A^{X_3}$	$\top$	$\top$	$\top$	X_2	X_2
t_B^X	=	$t_B^{X_1} \sqcap t_B^{X_3}$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
$t_A^{X_1}$	=	$t_\epsilon^{X_2}$	$\top$	$\top$	X_2	X_2	X_2
$t_B^{X_1}$	=	$t_\epsilon^{X_2}$	$\top$	$\top$	X_2	X_2	X_2
$t_A^{X_2}$	=	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$t_B^{X_2}$	=	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$t_A^{X_3}$	=	t_{AB}^X	$\top$	$\top$	$\top$	$\top$	$\top$
$t_B^{X_3}$	=	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$
$t_\epsilon^{X_2}$	=	X_2	$\top$	X_2	X_2	X_2	X_2
t_{AB}^X	=	$t_A^X \circ t_B$	$\top$	$\top$	$\top$	$\top$	$\top$

Table 4.6. Fixpoint computation for the PT-scheme Π.

In the last step of our model checking algorithm we have to determine the set of variables which hold for the terminating state ϵ. In the example given above only X_2 holds for ϵ, since ϵ cannot perform any action at all. Thus we obtain that the process A satisfies Φ, since $t_A^X(\{ X_2 \})$ yields true.

4.5 Expressiveness of the modal μ-calculus

The expressiveness of a temporal logic is usually evaluated by comparing the logic at hand with other already known temporal logics by means of properties expressible in one logic, but not in the other. For example, it is well-known that CTL (Computation Tree Logic) is strictly weaker than CTL*, which itself is subsumed by the modal μ-calculus. Despite the expressiveness of the modal μ-calculus it also has some deficiencies since it does not admit to express properties concerned with counting occurrences of events.

To overcome this problem Bouajjani, Echahed, and Robbana [BER94] proposed an extension of CTL with Presburger arithmetic, called PCTL, allowing to express constraints on numbers of occurrences of events. Strikingly, for a large fragment of PCTL it can be proved that the satisfaction problem is decidable. More importantly, PCTL allows to characterise processes

whose associated trace languages are nonregular. In particular, these languages may even be context-sensitive. PCTL is therefore quite different from "traditional" propositional temporal logics which may express only regular properties definable by finite-state automata on infinite trees (cf. [Tho90]).

In this section we propose a new viewpoint different from considerations concerned purely with satisfiability questions. Instead, we will investigate the *structure of sets of processes* satisfying a formula of a fixed logic. In particular, it will turn out that properties expressed in the modal μ-calculus characterise always *regular* sets of processes when interpreted on the class of pushdown processes. This is a consequence of the compositionality of the second-order semantics by means of an automata-based construction which will be presented in detail in the remainder of this section.

We fix now a pushdown specification $\mathcal{D} = (Q, \mathcal{Z}, Act, \mathcal{E}, q_1, Z_1)$ with $|Q| = n$, and an equational μ-formula $\mathcal{F}$ with variables $\mathcal{X}$. By definition of the second-order semantics we may then associate with each fragment $[q, \gamma]$ a property transformer $S_{[q,\gamma]} : (2^{\mathcal{X}})^n \longrightarrow 2^{\mathcal{X}}$. We are now interested in the set of fragments (or processes) which satisfy the formula represented by some variable $X_l \in \mathcal{X}$. More formally, given a state q_k of the pushdown process system under consideration we want to characterise the set of stack contents

$$\{\, \gamma \in \mathcal{Z}^* \mid [q_k, \gamma] \models X_l \,\}. \tag{4.13}$$

The key point to observe is that we may associate with q_k and X_l a deterministic finite automaton

$$\mathcal{A}^{q_k, X_l} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \theta_{\mathcal{A}}, q^1_{\mathcal{A}}, F_{\mathcal{A}})$$

which is defined by

$$\begin{array}{lcl} Q_{\mathcal{A}} & = & (2^{\mathcal{X}})^n \\ \Sigma_{\mathcal{A}} & = & \mathcal{Z} \\ \theta_{\mathcal{A}}(\bar{\Delta}_n, Z) & = & (S_{[q_i,Z]}(\bar{\Delta}_n))_{i=1}^n \\ q^1_{\mathcal{A}} & = & (Sat(\varepsilon, \mathcal{X}))_{i=1}^n \\ F_{\mathcal{A}} & = & \{\, \bar{\Delta}_n \mid X_l \in \Delta_k \,\} \end{array}$$

That this automaton exactly characterises the set (4.13) is stated in the following proposition.

Proposition 4.5.1. $\mathcal{L}\,(\mathcal{A}^{q_k, X_l}) = \{\, \gamma \in \mathcal{Z}^* \mid [q_k, \gamma] \models X_l \,\}$

Proof. Let $\Rightarrow^Z_{\mathcal{A}}$ denote the transition relation in the automaton, and $\Rightarrow^{\gamma}_{\mathcal{A}}$ its extension to words over $\mathcal{Z}$. The proposition is then a consequence of the stronger property

$$\begin{aligned} S_{[q_i,Z_r\ldots Z_1]}(\bar{\Delta}_n) = \Delta'_i, \quad &\text{for all } 1 \leq i \leq n \\ &\text{iff} \\ \bar{\Delta}_n \Rightarrow_{\mathcal{A}}^{Z_1\ldots Z_r} &\bar{\Delta'}_n \end{aligned} \tag{4.14}$$

which shows that a transition sequence $Z_1 \ldots Z_r$ in the automaton corresponds to a sequence of property transformer applications associated with $Z_r \ldots Z_1$. (4.14) will now be shown by induction on r. For $r = 0$ we have $S_{[q_i,\epsilon]}(\bar{\Delta}_n) = \Delta_i$, as well as $\bar{\Delta}_n \Rightarrow_{\mathcal{A}}^{\epsilon} \bar{\Delta}_n$. Now assume the property holds for r. Then we conclude

$$\begin{array}{ll} & (S_{[q_i,Z_{r+1}\ldots Z_1]}(\bar{\Delta}_n))_{i=1}^n = \bar{\Delta'}_n \\ \text{iff} & \text{[by compositionality of the second-order semantics]} \\ & (S_{[q_i,Z_{r+1}]}((S_{[q_j,Z_r\ldots Z_1]}(\bar{\Delta}_n))_{j=1}^n))_{i=1}^n = \bar{\Delta'}_n \\ \text{iff} & \text{[by induction hypothesis and definition of } \theta_{\mathcal{A}}] \\ & \bar{\Delta}_n \Rightarrow_{\mathcal{A}}^{Z_1\ldots Z_r} (S_{[q_i,Z_r\ldots Z_1]}(\bar{\Delta}_n))_{i=1}^n \Rightarrow_{\mathcal{A}}^{Z_{r+1}} \bar{\Delta'}_n \\ \text{iff} & \bar{\Delta}_n \Rightarrow_{\mathcal{A}}^{Z_1\ldots Z_{r+1}} \bar{\Delta'}_n \end{array}$$

Now it is easy to see that the proposition holds, as

$$\begin{array}{ll} & [q_k,\gamma] \models X_l \\ \text{iff} & \text{[by Theorem 4.3.2]} \\ & X_l \in S_{[q_k,\gamma]}(Sat(\epsilon,\mathcal{X}))_{i=1}^n \\ \text{iff} & \text{[by (4.14)]} \\ & (Sat(\epsilon,\mathcal{X}))_{i=1}^n = q_{\mathcal{A}}^1 \quad \Rightarrow_{\mathcal{A}}^{\gamma} \quad \bar{\Delta'}_n \text{ and } X_l \in \Delta'_k \\ \text{iff} & \gamma \in \mathcal{L}\ (\mathcal{A}^{q_k,X_l}) \end{array}$$

□

As an immediate consequence, we obtain the main theorem of this section.

Theorem 4.5.1. *The set of fragments satisfying a formula X_l is regular.*

Proof. By means of Proposition 4.5.1 we obtain

$$\{\, [q,\gamma] \mid [q,\gamma] \models X_l \,\} = \bigcup_{k=1}^{n} \{\, [q_k,\gamma] \mid \gamma \in \mathcal{L}\ (\mathcal{A}^{q_k,X_l}) \,\}$$

We close this section by computing the automaton for the example from Section 4.4.3.

Example 4.5.1. In Table 4.6 we have shown the fixpoint iteration for the verification problem given by the context-free process $\mathcal{C}$ and the equational μ-formula $\mathcal{F}_\Phi$, and in particular, the final semantic component property transformers. The associated automaton can now be constructed by starting with the initial state which in this case consists of the set $\{\, X_2 \,\}$, and following

transitions labelled with the nonterminals A or B until no more states are added to the automaton.

The A-transition from the initial set $\{ X_2 \}$ leads, for example, to the set $\{ X, X_2, X_3, X_4 \}$ as

$$\begin{array}{lcll} S_A^X(\{ X_2 \}) & = & X_2 \in \{ X_2 \} & = \ true, \\ S_A^{X_1}(\{ X_2 \}) & = & X_2 \in \{ X_2 \} & = \ true, \\ S_A^{X_2}(\{ X_2 \}) & = & \top(\{ X_2 \}) & = \ true, \\ S_A^{X_3}(\{ X_2 \}) & = & \top(\{ X_2 \}) & = \ true. \end{array}$$

Continuing in this way we, finally, obtain the automaton of Figure 4.6 where the initial state 1 represents $\{ X_2 \}$, the state 2 represents $\{ X_1, X_2 \}$, and the state 3 represents $\{ X, X_2, X_3, X_4 \}$, respectively.

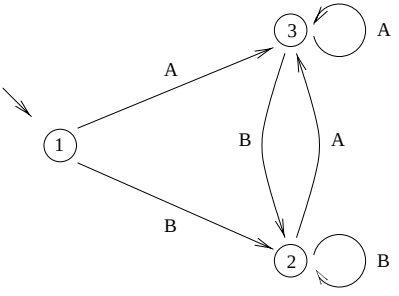

Fig. 4.6. The automaton for the context-free process $\mathcal{C}$ and the equational μ-formula $\mathcal{F}_\Phi$.

From the automaton we easily see that, for instance, any process AB^* satisfies the property represented by X, as

$$\{ X_2 \} \stackrel{B^*A}{\Rightarrow} \{ X, X_2, X_3, X_4 \} \text{ and } X \in \{ X, X_2, X_3, X_4 \},$$

whereas e.g. none of the processes B^* satisfies X, since in this case we have

$$\{ X_2 \} \stackrel{B^*}{\Rightarrow} \{ X_1, X_2 \} \text{ and } X \notin \{ X_1, X_2 \}.$$

5. Equivalence Checking

5.1 Introduction

In this chapter we present an elementary algorithm for deciding bisimulation equivalence between arbitrary context-free processes. This improves on the state of the art algorithm of Christensen, Hüttel and Stirling [CHS92] consisting of two semi-decision procedures running in parallel, which prohibits any complexity estimation. The point of our algorithm is the effective construction of a finite relation characterising all bisimulation equivalence classes, whose mere existence was exploited for the above mentioned decidability result.

In Section 5.2 we introduce the bisimulation equivalence problem for context-free processes and fix some notations used throughout this chapter. Section 5.3 investigates the notion of separability, while in Section 5.4 we present a new branching algorithm for normed context-free processes. By means of this algorithm we develop in Section 5.5 a bound on the number of transitions needed to separate two non-bisimilar normed BPA processes. Finally, Section 5.6 contains our new equivalence checking algorithm.

5.2 The Bisimulation Equivalence Problem

The question whether for any two given BPA systems

$$\mathcal{C}^{(1)} = (V^{(1)}, Act^{(1)}, \mathcal{E}^{(1)}, X_1^{(1)}) \text{ and } \mathcal{C}^{(2)} = (V^{(2)}, Act^{(2)}, \mathcal{E}^{(2)}, X_1^{(2)})$$

the roots are bisimilar, i.e. $X_1^{(1)} \sim X_1^{(2)}$, is called the *bisimulation equivalence problem* for context-free processes. However, taking the disjoint union of both BPA systems with variable renaming if required allows the reduction to the problem whether two sequences of nonterminals of a *single* BPA system are bisimulation equivalent.

In the remainder of this chapter we assume that we have a guarded BPA system $\mathcal{C} = (V, Act, \mathcal{E}, X_1)$ in K-GNF, where the variables $V = \{ X_1, \ldots, X_n \}$ are ordered by nondecreasing norm. Moreover, we divide the set of variables into two disjoint subsets

$$V_N =_{\text{df}} \{ X \in V \mid X \text{ is normed} \}$$

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354, pp. 115-149, 1997

and

$$V_U =_{\mathrm{df}} V \setminus V_N.$$

Due to the right-cancellation rule for unnormed processes (Lemma 2.6.3) we restrict our attention to variable sequences

$$\alpha \in \mathcal{P}(V^*) =_{\mathrm{df}} V_N^* \cup V_N^* V_U.$$

Additionally, we write $\mathcal{P}(V^+)$ for $\mathcal{P}(V^*) \setminus \{\epsilon\}$, and we denote the maximal finite norm of all given variables by $\mathcal{N}$.

Finally, we assume that the given BPA system is *normalised*, i.e. if α is an unnormed right-hand side summand of some normed variable X, then α must be of the form $a.Y$. This additional assumption does not impose any restriction, since every guarded BPA system in K-GNF can be normalised by means of the following transformation:

> If $a.\beta$ with $\beta \notin V$ is an unnormed right-hand side summand of some normed variable X, replace β by some fresh variable Y and add the equation $Y =_{\mathrm{df}} \beta$ to $\mathcal{E}$. When there are no more normed variables which satisfy the previous condition transform the resulting BPA system again into K-GNF.

That the transformation will eventually terminate can be seen from the following observations. The equations of all normed variables are still guarded after the normalisation transformation, while all the newly created equations $Y =_{\mathrm{df}} \beta$ define an unnormed variable with an unguarded right-hand side which has the form $\beta = Z\beta' \in V_N^+ V$. Replacing now every Z by its defining (guarded) equation and applying the right distributive law $(E_1 + E_2)F \sim E_1F + E_2F$ yields then again a BPA system in K-GNF.

5.3 Separability

The development of this section is based on a fixpoint construction, which Milner [Mil89] proposed to characterise bisimulation of finitely branching transition systems in terms of sequences of approximations. The notion of *separability* is defined in terms of the complements of these approximations. Therefore it characterises the number of transitions needed to separate two non-bisimilar processes. This notion will play a crucial role when developing our equivalence checking algorithm in Section 5.6.

Definition 5.3.1. *Let R be a binary relation between processes. Then $(p, q) \in \mathcal{F}(R)$ iff, for each $a \in Act$,*

1. $p \xrightarrow{a} p'$ *implies* $\exists\, q'.\ q \xrightarrow{a} q' \ \wedge\ (p', q') \in R$
2. $q \xrightarrow{a} q'$ *implies* $\exists\, p'.\ p \xrightarrow{a} p' \ \wedge\ (p', q') \in R$

Intuitively, the relation R represents the current approximation for bisimulation, while the mapping $\mathcal{F}$ is used to compute the succeeding approximation, respectively. In [Mil89] the following properties of $\mathcal{F}$ are proved.

Lemma 5.3.1.

1. *$\mathcal{F}$ is monotonic, i.e. if $R_1 \subseteq R_2$ then $\mathcal{F}(R_1) \subseteq \mathcal{F}(R_2)$.*
2. *R is a bisimulation iff $R \subseteq \mathcal{F}(R)$.*
3. *$\sim = \bigcup \{ R \mid R \subseteq \mathcal{F}(R) \}$.*
4. *If the labelled transition graph $(\mathcal{P}, Act, \rightarrow)$ is finitely branching, then $\mathcal{F}$ is continuous on $\mathcal{P} \times \mathcal{P}$. In this case we also have*

$$\sim = \bigcap \{ \mathcal{F}^i(\mathcal{P} \times \mathcal{P}) \mid i \geq 0 \}$$

Based on this iterative characterisation of bisimulation we define when two processes are said to be m-bisimilar.

Definition 5.3.2 (m-Bisimilarity).
Let $p, q \in \mathcal{P}$ be two processes. p and q are said to be m-bisimilar*, written $p \sim_m q$, if $(p, q) \in \mathcal{F}^{\,m}(\mathcal{P} \times \mathcal{P})$, for some $m \geq 0$.*

Considering the complements of these approximations leads immediately to the notion of *separability*.

Definition 5.3.3 (Separability).
Let $p, q \in \mathcal{P}$ be two processes. We define the separability *of p and q as*

$$Sep(p, q) =_{\text{df}} \sup\{ m \mid p \sim_m q \} + 1.$$

If the separability of p and q is finite[1]*, let us say m, then p and q are also called* m-separable*.*

The definitions given above can be illustrated in terms of an elegant game theoretic characterisation of bisimilarity [Sti93, NC94]. In this setting bisimulation is interpreted as a game between two persons, Player and Opponent, taking turns. Assuming that we have given two processes p and q, Player tries to prove the bisimilarity of p and q, while Opponent has the opposite intension. Opponent opens the game by choosing one of the processes together with a transition to a successor state. In turn, Player is now obliged to choose a matching transition of the other process, and the game continues with the successor states of both processes. If, however, the process chosen by Opponent is terminating the counter-move of Player is valid and the game continues if and only if also the other process is terminating.

Now we say that Player wins if the game continues forever and otherwise Opponent wins. Considered this way, p and q are bisimilar if Player has a

[1] Note that $p \sim q$ implies $\text{Sep}(p, q) = \infty$.

winning strategy, i.e. he can win every possible game starting with p and q. Moreover, m-bisimilarity of p and q means Player can respond at least m rounds in every game for p and q, while m-separability of p and q denotes that Opponent can win every game for p and q in at most m moves.

In contrast to bisimilarity, separability seems not to be very well elaborated. As a first step to overcome this problem, we present some new results concerning the separability of context-free processes.

Most of the results were first developed in [Cau89] for *simple grammars* a proper subclass of context-free grammars. Simple grammars can be interpreted as normed *deterministic* BPA systems, so it is surprising that these results also hold in the nondeterministic case.

Lemma 5.3.2. *If two processes $p, q \in \mathcal{P}$ are m-separable, then one of the following conditions hold.*

1. $\exists\, a, p'.\ p \xrightarrow{a} p'$ *such that* $\forall\, q'.\ q \xrightarrow{a} q' \Rightarrow p' \not\sim_{m-1} q'$*; or*
2. $\exists\, a, q'.\ q \xrightarrow{a} q'$ *such that* $\forall\, p'.\ p \xrightarrow{a} p' \Rightarrow p' \not\sim_{m-1} q'$.

In the first case we call $p \xrightarrow{a} p'$ a separating transition *for p and q, and in the latter case $q \xrightarrow{a} q'$, respectively. Moreover, we have in both cases $Sep(p', q') \leq m - 1$.*

Proof. Suppose p and q are m-separable. Then by definition of separability we have, in particular, $p \not\sim_m q$. The conditions stated in the lemma then follow from negation of m-bisimilarity, and $\mathrm{Sep}(p', q') \leq m - 1$ holds by definition of separability. □

Iterative application of the above lemma to a non-bisimilar pair of processes p and q yields a separability-reducing *transition track* for p and q.

Definition 5.3.4 (Transition Track).
Let $p \not\sim_m q$. A transition track *for (p, q) is a pair of transition sequences*

$$(p = p_m \xrightarrow{a_{m-1}} p_{m-1} \xrightarrow{a_{m-2}} \ldots \xrightarrow{a_r} p_r,\ q = q_m \xrightarrow{a_{m-1}} q_{m-1} \xrightarrow{a_{m-2}} \ldots \xrightarrow{a_r} q_r),$$

where $1 \leq r < m$, and the property $p_i \not\sim_i q_i$, for each $r \leq i \leq m$, holds.

Figure 5.1 gives an illustration of a transition track for two non-bisimilar processes p, q. A transition track can be seen as *separability-reducing* since $p_i \not\sim_i q_i$ implies $\mathrm{Sep}(p_i, q_i) \leq i$ and i is decreasing on the track.

It turns out that computing the separability of two processes is as hard as determining all relevant approximations $\sim_i$ containing both processes. For some restricted cases, however, we can give bounds for the separability. To start with, when two processes differ in norm the separability is bounded by the smaller norm plus one.

Lemma 5.3.3. *If $||\alpha|| < ||\beta||$ then $Sep(\alpha, \beta) \leq ||\alpha|| + 1$.*

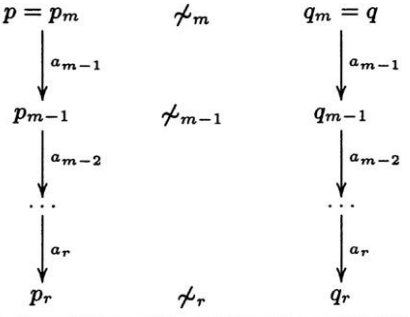

Fig. 5.1. A transition track for p, q.

Proof. Let $\alpha \xrightarrow{w} \epsilon$ be a norm-reducing transition sequence. Then either $\beta \not\xrightarrow{w}$ and consequently

$$\mathrm{Sep}(\alpha, \beta) \leq |w| = ||\alpha||,$$

or all β' with $\beta \xrightarrow{w} \beta'$ satisfy $\beta' \neq \epsilon$ due to $||\alpha|| < ||\beta||$, from which we conclude $\epsilon \not\sim_1 \beta'$ and

$$\mathrm{Sep}(\alpha, \beta) \leq ||\alpha|| + 1.$$

□

Lemma 5.3.4. *Let* $Sep(\alpha, \beta) = m+1$ *and* $\alpha \xrightarrow{w} \alpha'$ *with* $|w| \leq m$. *Then there exists some* β' *such that* $\beta \xrightarrow{w} \beta'$ *and* $Sep(\alpha, \beta) \leq |w| + Sep(\alpha', \beta')$.

Proof. The lemma is proved by induction on $|w|$. For $|w| = 0$ there is nothing to show. Now let

$$\mathrm{Sep}(\alpha, \beta) = m + 1$$

and therefore in particular $\alpha \sim_m \beta$. Moreover, assume $\alpha \xrightarrow{a} \alpha' \xrightarrow{w} \alpha''$ with $|w| \leq m - 1$. According to the definition of m-bisimilarity then there exists some β' satisfying $\beta \xrightarrow{a} \beta'$ and $\alpha' \sim_{m-1} \beta'$ which implies

$$\mathrm{Sep}(\alpha', \beta') \geq m = \mathrm{Sep}(\alpha, \beta) - 1.$$

The induction hypothesis now yields the existence of some β'' such that $\beta' \xrightarrow{w} \beta''$ and

$$\mathrm{Sep}(\alpha', \beta') \leq |w| + \mathrm{Sep}(\alpha'', \beta'').$$

Thus together we obtain as desired

$$\mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\alpha', \beta') + 1 \leq 1 + |w| + \mathrm{Sep}(\alpha'', \beta'').$$

□

Note that in case of simple grammars β' is uniquely determined, since each right-hand side of a variable is deterministic in the sense that for any $A \in V$, and any $a \in Act$ there exists at most one β such that $A \xrightarrow{a} \beta$. For arbitrary context-free processes systems, however, not every β' satisfies

$$\mathrm{Sep}(\alpha, \beta) \leq |w| + \mathrm{Sep}(\alpha', \beta'),$$

as illustrated by the following example.

Example 5.3.1. Let

$$\mathcal{E} = \{\, A =_{\mathrm{df}} a, B =_{\mathrm{df}} b, C =_{\mathrm{df}} a + aAB, D =_{\mathrm{df}} a + aA \,\},$$

be the set of equations of a guarded BPA specification and consider the two processes C and D whose transition graphs are shown in Figure 5.2. Fix $C \xrightarrow{a} AB$. Then for $D \xrightarrow{a} \epsilon$ we have

$$3 = \mathrm{Sep}(C, D) > |a| + \mathrm{Sep}(AB, \epsilon) = 1 + 1 = 2.$$

The β' whose existence is guaranteed by Lemma 5.3.4 is, however, A since $D \xrightarrow{a} A$ and

$$3 = \mathrm{Sep}(C, D) \leq |a| + \mathrm{Sep}(AB, A) = 1 + 2 = 3.$$

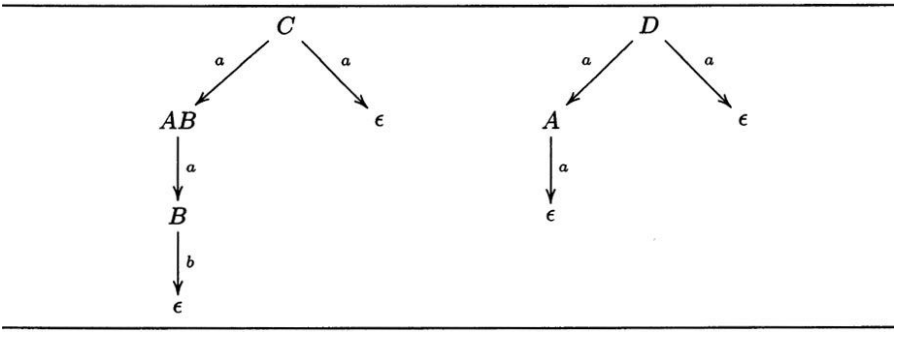

Fig. 5.2. The transition graphs for C and D.

Since sequencing of processes is the essential construction for context-free processes, we next investigate how separability behaves with respect to sequential composition.

Lemma 5.3.5. *For all $\alpha, \beta, \zeta, \eta \in V^*$ such that $\zeta \sim \eta$, we have*

1. $1 \leq Sep(\alpha, \beta) \leq Sep(\alpha\zeta, \beta\eta) \leq Sep(\alpha, \beta) + ||\zeta|| \leq Sep(\zeta\alpha, \eta\beta)$, *and*
2. $\min\{\, Sep(\alpha, \beta), Sep(\beta, \zeta) \,\} \leq Sep(\alpha, \zeta)$.

Proof.
(1.): Note that the first inequality $1 \leq \mathrm{Sep}(\alpha, \beta)$ is a direct consequence of the definition of separability. If one of the separabilities occurring in the lemma is infinite, we distinguish the following cases:

(a) If $\mathrm{Sep}(\alpha, \beta) = \infty$ then $\alpha \sim \beta$, and therefore also $\alpha\zeta \sim \beta\eta$, as we assumed $\zeta \sim \eta$. Hence we conclude $\mathrm{Sep}(\alpha\zeta, \beta\eta) = \infty$.

(b) If $\mathrm{Sep}(\alpha\zeta, \beta\eta) = \infty$ then $\alpha\zeta \sim \beta\eta$. Since there is nothing to show if $||\zeta||$ is infinite we may assume that $||\zeta||$ is finite. This implies that $||\alpha||$ and $||\beta||$ are either both infinite or both finite since, for example, the assumption $||\alpha||$ is finite, and $||\beta||$ is infinite yields $\alpha\zeta \sim \beta\eta \sim \beta$ and thus that $||\alpha\zeta||$ must be infinite which contradicts our assumption about the finiteness of $||\zeta||$. But, if $||\alpha||$ and $||\beta||$ are both infinite we conclude from Lemma 2.6.3 that $\alpha \sim \beta$, and if both norms are finite we similarly deduce by means of Lemma 2.6.1(2.) that $\alpha \sim \beta$, and therefore $\mathrm{Sep}(\alpha, \beta) = \infty$.

(c) If $\mathrm{Sep}(\alpha, \beta) = \infty$ then by definition $\alpha \sim \beta$, and hence also $\zeta\alpha \sim \eta\beta$. Therefore $\mathrm{Sep}(\zeta\alpha, \eta\beta) = \infty$. Finally, also from the assumption $||\zeta||$ is infinite we may deduce $\zeta\alpha \sim \zeta \sim \eta \sim \eta\beta$, and hence $\mathrm{Sep}(\zeta\alpha, \eta\beta) = \infty$.

Henceforth, we prove that the inequalities also hold, when we suppose that all separabilities occurring in the lemma are finite.

$\boxed{\text{(a)} \quad \mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\alpha\zeta, \beta\eta)}$

We first prove that

$$\alpha \sim_m \beta \quad \text{implies} \quad \alpha\zeta \sim_m \beta\eta \tag{5.1}$$

by induction on m. For $m = 0$ simply observe that $\alpha\zeta \sim_0 \beta\eta$ is always valid. Now assume (5.1) holds for m. Let $\alpha \sim_{m+1} \beta$, and consider $\alpha\zeta \xrightarrow{a} \alpha'\zeta$ for some action a and some α'. From $m+1$-bisimilarity of α and β we deduce there exists some β' such that $\beta \xrightarrow{a} \beta'$ and $\alpha' \sim_m \beta'$. However, this implies also $\beta\eta \xrightarrow{a} \beta'\eta$ and by induction hypothesis $\alpha'\zeta \sim_m \beta'\eta$. Since the dual case for $\beta\eta \xrightarrow{a} \beta'\eta$ is shown analogously, we obtain $\alpha\zeta \sim_{m+1} \beta\eta$. To continue, let now $\mathrm{Sep}(\alpha, \beta) = m + 1$ and hence $\alpha \sim_m \beta$. By (5.1) thus $\alpha\zeta \sim_m \beta\eta$ which, finally, yields

$$\mathrm{Sep}(\alpha, \beta) = m + 1 \leq \mathrm{Sep}(\alpha\zeta, \beta\eta).$$

$\boxed{\text{(b)} \quad \mathrm{Sep}(\alpha\zeta, \beta\eta) \leq \mathrm{Sep}(\alpha, \beta) + ||\zeta||}$

This inequality is shown by induction on $\mathrm{Sep}(\alpha, \beta)$. For $\mathrm{Sep}(\alpha, \beta) = 1$, we either have wlog. $\alpha = \epsilon$ and $\beta \neq \epsilon$, which implies by Lemma 5.3.3

$$\mathrm{Sep}(\alpha\zeta, \beta\eta) \leq ||\zeta|| + 1 = \mathrm{Sep}(\alpha, \beta) + ||\zeta||,$$

or we have $\alpha, \beta \neq \epsilon$, which implies

$$\mathrm{Sep}(\alpha\zeta, \beta\eta) = 1 = \mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\alpha, \beta) + ||\zeta||.$$

Now let $\mathrm{Sep}(\alpha, \beta) = m + 1$ and wlog. $\alpha \xrightarrow{a} \alpha'$ be a separating transition for α and β. Furthermore assume that $\mathrm{Sep}(\alpha\zeta, \beta\eta) = k + 1$. Thus $\alpha\zeta \sim_k \beta\eta$ and there exists some β' with $\beta \xrightarrow{a} \beta'$ and $\alpha'\zeta \sim_{k-1} \beta'\eta$. This yields

$$\mathrm{Sep}(\alpha'\zeta, \beta'\eta) \geq k = \mathrm{Sep}(\alpha\zeta, \beta\eta) - 1.$$

Since $\alpha \xrightarrow{a} \alpha'$ is separating, we conclude that $\mathrm{Sep}(\alpha', \beta') \leq m$, which allows us to apply the induction hypothesis. Summarising we obtain

$$\begin{aligned} \mathrm{Sep}(\alpha\zeta, \beta\eta) &\leq \mathrm{Sep}(\alpha'\zeta, \beta'\eta) + 1 \\ &\leq \mathrm{Sep}(\alpha', \beta') + 1 + ||\zeta|| \\ &\leq \mathrm{Sep}(\alpha, \beta) + ||\zeta||. \end{aligned}$$

(c) $\mathrm{Sep}(\alpha, \beta) + ||\zeta|| \leq \mathrm{Sep}(\zeta\alpha, \eta\beta)$

Let $\mathrm{Sep}(\zeta\alpha, \eta\beta) = m$, and $\mathrm{Sep}(\alpha, \beta) = r$. In order to show the inequality we construct a separability-reducing transition track for $(\zeta\alpha, \eta\beta)$ consisting of pairs $(\zeta_i\alpha, \eta_i\beta)$ such that the invariant $\zeta_i \sim \eta_i$ holds by means of the following strategy. If the actual pair is of the form $\zeta_i\alpha \not\sim_j \eta_i\beta$ let without loss of generality $\zeta_i\alpha \xrightarrow{a} \zeta_{i+1}\alpha$ be the separating transition[2]. Since by invariant $\zeta_i \sim \eta_i$ we can choose some η_{i+1} such that $\eta_i \xrightarrow{a} \eta_{i+1}$ and $\zeta_{i+1} \sim \eta_{i+1}$ which yields $\eta_i\beta \xrightarrow{a} \eta_{i+1}\beta$ as the matching transition. Moreover, as $\zeta_i\alpha \xrightarrow{a} \zeta_{i+1}\alpha$ was separating we obtain $\zeta_{i+1}\alpha \not\sim_{j-1} \eta_{i+1}\beta$. Eventually, this will lead us in $m - r \geq ||\zeta||$ steps to $\alpha \not\sim_r \beta$. Thus together we obtain

$$||\zeta|| + \mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\zeta\alpha, \eta\beta).$$

(2.): Assume without loss of generality that $m + 1 = \mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\beta, \zeta)$. Then $\alpha \sim_m \beta \sim_m \zeta$, and by transitivity also $\alpha \sim_m \zeta$. Thus we obtain $\mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\alpha, \zeta)$. □

Example 5.3.2. That the fourth inequality of the previous Lemma 5.3.5.1

$$\mathrm{Sep}(\alpha, \beta) + ||\zeta|| \leq \mathrm{Sep}(\zeta\alpha, \eta\beta)$$

may be proper is demonstrated by the example

$$\alpha = a, \beta = b \text{ and } \zeta = a + a\alpha + a\beta$$

given by Caucal. In this case

$$\mathrm{Sep}(\alpha, \beta) = 1, ||\zeta|| = 1, \text{ but } \mathrm{Sep}(\zeta\alpha, \zeta\beta) = 3.$$

Figure 5.3 shows the transition graphs for $\zeta\alpha$ and $\zeta\beta$. In the example every initial transition is separating which demonstrates that a separating transition needs not to be norm-reducing. In contrast, for simple grammars we always have equality since separating transitions are also norm-reducing.

[2] This transition needs not to be norm-reducing as in the case of simple grammars.

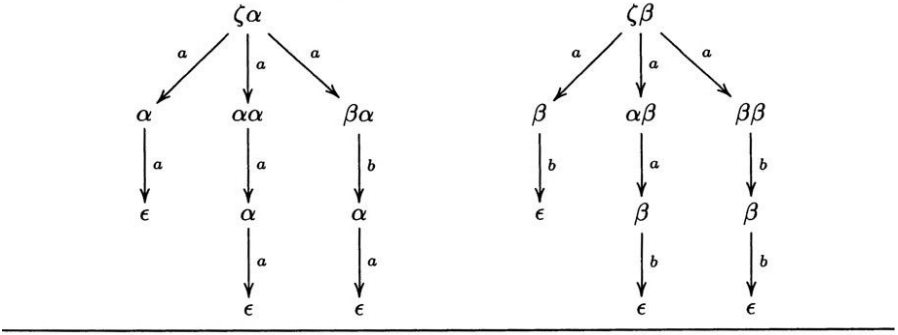

Fig. 5.3. The transition graphs for $\zeta\alpha$ and $\zeta\beta$.

We close this section by extending the notion of separability to relations.

Definition 5.3.5. *Let R be a binary relation over processes. Then*

$$\mathit{Sep}(R) =_{\mathrm{df}} \inf\{\, \mathit{Sep}(\alpha,\beta) \mid (\alpha,\beta) \in R \,\}.$$

Since we will be interested particularly in the least congruence of a given relation R the following lemma is important.

Lemma 5.3.6. *Let R be a binary relation on V^*. Then we have, for all $\alpha \leftrightarrow_R^* \alpha', \beta \leftrightarrow_R^* \beta'$,*

$$\mathit{Sep}(R \cup \{\, (\alpha,\beta) \,\}) \leq \mathit{Sep}(\alpha',\beta')$$

Proof. The one-step rewriting $\rightarrow_R$ can be extended to $(V^*)^2 \times (V^*)^2$ by

$$(\alpha,\beta) \underset{R}{\rightarrow} (\alpha',\beta') \quad \text{if} \quad (\alpha \underset{R}{\rightarrow} \alpha' \wedge \beta = \beta') \text{ or } (\alpha = \alpha' \wedge \beta \underset{R}{\rightarrow} \beta')$$

which allows us to define $(\alpha,\beta) \leftrightarrow_R^i (\alpha',\beta')$ in the usual fashion. The lemma is then shown by induction on i. For $i = 0$ simply observe that

$$\mathrm{Sep}(R \cup \{\, (\alpha,\beta) \,\}) = \min\{\, \mathrm{Sep}(R), \mathrm{Sep}(\alpha,\beta) \,\} \leq \mathrm{Sep}(\alpha,\beta).$$

Now assume $(\alpha,\beta) \leftrightarrow_R^i (\alpha'',\beta'') \leftrightarrow_R (\alpha',\beta')$ such that wlog. $\alpha'' \leftrightarrow_R \alpha'$ and $\beta'' = \beta'$. Then by induction hypothesis we have

$$\mathrm{Sep}(R \cup \{\, (\alpha,\beta) \,\}) \leq \mathrm{Sep}(\alpha'',\beta'') = \mathrm{Sep}(\alpha'',\beta')$$

and by definition

$$\mathrm{Sep}(R) \leq \mathrm{Sep}(\alpha'',\alpha').$$

Summarising we obtain by means of Lemma 5.3.5(2.)

$$\mathrm{Sep}(R \cup \{\, (\alpha,\beta) \,\}) \leq \min\{\, \mathrm{Sep}(\alpha',\alpha''), \mathrm{Sep}(\alpha'',\beta') \,\} \leq \mathrm{Sep}(\alpha',\beta').$$

□

5.4 Deciding Bisimilarity of Normed BPA

In this section we present a branching algorithm for deciding bisimilarity of normed context-free processes. Thereby we improve on a similar tableau system presented in [HS91] in two respects: first we introduce an explicit `SPLIT` rule which replaces the "eliminating subtableau" construction and secondly, we use a depth-first, left-to-right derivation strategy together with variable reduction. Both techniques together reduce the exponentiality of the size of a tableau. The aim of the branching algorithm is not to improve on the polynomial time-complexity of [HJM94], but to use it for giving a bound on the number of transitions needed to separate two non-bisimilar normed BPA processes. This bound was formerly shown for the restricted case of simple grammars in [Cau89].

Since our branching algorithm is tree-oriented, we start by presenting the necessary notations. Given a set M, an *M-valued tree* T is a map from the set of nodes $N(T) \subseteq \mathbb{N}_{>0}^*$ to M such that $N(T)$ is prefix closed and satisfies

$$uj \in N(T) \Rightarrow \forall\, i < j.\ ui \in N(T).$$

The branching algorithm will construct a tree where each node is labelled with an equation $\alpha = \beta$ for some $\alpha, \beta \in V^*$. Starting with the root each node is subsequently expanded in lexical order by using a *rule* which has the form

$$\frac{\alpha = \beta}{\alpha_1 = \beta_1 \qquad \ldots \qquad \alpha_m = \beta_m} \quad C$$

with side conditions C restricting the applicability. Intuitively, this rule expresses that the goal $\alpha \sim \beta$, the node under consideration, can be proved by verifying the subgoals $\alpha_1 = \beta_1 \quad \ldots \quad \alpha_m = \beta_m$ constituting sons in the tree. During the construction of the tree we will extract a *fundamental* relation which is defined as follows.

Definition 5.4.1. *Let R be a binary relation over V^*.*

– *The* domain *of R is the set*

$$dom(R) =_{\text{df}} \{\, \alpha \mid \alpha R \beta \,\},$$

while the image *of R is the set*

$$im(R) =_{\text{df}} \{\, \beta \mid \alpha R \beta \,\}.$$

– *R is called* fundamental*, if it satisfies the following three conditions:*

1. *$dom(R) \subseteq V$ and $im(R) \subseteq (V \setminus dom(R))^*$,*
2. *R is* functional*: if $XR\alpha$ and $XR\beta$ then $\alpha = \beta$, and*
3. *R is* norm-preserving*: if $XR\alpha$ then $||X|| = ||\alpha||$.*

It is easy to see that any fundamental relation R is confluent and noetherian. Thus any α possesses a unique normal form $\alpha \downarrow R$. Moreover, if *card* denotes the cardinality of either sets or relations, we have $card(R) < card(V)$

due to property (1) and R being functional. This puts us in a position to give the details of our branching algorithm.

Let BA be the recursive procedure which takes as input parameter a tuple (T, u, R) where T is a tree, u is a leaf node of the tree and R a fundamental relation, and returns either "failure" or a triple (T', u', R') of the same type as (T, u, R). Abbreviating $\{1, \ldots, n\}$ by $[n]$, the procedure $\mathtt{BA}(T, u, R)$ is defined by means of the rules listed in Table 5.1 and the instructions given in Table 5.2.

REDUCE $\dfrac{\alpha = \beta}{\alpha \downarrow R = \beta \downarrow R}$	where $\alpha \neq \alpha \downarrow R$ or $\beta \neq \beta \downarrow R$.
SWAP $\dfrac{X_j\beta = X_i\alpha}{X_i\alpha = X_j\beta}$	where $X_j\beta, X_i\alpha$ are in normal form, and $j > i$.
LCANCEL $\dfrac{X_i\alpha = X_i\beta}{\alpha = \gamma\beta}$	where $X_i\alpha, X_i\beta$ are in normal form, $\|\|X_i\alpha\|\| = \|\|X_i\beta\|\|$, and $\exists\ w.\ \|w\| = \|\|X_i\|\|,\ X_i \stackrel{w}{\rightarrow} \epsilon$ and $X_i \stackrel{w}{\rightarrow} \gamma$.
SPLIT $\dfrac{X_i\alpha = X_j\beta}{X_i\gamma = X_j \qquad \alpha = \gamma\beta}$	where $X_i\alpha, X_j\beta$ are in normal form, $i < j, \|\|X_i\alpha\|\| = \|\|X_j\beta\|\|$, $\alpha, \beta \neq \epsilon$, and $\exists\ w.\ \|w\| = \|\|X_i\|\|, X_i \stackrel{w}{\rightarrow} \epsilon$ and $X_j \stackrel{w}{\rightarrow} \gamma$.
UNFOLD $\dfrac{X_i\alpha = X_j}{\{\, \alpha_i\alpha = \beta_{f(i)} \,\}_{i=1}^{k} \ \{\, \alpha_{g(j)}\alpha = \beta_j \,\}_{j=1}^{l}}$	where $X_i\alpha, X_j$ are in normal form, $i < j, \|\|X_i\alpha\|\| = \|\|X_j\|\|, X_i \sim_1 X_j$, $X_i =_{\mathrm{df}} \sum_{i=1}^{k} a_i\alpha_i, X_j =_{\mathrm{df}} \sum_{j=1}^{l} b_j\beta_j$, $f : [k] \longrightarrow [l],\ \forall\, i \in [k].\ a_i = b_{f(i)}$, $g : [l] \longrightarrow [k],\ \forall\, j \in [l].\ a_{g(j)} = b_j$.

Table 5.1. The Expansion Rules of the Branching Algorithm.

There are two points worth mentioning here. First, even though there is at most one rule applicable during the execution of the procedure the algorithm is still *nondeterministic*, since we have, for example, to guess the right functions f and g for every application of the UNFOLD rule. Thus there may be many execution sequences of BA, which we call *runs* of BA in the sequel.

Let $T(u)$ be labelled by $\alpha = \beta$.

1. If no rule is applicable then
   - If $\alpha, \beta = \epsilon$
     then if u has a successor v in the lexical order
     - then continue with $\texttt{BA}(T, v, R)$
     - else stop successfully with (T, u, R).
   - Otherwise stop with "failure".
2. Else apply the unique applicable rule r to u. Let $e_1, \ldots, e_m$ be the consequents of the rule application, $T' = T \cup \{ (ui, e_i) \mid 1 \leq i \leq m \}$ and $u' = u1$. If r is the UNFOLD rule
   - then continue with $\texttt{BA}(T', u', \{ (X, \beta \downarrow \{ (X_j, X_i\alpha) \}) \mid XR\beta \} \cup \{ (X_j, X_i\alpha) \})$
   - else continue with $\texttt{BA}(T', u', R)$.

Table 5.2. The Instructions of the Branching Algorithm.

Secondly, we have used a less restrictive LCANCEL rule compared to the usual definition:

$$\texttt{LCANCEL}' \quad \frac{X_i\alpha = X_i\beta}{\alpha = \beta} \qquad \text{where } X_i\alpha, X_i\beta \text{ are in normal form, and } ||X_i\alpha|| = ||X_i\beta||$$

since in our version we do not require that γ must equal ϵ. The reason for this relaxation is that we want later on to bound the separability of $X_i\alpha$ and $X_i\beta$ as formally stated in Lemma 5.5.2.

To determine now whether $\alpha \sim \beta$, we start the branching algorithm with

$$\texttt{BA}(\{ (\epsilon, \alpha = \beta) \}, \epsilon, \emptyset),$$

i.e. we start at the root of the tree T which has only a single node $\alpha = \beta$ at the root, and our fundamental relation R is initially empty. Before proving the soundness, completeness and termination of the algorithm, we summarise some basic facts about rule applications. Let henceforth $||\alpha, \beta||$ abbreviate $\min\{ ||\alpha||, ||\beta|| \}$.

Lemma 5.4.1. *Concerning the minimal norm of left and right-hand sides of equations occurring as premises in rule applications, the* REDUCE *and the* SWAP *rule are norm-preserving, while the* LCANCEL *and the* SPLIT *rule are strictly norm-reducing, i.e.*

REDUCE:	$\|\|\alpha, \beta\|\|$	$=$	$\|\|\alpha \downarrow R, \beta \downarrow R\|\|$
SWAP:	$\|\|X_j\beta, X_i\alpha\|\|$	$=$	$\|\|X_i\alpha, X_j\beta\|\|$
LCANCEL:	$\|\|X_i\alpha, X_i\beta\|\|$	$>$	$\|\|\alpha, \gamma\beta\|\|$
SPLIT:	$\|\|X_i\alpha, X_j\beta\|\|$	$>$	$\max\{ \|\|X_i\gamma, X_j\|\|, \|\|\alpha, \gamma\beta\|\| \}$

Proof. The equality for the REDUCE rule follows from the norm-preserving property of R, while the other equations are a direct consequence of the definitions. □

Proposition 5.4.1. *Each run for $\alpha = \beta$ is finite.*

Proof. Consider the sequence $(T_i, u_i, R_i)_{i \geq 0}$ of successive call parameters of BA with $T_0 = \{ (\epsilon, (\alpha, \beta)) \}, u_0 = \epsilon$ and $R_0 = \emptyset$. By induction on i we show that each R_i is a fundamental relation. For the base case observe that $R_0 = \emptyset$ is clearly fundamental. Now let (T_k, u_k, R_k) be the current call parameter of BA where we assume that R_k is fundamental. Suppose the UNFOLD rule is applied to u_k and thus

$$R_{k+1} = \{ (X, \beta \downarrow \{ (X_j, X_i\alpha) \}) \mid X R_k \beta \} \cup \{ (X_j, X_i\alpha) \},$$

since otherwise $R_{k+1} = R_k$ is obviously fundamental. First we deduce that

$$X_i\alpha \in (V \setminus dom(R_k))^*$$

from $X_i\alpha \downarrow R_k = X_i\alpha$, and that X_j does not occur in $X_i\alpha$ from $||X_i\alpha|| = ||X_j||$, respectively. Overall, this yields

$$X_i\alpha \in (V \setminus dom(R_{k+1}))^*.$$

Hence, by induction hypothesis we have

$$\begin{array}{rcl} dom(R_{k+1}) & = & dom(R_k) \cup \{ X_j \} \subseteq V, \text{ and} \\ im(R_{k+1}) & = & \{ \beta \downarrow \{ (X_j, X_i\alpha) \} \mid X R_k \beta \} \cup \{ X_i\alpha \} \\ & \subseteq & (V \setminus dom(R_{k+1}))^* \end{array}$$

Second, R_{k+1} is functional since $X_j \notin dom(R_k)$ due to $X_j \downarrow R_k = X_j$, and as by induction hypothesis R_k is functional. Third, the precondition of the UNFOLD rule assures that $||X_i\alpha|| = ||X_j||$. Together with the norm-preserving property of R_k following from induction hypothesis this, finally, yields that also R_{k+1} is norm-preserving. Summarising, we see that each R_k occurring during a run of BA is fundamental.

Now let T be the tree constructed by a run of BA. Then the sequence of rules applied to the nodes occurring on a path in the tree are a word of the regular language represented by the BNF-expression:

```
(  [REDUCE] LCANCEL |
   [REDUCE] [SWAP] (SPLIT | UNFOLD)  )*
```

Since each R_i is fundamental, we have at most $card(V) - 1$ many UNFOLD applications. Considering $||\alpha, \beta||$ for each node label $\alpha = \beta$, we deduce from Lemma 5.4.1 that every path in T has finite length. Since also every rule application has finitely many consequents only, we obtain by König's Lemma that T must be finite. □

It is also possible to show the slightly stronger result that the length of each path is not only finite, but also bounded. This implies that the number of runs for $\alpha = \beta$ is also finite which is needed for the decidability of the bisimulation equivalence problem by means of BA. However, since we will only concentrate on developing a bound for separability it is sufficient for this purpose to show the soundness and completeness of the branching algorithm.

Proposition 5.4.2. *The branching algorithm is sound and complete, i.e. we have $\alpha \sim \beta$ iff there exists a successful run for $\alpha = \beta$.*

Proof. Assume $\alpha \sim \beta$. Then we build a tree for $\alpha = \beta$ in such a way that for each node labelled with $\zeta = \eta$ we have $\zeta \sim \eta$. It is important to note that we can always apply one of the rules to such a node whenever $\zeta \sim \eta$.

If we have now to apply the LCANCEL rule to a node of the form $X_i\alpha = X_i\beta$, let $\gamma = \epsilon$ and thus $X_i \xrightarrow{w} \beta$. From Lemma 2.6.1(1.) it easily follows that the consequents resulting from such an application are bisimilar if the antecedent is. Obviously the same holds for any application of the SWAP rule. If we have to apply the SPLIT rule choose γ as given by the splitting Lemma 2.6.2, and in the case of an UNFOLD application simply choose the matching transitions which must exist due to the definition of bisimulation. This way only bisimilar pairs are added to R during an UNFOLD application. Then the congruence property of bisimulation wrt. sequential composition yields the correctness of the REDUCE rule. By Proposition 5.4.1 this construction must terminate, and from instruction (1) of the branching algorithm we see that all resulting terminals are of the form $\epsilon = \epsilon$. Hence, the branching algorithm is complete.

Let now (T, u, R) be the value returned by BA. Then we prove soundness by showing that

$$S =_{\mathrm{df}} \{\, (\alpha, \beta) \mid \alpha = \beta \text{ is a node labelling in the tree } T \,\}$$

is a self-bisimulation, i.e. that $\alpha \cong_S \beta$ whenever $(\alpha, \beta) \in S$.

First, we deal with the easiest case, i.e. the UNFOLD rule is applied to $\zeta = \eta$. This guarantees that for each $\zeta \xrightarrow{a} \zeta'$ we have $\eta \xrightarrow{a} \eta'$ for some η' such that $(\zeta', \eta') \in S$ and symmetrically, for each $\eta \xrightarrow{a} \eta'$ we have $\zeta \xrightarrow{a} \zeta'$ for some ζ' such that $(\zeta', \eta') \in S$, as desired.

The remaining cases are now proved by induction on the depth of subtrees. For leaves ($\epsilon = \epsilon$) nothing needs to be shown. Thus assume $\zeta = \eta$ is an inner node.

If $\zeta = \eta$ is the premise of a SWAP application the self-bisimulation condition follows by induction hypothesis from the self-bisimulation property of (η, ζ).

Second, if the SPLIT rule is applied to $\zeta = \eta$, ζ must be of the form $X_i\alpha$, while η must have the form $X_j\beta$. Then let $X_i\gamma = X_j$ and $\alpha = \gamma\beta$ be the consequents. Now consider $X_i\alpha \xrightarrow{a} \alpha'\alpha$ which implies $X_i\gamma \xrightarrow{a} \alpha'\gamma$. Then we know that $X_j \xrightarrow{a} \beta'$ for some β' such that by induction hypothesis

$\alpha'\gamma \leftrightarrow_S^* \beta'$. This yields $X_j\beta \xrightarrow{a} \beta'\beta$, and observing that $(\alpha, \gamma\beta) \in S$ we obtain $\alpha'\alpha \leftrightarrow_S \alpha'\gamma\beta \leftrightarrow_S^* \beta'\beta$. The symmetric case is shown in a similar way.

Third, since T is the result of a successful run of `BA`, in case of an `LCANCEL` application to $X_i\alpha = X_i\beta$, it is easy to prove by considering the norm that the consequent must be $\alpha = \beta$. Thus by induction hypothesis we know $\alpha \leftrightarrow_S^* \beta$, and the self-bisimulation condition easily follows from $\gamma\alpha \leftrightarrow_S^* \gamma\beta$ for all γ with $X_i \xrightarrow{a} \gamma$.

Fourth, in the remaining case of a `REDUCE` application to $X_i\alpha = X_j\beta$, the self-bisimulation condition is a consequence of the induction hypothesis $X_i\alpha \downarrow R \cong_S X_j\beta \downarrow R$ and the following claim, as by Lemma 2.6.5 $\cong_S$ is a transitive relation and we therefore have

$$X_i\alpha \;\cong_S\; X_i\alpha \downarrow R \;\cong_S\; X_j\beta \downarrow R \;\cong_S\; X_j\beta.$$

Now let

$$U =_{\text{df}} \{\, (X_j, X_i\alpha) \mid X_i\alpha = X_j \;\; \text{is an } \texttt{UNFOLD} \text{ node labelling in the tree } T \,\}$$

Claim : For any $\alpha, \beta \in V^*$ we have that

$$\alpha \xrightarrow{*}_U \beta \quad \text{implies} \quad \alpha \cong_S \beta.$$

Proof. The claim is shown by induction on the number of rewrite steps from α to β. For $n = 0$ we have $\alpha = \beta$ and there is nothing to show. Now let the claim hold for n rewrite steps, and consider the $n + 1$ derivation

$$\alpha = \alpha_1 X \alpha_2 \to_U \alpha_1 \eta \alpha_2 \xrightarrow{n}_U \beta$$

When $\alpha_1 = \epsilon$ we use the fact that $X \cong_S \eta$ since $\eta = X$ is the labelling of an `UNFOLD` node in the tree T to obtain by induction hypothesis

$$\alpha = X\alpha_2 \cong_S \eta\alpha_2 \cong_S \beta$$

If, however, $\alpha_1 \neq \epsilon$ let $\alpha_1 X \alpha_2 \xrightarrow{a} \zeta X \alpha_2$ for some ζ. But then we have similarly $\alpha_1\eta\alpha_2 \xrightarrow{a} \zeta\eta\alpha_2$ and, moreover, $\zeta X\alpha_2 \leftrightarrow_S^* \zeta\eta\alpha_2$ since $U \subseteq S$. As the case $\alpha_1\eta\alpha_2 \xrightarrow{a} \zeta\eta\alpha_2$ for some ζ is shown analogously we conclude again by induction hypothesis

$$\alpha = \alpha_1 X \alpha_2 \cong_S \alpha_1 \eta \alpha_2 \cong_S \beta$$

□

Since each rewrite step $\alpha \to_R \beta$ corresponds by construction of R to some rewrite sequence $\alpha \to_U^* \beta$ we may thus conclude $X_i\alpha \cong_S X_i\alpha \downarrow R$, for any process $X_i\alpha \in V^+$.
Overall, this shows that the branching algorithm is sound and complete. □

5.5 A Bound for Separability

In this section we develop a bound for the separability of two non-bisimilar normed BPA processes from the branching algorithm presented in the previous section. To start with we give a bound for the separability of two processes occurring in a node which cannot be further expanded.

Lemma 5.5.1. *If no rule is applicable to an equation $X_i\alpha = X_j\beta$, we have*

$$Sep(X_i\alpha, X_j\beta) \leq ||X_i\alpha, X_j\beta|| + 1.$$

Proof. Since none of the rules `REDUCE` and `SWAP` is applicable, we conclude that $X_i\alpha$ as well as $X_j\beta$ are in normal form, and that $i \leq j$. Thus either the left and right-hand side of the equation have different norms and the inequality follows from Lemma 5.3.3, or we have $||X_i\alpha|| = ||X_j\beta||$. As `LCANCEL` is also not applicable we deduce $i < j$, and must further distinguish between $\min\{\,|X_i\alpha|, |X_j\beta|\,\} = 1$ and $\min\{\,|X_i\alpha|, |X_j\beta|\,\} > 1$. The non-applicability of the `UNFOLD` rule guarantees in the first case $\mathrm{Sep}(X_i\alpha, X_j\beta) = 1$, while the non-applicability of the `SPLIT` rule implies in the latter case

$$\mathrm{Sep}(X_i\alpha, X_j\beta) \leq ||X_i|| + 1 \leq ||X_i\alpha|| + 1.$$

□

Now we successively relate the separabilities of premise and consequents for `SWAP`, `LCANCEL`, and `SPLIT` rule applications. As the result we will obtain that if $\alpha = \beta$ is the premise there exists some rule application such that for any consequent $\zeta = \eta$ the following holds.

$$\mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(\zeta, \eta) + ||\alpha, \beta|| - ||\zeta, \eta|| \tag{5.2}$$

It is worth to remark that the restriction that only for *some* rule application the inequality is guaranteed to hold is due to the nondeterminism of context-free processes. In contrast, for the deterministic variant of simple grammars it is known that any rule application of `LCANCEL` and `SPLIT` satisfies Inequality 5.2.

Remark 5.5.1. For any `SWAP` application to $X_j\beta = X_i\alpha$ we obviously have:

$$\mathrm{Sep}(X_j\beta, X_i\alpha) \leq \mathrm{Sep}(X_i\alpha, X_j\beta) + ||X_j\beta, X_i\alpha|| - ||X_i\alpha, X_j\beta||.$$

Lemma 5.5.2. *If the `LCANCEL` rule is applicable to $X_i\alpha = X_i\beta$ then there exists an application such that for the consequent $\alpha = \gamma\beta$ we have:*

$$Sep(X_i\alpha, X_i\beta) \leq Sep(\alpha, \gamma\beta) + ||X_i\alpha, X_i\beta|| - ||\alpha, \gamma\beta||$$

Proof. Let $X_i\alpha \xrightarrow{w} \alpha$ with $|w| = ||X_i||$. Due to Lemma 5.3.4 there exists some γ with $X_i\beta \xrightarrow{w} \gamma\beta$ and

$$\mathrm{Sep}(X_i\alpha, X_i\beta) \leq |w| + \mathrm{Sep}(\alpha, \gamma\beta).$$

Thus we obtain:

$$\begin{aligned} & \mathrm{Sep}(X_i\alpha, X_i\beta) \\ \leq\ & |w| + \mathrm{Sep}(\alpha, \gamma\beta) \\ =\ & \mathrm{Sep}(\alpha, \gamma\beta) + ||X_i|| + ||\alpha, \beta|| - ||\alpha, \beta|| \\ =\ & \mathrm{Sep}(\alpha, \gamma\beta) + ||X_i\alpha, X_i\beta|| - ||\alpha, \gamma\beta|| \end{aligned}$$

□

Lemma 5.5.3. *If the* `SPLIT` *rule is applicable to $X_i\alpha = X_j\beta$ then there exists an application such that for every consequent $\zeta = \eta$ we have:*

$$\mathit{Sep}(X_i\alpha, X_j\beta) \leq \mathit{Sep}(\zeta, \eta) + ||X_i\alpha, X_j\beta|| - ||\zeta, \eta||.$$

Proof. By precondition of the `SPLIT` rule we know $||X_i\alpha|| = ||X_j\beta||$, and that $X_i \xrightarrow{w} \epsilon$ is a norm-reducing transition sequence. Since $X_j \xrightarrow{w} \gamma$ does not need to be also norm-reducing, we obtain for every application of the `SPLIT` rule

$$\begin{aligned} &||\alpha|| = ||X_i\alpha|| - ||X_i|| = ||X_j\beta|| - |w| \leq ||\gamma\beta||, \text{ and} \\ &||X_j|| = ||X_j\beta|| - ||\beta|| \leq ||X_i\alpha|| + ||\gamma|| - ||\alpha|| = ||X_i\gamma||, \end{aligned}$$

from which the following two equalities can be deduced.

$$||X_i\alpha, X_j\beta|| = ||X_i\alpha|| = ||X_i|| + ||\alpha|| = ||X_i|| + ||\alpha, \gamma\beta||, \tag{5.3}$$

and

$$||X_i\alpha, X_j\beta|| = ||X_j\beta|| = ||X_j|| + ||\beta|| = ||\beta|| + ||X_i\gamma, X_j|| \tag{5.4}$$

In case that now $|w| + 1 \leq \mathrm{Sep}(X_i\alpha, X_j\beta)$ we apply Lemma 5.3.4. Observing that $X_i\alpha \xrightarrow{w} \alpha$ is also a norm-reducing transition sequence we obtain that there exists some γ such that $X_j\beta \xrightarrow{w} \gamma\beta$ and

$$\mathrm{Sep}(X_i\alpha, X_j\beta) \leq |w| + \mathrm{Sep}(\alpha, \gamma\beta) = ||X_i|| + \mathrm{Sep}(\alpha, \gamma\beta). \tag{5.5}$$

If on the other side $|w| + 1 > \mathrm{Sep}(X_i\alpha, X_j\beta)$ then equation (5.5) follows trivially.
Fixing this γ, the lemma follows now for the consequent $\alpha = \gamma\beta$ from

$$\begin{aligned} & \mathrm{Sep}(X_i\alpha, X_j\beta) \\ \leq\ & ||X_i|| + \mathrm{Sep}(\alpha, \gamma\beta) && \text{[by (5.5)]} \\ =\ & \mathrm{Sep}(\alpha, \gamma\beta) + ||X_i\alpha, X_j\beta|| - ||\alpha, \gamma\beta|| && \text{[by (5.3)]} \end{aligned}$$

while for the consequent $X_i\gamma = X_j$ it follows from the inequality

$$\begin{aligned} & \mathrm{Sep}(X_i\alpha, X_j\beta) \\ \leq\ & ||X_i|| + \mathrm{Sep}(\alpha, \gamma\beta) && \text{[by (5.5)]} \\ \leq\ & \mathrm{Sep}(X_i\alpha, X_i\gamma\beta) && \text{[by Lemma 5.3.5(1)]} \end{aligned} \tag{5.6}$$

by means of

$$\begin{aligned}
& \mathrm{Sep}(X_i\alpha, X_j\beta) \\
= \;& \min\{\, \mathrm{Sep}(X_i\gamma\beta, X_i\alpha), \mathrm{Sep}(X_i\alpha, X_j\beta) \,\} && \text{[by (5.6)]} \\
\leq \;& \mathrm{Sep}(X_i\gamma\beta, X_j\beta) && \text{[by Lemma 5.3.5(2)]} \\
\leq \;& \mathrm{Sep}(X_i\gamma, X_j) + ||\beta|| && \text{[by Lemma 5.3.5(1)]} \\
= \;& \mathrm{Sep}(X_i\gamma, X_j) + ||X_i\alpha, X_j\beta|| - ||X_i\gamma, X_j|| && \text{[by (5.4)]}
\end{aligned}$$

□

Now let $\alpha \not\sim \beta$. Although by soundness of the branching algorithm *every* run of `BA` for $\alpha = \beta$ is unsuccessful, and therefore also *every* tree constructed during a run, we are merely interested in the particular tree T with root $\alpha = \beta$ which satisfies the following two assumptions:

U-assumption: every consequent $\zeta = \eta$ of an `UNFOLD` application to $X_i\alpha = X_j$ in T satisfies $\mathrm{Sep}(X_i\alpha, X_j) \leq 1{+}\mathrm{Sep}(\zeta, \eta)$, and

SLS-assumption: every `SWAP`, `LCANCEL`, and `SPLIT` application in T satisfies the Inequality 5.2.

While the first assumption can be guaranteed by Lemma 5.3.4, the existence of appropriate `LCANCEL`, and `SPLIT` applications is stated in the related Lemmata 5.5.2, and 5.5.3, respectively. Finally, Remark 5.5.1 deals with `SWAP` applications.

Henceforth, we order the nodes of T by the usual lexical ordering $\leq_{lex}$ (depth-first, left-to-right traversal of the tree), split $N(T)$ into the set of nodes $N_U(T)$ whose labels have been expanded by the `UNFOLD` rule, and abbreviate $N(T) \setminus N_U(T)$ by $N_O(T)$. To shorten notation we omit references to T in the sequel. Moreover, we order $N_U = \{\, u_1, \ldots, u_p \,\}$ with $p < n = card(V)$ such that $u_i \leq_{lex} u_{i+1}$. For any $u \in N$, let now

$$L_u \;=_{\mathrm{df}}\; \{\, n \in N_U \mid n \leq_{lex} u \,\}, \text{ and}$$

$$D_u \;=_{\mathrm{df}}\; \{\, uv \in N \mid \forall w \neq \epsilon.\ w \text{ is a proper prefix of } v, uw \notin N_U \,\}$$

Intuitively, D_u denotes the subtree of T rooted at u where subtrees below nodes $uv \in N_U, v \neq \epsilon$ are cut off. In particular, this means that if $\epsilon \in N_U$ the set of tree nodes can be decomposed as

$$N = \bigcup \{\, D_{u_i} \setminus \{\, u_i \,\} \mid i \in [p] \,\} \cup \{\, \epsilon \,\}$$

Writing

$$\mathcal{M} =_{\mathrm{df}} \max\{\, ||\alpha_i|| \mid X =_{\mathrm{df}} \sum_{i=1}^{m} a_i\alpha_i \in \mathcal{E} \,\}$$

for the greatest norm of a variable sequence occurring in a right-hand side summand, the next two lemmata state properties about the separability of processes occurring as labels of nodes contained in D_u.

Lemma 5.5.4. $\forall u \in N_O,\ \forall v \in D_u$

- $||T(v)|| \leq ||T(u)||$
- $Sep(T(L_u) \cup \{\, T(u) \,\}) \leq Sep(T(v)) + ||T(u)|| - ||T(v)||$

Proof. Both parts will be shown by induction on $|v|-|u|$. If $|v| = |u|$ we simply conclude $u = v$ and the lemma is trivially true. Now let for the induction step

$$u \to v_1 \to \ldots \to v_n = v$$

be a path in the tree. Then the first part follows from the induction hypothesis and Lemma 5.4.1:

$$||T(u)|| \geq ||T(v_{n-1})|| \geq ||T(v_n)||.$$

For the second part we distinguish whether v_{n-1} was expanded by the `REDUCE` rule or one of the rules `SWAP`, `LCANCEL` or `SPLIT`. In the latter case we conclude as follows:

$$\begin{array}{ll} & \mathrm{Sep}(T(L_u) \cup \{\, T(u) \,\}) \\ \leq & \text{[by induction hypothesis]} \\ & \mathrm{Sep}(T(v_{n-1})) + ||T(u)|| - ||T(v_{n-1})|| \\ \leq & \text{[by SLS-assumption]} \\ & \mathrm{Sep}(T(v_n)) + ||T(v_{n-1})|| - ||T(v_n)|| + ||T(u)|| - ||T(v_{n-1})|| \\ = & \mathrm{Sep}(T(v)) + ||T(u)|| - ||T(v)||. \end{array}$$

If, however, v_{n-1} was expanded by the `REDUCE` rule, observe that $||T(v_{n-1})|| = ||T(v_n)||$ since reduction with respect to R preserves the norm. We distinguish two cases:

Case 1:: $\min\{\, \mathrm{Sep}(T(L_u)), \mathrm{Sep}(T(v_{n-1})) \,\} = \mathrm{Sep}(T(v_{n-1}))$

From Lemma 5.3.6, we obtain

$$\mathrm{Sep}(T(v_{n-1})) = \mathrm{Sep}(T(L_u) \cup \{\, T(v_{n-1}) \,\}) \leq \mathrm{Sep}(T(v_n)),$$

while Lemma 5.4.1 yields $||T(v_{n-1})|| = ||T(v_n)||$. Thus we can conclude as above.

Case 2:: $\min\{\, \mathrm{Sep}(T(L_u)), \mathrm{Sep}(T(v_{n-1})) \,\} = \mathrm{Sep}(T(L_u))$

In this case we have

$$
\begin{array}{lll}
 & \mathrm{Sep}(T(L_u) \cup \{\, T(u) \,\}) & \\
= & \min\{\, \mathrm{Sep}(T(L_u)), \mathrm{Sep}(T(u)) \,\} & \text{[by definition]} \\
\leq & \mathrm{Sep}(T(L_u)) & \\
= & \mathrm{Sep}(T(L_u) \cup \{\, T(v_{n-1}) \,\}) & \text{[by assumption]} \\
\leq & \mathrm{Sep}(T(v_n)) & \text{[by Lemma 5.3.6]} \\
\leq & \mathrm{Sep}(T(v_n)) + ||T(u)|| - ||T(v_n)|| & \text{[by the first part]} \\
= & \mathrm{Sep}(T(v)) + ||T(u)|| - ||T(v)|| &
\end{array}
$$

□

Lemma 5.5.5. $\forall u \in N_U,\ \forall v \in D_u \setminus \{\, u \,\}$

– $||T(v)|| \leq \mathcal{M}$

– $\mathit{Sep}(T(L_u)) \leq \mathit{Sep}(T(v)) + \mathcal{M} + 1 - ||T(v)||$

Proof. Let u be labelled by $X_i\alpha = X_j$, and

$$u \overset{\texttt{UNFOLD}}{\rightarrow} v_1 \rightarrow \ldots \rightarrow v_n = v$$

be a path in the tree such that $v_i \in D_u$, for all $1 \leq i \leq n$. Since the `UNFOLD` rule was applied to u we have $X_j \overset{a}{\rightarrow} \gamma$ implies $||\gamma|| \leq \mathcal{M}$, for any $a \in \mathit{Act}$, and hence we deduce $||T(v_1)|| \leq \mathcal{M}$. Moreover, we have by assumption about the `UNFOLD` rule

$$
\begin{array}{ll}
 & \mathrm{Sep}(T(L_u)) \\
= & \min\{\, \mathrm{Sep}(T(L_u \setminus \{\, u \,\})), \mathrm{Sep}(T(u)) \,\} \\
\leq & \mathrm{Sep}(T(u)) \\
\leq & 1 + \mathrm{Sep}(T(v_1)).
\end{array}
\tag{5.7}
$$

We proceed by case analysis.

Case 1: $v_1 \in N_U$

The definition of D_u implies that $v = v_1$. Thus $||T(v)|| \leq \mathcal{M}$, as well as

$$
\begin{array}{lll}
 & \mathrm{Sep}(T(L_u)) & \\
\leq & \mathrm{Sep}(T(v_1)) + 1 & \text{[by (5.7)]} \\
\leq & \mathrm{Sep}(T(v_1)) + 1 + \mathcal{M} - ||T(v_1)|| & \text{[by the first part]}
\end{array}
$$

Case 2: $v_1 \notin N_U$

In this case Lemma 5.5.4(1) yields: $||T(v)|| \leq ||T(v_1)|| \leq \mathcal{M}$. Moreover, we have $L_{v_1} = L_u$ by definition of D_u, and thus

$$
\begin{array}{ll}
& \mathrm{Sep}(T(L_u)) \\
\leq & \min\{\, \mathrm{Sep}(T(L_{v_1})), 1 + \mathrm{Sep}(T(v_1)) \,\} \qquad \text{[by (5.7)]} \\
\leq & \mathrm{Sep}(T(L_{v_1}) \cup \{\, T(v_1) \,\}) + 1 \\
\leq & \mathrm{Sep}(T(v)) + 1 + ||T(v_1)|| - ||T(v)|| \qquad \text{[by Lemma 5.5.4(2)]} \\
\leq & \mathrm{Sep}(T(v)) + 1 + \mathcal{M} - ||T(v)|| \qquad \text{[by the first part]}
\end{array}
$$

□

Lemma 5.5.6. *Whenever the root $\alpha = \beta$ of the tree is expanded by the* `UNFOLD` *rule, i.e. $T(\epsilon) \in N_U$, we have, for every $v \in N \setminus \{\, \epsilon \,\}$:*

$$\mathit{Sep}(\alpha, \beta) \leq \mathit{Sep}(T(v)) + (n-1)\mathcal{M} + 1 - ||T(v)||$$

Proof. Recall that the set of tree nodes can be decomposed as

$$N \setminus \{\, \epsilon \,\} = \bigcup \{\, D_{u_i} \setminus \{\, u_i \,\} \mid i \in [p] \,\}$$

The lemma is then proved by showing, for any $v \in D_{u_i} \setminus \{\, u_i \,\}$:

$$\mathrm{Sep}(\alpha, \beta) \leq \mathrm{Sep}(T(v)) + i\mathcal{M} + 1 - ||T(v)||$$

by induction on i. For $i = 1$ and $v \in D_{u_1} \setminus \{\, u_1 \,\}$ Lemma 5.5.5 yields

$$
\begin{aligned}
\mathrm{Sep}(\alpha, \beta) &= \mathrm{Sep}(T(u_1)) = \mathrm{Sep}(T(L_{u_1})) \\
&\leq \mathrm{Sep}(T(v)) + \mathcal{M} + 1 - ||T(v)||.
\end{aligned}
$$

In order to show the induction step let now $v \in D_{u_{i+1}} \setminus \{\, u_{i+1} \,\}$. Now either

$$
\begin{array}{ll}
& \mathrm{Sep}(T(u_1)) \\
= & \mathrm{Sep}(T(L_{u_{i+1}})) \\
\leq & \text{[by Lemma 5.5.5(2)]} \\
& \mathrm{Sep}(T(v)) + \mathcal{M} + 1 - ||T(v)|| \\
\leq & \mathrm{Sep}(T(v)) + (i+1)\mathcal{M} + 1 - ||T(v)||
\end{array}
$$

or $\mathrm{Sep}(T(L_{u_{i+1}})) = \mathrm{Sep}(T(u_j))$, for some $1 < j \leq i+1$, with $u_j \in D_{u_{j-1}}$ from which we, finally, conclude:

$$
\begin{array}{ll}
& \mathrm{Sep}(\alpha, \beta) \\
\leq & \text{[by induction hypothesis]} \\
& \mathrm{Sep}(T(u_j)) + (j-1)\mathcal{M} + 1 - ||T(u_j)|| \\
\leq & \text{[by Lemma 5.5.5(2)]} \\
& \mathrm{Sep}(T(v)) + \mathcal{M} + 1 - ||T(v)|| + (j-1)\mathcal{M} + 1 - ||T(u_j)|| \\
\leq & \mathrm{Sep}(T(v)) + j\mathcal{M} + 1 - ||T(v)|| \\
\leq & \mathrm{Sep}(T(v)) + (i+1)\mathcal{M} + 1 - ||T(v)||
\end{array}
$$

□

Finally, we obtain the main theorem of this section.

Theorem 5.5.1 (Bounded Separability Theorem).
Let $\alpha, \beta \in V^*$ be normed. If α is not bisimilar to β, then $Sep(\alpha, \beta) \leq \mathcal{B}_{\alpha,\beta}$, where

$$\mathcal{B}_{\alpha,\beta} = \begin{cases} ||\alpha, \beta|| + 1 & \text{if } ||\alpha|| \neq ||\beta||, \text{ and} \\ (n-1)\mathcal{M} + 1 + ||\alpha, \beta|| & \text{if } ||\alpha|| = ||\beta||. \end{cases}$$

Proof. Let $\alpha \not\sim \beta$, and T be an unsuccessful tree satisfying the U- and SLS-assumption. If $||\alpha|| \neq ||\beta||$, we conclude by Lemma 5.3.3 that

$$\mathrm{Sep}(\alpha, \beta) \leq ||\alpha, \beta|| + 1 = \mathcal{B}_{\alpha,\beta}.$$

Now suppose $||\alpha|| = ||\beta||$. The theorem is then proved by induction on $||\alpha, \beta||$. For the base case assume $||\alpha, \beta|| = 1$, and let z be the last node of the tree construction which could not be further expanded. By Lemma 5.5.1 we then know

$$\mathrm{Sep}(T(z)) \leq ||T(z)|| + 1. \tag{5.8}$$

If $z = \epsilon$, we immediately get

$$\mathrm{Sep}(\alpha, \beta) \leq 1 + ||\alpha, \beta|| \leq \mathcal{B}_{\alpha,\beta}.$$

Otherwise we have to distinguish which rule has been applied to $\alpha = \beta$. It is easy to see that the `REDUCE` rule is not applicable to $\alpha = \beta$ due to $R_0 = \emptyset$. The same applies for the `LCANCEL` rule, since $||\alpha, \beta|| = 1$ would imply that α, β are identical to some X_i yielding a contradiction to the assumption $\alpha \not\sim \beta$. If the root was expanded by the `UNFOLD` rule, we obtain:

$$\begin{aligned} & \mathrm{Sep}(\alpha, \beta) \\ \leq\ & (n-1)\mathcal{M} + 1 + \mathrm{Sep}(T(z)) - ||T(z)|| && \text{[by Lemma 5.5.6]} \\ \leq\ & (n-1)\mathcal{M} + 1 + 1 && \text{[by (5.8)]} \\ =\ & (n-1)\mathcal{M} + 1 + ||\alpha, \beta|| && \text{[by assumption]} \\ =\ & \mathcal{B}_{\alpha,\beta} \end{aligned}$$

Moreover, in case of a `SWAP` rule application, we have $\mathrm{Sep}(\alpha, \beta) = \mathrm{Sep}(\beta, \alpha)$. Thus we can conclude for the consequent as in the former cases.

For the induction step assume $||\alpha, \beta|| = i + 1$, and that the theorem holds for all $\zeta \not\sim \eta$ with $||\zeta, \eta|| < i + 1$. Then the `REDUCE` rule is still not applicable to the root. In case of an `UNFOLD` expansion we apply Lemma 5.5.6 again, and for a `SWAP` rule application the arguments for the base case apply. Let now the root be of the form $X_i\alpha' = X_i\beta'$ such that the `LCANCEL` rule is applied to it yielding the consequent $\alpha' = \gamma\beta'$. Then

$$\begin{aligned}
& \mathrm{Sep}(\alpha,\beta) = \mathrm{Sep}(X_i\alpha', X_i\beta') \\
\leq\ & ||X_i|| + \mathrm{Sep}(\alpha',\gamma\beta') && \text{[by SLS-assumption]} \\
\leq\ & ||X_i|| + \mathcal{B}_{\alpha',\gamma\beta'} && \text{[by induction hypothesis]} \\
\leq\ & ||X_i|| + (n-1)\mathcal{M} + 1 + ||\alpha',\gamma\beta'|| \\
=\ & (n-1)\mathcal{M} + 1 + ||X_i\alpha', X_i\beta'|| \\
=\ & \mathcal{B}_{\alpha,\beta}.
\end{aligned}$$

Finally, we consider a SPLIT rule application to a root of the form $X_i\alpha' = X_j\beta'$. Since by Lemma 5.4.1 $||X_i\alpha', X_j\beta'|| > ||\zeta,\eta||$ holds for every consequent $\zeta = \eta$ we may apply the induction hypothesis to obtain

$$\begin{aligned}
& \mathrm{Sep}(\alpha,\beta) = \mathrm{Sep}(X_i\alpha', X_j\beta') \\
\leq\ & \mathrm{Sep}(\zeta,\eta) + ||X_i\alpha', X_j\beta'|| - ||\zeta,\eta|| && \text{[by SLS-assumption]} \\
\leq\ & \mathcal{B}_{\zeta,\eta} + ||X_i\alpha', X_j\beta'|| - ||\zeta,\eta|| && \text{[by induction hypothesis]} \\
\leq\ & \mathcal{B}_{\alpha,\beta}.
\end{aligned}$$

This completes the proof. □

To emphasise that the bound is valid for processes α, β defined with respect to a BPA system $\mathcal{C}$ we will write $\mathcal{B}^{\mathcal{C}}_{\alpha,\beta}$.

In the next section we will apply this theorem to a slightly larger class of processes, namely BPA_δ. The constant symbol δ represents *deadlock*, i.e. a process which cannot proceed. Its behaviour is captured by the axioms $E+\delta = E$ and $\delta E = \delta$. A BPA_δ system is *normed*, if each variable X is either normed in the usual sense or X deadlocks after finitely many steps. It is easy to extend the proofs given in this section in order to cope with normed BPA_δ systems, which allows us to apply the bound given above. This technical but straightforward variant is important for the proof of Proposition 5.6.2.

5.6 The Algorithm

In this section we present our three step algorithm for deciding bisimilarity of context-free processes. The first two steps are required for the construction of a *bisimulation base* B. Bisimulation bases characterise bisimulation equivalence as the least congruence w.r.t. sequential composition containing B. This can be exploited for a branching algorithm, which completes the decision procedure in a third step. We will concentrate on the first step here, as the second step can be obtained by combining results from [CHS92, HJM94], and the third step is rather straightforward.

We start by sketching the second step, which also serves as a good motivation for the subsequently presented first step of our algorithm. Section 5.6.3

then presents the branching algorithm for deciding bisimilarity, and finally, we summarise the results to obtain the complete decision procedure.

5.6.1 Bisimulation Bases

An important difference between the theory of normed and unnormed BPA processes is the existence of two kinds of bisimilar pairs, *decomposable* and *elementary* pairs (cf. [CHS92]).

Definition 5.6.1. *Let $X_i\alpha \sim X_j\beta$ with $i \leq j$ and $X_i, X_j \in V_N$. We say that the pair $(X_i\alpha, X_j\beta)$ is* decomposable *if there exists some γ such that $\alpha \sim \gamma\beta$ and $X_i\gamma \sim X_j$. If the pair is not decomposable it is said to be* elementary.

Observe that whenever $(X_i\alpha, X_j\beta)$ is decomposable γ must be normed due to $X_i\gamma \sim X_j$. Moreover, an immediate consequence of this definition is that if $(X_i\alpha, X_j\beta)$ is elementary, then α and β must be unnormed. Thus in the normed case only decomposable pairs can occur.

In order to prove termination of the branching algorithm presented in Section 5.6.3, we need to extend the notion of norm to a *seminorm* on arbitrary context-free processes as follows.

Definition 5.6.2. *We define the* seminorm *of a variable sequence $\alpha \in \mathcal{P}(V^*)$ as*

$$||\alpha X||_s =_{\mathrm{df}} \begin{cases} ||\alpha X|| & \text{if } X \in V_N \\ ||\alpha|| & \text{otherwise} \end{cases}$$

The seminorm is used to define a well-founded quasi-order on $\mathcal{P}(V^*) \times \mathcal{P}(V^*)$ by

$$(\alpha_1, \alpha_2) \sqsubseteq (\beta_1, \beta_2) \quad \textit{iff} \quad \max\{\, ||\alpha_1||_s, ||\alpha_2||_s \,\} \leq \max\{\, ||\beta_1||_s, ||\beta_2||_s \,\}.$$

Next we introduce some notions concerning *bases* which are binary relations over $\mathcal{P}(V^+)$.

Definition 5.6.3.

- *A* base *is a binary relation consisting of pairs $(X_i\alpha, X_j\beta) \in \mathcal{P}(V^+) \times \mathcal{P}(V^+)$ with $i \leq j$.*
- *A base B is called* bisimulation-complete *if whenever $X_i\alpha \sim X_j\beta$ with $i \leq j$ then one of the following conditions hold:*
  1. *$(X_i\alpha, X_j\beta)$ is decomposable and there are γ, γ' with $X_i\gamma \sim X_j$, $\gamma' \sim \gamma$ and $(X_i\gamma', X_j) \in B$.*
  2. *$(X_i\alpha, X_j\beta)$ is elementary and $(X_i\alpha', X_j\beta') \in B$ for some $\alpha' \sim \alpha$ and $\beta' \sim \beta$ such that $(\alpha', \beta') \sqsubseteq (\alpha, \beta)$.*
- *The relation*

$$\equiv_B =_{\mathrm{df}} \bigcup_{i \geq 0} \equiv_B^i$$

is defined recursively by:

1. $\epsilon \equiv_B^0 \epsilon$ *and*
2. $X_i\alpha \equiv_B^{i+1} X_j\beta$ *iff*
   a) $(X_i\gamma, X_j) \in B$ *and* $\alpha \equiv_B^j \gamma\beta$, *for some* $j \leq i$, *or*
   b) $(X_i\alpha', X_j\beta') \in B$, $\alpha \equiv_B^{j_1} \alpha'$ *and* $\beta \equiv_B^{j_2} \beta'$, *for some* $j_1, j_2 \leq i$.

The importance of the relation $\equiv_B$ is revealed by the following lemma.

Lemma 5.6.1.

1. $\equiv_B \subseteq \leftrightarrow_B^*$
2. *If* B *is bisimulation-complete then* $\sim \subseteq \equiv_B$.

Proof. The first part is easily proved by induction on i. The second part is shown by induction with respect to $\sqsubseteq$. First observe that $\epsilon \equiv_B \epsilon$. Let us now assume $X_i \sim X_j$ for some $X_i, X_j \in V_U$. The pair (X_i, X_j) is elementary, so by bisimulation-completeness of B we know $(X_i, X_j) \in B$ which implies $X_i \equiv_B X_j$. Thus the induction base holds. For the induction step now assume $X_i\alpha \sim X_j\beta$. We have to distinguish two cases. First assume that $(X_i\alpha, X_j\beta)$ is decomposable into $X_i\gamma \sim X_j$ and $\alpha \sim \gamma\beta$ for some γ. By bisimulation-completeness of B we have $(X_i\gamma', X_j) \in B$ for some $\gamma' \sim \gamma$ which implies $\alpha \sim \gamma'\beta$. Observing that $(\alpha, \gamma'\beta) \sqsubset (X_i\alpha, X_j\beta)$ the induction hypothesis yields $\alpha \equiv_B \gamma'\beta$ and therefore $X_i\alpha \equiv_B X_j\beta$. To show the remaining case assume that $(X_i\alpha, X_j\beta)$ is elementary. Then the bisimulation-completeness of $\mathcal{B}$ delivers $(X_i\alpha', X_j\beta') \in B$ for some $\alpha' \sim \alpha$ and $\beta' \sim \beta$ such that $(\alpha', \beta') \sqsubseteq (\alpha, \beta)$. Using $(\alpha', \beta') \sqsubseteq (\alpha, \beta) \sqsubset (X_i\alpha, X_j\beta)$ we conclude by induction $\alpha \equiv_B \alpha'$ and $\beta \equiv_B \beta'$ and therefore $X_i\alpha \equiv_B X_j\beta$. □

The key structure for our decidability result are *bisimulation bases* B. They are important as they characterise bisimulation equivalence as the least congruence w.r.t. sequential composition containing B.

Definition 5.6.4 (Bisimulation Base).
A relation B *satisfying* $\sim = \leftrightarrow_B^*$ *is said to be a* bisimulation base.

As a consequence of Lemma 5.6.1, one of the inclusions, $\sim \subseteq \leftrightarrow_B^*$, is guaranteed for bisimulation-complete relations B. A sufficient condition for the inverse inclusion, which can be proved for our construction, is 'self-bisimulation'.

Definition 5.6.5. *Given a base* B, *define the* sub-base $\mathcal{R}(B)$ *by:* $(\alpha, \beta) \in \mathcal{R}(B)$ *iff* $(\alpha, \beta) \in B$ *and*

1. $\alpha \xrightarrow{a} \alpha'$ *implies* $\exists\ \beta'.\ \beta \xrightarrow{a} \beta'\ \wedge\ \alpha' \equiv_B \beta'$
2. $\beta \xrightarrow{a} \beta'$ *implies* $\exists\ \alpha'.\ \alpha \xrightarrow{a} \alpha'\ \wedge\ \alpha' \equiv_B \beta'$

That $\mathcal{R}$ is a good candidate for successively reducing a bisimulation-complete relation to a bisimulation base is a consequence of the following lemma.

Lemma 5.6.2. *Let B be a bisimulation-complete base. Then the following holds:*

1. *If $(\alpha, \beta) \in B$ and $\alpha \sim \beta$ then $(\alpha, \beta) \in \mathcal{R}(B)$.*
2. *$\mathcal{R}(B)$ is bisimulation-complete.*

Proof. For the first part assume $(\alpha, \beta) \in B$ and $\alpha \sim \beta$. Whenever $\alpha \stackrel{a}{\to} \alpha'$ then $\beta \stackrel{a}{\to} \beta'$ and $\alpha' \sim \beta'$ for some β'. Thus by Lemma 5.6.1 we have $\alpha \equiv_B \beta$. Similarly, if $\beta \stackrel{a}{\to} \beta'$ then $\alpha \stackrel{a}{\to} \alpha'$ and $\alpha' \sim \beta'$ for some α'. Again by Lemma 5.6.1 we conclude $\alpha \equiv_B \beta$. So $(\alpha, \beta) \in \mathcal{R}(B)$.

For the second part, assume $X_i\alpha \sim X_j\beta$. If $(X_i\alpha, X_j\beta)$ is decomposable, i.e. if $X_i\gamma \sim X_j$ and $\alpha \sim \gamma\beta$ for some γ, then by the bisimulation-completeness of B we know $(X_i\gamma', X_j) \in B$ for some $\gamma' \sim \gamma$. Thus $X_i\gamma' \sim X_j$, and by part one the pair $(X_i\gamma', X_j)$ is also contained in $\mathcal{R}(B)$. On the other side, if $(X_i\alpha, X_j\beta)$ is elementary we have by bisimulation-completeness of B that $(X_i\alpha', X_j\beta') \in B$ for some $\alpha' \sim \alpha$ and $\beta' \sim \beta$. Using the first part again, we conclude from $X_i\alpha' \sim X_i\alpha \sim X_j\beta \sim X_j\beta'$ that $(X_i\alpha', X_j\beta') \in \mathcal{R}(B)$. □

Along the lines of [HJM94], this can straight-forwardly be exploited to verify that the successive $\mathcal{R}$-refinement of a bisimulation-complete relation yields indeed a bisimulation base: the (additional) fixpoint property of $B_\sim$ is sufficient to establish that $B_\sim$ is a self-bisimulation.

Theorem 5.6.1. *If B_0 is a finite bisimulation-complete base then*

$$B =_{\mathrm{df}} \bigcap \{\, \mathcal{R}^i(B_0) \mid i \geq 0 \,\}$$

is a bisimulation base, i.e. we have $\sim \; = \; \leftrightarrow_B^*$.

Proof. By definition of sub-base construction we have

$$\mathcal{R}^{i+1}(B_0) \subseteq \mathcal{R}^i(B_0),$$

for all $i \geq 0$. Since $2^{V^+ \times V^+}$ is well-founded there must exist some k such that

$$B = \mathcal{R}^{k+1}(B_0) \subseteq \mathcal{R}^k(B_0),$$

and hence B is a fixpoint of $\mathcal{R}$, i.e. $\mathcal{R}(B) = B$. Lemma 5.6.2 then guarantees that B is bisimulation-complete which yields $\sim \subseteq \equiv_B \subseteq \leftrightarrow_B^*$ by Lemma 5.6.1. Moreover, the fixpoint property of B implies by means of Lemma 5.6.1(1) that B is also a self-bisimulation. We thus conclude from Lemma 2.6.4 that also $\leftrightarrow_B^* \subseteq \sim$ holds which proves that B is a bisimulation base. □

5.6.2 The Computation of an Initial Base

Now we attack the most difficult step of our algorithm which deals with the computation of an initial base. In this step we collect a sufficiently large set of pairs such that after completion we may ensure bisimulation-completeness for the obtained base. Our key result that the search for candidate pairs to be

inserted in the initial base may be bounded will guarantee the effectiveness of this procedure. To start with, we apply the separability bound presented in Section 5.5 to a BPA system $\mathcal{C}'$ representing the normed part of the BPA system $\mathcal{C}$ at hand. Subsequently, this reduction will admit the development of a finite search for initial candidate pairs.

Recall that $\mathcal{C} = (V, Act, \mathcal{E}, X_1)$ is the normalised unnormed BPA system under consideration. We say a variable $Y \in V_U$ is *crossing*, if Y occurs on the right-hand side of some $X \in V_N$, and we denote the set of all crossing variables by V_C. Intuitively, starting with a normed process we can only reach an unnormed one by "crossing" a variable of V_C. If $\sim_{\mathcal{C}}$ denotes bisimilarity wrt. $\mathcal{C}$, we construct a normed BPA_δ system $\mathcal{C}' = (V', Act', \mathcal{E}', X_1)$ from $\mathcal{C}$ as follows.

- $V' = V_N \cup V_C$,
- $Act' = Act \cup \{\, a_{[Y]_{\sim_{\mathcal{C}}}} \mid Y \in V_C \,\}$, where each $a_{[Y]_{\sim_{\mathcal{C}}}}$ is an action not occurring in Act,
- $\mathcal{E}' = \{\, X =_{\mathrm{df}} E \in \mathcal{E} \mid X \in V_N \,\} \cup \{\, Y =_{\mathrm{df}} a_{[Y]_{\sim_{\mathcal{C}}}} \delta \mid Y \in V_C \,\}$.

The constructed BPA_δ system $\mathcal{C}'$ represents the labelled transition graph of the normed part of $\mathcal{C}$ as illustrated in Figure 5.4.

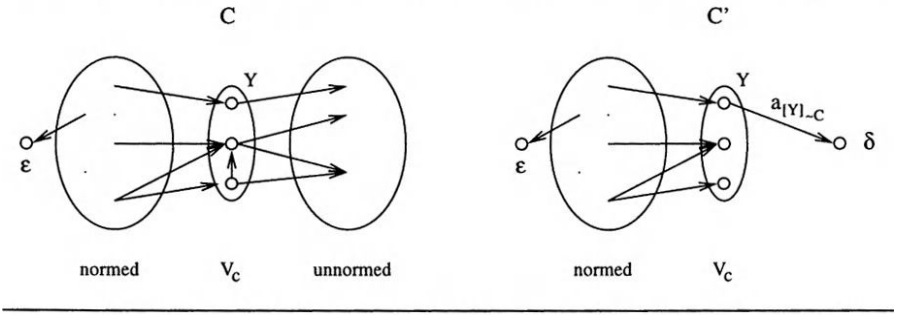

Fig. 5.4. Illustration of the labelled transition graphs of $\mathcal{C}$ and $\mathcal{C}'$.

The main idea of the construction is to encode the behaviour of crossing variables into transitions leading to a deadlocked process thereby obtaining some notion of normedness. Note, however, that the deadlock δ is used in order to preserve the bisimilarity of $Y\alpha \sim Y$ for $Y \in V_C, \alpha \in V_N^*$. With respect to $\mathcal{C}$ this bisimilarity is valid due to the unnormedness of Y, while $\delta\alpha \sim \delta$ guarantees that $Y\alpha \sim Y$ also holds with respect to $\mathcal{C}'$. Formally, we may relate the bisimulations associated with $\mathcal{C}$ and $\mathcal{C}'$ as stated in the following proposition.

Proposition 5.6.1. *For all $\alpha, \beta \in V_N^* \cup V_C$ we have: $\alpha \sim_{\mathcal{C}} \beta$ iff $\alpha \sim_{\mathcal{C}'} \beta$.*

Proof. To prove the direction from left to right, we will show that

$$R =_{\mathrm{df}} \sim_{\mathcal{C} \mid V_N^* \cup V_C} \cup \{ (\delta, \delta) \}$$

is a bisimulation wrt. $\mathcal{C}'$. As δ cannot perform any action we distinguish two cases:

Case 1: $\alpha \sim_{\mathcal{C}} \beta$ where $\alpha, \beta \in V_C$.

Let

$$\alpha \overset{a_{[\alpha]_{\sim_{\mathcal{C}}}}}{\rightarrow}_{\mathcal{C}'} \delta$$

be the only possible transition of α. Since $\alpha \sim_{\mathcal{C}} \beta$ and therefore $[\alpha]_{\sim_{\mathcal{C}}} = [\beta]_{\sim_{\mathcal{C}}}$ we obtain by construction of $\mathcal{C}'$ that

$$\beta \overset{a_{[\beta]_{\sim_{\mathcal{C}}}}}{\rightarrow}_{\mathcal{C}'} \delta$$

and $(\delta, \delta) \in R$. The other direction is shown by symmetric arguments.

Case 2: $\alpha \sim_{\mathcal{C}} \beta$ where $\alpha, \beta \in V_N^*$.

Let $\alpha \overset{a}{\rightarrow}_{\mathcal{C}'} \alpha'$. Since α is normed we know by construction of $\mathcal{C}'$ that also $\alpha \overset{a}{\rightarrow}_{\mathcal{C}} \alpha'$, and as $\alpha \sim_{\mathcal{C}} \beta$ this together implies $\beta \overset{a}{\rightarrow}_{\mathcal{C}} \beta'$ and $\alpha' \sim_{\mathcal{C}} \beta'$ for some β'. Again from the construction of $\mathcal{C}'$ and the fact that $\alpha', \beta' \in V_N^* \cup V_C$ we finally conclude that $\beta \overset{a}{\rightarrow}_{\mathcal{C}'} \beta'$ and $(\alpha', \beta') \in R$. The other direction follows in a similar fashion.

For the remaining direction from right to left., we first assume that $\alpha \sim_{\mathcal{C}'} \beta$ and $\alpha, \beta \in V_C$. Since

$$\alpha \overset{a_{[\alpha]_{\sim_{\mathcal{C}}}}}{\rightarrow}_{\mathcal{C}'} \delta \quad \text{and} \quad \beta \overset{a_{[\beta]_{\sim_{\mathcal{C}}}}}{\rightarrow}_{\mathcal{C}'} \delta$$

are the only possible transitions in $\mathcal{C}'$, we conclude from the assumption that $[\alpha]_{\sim_{\mathcal{C}}} = [\beta]_{\sim_{\mathcal{C}}}$ and therefore $\alpha \sim_{\mathcal{C}} \beta$. Finally, the proof is completed by showing that

$$R =_{\mathrm{df}} \sim_{\mathcal{C}' \mid V_N^*} \cup \sim_{\mathcal{C} \mid V_N^* V_U}$$

is a bisimulation wrt. $\mathcal{C}$. As the case $\alpha \sim_{\mathcal{C}} \beta$, where $\alpha, \beta \in V_N^* V_U$, is trivial, we only consider the situation when $\alpha \sim_{\mathcal{C}'} \beta$ with $\alpha, \beta \in V_N^*$. Thus let $\alpha \overset{a}{\rightarrow}_{\mathcal{C}} \alpha'$. From the construction of $\mathcal{C}'$, we conclude that $\alpha \overset{a}{\rightarrow}_{\mathcal{C}'} \alpha'$, and as $\alpha \sim_{\mathcal{C}'} \beta$ we have $\beta \overset{a}{\rightarrow}_{\mathcal{C}'} \beta'$ and $\alpha' \sim_{\mathcal{C}'} \beta'$ for some β'. Now we have either $\alpha', \beta' \in V_C$, which according to the first case yields $\alpha' \sim_{\mathcal{C}} \beta'$ and therefore $(\alpha', \beta') \in R$, or $\alpha', \beta' \in V_N^*$, which also guarantees $(\alpha', \beta') \in R$. Since the other direction is symmetric this completes the proof. □

Using this proposition and the following lemma, we are now able to apply the separability bound $\mathcal{B}_{\alpha,\beta}$ obtained in the previous section.

Lemma 5.6.3. *Let $\alpha \overset{w}{\rightarrow} \alpha'$ with α, α' normed and $|w| = l$. Then*

$$||\alpha'|| \leq (l(K-1) + |\alpha|)\mathcal{N}.$$

Proof. The proof is accomplished by induction on l. For $l = 0$ we clearly obtain $||\alpha'|| = ||\alpha|| \leq |\alpha|\mathcal{N}$. Now assume $\alpha = X\beta \xrightarrow{a} \gamma\beta \xrightarrow{w} \alpha'$ with $|w| = l$. Then we conclude by induction hypothesis

$$\begin{aligned} ||\alpha'|| &\leq (l(K-1) + |\gamma\beta|)\mathcal{N} \\ &\leq (l(K-1) + (K + |\beta|))\mathcal{N} \\ &= (l(K-1) + ((K-1) + |X\beta|))\mathcal{N} \\ &\leq ((l+1)(K-1) + |\alpha|)\mathcal{N}. \end{aligned}$$

□

Proposition 5.6.2. *Consider the BPA system $\mathcal{C}$ and let ζ be normed with $\zeta \not\sim \eta$, $||\zeta|| \leq ||\eta||$ and $\zeta\beta \sim \eta\beta$. Then $\beta \sim X$, where $X =_{\mathrm{df}} \gamma X$ for some $\gamma \neq \epsilon$ such that the following conditions hold.*

- *If $||\zeta|| < ||\eta||$ then $\eta \xrightarrow{w} \gamma$ for some fixed w where $\zeta \xrightarrow{w} \epsilon$ is a norm-reducing transition sequence.*
- *If $||\zeta|| = ||\eta||$ then without loss of generality $\zeta \xrightarrow{w} \epsilon$ and $\eta \xrightarrow{w} \gamma$ in $|w| \leq \mathcal{B}(\zeta, \eta)$ steps where*

$$\mathcal{B}(\zeta, \eta) =_{\mathrm{df}} \mathcal{B}^{\mathcal{C}'}_{\zeta,\eta} + (\mathcal{B}^{\mathcal{C}'}_{\zeta,\eta}(K-1) + |\zeta|)\mathcal{N}.$$

Proof. First assume $||\zeta|| < ||\eta||$. Let $\zeta \xrightarrow{w} \epsilon$ be a norm-reducing transition sequence. Then $\zeta\beta \xrightarrow{w} \beta, \eta\beta \xrightarrow{w} \gamma\beta$ and $\beta \sim \gamma\beta$ for some γ such that $\eta \xrightarrow{w} \gamma$ in $|w| = ||\zeta||$ steps. Since any system of guarded equations has a unique solution up to bisimulation, we conclude that $\beta \sim X$ where $X =_{\mathrm{df}} \gamma X$, which completes the first part.

Now suppose $||\zeta|| = ||\eta||$. By Proposition 5.6.1, $\mathrm{Sep}_{\mathcal{C}'}(\zeta, \eta) = m$ for some $m \geq 1$. Thus we have the following situation: $\zeta = \zeta_m \not\sim_{m,\mathcal{C}'} \eta_m = \eta$. Let $\zeta_m \xrightarrow{a} \zeta_{m-1}$ wlog. be a separating transition. Since $\zeta_m\beta \xrightarrow{a} \zeta_{m-1}\beta$ we also have $\eta_m\beta \xrightarrow{a} \eta_{m-1}\beta$ for some η_{m-1} such that $\zeta_{m-1}\beta \sim_{\mathcal{C}} \eta_{m-1}\beta$ and $\zeta_{m-1} \not\sim_{m-1,\mathcal{C}'} \eta_{m-1}$ due to the separating property of the transition. This construction can repeatedly be applied to obtain sequences $\zeta_m, \ldots, \zeta_1$ and $\eta_m, \ldots, \eta_1$ such that $\zeta_i \not\sim_{i,\mathcal{C}'} \eta_i$ and $\zeta_i\beta \sim_{\mathcal{C}} \eta_i\beta$ for all $1 \leq i \leq m$. An illustration is given in Figure 5.5. As the situation is symmetric in this case we may assume $||\zeta_1|| \leq ||\eta_1||$. Now observe that by Theorem 5.5.1 we have

$$m - 1 = \mathrm{Sep}_{\mathcal{C}'}(\zeta, \eta) - 1 \leq \mathcal{B}^{\mathcal{C}'}_{\zeta,\eta} - 1.$$

In order to complete the proof we consider two cases:

Case 1: $||\eta_1|| < \infty$. This implies $||\zeta_1|| < \infty$, and since $\mathcal{C}$ and $\mathcal{C}'$ coincide on $V_N^* \times V_N^*$, we have also $\zeta_1 \not\sim_{1,\mathcal{C}} \eta_1$. As $\zeta_1\beta \sim_{\mathcal{C}} \eta_1\beta$, we obtain $\zeta_1 = \epsilon$. Hence $\gamma = \eta_1$ suits and we have $|w| = m - 1 \leq \mathcal{B}^{\mathcal{C}'}_{\zeta,\eta} - 1$.

Case 2: $||\eta_1|| = \infty$. From $\zeta_1 \not\sim_{\mathcal{C}'} \eta_1$ we conclude by Proposition 5.6.1 that $\zeta_1 \not\sim_{\mathcal{C}} \eta_1$. Since $\zeta_1\beta \sim_{\mathcal{C}} \eta_1\beta$, we have $||\zeta_1|| < \infty$. Thus Lemma 5.6.3 yields

$$||\zeta_1|| \leq ((m-1)(K-1) + ||\zeta||)\mathcal{N}.$$

Moreover, for the norm-reducing transition sequence $\zeta_1 \stackrel{w'}{\to} \epsilon$ there exists some γ such that $\eta_1 \stackrel{w'}{\to} \gamma$ and $\beta \sim \gamma\beta$. Hence for $w =_{\rm df} a_{m-1} \ldots a_1 w'$ we obtain $\eta = \eta_m \stackrel{a_{m-1}\ldots a_1}{\to} \eta_1 \stackrel{w'}{\to} \gamma$ in

$$|w| = m - 1 + ||\zeta_1|| \leq \mathcal{B}^{\mathcal{C}'}_{\zeta,\eta} + (\mathcal{B}^{\mathcal{C}'}_{\zeta,\eta}(K-1) + |\zeta|)\mathcal{N}$$

steps and $\beta \sim X$ where $X =_{\rm df} \gamma X$.

□

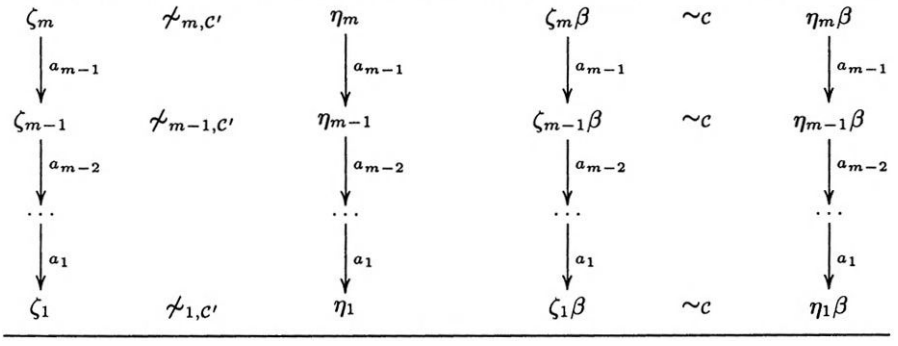

Fig. 5.5. The construction of a separating transition track.

The first step of the initial base construction consists of a *completion* procedure for the given BPA system.

Algorithm 5.6.1. The *elementary completion* $\mathcal{E}^\omega$ of a set of equations $\mathcal{E}$ is defined as follows:

- Every equation of $\mathcal{E}$ is contained in $\mathcal{E}^\omega$.
- For each i and j in the range $1 \leq i \leq j \leq n$ such that $X_i, X_j \in V_N$ fix some norm-reducing transition sequence $X_i \stackrel{u}{\to} \epsilon$. Then for each pair $(X_i\gamma, X_j)$ such that $X_j \stackrel{u}{\to} \gamma$ for some γ we do the following:
  1. If $||X_i\gamma|| < ||X_j||$ then fix some norm-reducing transition sequence $X_i\gamma \stackrel{w}{\to} \epsilon$ and add for each ζ such that $X_j \stackrel{w}{\to} \zeta$ the equations $Y =_{\rm df} \zeta Y$ and $Y' =_{\rm df} \gamma Y$ to $\mathcal{E}^\omega$ where Y, Y' are fresh variables.
  2. If $||X_i\gamma|| > ||X_j||$ then fix some norm-reducing transition sequence $X_j \stackrel{w}{\to} \epsilon$ and add for each ζ such that $X_i\gamma \stackrel{w}{\to} \zeta$ the equations $Y =_{\rm df} \zeta Y$ and $Y' =_{\rm df} \gamma Y$ to $\mathcal{E}^\omega$ where Y, Y' are fresh variables.
  3. If $||X_i\gamma|| = ||X_j||$ then add for each ζ such that
     - $X_i\gamma \stackrel{w}{\to} \epsilon, X_j \stackrel{w}{\to} \zeta$ in $|w| \leq \mathcal{B}(X_i\gamma, X_j)$ steps, or
     - $X_j \stackrel{w}{\to} \epsilon, X_i\gamma \stackrel{w}{\to} \zeta$ in $|w| \leq \mathcal{B}(X_i\gamma, X_j)$ steps

the equations $Y =_{\mathrm{df}} \zeta Y$ and $Y' =_{\mathrm{df}} \gamma Y$ to $\mathcal{E}^\omega$ where Y, Y' are fresh variables.

The equations generated from a pair $(X_i\gamma, X_j)$ are also called the *elementary completion* of $(X_i\gamma, X_j)$.

During this completion we will add new equations to $\mathcal{E}$ in order to explicitly denote some needed bisimulation classes. Note, however, that each equation added to $\mathcal{E}$ defines an unnormed variable which is bisimilar to $\gamma\zeta^\omega$ for some γ, ζ, and that no new variable is reachable from any of the original variables. Thus the bisimulation classes generated by finite sequences over V are unchanged after completion.

Algorithm 5.6.2 (Initial base B_0 construction).
Let $\mathcal{E}^\omega$ be the elementary completion of $\mathcal{E}$ with variables V_N and $V_U^\omega = V_U \cup V_U^{new}$.

Step 1: For each i and j in the range $1 \leq i \leq j \leq n$ such that $X_i, X_j \in V_N$ fix some w and some $[X_j]_{||X_i||}$ such that $X_j \xrightarrow{w} [X_j]_{||X_i||}$ in $|w| = ||X_i||$ norm-reducing steps. Then $(X_i[X_j]_{||X_i||}, X_j) \in B_0$.

Step 2: For each i and j in the range $1 \leq i \leq j \leq n$ such that $X_i, X_j \in V_N$ let u be the path labelling chosen during the elementary completion. Then for each γ such that $X_j \xrightarrow{u} \gamma$ let $(X_iY', X_jY) \in B_0$, if $Y =_{\mathrm{df}} \zeta Y$ and $Y' =_{\mathrm{df}} \gamma Y$ are contained in the elementary completion of $(X_i\gamma, X_j)$.

Step 3: For each i and j in the range $1 \leq i \leq j \leq n$ such that $X_i, X_j \in V_U^\omega$ let $(X_i, X_j) \in B_0$.

Step 4: For each $X_i \in V_N$ and $X_j \in V_U^\omega$ let

$$\{\, (X_i[X_j]_{||X_i||}, X_j) \mid X_j \xrightarrow{w} [X_j]_{||X_i||} \text{ in } |w| = ||X_i|| \text{ steps} \,\} \subseteq B_0.$$

Moreover, let $\mathcal{S}$ be the maximal seminorm of all $[X_j]_{||X_i||}$ obtained. Then add also $\{\, (X_i\alpha, X_j) \mid ||\alpha||_s < \mathcal{S} \,\}$ to B_0.

Intuitively, in step 1 we collect all candidate pairs $(X_i\gamma, X_j)$ needed for the reduction of decomposable pairs, while step 2 collects a sufficiently large set of candidate elementary pairs (X_iY', X_jY). Step 3 simply includes all pairs (X, Y) where X, Y are unnormed since there may also exist elementary pairs of this form. Finally, step 4 collects possible elementary pairs of the form $(X_i\alpha, X_j)$, together with all smaller pairs thereby ensuring that the initial base will contain all minimal elementary pairs as required by condition (2) of bisimulation-completeness. That the constructed base admits the computation of a bisimulation base is now a consequence of the following theorem.

Theorem 5.6.2. *The initial base B_0 is bisimulation-complete.*

Proof. Assume $X_i\alpha \sim X_j\beta$, and let us first consider the case when $(X_i\alpha, X_j\beta)$ is decomposable, i.e. $X_i\gamma \sim X_j$ and $\alpha \sim \gamma\beta$ for some γ. Let w be the norm-reducing transition sequence of length $||X_i||$ chosen in step (1) of the initial base construction. Then due to $X_j \stackrel{w}{\to} [X_j]_{||X_i||}$ we have $X_i\gamma \stackrel{w}{\to} \gamma$ and $\gamma \sim [X_j]_{||X_i||}$. Thus the pair $(X_i[X_j]_{||X_i||}, X_j) \in B_0$ satisfies condition (1) for bisimulation-completeness.
Now assume $(X_i\alpha, X_j\beta)$ is elementary.

Case 1: $X_i, X_j \in V_N$

Let $X_i \stackrel{u}{\to} \epsilon$ be the norm-reducing transition sequence chosen during the elementary completion. Then $X_i\alpha \stackrel{u}{\to} \alpha, X_j\beta \stackrel{u}{\to} \gamma\beta$ and $\alpha \sim \gamma\beta$ for some γ such that $X_j \stackrel{u}{\to} \gamma$. Thus we also have $X_i\gamma\beta \sim X_i\alpha \sim X_j\beta$. Moreover, we know $X_i\gamma \not\sim X_j$ due to the undecomposability of the pair $(X_i\alpha, X_j\beta)$. Observing that $\min\{\, ||X_i\gamma||, ||X_j|| \,\}$ is finite allows us to apply Lemma 5.6.2. We distinguish the following cases.

1. $||X_i\gamma|| < ||X_j||$
   For the norm-reducing transition sequence $X_i\gamma \stackrel{w}{\to} \epsilon$ fixed during elementary completion we have $X_j \stackrel{w}{\to} \zeta$ and $\beta \sim \zeta\beta$ for some ζ. So there exist equations $Y =_{\mathrm{df}} \zeta Y,\ Y' =_{\mathrm{df}} \gamma Y \in \mathcal{E}^\omega$ with $Y \sim \beta$ and $Y' \sim \gamma Y \sim \gamma\beta \sim \alpha$. Due to step (2.) we have $(X_iY', X_jY) \in B_0$ and obviously this pair is minimal wrt. $\sqsubseteq$.
2. $||X_i\gamma|| > ||X_j||$
   This case follows a similar way as the previous one.
3. $||X_i\gamma|| = ||X_j||$
   Then without loss of generality for a transition sequence $X_i\gamma \stackrel{w}{\to} \epsilon$ with $|w| \leq \mathcal{B}(X_i\gamma, X_j)$ we have $X_j \stackrel{w}{\to} \zeta$ and $\beta \sim \zeta\beta$. Again the elementary completion ensures the existence of equations $Y =_{\mathrm{df}} \zeta Y,\ Y' =_{\mathrm{df}} \gamma Y \in \mathcal{E}^\omega$ with $Y \sim \beta$ and $Y' \sim \gamma Y \sim \gamma\beta \sim \alpha$. Due to step (2.) we have $(X_iY', X_jY) \in B_0$ and obviously this pair is also minimal wrt. $\sqsubseteq$.

Case 2: $X_i, X_j \in V_U^\omega, \alpha = \beta = \epsilon$

Then by step (3.) $(X_i, X_j) \in B_0$. Note that this pair is minimal wrt. $\sqsubseteq$.

Case 3: $X_i \in V_N, X_j \in V_U^\omega, \beta = \epsilon$

Let $X_i \stackrel{w}{\to} \epsilon$ be a norm-reducing transition sequence. Then $X_i\alpha \stackrel{w}{\to} \alpha, X_j \stackrel{w}{\to} \alpha'$ and $\alpha \sim \alpha'$ for some α'. Due to step (4.) we have $(X_i\alpha', X_j) \in B_0$. If there exists α'' with $\alpha'' \sim \alpha'$ and $||\alpha''||_s < ||\alpha'||_s$ then the seminorm down closure ensures that $(X_i\alpha'', X_j)$ is also contained in B_0.

□

5.6.3 The Branching Algorithm

Now assume that we have refined our initial base B_0 constructed in the previous section by means of the sub-base construction $\mathcal{R}$ to a bisimulation

base B. Deciding bisimilarity of two processes α, β is then accomplished by using a tableaux system in the style of [HS91].

A *tableau system* is a goal-directed proof system defined by a set of *proof-rules*, and thus similar to a branching algorithm. The difference lies in the proof strategy, i.e. the choice to which of the current subgoals a rule should be applied next, which is defined for branching algorithms, while left unspecified for tableau systems.

To determine whether $\alpha = \beta$ we construct a tableau for $\alpha = \beta$ using the rules given in Table 5.3. As usual, a *tableau* for $\alpha = \beta$ is a maximal finite proof tree with root $\alpha = \beta$ such that the equations labelling an immediate successor of a node are obtained by application of some rule. If none of the rules is applicable to a node it is called a *terminal*. Moreover, we call a terminal *successful* if it is labelled with $\epsilon = \epsilon$, and *unsuccessful* otherwise. Accordingly, a tableau is said to be *successful* if all its leaves are successful terminals.

$\mathtt{I}:$	$\dfrac{\alpha = \beta}{\epsilon = \epsilon}$	$(\alpha, \beta) \in B$
$\overline{\mathtt{I}}:$	$\dfrac{\beta = \alpha}{\epsilon = \epsilon}$	$(\alpha, \beta) \in B$
$\mathtt{D}:$	$\dfrac{X_i\alpha = X_j\beta}{\alpha = \gamma\beta}$	$(X_i\gamma, X_j) \in B$
$\overline{\mathtt{D}}:$	$\dfrac{X_j\beta = X_i\alpha}{\alpha = \gamma\beta}$	$(X_i\gamma, X_j) \in B$
$\mathtt{E}:$	$\dfrac{X_i\alpha = X_j\beta}{\alpha = \alpha' \quad \beta = \beta'}$	$(X_i\alpha', X_j\beta') \in B$ and $(\alpha', \beta') \sqsubseteq (\alpha, \beta)$
$\overline{\mathtt{E}}:$	$\dfrac{X_j\beta = X_i\alpha}{\alpha = \alpha' \quad \beta = \beta'}$	$(X_i\alpha', X_j\beta') \in B$ and $(\alpha', \beta') \sqsubseteq (\alpha, \beta)$

Table 5.3. The tableau rules for deciding bisimulation.

The decidability, soundness, and completeness of the tableau method is proved in the following propositions.

Proposition 5.6.3. *There exist only finitely many tableaux for $\alpha = \beta$, and each of them is finite.*

Proof. Let T be a tableau for $\alpha = \beta$. Observe that T is finitely branching since the rules applied have only one or two consequents. Moreover, for each consequent $\zeta' = \eta'$ of a premise $\zeta = \eta$ we have $(\zeta', \eta') \sqsubset (\zeta, \eta)$ which bounds the length of each path in T by (α, β). Hence, according to König's Lemma T must be finite.

Since the number of applicable rules, as well as the bisimulation base, is finite we conclude from the bounded length of each path in a tableau that only finitely many tableaux can exist for $\alpha = \beta$. □

Proposition 5.6.4. *The tableau method is sound and complete, i.e. we have $\alpha \sim \beta$ iff there exists a successful tableau for $\alpha = \beta$.*

Proof. Assume $\alpha \sim \beta$. Due to bisimulation-completeness of B it is possible to build a tableau for $\alpha = \beta$ such that for every node labelling $\zeta = \eta$ we have $\zeta \sim \eta$. Since by Proposition 5.6.3 this construction must terminate all leaves are labelled by $\epsilon = \epsilon$, and thus successful. Hence, the tableau itself is also successful.

Now suppose T is a successful tableau for $\alpha = \beta$. To prove $\alpha \sim \beta$ we show $\zeta \sim \eta$ for every node labelling in T by induction on tree depth. The proof relies on the bisimilarity of $\gamma \sim \sigma$ for each $(\gamma, \sigma) \in B$. Obviously, we have $\epsilon \sim \epsilon$ for every successful leave. Now let $\zeta = \eta$ be the labelling of an inner node and assume that $\zeta' \sim \eta'$ for all consequents $\zeta' = \eta'$ of $\zeta = \eta$. First, if the I-rule is applied to $\zeta = \eta$ we deduce $\zeta \sim \eta$ due to $(\zeta, \eta) \in B$. Second, if $\zeta = \eta$ is the premise of a D-rule application let $\zeta = X_i\alpha'$, $\eta = X_j\beta'$, and $\alpha' = \gamma\beta'$ be the consequent. Then we have $X_i\gamma \sim X_j$ due to $(X_i\gamma, X_j)B$, and $\alpha' \sim \gamma\beta'$ by induction hypothesis. From this we conclude $X_i\alpha' \sim X_i\gamma\beta' \sim X_j\beta'$, as desired. Finally, in case of an E-rule application to $\zeta = \eta$ let $\zeta = X_i\alpha'$, $\eta = X_j\beta'$, and $\alpha' = \alpha''$, $\beta' = \beta''$ be the consequents. By induction hypothesis we then know $\alpha' \sim \alpha''$ and $\beta' \sim \beta''$ which yields $X_i\alpha' \sim X_i\alpha'' \sim X_j\beta'' \sim X_j\beta'$ due to $(X_i\alpha'', X_j\beta'') \in B$. The remaining cases where a dual rule is applied to $\zeta = \eta$ are dealt with entirely similar. □

5.6.4 Summary of the Decision Procedure

The overall algorithm proceeds in three steps:

1. Computation of an initial base B_0 by means of the algorithm presented in Section 5.6.2. This algorithm terminates with a base of elementary size.
2. Refinement of B_0 by iterative application of $\mathcal{R}$ until a bisimulation base is reached, which is the case after at most $|B_0|$ iterations. As B_0 is of elementary size and each iteration itself is elementary also this step is elementary.
3. Decision of $\alpha \sim \beta$ by means of a straightforward (obviously elementary) branching algorithm.

Summarising we can conclude:

Theorem 5.6.3 (Main Theorem).
Our three step bisimulation decision algorithm is elementary.

Remark: Our complexity analysis is quite rough. A more careful analysis would reveal quite a small 'exponentiality' which, nevertheless, is intractable for practical purposes. Our main result established is therefore the effectivity of the bisimulation decision procedure for arbitrary context-free processes.

6. Summary and Perspectives

In this monograph we have considered syntactical, logical and semantical analysis and verification methods for the class of context-free processes, as well as its generalisation the class of pushdown processes. By exploiting structural properties of the underlying process model the presented algorithms have clearly demonstrated that an automated reasoning about these restricted, but nevertheless expressive, classes of infinite-state systems is indeed possible. Although the developed algorithms are quite intricate as the associated decision problems are already very close to the dividing line between decidability and undecidability, we hope that they provide a better understanding of the theory of infinite-state concurrent systems.

In this final chapter we will summarise our achievements, present relevant results which have been obtained since the submission of my Ph.D. thesis and give some directions for further research.

6.1 Summary of the Main Results

In *Chapter 3* we proposed Pushdown Process Algebra (PDPA) as a suitable framework for modelling pushdown processes. We introduced the PDPA laws and showed a normal form theorem stating that any guarded PDPA system may effectively be transformed into an up to bisimilarity equivalent PDPA system in Pushdown Normal Form. We furthermore investigated parallel compositions involving pushdown processes. In particular, we showed that the class of pushdown processes is up to relabelling the smallest extension of the class of context-free processes closed under parallel composition with finite-state systems.

In *Chapter 4* we developed an iterative model checker that decides the alternation-free fragment of the μ-calculus for pushdown processes thus complementing local tableaux-based methods investigated in [HS93]. The correctness and soundness of our algorithm relies on a second-order variant of the ordinary semantics of μ-formulas which raises the iteration domain from set of states, or dually sets of formulas, to functions over sets of formulas. The complexity of the algorithm is exponential in the size of the formula to be verified, but only quadratic in the size of the PDPA system in question.

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 151-154, 1997

We furthermore showed a regularity property of the semantics of μ-formulas when interpreted on pushdown transition graphs.

In *Chapter 5* we presented a branching algorithm for deciding bisimilarity of normed context-free processes. Exploiting the tree structure of this branching algorithm we proved that the number of transitions needed to distinguish two non-bisimilar normed context-free processes is bounded by a constant which only depends on the given processes and the BPA system under consideration. This result generalises work of Caucal [Cau89] who considered only the deterministic case. Using our new bound we furthermore showed how to effectively compute a bisimulation base for a given BPA system extending the result of Christensen, Hüttel and Stirling [CHS92] who proved the mere existence of such a base. Finally, we obtained by combining this base construction with a tableaux system in the style of Hüttel and Stirling [HS91] an elementary decision procedure for bisimulation equivalence of arbitrary context-free processes.

6.2 Perspectives

In the rest of this chapter we give directions for further research and describe open problems related to this monograph. That the analysis and verification of infinite-state systems is currently receiving a lot of attention in the concurrency theory community is underpined by the fact that most of the open problems we stated at the submission time of my thesis in summer 1995 have been solved in the meantime. The following exposition summarises the state of the art, and, in particular, provides the relevant references.

6.2.1 Model Checking

An important problem not addressed in this monograph and only recently solved in [BS97] is how to extend the model checking algorithm to the full modal μ-calculus. Even though the semantics of μ-formulas becomes discontinuous when allowing alternating nested fixpoints the second-order semantics which is more oriented at the regular structure of pushdown processes seems to elegantly cope with this problem. However, the difficulty of establishing the correctness and soundness of an extended model checker that decides the full modal μ-calculus lies in the orthogonality of property transformers and state sets. As we have shown, a family of property transformers may be used to define state sets representing the semantics of subformulas, and on the other hand, state sets induce a family of property transformers. In the alternation-free case these two points of view are easy to match, whereas deeper alternation depths where some sort of induction is required make the task of matching much harder. As proved in [BS97], it turns out that the straight-forward extension of our model checker where according to the parities of the equational system also in the PT-scheme alternating fixpoints

occur is indeed the desired algorithm. The only complication arises from the need that now also valuations have to be taken into account when proving the correctness of the algorithm. Fortunately, all this additional complexity is only required for the proof and needs not be considered for an implementation.

A different line of research has been pursued in [BQ97] and [Bur97] where our model checking algorithm has been generalised in order to deal with families of infinite-state systems more expressive than pushdown processes. Whereas the first paper presents an adaptation of the notion of predicate transformer to the framework of hypergraph grammars as introduced in Section 3.8.4 which allows the handling of infinite-branching transition graphs, the second paper shows how to verify an even more expressive class of transition graphs introduced in [Cau96] by encoding the acceptance of regular languages into μ-formulas and using the model checker for the full μ-calculus.

Another interesting question is whether our automata-based construction showing the regularity of μ-formula semantics on pushdown transition graphs may be exploited for obtaining new results about the expressiveness of μ-formulas. Using radically different techniques, the open problem whether the alternation depth of μ-formulas induces a hierarchy with respect to expressiveness has been answered by Lenzi [Len96] and Bradfield [Bra96] in the affirmative. While Lenzi uses a topological approach for the proof, Bradfield transfers a strictness result for an alternation hierarchy of arithmetic with fixpoints.

6.2.2 Equivalence Checking

Decidability of bisimulation equivalence for arbitrary context-free processes was first presented in [CHS92]. The result relies on the existence of two semi-decision procedures: one searching a finite bisimulation base and one testing for non-bisimilarity. Hence, no complexity measure could be given. The purpose of our studies was therefore to find a single decision procedure which would allow to open a new area for complexity improvements. However, it turned out that the mere exploitation of the separability bound gives a highly nondeterministic algorithm which prevents it from any practical applicability. Our work can thus only be seen as a first step in the direction of obtaining a more realistic decision algorithm, also reflected by our rough complexity estimation. Moreover, the lower complexity bound for deciding bisimilarity of context-free processes remains a question still to be answered. Nevertheless, the new results underpin the importance of bisimulation bases for the theory of BPA processes. So we may ask, whether it is possible to find structural properties of bisimulation bases which would allow for better decision algorithms.

A related issue is the bisimulation equivalence problem for the larger class of pushdown processes. Since this problem is closely connected to the problem of deciding language equivalence for deterministic pushdown automata, as

well as to the problem of deciding equivalence for monadic recursion schemes, we cannot expect a simple answer. Obviously, the approach using bisimulation bases is not applicable to this class of processes as a vertex in a pushdown transition graph corresponds not only to a sequence of nonterminals, but incorporates also an additional control state. However, our characterisation theorem indicates that processes of the form

$$\varphi(\mathcal{C} \parallel \mathcal{R})$$

where $\mathcal{C}$ is a context-free process and $\mathcal{R}$ is a regular process are candidates for further investigation. The difficulty in applying the approach of bisimulation bases to the context-free component lies in the required synchronisation with the regular process leading to forbidden transitions.

Nevertheless, two outstanding breakthroughs have been achieved in this area in the recent past. First, Stirling [Sti96] proved that bisimulation equivalence is decidable for normed pushdown processes. He successfully attacked this problem by introducing a new tableau system in conjunction with recursive constants capturing the possible termination behaviour of pushdown processes. In a second landmark paper Sénizergues [Sén97] showed the decidability of language equivalence for deterministic pushdown automata. His result is based on considering deterministic rational series and applying a triangulation technique from linear algebra. However, the proof is very intricate and a careful analysis is required in order to obtain an intuitive understanding.

6.2.3 Regularity of Context-Free Processes

From classical language theory it is known that regularity of context-free languages is undecidable. Nevertheless, the decidability of bisimulation equivalence for context-free processes indicates that in the finer setting of bisimulation semantics the situation may be different. As a first step in this direction, Mauw and Mulder [MM94] have shown that it is decidable whether a complete BPA specification is regular, i.e. whether *all* variables define a regular process. However, the more interesting question seems to be whether a given *single* variable is regular.

Exploiting the effective characterisation of all bisimulation equivalence classes, as given by our decision algorithm, the decidability of this problem has recently been established in [BCS96] where it was shown that the factorisation of a context-free process with respect to bisimulation equivalence is effectively a transition graph representable by means of a hypergraph grammar. Since finiteness is decidable for these transition graphs, now called *regular graphs*, this yields, as a corollary, a decision procedure for the regularity problem of context-free processes.

References

[BB91] J.C.M. Baeten and J.A. Bergstra. Recursive Process Definitions with the State Operator. *Theoretical Computer Science*, 82:285–302, 1991.

[BBK87a] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of Bisimulation Equivalence for Processes Generating Context-Free Languages. In *PARLE '87*, LNCS 259, pages 94–113. Springer, 1987.

[BBK87b] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of Bisimulation Equivalence for Processes Generating Context-Free Languages. Technical Report CS-R8632, CWI, September 1987.

[BCS95] O. Burkart, D. Caucal, and B. Steffen. An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes. In *MFCS '95*, LNCS 969, pages 423–433. Springer, 1995.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation Collapse and the Process Taxonomy. In *CONCUR '96*, LNCS 1119, pages 247–262. Springer, 1996.

[BER94] A. Bouajjani, R. Echahed, and R. Robbana. Verification of Nonregular Temporal Properties for Context-Free Processes. In *CONCUR '94*, LNCS 836, pages 81–97. Springer, 1994.

[BK84] J.A. Bergstra and J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60:109–137, 1984.

[BK88] J.A. Bergstra and J.W. Klop. Process Theory Based on Bisimulation Semantics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 50–122. Springer, 1988.

[BQ97] O. Burkart and Y.-M. Quemener. Model-Checking of Infinite Graphs Defined by Graph Grammars. In *INFINITY '96*, volume 6 of *ENTCS*, 15 pages. Elsevier Science B.V., 1997.

[Bra91] J.C. Bradfield. *Verifying Temporal Properties of Systems*. Birkhäuser, Boston, Massachussets, 1991.

[Bra96] J.C. Bradfield. The Modal mu-Calculus Alternation Hierarchy is Strict. In *CONCUR '96*, LNCS 1119, pages 233–246. Springer, 1996.

[BS92a] J.C. Bradfield and C. Stirling. Local Model Checking for Infinite State Spaces. *Theoretical Computer Science*, 96:157–174, 1992.

[BS92b] O. Burkart and B. Steffen. Model Checking for Context-Free Processes. In *CONCUR '92*, LNCS 630, pages 123–137. Springer, 1992.

[BS94] O. Burkart and B. Steffen. Pushdown Processes: Parallel Composition and Model Checking. In *CONCUR '94*, LNCS 836, pages 98–113. Springer, 1994.

[BS97] O. Burkart and B. Steffen. Model-Checking the Full-Modal Mu-Calculus for Infinite Sequential Processes. In *ICALP '97*, LNCS 1256, pages 419–429. Springer, 1997.

[Bur97] O. Burkart. Model-Checking Rationally Restricted Right Closures of Recognizable Graphs. In *INFINITY '97*, UPMAIL Tech. Report 148, pages 19–29. Uppsala, July 1997.

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 155-159, 1997

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in TCS*. Cambridge University Press, 1990.

[Cau89] D. Caucal. A Fast Algorithm to Decide on Simple Grammars Equivalence. In *Int. Symposium on Optimal Algorithms*, LNCS 401, pages 66–85. Springer, 1989.

[Cau90] D. Caucal. Graphes Canoniques de Graphes Algébriques. *RAIRO*, 24(4):339–352, 1990.

[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 *ICALP '96*, LNCS 1099, pages 194–205. Springer, 1996.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGL94] E. Clarke, O. Grumberg, and D. Long. Verification Tools for Finite-State Concurrent Systems. In *A Decade of Concurrency, LNCS 803*, pages 124–175. Springer, 1994.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation Equivalence is Decidable for all Basic Parallel Processes. In *CONCUR '93*, LNCS 715, pages 143–157. Springer, 1993.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, The University of Edinburgh, Department of Computer Science, 1993.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation Equivalence is Decidable for all Context-Free Processes. In *CONCUR '92*, LNCS 630, pages 138–147. Springer, 1992.

[CKS92] R. Cleaveland, M. Klein, and B. Steffen. Faster Model Checking for the Modal Mu-Calculus. In *CAV '92*, LNCS 663, pages 410–422, 1992.

[Cle90] R. Cleaveland. Tableau-Based Model Checking in the Propositional Mu-Calculus. *Acta Informatica*, 27:725–747, 1990.

[CM90] D. Caucal and R. Monfort. On the Transition Graphs of Automata and Grammars. In *Graph-Theoretic Concepts in Computer Science*, LNCS 484, pages 311–337. Springer, 1990.

[Cou89] B. Courcelle. The Definability of Equational Graphs in Monadic Second-Order Logic. In *ICALP '89*, LNCS 372, pages 207–221. Springer, 1989.

[Cou90] B. Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 5, pages 193–242. Elsevier Science Publisher B.V., 1990.

[CS91] R. Cleaveland and B. Steffen. Computing Behavioural Relations, Logically. In *ICALP '91*, LNCS 510, pages 127–138. Springer, 1991.

[CS92] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. In *CAV '91*, LNCS 575, pages 48–58. Springer, 1992.

[DP90] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[EH86] E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not never" Revisited: On Branching Versus Linear Time. *Journal of the ACM*, 33:151–178, 1986. Extended version of POPL '83, pp. 127– 140.

[EJ88] E.A. Emerson and C.S. Jutla. The Complexity of Tree Automata and Logics of Programs. In *FOCS '88*, pages 328–337. IEEE Computer Society Press, 1988.

[EJ91] E.A. Emerson and C.S. Jutla. Tree Automata, Mu-Calculus and Determinacy. In *FOCS '91*, pages 368–377. IEEE Computer Society Press, 1991.

[EK95] J. Esparza and A. Kiehn. On the Model Checking Problem for Branching Time Logics and Basic Parallel Processes. In *CAV '95*, LNCS 939, pages 353–366. Springer, 1995.

[EL86] E.A. Emerson and C.-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *LICS '86*, pages 267–278. IEEE Computer Society Press, 1986.

[Eme90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science Publisher B.V., 1990.

[Esp94] J. Esparza. On the Decidability of Model Checking for Several μ-calculi and Petri Nets. In *CAAP '94*, LNCS 787, pages 115–129. Springer, 1994.

[FL79] M.J. Fischer and R.E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[Flo67] R.W. Floyd. Assigning Meanings to Programs. In *Symposium on Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, 1967.

[Fri76] E. P. Friedman. The Inclusion Problem for Simple Languages. *Theoretical Computer Science*, 1:297–316, 1976.

[GH94] J.F. Groote and H. Hüttel. Undecidable Equivalences for Basic Process Algebra. *Information and Computation*, 115(2):354–371, 1994.

[Gla90] R.J. van Glabbeek. The Linear Time - Branching Time Spectrum. In *CONCUR 90*, LNCS 458, pages 278–297. Springer, 1990.

[Gro91] J.F. Groote. A Short Proof of the Decidability of Bisimulation for Normed BPA-Processes. *Information Processing Letters*, 42:167–171, 1991.

[Har78] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[HJM94] Y. Hirshfeld, M. Jerrum, and F. Moller. A Polynomial Algorithm for Deciding Bisimilarity of Normed Context-Free Processes. In *FOCS '94*, pages 623–631. IEEE Computer Society Press, 1994.

[HKP82] D. Harel, D. Kozen, and R. Parikh. Process Logic: Expressiveness, Decidability and Completeness. *Journal of Computer and System Sciences*, 25:144–201, 1982.

[HM94] Y. Hirshfeld and F. Moller. A Fast Algorithm for Deciding Bisimilarity of Normed Context-Free Processes. In *CONCUR '94*, LNCS 836, pages 48–63. Springer, 1994.

[Hoa69] C.A.R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–583, 1969.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice–Hall, 1985.

[HS91] H. Hüttel and C. Stirling. Actions Speak Louder than Words: Proving Bisimilarity for Context-Free Processes. In *LICS '91*, pages 376–386. IEEE Computer Society Press, 1991.

[HS93] H. Hungar and B. Steffen. Local Model-Checking for Context-Free Processes. In *ICALP '93*, LNCS 700, pages 593–605, 1993.

[HT94] D.T. Huynh and L. Tian. Deciding Bisimilarity of Normed Context-Free Processes is in Σ_2^p. *Theoretical Computer Science*, 123:183–197, 1994.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Hüt91] H. Hüttel. *Decidability, Behavioural Equivalences and Infinite Transition Graphs*. PhD thesis, University of Edinburgh, Dec 1991. CST-86-91.

[KH66] A.J. Korenjak and J.E. Hopcroft. Simple Deterministic Languages. In *7 th Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.

[Kna28] B. Knaster. Un théorème sur les fonctions d'ensembles. *Ann. Soc. Pol. Math.*, 6:133–134, 1928.

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Lam94] L. Lamport. Verification and Specification of Concurrent Programs. In *A Decade of Concurrency, REX School/Symposium*, LNCS 803, pages 347–374. Springer, 1994.

[Lar88] K.G. Larsen. Proof Systems for Hennessy–Milner Logic with Recursion. In *CAAP '88*, LNCS 299, pages 215–230. Springer, 1988.

[Len96] G. Lenzi. A Hierarchy Theorem for the Mu-Calculus. In *ICALP '96, LNCS 1099*, pages 87–109. Springer, 1996.

[LNS82] J.L. Lassez, V.L. Nguyen, and E.A. Sonenberg. Fixed Point Theorems and Semantics: A Folk Tale. *Information Processing Letters*, 14(3):112–116, 1982.

[Mad92] E. Madelaine. Verification Tools from the CONCUR Project. *European Association for Theoretical Computer Science*, 47:110–128, June 1992.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer, 1980.

[Mil84] R. Milner. A Complete Inference System for a Class of Regular Behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice–Hall, 1989.

[MM94] S. Mauw and H. Mulder. Regularity of BPA-Systems is Decidable. In *CONCUR '94*, LNCS 836, pages 34–47. Springer, 1994.

[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.

[NC94] M. Nielsen and Ch. Clausen. Bisimulation for Models in Concurrency. In *CONCUR '94*, LNCS 836, pages 385–400, 1994.

[Niw86] D. Niwinski. On fixed-point clones. In *ICALP '86*, LNCS 226, pages 464–473. Springer, 1986.

[Niw88] D. Niwinski. Fixed Points vs. Infinite Generation. In *LICS '88*, pages 402–409. IEEE Computer Society Press, 1988.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In *5th GI Conference*, LNCS 104, pages 167–183. Springer, 1981.

[PI93] S. Purushothaman Iyer. A Note on Model Checking Context-Free Processes. In *NAPAW '93, Technical Report TR 93-1369.* Cornell, Ithaca, NY, 1993.

[Plo81] G. Plotkin. A Structural Approach to Operational Semantics. Daimi FN-19, Aarhus University, Computer Science Department, 1981.

[Rab69] R.O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the AMS*, 141:1–35, 1969.

[Sén97] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. Accepted for ICALP '97, 1997.

[Sti92] C. Stirling. Modal and Temporal Logics. In *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford Science Publications, 1992.

[Sti93] C. Stirling. Modal and Temporal Logics for Processes. Notes for Summer School in Logic Methods in Concurrency, Department of Computer Science, Aarhus University, 1993.

[Sti96] C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. In *CONCUR '96*, LNCS 1119, pages 217–232. Springer, 1996.

[Str82] R.S. Street. Propositional Dynamic Logic of Looping and Converse is Elementary Decidable. *Information and Computation*, 54:121–141, 1982.

[SW89] C. Stirling and D. Walker. Local Model Checking in the Modal Mu-Calculus. In *TAPSOFT '89*, LNCS 351, pages 369–383. Springer, 1989.

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tho90] W. Thomas. Automata on Infinite Objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–191. Elsevier Science Publishers B.V., 1990.

[Wal93] I. Walukiewicz. On Completeness of the μ-calculus. In *LICS '93*, pages 136–146. IEEE Computer Society Press, 1993.

[Wec92] W. Wechler. *Universal Algebra for Computer Scientists*. Springer, 1992.

[Win89] G. Winskel. A Note on Model Checking the Modal Mu-Calculus. In *ICALP '89*, LNCS 372, pages 761–772. Springer, 1989.

Index

O. Burkart: Automatic Verification of Sequential Infinite-State Processes, LNCS 1354 , pp. 161-163, 1997