Decidability, Behavioural Equivalences and Infinite Transition Graphs

Hans Hüttel

Doctor of Philosophy

University of Edinburgh

1991

Abstract

This thesis studies behavioural equivalences on labelled infinite transition graphs and the role that they can play in the context of modal logics and notions from language theory. A natural class of such infinite graphs is that corresponding to the SnS -definable tree languages first studied by Rabin. We show that a modal mu-calculus with label set $\{0, \ldots, n-1\}$ can define these tree languages up to an observational equivalence.

Another natural class of infinite transition graphs is that of normed BPA processes, which correspond to the graphs of leftmost derivations in context-free grammars without useless productions. A remarkable result is that strong bisimulation is decidable for these graphs. After an outline of the existing proofs due to Baeten et al. and Caucal we present a much simpler proof using a tableau system closely related to the branching algorithms employed in language theory following Korenjak and Hopcroft. We then present a result due to Colin Stirling, giving a weakly sound and complete sequent-based equational theory for bisimulation equivalence for normed BPA processes from the tableau system. Moreover, we show how to extract a fundamental relation (as in the work of Caucal) from a successful tableau.

We then introduce silent actions and consider a class of normed BPA processes with the restriction that processes cannot terminate silently, showing that the decidability result for strong bisimilarity can be extended to van Glabbeek's branching bisimulation equivalence for this class of processes.

We complete the picture by establishing that *all* other known behavioural equiv-

alences and a number of preorders are undecidable for normed BPA processes.

Acknowledgements

First of all I want to thank my supervisor Colin Stirling for our fruitful discussions and his many useful comments that greatly influenced the contents and presentation of this thesis.

Thanks are also due to Didier Caucal for his important insights into the topics of Chapters 4 to 6 that inspired much of my work; in particular thanks for pointing out some serious errors in an early version of what was to become Chapter 6. I also want to thank him, his wife Catherine and Roland Monfort for their immense hospitality during my visit to IRISA in June 1990. – Had it not been for Jan Friso Groote and the discussions we had during his visit to the LFCS in May of 1991, Chapter 6 would have been very short and boring. Many of the results in that chapter are due to him. My work on branching bisimulation in Chapter 5 began as the result of a discussion I had in Aalborg with Kim Larsen. Example 5.2.2 is his.

The Department of Computer Science at Edinburgh provided me with an interesting work environment and I want to thank all the people I got to know there during my stay. Special thanks to Kees and Fabio for being such good office-mates.

An important aspect of going abroad is that you get to make new friends. My warm thanks go to Mads and Charlotte for providing me with a proper perspective of many things in life, to Bjarne for drinks osv. and for keeping me in touch with things Danish, to David for meals and Japanese film and theatre (which will *never* be the same) and to Sonia for food, company etc.

And thanks to Eduardo, Nigel, Hans Jørgen & Nomi and last – but by no means least – Søren for sharing accommodation with me at various times during my years in Edinburgh and for generally tolerating my strange whims.

The constant emotional and practical support that my mother has given me throughout my self-imposed exile and whenever I was in Denmark has been all--important; without her, things could have looked very grim indeed and I cannot thank her enough for being there.

My grandmother did not live to see the end of my stay in Edinburgh; it is to the memory of her that my thesis is dedicated.

*

The work in this thesis was made possible by a research position at the Department of Mathematics at Århus University, travel grants and payment of research costs by Aalborg University Centre and C.W. Obel Fonden and financial support from the Danish Research Academy.

Declaration

This is the revised version of my thesis incorporating the required corrections suggested by my examiners Robin Milner and Matthew Hennessy. The thesis was composed by myself, and the work reported has not been presented for any university degree before. The ideas and results that I do not attribute to others are my own.

Parts of the thesis have already been published elsewhere. Chapter 3 is a slightly revised version of [Hüt90]. Chapter 4 contains an expanded version of [HS91]. Chapter 5 is essentially [Hüt91], and Chapter 6 is essentially [GH91].

Hans Hüttel

Table of Contents

Chapter 1

Introduction

The problem of determining if a program satisfies a given specification is one of the central motivating problems in theoretical computer science, and several approaches exist.

Denotational semantics can serve as a valuable tool for program verification. But in the case of nondeterministic, parallel or non-terminating programs an operational account is often preferred. Over the past decade much attention has been devoted to the study of process calculi such as CCS [Mil80,Mil89], ACP [BK84,BK88] and CSP [Hoa88]. Of particular interest has been the study of the behavioural semantics of these calculi as given by transition graphs arising from structural operational semantics in the tradition originated by [Plo81]. A particularly important question is when processes can be said to exhibit the same behaviour, and a plethora of *behavioural equivalences* exist today.

1.1 Determining the qualities of behavioural equivalences

The main rationale behind the various behavioural equivalences that have been proposed has been to capture behavioural aspects that the language equivalence known from language theory does not take into account. For instance,

$$ab + ac = a(b + c)$$

holds for language equivalence, but is an identification that these other notions of behaviour and behavioural equivalence do not make, since these two process expressions do not exhibit the same deadlock properties: after an initial a-action the former is only able to perform one action (b or c), whereas the latter has a choice between b and c.

Equivalences are usually classified according to their *coarseness*, i.e. how many identifications they make with respect to the branching behaviour of processes. This *linear/branching time hierarchy* is illustrated in Figure 1–1 (after [vG90a]) . The coarsest equivalences are then the trace equivalence and the completed trace equivalence; the latter differs from trace equivalence in that only the *completed* trace languages are compared and is thus the usual language equivalence. Directly above we have the testing/failures equivalence investigated by Hennessy and deNicola et al. (see e.g. [Hen89]). At the top of the diagram is bisimulation equivalence (or bisimilarity), a notion introduced by Park in [Par81] and subsequently used by Milner and others in the CCS tradition (as exemplified in [Mil89]).

It has also been argued that behavioural equivalences should be judged according to how well they obey some computationally justifiable criterion of *observability*. For instance, while bisimulation equivalence has many nice mathematical properties it fails to have a computational justification in that (in)equivalence is not intuitively observable. Indeed, within the framework of testing some very unintuitive testing operators must be used. Abramsky has shown [Abr87] that bisimulation can be characterized by a test language that contains a testing operator that enumerates all next-states of the process subjected to testing. In [Gro89] Groote presents another test language; here lookahead in combination with the possibility to check for the absence of activity is needed. Bloom et al. argue [BIM90] that only completed traces should count as observations and define an equivalence which is a (completed) trace congruence under a 'reasonable' set of process constructs. This equivalence is the ready simulation equivalence of Figure 1–1.

The study of processes with unobservable actions leads to another linear/branching time hierarchy similar to Figure 1–1 except that we now have the corresponding *weak equivalences* based on the weak transition relation $\stackrel{a}{\Longrightarrow}$ where unobservable actions are disregarded. The best-known weak equivalence is Milner's weak bisimulation equivalence [Mil89]; however, it has been argued that this is not the proper weak version of bisimilarity since it does not reflect the changes in branching properties that may happen as the result of performing an unobservable action. Moreover, weak bisimulation is not robust under a simple notion of action refinement.

The notion of branching bisimilarity, put forward by van Glabbeek and Weijland in [vGW89b], reflects these concerns. According to this definition, all intermediate unobservable steps in a weak transition must be matched. The idea is not new, however. De Nicola and Vaandrager have proved [DNV90] that branching bisimulation corresponds to the stuttering equivalence on Kripke structures considered in e.g. [BCG88]. Recently it has been shown [DNMV90] that there is a natural connection between branching bisimulation and weak bisimulation in that the former is the so-called 'back and forth' variant of the latter. Branching bisimilarity has many pleasant properties; in particular a complete equational theory for finite BPA, the class of finite processes with choice and sequential composition, can be obtained by adding just two new axioms to those for strong bisimulation for finite BPA processes. Moreover, this axiomatization can easily be turned into a complete term rewriting system [vG90a], something that is not the case for weak bisimulation equivalence.

Finally, there is a similar hierarchy of *preorders* on processes which is yet to be determined in detail. Examples of such preorders include the simulation preorder [Par81], the testing/failures preorder [Hen89] and the ready simulation preorder of [BIM90].

1.2 Infinite-state systems

Milner [Mil84] has shown that the class of finite transition graphs corresponds to that of the transition graphs for *regular processes*, i.e. the recursively defined CCS processes over the signature $\{a., +\}$ where $a.$ is an action prefixing operator for every a in a set of atomic actions Act and $+$ is nondeterministic choice. Regular CCS processes correspond to the usual finite automata; their finitary trace languages are the finitary regular languages.

However, this result also says that as soon as we move beyond these constructs (known as the dynamic process constructs), recursively defined processes can have transition graphs with infinitely many states and trace languages that are no longer regular. This includes many realistic cases; in particular processes that are defined using various notions of parallel composition such as the asynchronous parallel operator $|$ of CCS can have infinitely many states.

In fact, the theory of finite-state systems and their equivalences can now be said to be well-established. One may be led to wonder what the results will look like for *infinite-state* systems.

1.3 Behavioural equivalences and program logics

There is a striking relationship between behavioural equivalences and some program logics. A *modal characterization* of bisimilarity exists [HM85] in that two processes are bisimilar iff they satisfy the same formulae in a modal logic now known as Hennessy-Milner logic. Many other related modal and temporal logics also characterize bisimulation equivalence in this way [BCG88,Sti87,Sti91]. Similar characterizations exist for the other parts in the linear/branching time hierarchy; in the logics that characterize these equivalences either restrictions of Hennessy-Milner logic are made or operators of linear-time temporal logic are introduced in place of the Hennessy-Milner modalities.

Indeed, for the description of program properties the tendency is to prefer modal and temporal logics for describing properties of programs. One such logic is the modal mu-calculus [Koz83] , which also characterizes bisimilarity in the sense that two processes are bisimilar iff they satisfy the same closed formulae [Sti91].

1.3.1 Tableau techniques

An important problem in the context of program logics is that of *model checking*. Model checking consists in determining whether a process state p satisfies a formula F – written $p \models F$. In [SW89] Stirling and Walker have given a model checker for the modal mu-calculus and finite transition graphs in the form of a *tableau system*, a goal-directed proof system for the relation $p \models F$. An advantage of the tableau-based approach to model checking is that it is *local* in the sense that only those states relevant to determining whether or not $p \models F$ need to be examined. In [BS90] Bradfield and Stirling have given a tableau system that deals with infinite transition graphs. In this thesis we use a related tableau technique approach to look at decision problems for behavioural equivalences. Our approach turns out also to be closely related to the branching algorithms for equivalences studied in formal language theory. The method, introduced by Korenjak and Hopcroft in [KH66] has been widely applied for giving decision procedures for various equivalence problems – see e.g. [Cou83].

1.3.2 Expressiveness

The modal mu-calculus is a very expressive logic; it incorporates the full expressibility of CTL^* [Dam90] and thus serves as a natural branching time logic for expressing properties of processes. Moreover, the modal mu-calculus is decidable, in fact even elementary [ES84]. Another important decidable theory, in fact one of the most general decidable theories around, is the second-order monadic theory of n successors, SnS , as introduced by Rabin [Rab69]. SnS is the generalization of $S1S$, the second-order monadic theory of 1 successor, which was shown to be decidable by Büchi in [Büc60] by automata-theoretic means similar to those later used by Rabin.

Since SnS is a very general theory, several other theories have been shown to be decidable by interpretations into SnS [Rab69,Rab77]; examples include the weak second-order theory of linearly ordered sets, the second-order theory of totally ordered sets with a countable domain and various propositional modal logics [Rab77]. The class of SnS-definable sets corresponds to that of the sets of infinite n-ary node-labelled trees accepted by Rabin automata [Rab69] and is a well-known class of infinite-state systems. The acceptance condition of a Rabin automaton can be seen as describing a fairness property along tree paths. Thus it would seem that the SnS-definable sets can be defined through a tree property that can be described in a somewhat more natural way, namely through using a branching-time temporal logic, since such a logic can be interpreted on infinite trees in a straightforward fashion.

So since SnS is a very powerful decidable theory and the modal mu-calculus also is very powerful, subsuming many well-known modal and temporal logics, a natural and interesting question is how these two logics are related with respect to expressiveness.

Some work has already been done in this field. In [VWS83] Vardi et al. show that the temporal logic ETL [Wol83] can define exactly the class of ω-regular languages, corresponding to the $S1S$ -definable sets. And in [Niw88] it was shown by Niwinski that a fixed-point calculus whose signature apart from maximal and minimal fixed points and disjunction includes the usual operators on trees can define exactly the SnS-definable sets.

Finally, Hafer and Thomas have proved [HT87] that a restricted version of SnS with set quantification restricted to paths is expressively equivalent to CTL^* for binary tree models. However, there are certainly bound to be differences. For one thing, the full SnS is non-elementary [Mey75], whereas the modal mu-calculus is elementary [ES84]. Moreover, as was also shown in [HT87] the full SnS can express properties which have no correlate in a branching time temporal logic which does not have operators that incorporate information about the ordering of nodes in a tree. An example is counting the nodes in a tree which are incomparable to a node x w.r.t. to the ancestral ordering, $\leq$:

$$A(x) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_m. \bigwedge_{i=1}^{m} (x \not\leq x_i \wedge x_i \not\leq x) \wedge \bigwedge_{i,j} (i \neq j \supset x_i \neq x_j)$$

This problem would not arise if we could somehow refer to the ancestral ordering in our modal operators. In fact, in this thesis we show that a modal mu-calculus with label set $\{0, \ldots, n-1\}$ can characterize SnS up to a bisimulation-like equivalence on node-labelled trees.

1.4 Decidability of behavioural equivalences

Language equivalence is known to be decidable for finite automata. However, it is also well known (see e.g. [HU79]) that language equivalence becomes undecidable when one moves beyond finite automata to context-free languages.

For finite-state processes all of the equivalences of Figure 1–1 can be seen to be decidable. For instance, the bisimilarity problem $p \sim q$ for processes p and q is decidable for finite transition graphs because we can enumerate all the finitely many binary relations over the state set and search for a bisimulation among them containing the pair (p, q). Moreover, for regular CCS complete equational theories exist for strong bisimilarity [Mil84].

In this thesis we argue that *decidability* or lack thereof should be thought of as another criterion for determining the computational merits and deficiencies of behavioural equivalences.

A natural question is then whether the decidability can be extended beyond the finite-state case. One limitation that should be noted is that strong bisimulation equivalence ($\sim$) in a process language with general static constructs and recursive definitions is *undecidable*. In fact, the general bisimilarity problem is not even r.e. For, using the operators communication ($|$), restriction ($\backslash$) and sequential composition one can code any Turing machine M and input string w as a process expression $p_{M,w}$ such that all moves of M are represented by internal (τ-)actions of $p_{M,w}$ and such that the possible eventual halting is represented by a special success action d. The encoding consists in expressing the tape of M as two stacks, one of which has been initialized to hold w. The stacks communicate with the finite control, represented by a regular process. The problem 'Does M diverge on input w ?' can now be expressed as

$$p_{M,w} \sim \mu x.(\tau.x) \tag{1.1}$$

However, (1.1) is undecidable since the above divergence problem is *not* r.e.

In the setting of process algebra, an example of infinite-state systems is that of the transition graphs of processes in the calculus BPA (Basic Process Algebra) [BK88]. These are recursively defined processes over the signature $\{a, +, .\}$ where a

ranges over a set of atomic actions, $+$ is nondeterministic choice and $\cdot$ is sequential composition.

A BPA process is defined by a system of recursion equations of the form

$$\begin{array}{rcl} X_0 & \stackrel{\mathrm{def}}{=} & E_0(X_0,\ldots,X_n) \\ & \vdots & \\ X_n & \stackrel{\mathrm{def}}{=} & E_n(X_0,\ldots,X_n) \end{array}$$

where the E_i's are BPA expressions. A system of the above form where every occurrence of a variable in any expression E_i is within the scope of an atomic action is said to be *guarded*. Any guarded system of equations can effectively be put in the BPA equivalent of *Greibach Normal Form* (GNF), i.e. a system of equations where all equations are of the form $X_i \stackrel{\mathrm{def}}{=} \sum_j a_{ij}\alpha_{ij}$ where the α_{ij}'s are compositions of process variables.

A special case is that of *normed BPA* processes. The *norm* of a process is defined as the least number of transitions necessary to terminate. A process is said to be normed if every state has a finite norm. Even though normed BPA does not incorporate all regular processes, systems defined in this calculus can in general have infinitely many states.

There is an obvious correspondence between process equations of the form $X_i \stackrel{\mathrm{def}}{=} \sum_j a_{ij}\alpha_{ij}$ and the GNF context-free productions $X \rightarrow a_{i1}\alpha_{i1} \mid \ldots \mid a_{ik}\alpha_{ik}$, so normed BPA processes correspond to context-free grammars without useless or empty productions. It is therefore easy to see that both trace equivalence and completed trace equivalence (or language equivalence) are undecidable for normed BPA processes.

However, a recent result shows that strong bisimilarity for normed BPA processes is *decidable*. Two proofs of this result exist, one by Baeten, Bergstra and Klop [BBK87a] and another due to Caucal [Cau88,Cau90a]. These proofs are very different (and are sketched in Chapter 4). The (lengthy and impenetrable) proof in

[BBK87a] consists in showing that one can exhibit a decomposition of the process graph with certain regularities. The proof in [Cau88,Cau90a] consists in showing that the maximal bisimulation is finitely representable by a confluent and strongly normalizing Thue system, and that there are only finitely many candidates for this Thue system.

These proofs do not correspond to one's intuition about how to determine whether or not two normed BPA processes are bisimilar. And they do not lead to complete proof systems for the process algebra involved. But what these proofs *do* tell us is that we can go beyond finite-state systems while maintaining the decidability of bisimulation equivalence. Moreover, somewhat unexpectedly, we have gained something by using an equivalence different from language equivalence in a setting that involves structures from language theory. On the other hand, Huynh and Tian [HT90] have proved the negative result that the failures and readiness equivalences are undecidable for normed BPA. Their proof consists in giving a special class of normed BPA processes for which these equivalences coincide with language equivalence and then showing that the language equivalence problem for arbitrary normed BPA processes reduces to that for the special class.

A natural question in the light of this is now for which equivalences and for which process signatures we have that the equivalence in question is decidable for normed processes. In this thesis we show that in fact *none* of the other behavioural equivalences in Figure 1–1 are decidable for normed BPA processes.

Related questions are what happens when we introduce *silent actions* and what the situation looks like for *preorders*. For the latter there are bound to be some differences. Friedman has shown [Fri76] that the language inclusion preorder is undecidable for so-called simple grammars, a class of context-free grammars that correspond to that of deterministic normed BPA processes. However, Korenjak and Hopcroft have shown that the language equivalence problem for this class of grammars/processes is decidable [KH66].

Finally, another important question is whether we can find a 'natural' and 'convenient' method of determining whether or not two normed BPA processes are bisimulation equivalent. In this thesis we show that it is indeed possible; our method is a tableau technique related to the local model checking systems of [SW89,BS90]. This method is also closely related to the branching algorithms for equivalence problems on grammars introduced by Korenjak and Hopcroft in [KH66].

We use the same tableau method to prove that the branching bisimulation equivalence of Weijland and van Glabbeek [vGW89a,vGW89b] is decidable for a class of normed BPA processes with silent actions.

1.5 Layout of the Thesis

In this final section we outline the contents of the rest of this thesis.

In **Chapter 2** we present the necessary background material for the chapters that follow. We give the definitions of infinite n-ary trees and Rabin automata (studied in Chapter 3) and introduce in greater detail the process calculus BPA (studied in Chapters 4 to 6) and the notion of bisimulation equivalence. We describe the subclass of normed BPA and show how every system of BPA equations can be effectively rewritten into 3-GNF.

In **Chapter 3** we show that a modal mu-calculus with label set $\{0, \ldots, n-1\}$ can define the SnS-definable n-ary tree languages up to an observational equivalence. The main idea is to use Rabin's theorem stating that the SnS-definable languages correspond to the n-ary Rabin-recognizable tree languages, which are the sets of infinite n-ary labelled trees recognizable by Rabin automata. Thus our result also underpins the idea that equivalences other than those normally used can be of use in problems related to language theory.

In **Chapter 4** we first outline the existing proofs of the decidability of bisimulation equivalence for normed BPA processes due to Baeten, Bergstra and Klop

[BBK87b,BBK87a] and Caucal [Cau88,Cau90a] and then give an alternative and much simpler proof of this result. Our decidability proof uses a tableau system which is similar to the tableau systems used for model-checking the modal mu-calculus [SW89,BS90] and closely related to the branching algorithms of language theory [KH66,Cou83]. If a successful tableau for an equation $\alpha = \beta$ exists, the tableau provides us with a finite witness for a bisimulation containing (α, β), the witness being a self-bisimulation in the sense of [Cau88,Cau90a]. We give a complexity bound for the tableau method in terms of the length of the longest possible path in any tableau for a given equation. Then we present a result due to Colin Stirling, a sequent-based equational theory for bisimulation equivalence for normed BPA processes in 3-GNF extracted from the tableau system. The theory is shown to be strongly sound and weakly complete. Finally we show how one can find a fundamental relation (as in the work of [Cau88,Cau90a]) from a successful tableau. This is done via another so-called auxiliary tableau system.

In **Chapter 5** we introduce silent actions into normed BPA. We consider a class of BPA processes with the restriction that process termination must involve performing an observable action. We then show how the decidability result of Chapter 4 can be extended to branching bisimulation equivalence, giving complexity bounds for the tableau method.

In **Chapter 6** we show that *all* equivalences below bisimulation in the linear/branching time hierarchy are undecidable for normed BPA processes in 3-GNF and thus that they are undecidable for BPA processes in general. The proofs involve reductions to the language inclusion problem for simple grammars of [Fri76] and the language and trace equivalence problems for normed BPA processes.

Chapter 7 sums up the conclusions of this thesis and give directions for further work.

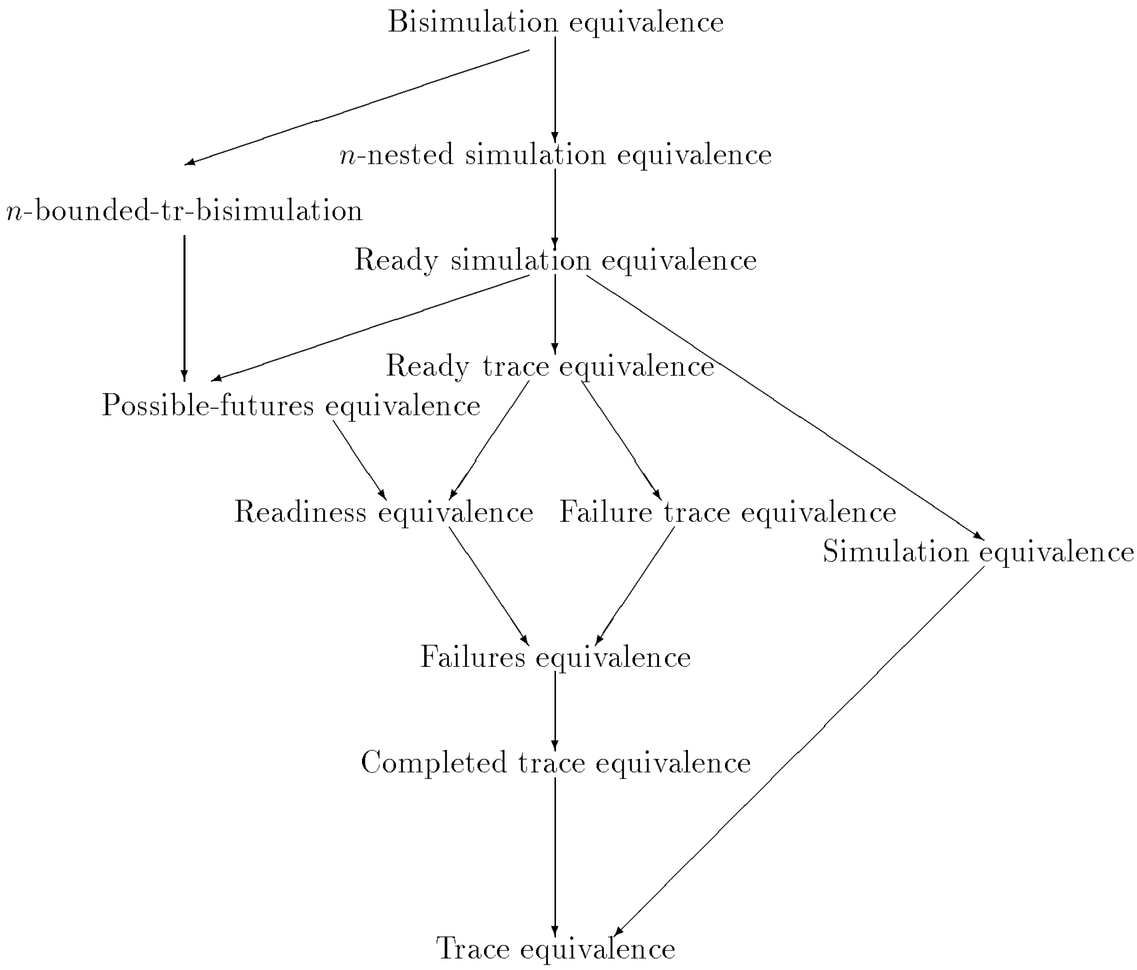

Figure 1–1: The linear/branching time hierarchy of equivalences (based on [vG90a]) .

Chapter 2

Background

In this chapter we give various definitions that will be used throughout the rest of this thesis. Section 2.1 introduces the notions of infinite node-labelled trees, bisimulations on such trees and Rabin automata that will be used in Chapter 3. In Section 2.2 we introduce the class of normed BPA processes studied in Chapters 4 to 6 and the notion of bisimulation equivalence for such processes.

2.1 Infinite trees and Rabin automata

In Chapter 3 we shall look at SnS [Rab69] and a version of the modal mu-calculus [Koz83], both of which are logics interpreted on infinite node-labelled trees of fixed arity.

2.1.1 Infinite trees

An n-ary infinite tree can be seen as a prefix-closed set, with suffixing representing the successor relation.

Definition 2.1.1 *The full infinite n-ary tree is the set $\{0,\ldots,n-1\}^*$ with the successor relation $\rightarrow \subseteq \{0,\ldots,n-1\}^* \times \{0,\ldots,n-1\}^+$ defined by $w \rightarrow wi$ for*

$i \in \{0, \ldots, n-1\}$. *wi is called the ith successor of w. The root is ϵ.*

In other words, any node wx where $x = i_1 i_2 \ldots i_m$ is the unique node reachable from w via the path $w \to wi_1 \to wi_1 i_2 \cdots \to wi_1 i_2 \ldots i_m$.

We shall sometimes need an ordering on the node set.

Definition 2.1.2 *The* ancestral ordering *on $\{0, \ldots, n-1\}^*$ is given by $w' < w''$ if there exists a $w''' \in \{0, \ldots, n-1\}^+$ such that $w'w''' = w''$.*

A labelled tree t whose labels are in the alphabet A is defined as a labelling function on $\{0, \ldots, n-1\}^*$.

Definition 2.1.3 *An n-ary A-labelled tree is a function $t : \{0, \ldots, n-1\}^* \to A$. The set of all A-labelled trees is denoted by $\mathcal{T}_A^\omega$. A set of n-ary A-labelled trees is called a* tree language *over A.*

In the rest of this section we assume without loss of generality that all trees considered are binary, meaning $n = 2$.

We now define a notion of equivalence on trees which states that two nodes have the same branching properties with respect to some subalphabet A'. First we define the notion of an A'-descendant.

Definition 2.1.4 *For any tree $t \in \mathcal{T}_A^\omega$ and $A' \subseteq A$, the A'-*descendant relation *$\underset{A'}{\Rightarrow} \subseteq \{0,1\}^* \times (\{0,1\}^* \times \{0,1\}^*)$ is given by $w \underset{A'}{\Rightarrow} (u0v_0, u1v_1)$ whenever $w < u0v_0$ and $w < u1v_1$ where $t(w') \notin A'$ whenever $w < w' < u0v_0$ or $w < w' < u1v_1$ but $t(u0v_0), t(u1v_1) \in A'$.*

Thus, the A'-descendants of a node w are the first descendants of w labelled by elements from A' along a pair of incomparable paths. For A-labelled trees $\underset{A}{\Rightarrow}$ clearly reduces to $\to$. If $A = A' \cup \{\tau\}$ where τ is a special 'invisible' label, $\tau \notin A'$, $\underset{A'}{\Rightarrow}$ can be seen as the successor relation modulo invisible labels. Thus, the relation is similar to the weak transition relations for edge-labelled transition graphs with silent actions (cf. Definition 5.1.1) and gives rise to an equivalence of nodes that is similar to the observational equivalence of [Mil89] and essentially is the equivalence on trees of [BCG88]:

Definition 2.1.5 *For any trees $t_1 \in \mathcal{T}^{\omega}_{A_1}, t_2 \in \mathcal{T}^{\omega}_{A_2}$ and $A' \subseteq A_1 \cap A_2$, an A-*bisimulation *is a relation $R_{A'} \subseteq \{0,1\}^* \times \{0,1\}^*$ such that whenever $(w', w'') \in R_{A'}$ we have*

1. $t_1(w') = t_2(w'')$ *and* $t_1(w') \in A'$
2. $w' \underset{A'}{\Rightarrow} (w'_0, w'_1) \Longrightarrow \exists (w''_0, w''_1) : w'' \underset{A'}{\Rightarrow} (w''_0, w''_1)$ *with* $w'_0 R w''_0$ *and* $w'_1 R w''_1$
3. $w'' \underset{A'}{\Rightarrow} (w''_0, w''_1) \Longrightarrow \exists (w'_0, w'_1) : w' \underset{A'}{\Rightarrow} (w'_0, w'_1)$ *with* $w'_0 R w''_0$ *and* $w'_1 R w''_1$

*We define $\simeq_{A'}$ by $\simeq_{A'} = \{(w', w'') \mid w' R_{A'} w''$ for some A'-bisimulation $R_{A'}\}$. If $w' \simeq_{A'} w''$ we say that w' and w'' are A'-*bisimilar.

Thus, $\simeq_{A'}$ identifies two trees labelled by alphabets containing A' if their ancestral information w.r.t. A' is the same. We have

Proposition 2.1.1 *$\simeq_{A'}$ is an equivalence relation on $\{0,1\}^* \times \{0,1\}^*$.*

2.1.2 Rabin automata

The use of Rabin automata is crucial to the equi-expressiveness proof in Chapter 3. The Rabin automaton, introduced in [Rab69], is an important type of automaton on infinite trees, first used as an auxiliary notion in the proof of the decidability of SnS. It provides a generalization of the Büchi automata on infinite sequences introduced in Büchi's proof of the decidability of $S1S$ [Büc60].

Definition 2.1.6 *A* Rabin automaton *on binary A-labelled trees is a quadruple $\mathcal{A} = (Q, q_0, \{\stackrel{a}{\to} \mid a \in A\}, \Omega)$, where Q is a finite set of* states, *q_0 is the* start state,

$\{\stackrel{a}{\to} \subseteq Q \times (Q \times Q) \mid a \in A\}$ *is a finite family of finite* transition relations *and* $\Omega \subseteq \mathbf{2}^Q \times \mathbf{2}^Q$ *is a finite collection of finite* acceptance pairs. *Whenever* $(q, (q_1, q_2)) \in \stackrel{a}{\to}$ *we write* $q \stackrel{a}{\to} (q_1, q_2)$.

Rabin automata are thus nondeterministic. This feature is essential; deterministic Rabin automata can be shown to be strictly less powerful than their nondeterministic counterparts [Tho90].

Definition 2.1.7 *A* run *of the Rabin automaton* $\mathcal{A} = (Q, q_0, \{\stackrel{a}{\to} \mid a \in A\}, \Omega)$ *on an* A*-labelled tree* t *is any* Q*-labelled tree* r *such that*

$$r(\epsilon) = q_0$$

$$\textit{if } t(s) = a \textit{ then } r(s) = q, r(s0) = q', r(s1) = q'' \textit{ for some } q \stackrel{a}{\to} (q', q'')$$

Rabin acceptance states that there is a run where every path satisfies a fairness condition in that along any path there is an acceptance pair (L_i, U_i) such that states in U_i occur infinitely often and states in L_i do not. We let $In(r \mid \pi)$ denote the set of states occurring infinitely often along a path π in the run r.

Definition 2.1.8 *A run* r *of* $\mathcal{A}$ *is* accepting *if for all paths* π *in* r *there is an acceptance pair* $(L_i, U_i) \in \Omega$ *such that*

$$In(r \mid \pi) \cap L_i = \emptyset \textit{ and } In(r \mid \pi) \cap U_i \neq \emptyset$$

Definition 2.1.9 *A tree language* L *is* Rabin-recognizable *if there is a Rabin automaton* $\mathcal{A}$ *such that* $t \in L$ *iff* t *admits an accepting run of* $\mathcal{A}$.

Example 2.1.1 For the alphabet $\{a, b\}$ consider the Rabin automaton

$$\mathcal{A} = (\{q_1, q_2\}, q_1, \{q_1 \stackrel{a}{\to} (q_1, q_1), q_1 \stackrel{b}{\to} (q_2, q_2), q_2 \stackrel{a}{\to} (q_1, q_1), q_2 \stackrel{b}{\to} (q_2, q_2)\}, \{(\{q_2\}, \{q_1\})\})$$

The state q_2 is assumed exactly when a b is encountered, so from the acceptance condition we see that the tree language recognized by $\mathcal{A}$ is that of the trees that do not have a path containing infinitely many b's. □

In [Rab69] Rabin proved that a set of trees can be defined in SnS if and only it it is Rabin-recognizable. This result, stated in this thesis as Theorem 3.3.1, provided a generalization of Büchi's result that $S1S$-definability corresponds to the notion of the notion of being Büchi-recognizable and will play a crucial role in Chapter 3.

2.2 Normed recursive BPA processes

In Chapters 4 to 6 we shall look at (edge)-labelled transition graphs that arise from the structural operational semantics of process calculi.

Definition 2.2.1 *A* labelled transition graph $\mathcal{G} = (Pr, Act, \{\stackrel{a}{\rightarrow}\})$ *is a triple consisting of a set of* states *or* processes Pr*, a set of* atomic actions Act *and a family of transition relations* $\{\stackrel{a}{\rightarrow} \subseteq Pr \times Pr \mid a \in Act\}$*. We refer to processes in general by* $p, q, \ldots$ *Whenever* $(p, q) \in \stackrel{a}{\rightarrow}$ *we write* $p \stackrel{a}{\rightarrow} q$*. If there is no* q *such that* $p \stackrel{a}{\rightarrow} q$ *we write* $p \stackrel{a}{\not\rightarrow}$*. The transitive closure* $\{\stackrel{u}{\rightarrow} \mid u \in Act^+\}$ *of the transition relations is defined for* $w \in Act^+$ *by* $p \stackrel{aw}{\rightarrow} q$ *if* $p \stackrel{a}{\rightarrow} p'$ *and* $p' \stackrel{w}{\rightarrow} q$ *for some* p'*.*

In particular we look at the transition graphs defined by the class of guarded recursive normed BPA (Basic Process Algebra) processes (see e.g. [BBK87a,BK88]).

2.2.1 Syntax and semantics

In the rest of this chapter and in Chapters 4 to 6 $E, F, G \ldots$ will be used to denote BPA process expressions. These are given by the abstract syntax

$$E ::= a \mid X \mid E_1 + E_2 \mid E_1 \cdot E_2$$

Here a ranges over a set of atomic actions Act, and X over a family of variables. The operator $+$ is nondeterministic choice while $E_1 \cdot E_2$ is the sequential composition of E_1 and E_2 – we usually omit the '$\cdot$'. A BPA process is defined by a finite system of recursive process equations

$$\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq k\}$$

where the X_i are distinct, and the E_i are BPA expressions with free variables in $Var = \{X_1, \ldots, X_m\}$. One variable (generally X_1) is singled out as the *root*. We shall occasionally write $\Delta_1 R \Delta_2$ for binary relations R; this should be read as stating that the roots of Δ_1 and Δ_2 are related by R. Often we shall only look at relations within the transition graph for a single Δ. We can do so without loss of generality, since we can let Δ be the disjoint union of the Δ_1 and Δ_2 that we are comparing (with suitable renamings of variables, if required); its transition graph is then the disjoint union of those for Δ_1 and Δ_2.

We restrict our attention to *guarded* systems of recursive equations.

Definition 2.2.2 *A BPA expression is* guarded *if every variable occurrence is within the scope of an atomic action. The system $\Delta = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq k\}$ is guarded if all E_i are for $1 \leq i \leq k$.*

Here and in Chapters 4 to 6 we use $X, Y, \ldots$ to range over variables in Var and Greek letters $\alpha, \beta, \ldots$ to range over elements in Var^*. In particular, ϵ denotes the empty variable sequence.

Definition 2.2.3 *Any system of process equations Δ defines a labelled transition graph. The transition relations are given as the least relations satisfying the following rules:*

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

$$\frac{E \xrightarrow{a} E'}{EF \xrightarrow{a} E'F} \qquad a \xrightarrow{a} \epsilon \quad a \in Act$$

$$\frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \quad X \stackrel{\text{def}}{=} E \in \Delta \qquad \frac{E \xrightarrow{a} \epsilon}{EF \xrightarrow{a} F}$$

Finally, the important extra restriction on a family Δ is *normedness.*

Definition 2.2.4 *The* norm *of a BPA expression E is defined as*

$$|E| = \min\{length(w) \mid E \xrightarrow{w} \epsilon, w \in Act^+\}$$

A system of defining equations Δ is normed *if for any variable $X \in Var$ $|X| < \infty$. The maximal norm of any variable in Δ is $m_\Delta = \max\{|X| \mid X \in Var\}$.*

An important property of the norm is that it is additive under sequential composition and for nondeterministic choice corresponds to taking the minimum. This reduces the calculation of norms of variables in a system of process equation to solving systems of equations over the natural numbers.

Proposition 2.2.1 *For BPA expressions E, F we have $|EF| = |E| + |F|$ and $|E + F| = \min(|E|, |F|)$.*

Proof: Let w_E be any shortest string with the property that $E \xrightarrow{w_E} \epsilon$ and let w_F be any shortest string with the property that $F \xrightarrow{w_F} \epsilon$. Clearly, $w_E w_F$ is a shortest string u such that $EF \xrightarrow{u} \epsilon$. Also it is obvious that the shorter of w_E and w_F is the shortest string u with the property that $E + F \xrightarrow{u} \epsilon$. □

As from Section 2.2.4 we restrict our attention to the class of BPA processes in GNF, whose states are all members of Var^*. The following definitions and results are only stated for states that are indeed strings of variables.

Definition 2.2.5 *The language $L(\alpha)$ accepted by $\alpha \in Var^*$ is defined by*

$$L(\alpha) = \{w \in Act^+ \mid \alpha \xrightarrow{w} \epsilon\}$$

We say that α and β are language equivalent *iff $L(\alpha) = L(\beta)$.*

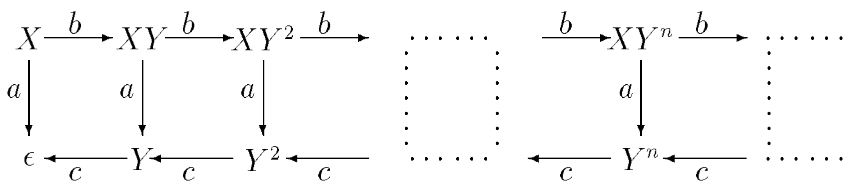

Figure 2–1: Transition graph for $X \stackrel{\text{def}}{=} a + bXY; \quad Y \stackrel{\text{def}}{=} c$ (Example 1)

Definition 2.2.6 *The set of traces $Tr(\alpha)$ for $\alpha \in Var^*$ is given by*

$$Tr(\alpha) = \{w \in Act^+ \mid \alpha \stackrel{w}{\rightarrow}\}$$

We say that α and β are trace equivalent *iff $Tr(\alpha) = Tr(\beta)$.*

Example 2.2.1 Consider the system $\Delta = \{X \stackrel{\text{def}}{=} a + bXY; \quad Y \stackrel{\text{def}}{=} c\}$. Here $|X| = |Y| = 1$, so Δ is normed. By the transition rules in Definition 2.2.3 X generates the transition graph in Figure 2–1. We have that $L(Y) = \{c\}$ whereas $L(X) = \{b^n ac^n \mid n \geq 1\}$ and $Tr(Y) = \{c\}$ but $Tr(X) = \{b^n \mid n \geq 1\} \cup \{b^n ac^j \mid j \leq n, n \geq 1\}$. □

Example 2.2.2 The system of equations $\Delta = \{X \stackrel{\text{def}}{=} aX; \quad Y \stackrel{\text{def}}{=} c + aX\}$ is *not* normed, since there is no w such that $X \stackrel{w}{\rightarrow} \epsilon$. □

Thus, because of the normedness restriction, normed BPA does not include all regular processes. Nevertheless, it is a very rich family with processes that can have infinitely many states even after quotienting by any behavioural equivalence in the linear/branching time hierarchy.

For instance, in Example 2.2.1 above, for any two distinct states α_1 and α_2 we have $Tr(\alpha_1) \neq Tr(\alpha_2)$, so no two distinct states are related by *any* equivalence in the linear/branching time hierarchy, since they are not even trace equivalent. Thus, the transition graph is left unchanged with infinitely many states after quotienting by any such equivalence.

2.2.2 Bisimulation equivalence on BPA processes

In Chapter 4 we prove that bisimulation equivalence [Par81,Mil89] over normed BPA processes is decidable.

Definition 2.2.7 *A relation R between processes is a* bisimulation *if whenever pRq then for each $a \in Act$*

1. $p \xrightarrow{a} p' \Rightarrow \exists q' : q \xrightarrow{a} q'$ *with* $p'Rq'$
2. $q \xrightarrow{a} q' \Rightarrow \exists p' : p \xrightarrow{a} p'$ *with* $p'Rq'$

We define $\sim$ by $\sim = \{(p, q) \mid pRq$ for some bisimulation $R\}$. If $p \sim q$, p and q are said to be bisimulation equivalent *or* bisimilar.

Proposition 2.2.2 [BK88] $\sim$ *is a congruence relation w.r.t.* $+$ *and* $\cdot$

Proposition 2.2.3 *For any normed system Δ, $\alpha \sim \beta$ implies that $L(\alpha) = L(\beta)$, and thus also $|\alpha| = |\beta|$.*

Example 2.2.3 An example of bisimilar BPA process expressions is given by $\{X \stackrel{\text{def}}{=} aYX + b, Y \stackrel{\text{def}}{=} bX, A \stackrel{\text{def}}{=} aC + b, C \stackrel{\text{def}}{=} bAA\}$. We have that $X \sim A$, since the relation $\{(X^n, A^n) \mid n \geq 0\} \cup \{(YX^{n+1}, CA^n) \mid n \geq 0\}$ is a bisimulation (where V^n here denotes n successive Vs, $V \in Var$). □

2.2.3 Axiomatizations of bisimulation equivalence

In the usual presentation of BPA (see e.g. [BK88]), much attention is usually devoted to the so-called BPA laws – presented here in Table 2–1. The BPA laws

$$E_1 + E_2 = E_2 + E_1 \qquad \text{A1}$$
$$(E_1 + E_2) + E_3 = E_1 + (E_2 + E_3) \qquad \text{A2}$$
$$E_1 + E_1 = E_1 \qquad \text{A3}$$
$$(E_1 + E_2)E_3 = E_1E_3 + E_2E_3 \qquad \text{A4}$$
$$(E_1E_2)E_3 = E_1(E_2E_3) \qquad \text{A5}$$

Table 2–1: The BPA laws

are easily shown to be sound w.r.t. bisimilarity[1], irrespective of any restrictions on the processes involved.

Proposition 2.2.4 [BK88] *For any BPA expressions E_1, E_2 and E_3 we have that $E_1+E_2 \sim E_2+E_1$, $(E_1+E_2)+E_3 \sim E_1+(E_2+E_3)$, $E_1+E_1 \sim E_1$, $(E_1+E_2)E_3 \sim E_1E_3+E_2E_3$ and $(E_1E_2)E_3 \sim E_1(E_2E_3)$.*

The BPA laws do not form a complete axiomatization of BPA; some notion of fixed-point induction must be added in order to prove equations involving recursively defined processes. In Section 4.3 we show how such an induction principle arises from the tableau method used to decide strong bisimilarity and use it together with an encoding of the BPA laws and congruence laws of Proposition 2.2.2 to give an equational theory for normed BPA processes in 3-GNF (see below).

[1] In fact the BPA laws are sound for *all* equivalences in the linear/branching hierarchy [vG90a].

2.2.4 Normed recursive BPA processes in Greibach Normal Form

Any system Δ of guarded BPA equations has a unique solution up to bisimulation equivalence [BK84]. Moreover, in [BBK87a] it is shown that any such system can be effectively presented in what we here call *Greibach Normal Form*. In this thesis we restrict our attention to normed BPA processes given in 3-GNF.

Definition 2.2.8 *A system of BPA equations Δ is said to be in* Greibach Normal Form (GNF) *if all equations are of the form*

$$\{X_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} a_{ij}\alpha_{ij} \mid 1 \leq i \leq m\}$$

If for each i, j the variable sequence α_{ij} has $length(\alpha_{ij}) < k$, Δ is said to be in k-GNF.

The normal form is called Greibach Normal Form by analogy with context-free grammars (without the empty production) in Greibach Normal Form (see e.g. [HU79]). There is an obvious correspondence with grammars in GNF: process variables correspond to non-terminals, the root is the start symbol, actions correspond to terminals, and each equation $X_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} a_{ij}\alpha_{ij}$ can be viewed as the family of productions $\{X_i \rightarrow a_{ij}\alpha_{ij} \mid 1 \leq j \leq n_i\}$. The notion of normedness says that the grammar must not have useless productions. It is well-known that any context-free language (without the empty string) is generated by a grammar in 3-GNF [HU79]. One should also notice that for systems in GNF, a transition step in the operational semantics of Definition 2.2.3 corresponds to a leftmost derivation step in the corresponding grammar.

Theorem 2.2.1 [BBK87a] *If Δ is a guarded system of BPA equations, we can effectively find a system Δ' in 3-GNF such that $\Delta' \sim \Delta$. Moreover, when Δ is normed, so is Δ'.*

PROOF: An effective procedure for rewriting Δ into 3-GNF consists in first rewriting Δ into GNF and then rewriting the resulting system into 3-GNF. (We assume that the right-distributive law A4 (Figure 2–1) has been applied (from left to right) as far as possible.)

For the rewriting into GNF, we first replace all internal occurrences of atomic actions by equations. Thus, for each atomic action a occurring in the definitions introduce a new variable X_a, replace as many occurrences of a as possible while keeping the resulting system guarded and add the equation $X_a \stackrel{\text{def}}{=} a$ to Δ.

We then remove all unresolved sums from the outside in. An *unresolved sum* is a sum $F + G$ occurring in an expression of the form $E(F + G)$. We now repeat the following loop until there are no unresolved sums left:

- For each outermost unresolved sum $F+G$ introduce a new variable X_{F+G} and replace all occurrences of $F + G$ by X_{F+G}. Add the equation $X_{F+G} \stackrel{\text{def}}{=} F + G$ to Δ. (This may make the resulting system unguarded.) Call the equations added in traversal i of the loop the ith *stratum* (letting the original Δ be stratum 0).

After all unresolved sums have been removed, all equations in the resulting system are of the form

$$X_i \stackrel{\text{def}}{=} \sum_j a_{ij}\alpha_{ij} + \sum_l \alpha_{il}$$

We then make all unguarded summands guarded. Notice that all variables introduced in stratum i have definitions using only variables in strata $< i$. We replace unguarded variables by their definitions, using the following loop. It is easy to see that when we reach stratum i all equations in strata $< i$ are now guarded.

- For each successive stratum do the following: For every equation in the stratum, for any unguarded summand $X'_{kl}\alpha'_{kl}$ replace X'_{kl} by its definition $\sum_{ij} a_{ij}\alpha_{ij}$ and use the right-distributive law (A4) to obtain the new summand $\sum_{ij} a_{ij}\alpha_{ij}\alpha'_{kl}$.

We now have a system of process equations in k-GNF for some k. We can then rewrite the system into 3-GNF in the following way. We introduce a new 'pair variable' U_{XY} for every occurring variable pair XY, adding the equation $U_{XY} \stackrel{\text{def}}{=} XY$. We then replace every occurrence of XY by U in each equation, going from left to right. For each of the new unguarded equations $U_{XY} \stackrel{\text{def}}{=} XY$ we use the same trick as above: X is replaced by its definition and A4 is applied. This may have introduced new instances of the variable pairs, which we then have to replace by appropriate 'pair variables'. The resulting system is now in $\lceil \frac{k}{2} \rceil$-GNF. The whole procedure is repeated until we reach 3-GNF.

Since all steps used in the algorithm described here either simply introduce new variables that rename expressions or use the BPA laws, we see that bisimilarity and thus normedness must be preserved, so clearly $\Delta' \sim \Delta$. □

Example 2.2.4 Let us rewrite the system Δ:

$$\begin{aligned} X &\stackrel{\text{def}}{=} a(Y + ZX) + aXb \\ Y &\stackrel{\text{def}}{=} aZ(Y + bXXX) + aZ \\ Z &\stackrel{\text{def}}{=} a \end{aligned}$$

in 3-GNF.

After removing internal occurrences of actions, it becomes

$$\begin{aligned} X &\stackrel{\text{def}}{=} a(Y + ZX) + aXX_b \\ Y &\stackrel{\text{def}}{=} aZ(Y + X_bXXX) + aZ \\ Z &\stackrel{\text{def}}{=} a \\ X_b &\stackrel{\text{def}}{=} b \end{aligned}$$

We then remove unresolved sums, getting

$$\begin{aligned}
X &\stackrel{\text{def}}{=} aX_{Y+ZX} + aXX_b \\
Y &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ \\
Z &\stackrel{\text{def}}{=} a \\
X_b &\stackrel{\text{def}}{=} b \\
X_{Y+ZX} &\stackrel{\text{def}}{=} Y + ZX \\
X_{Y+X_bXXX} &\stackrel{\text{def}}{=} Y + X_bXXX
\end{aligned}$$

After we have got rid of all unguarded sums, we have

$$\begin{aligned}
X &\stackrel{\text{def}}{=} aX_{Y+ZX} + aXX_b \\
Y &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ \\
Z &\stackrel{\text{def}}{=} a \\
X_b &\stackrel{\text{def}}{=} b \\
X_{Y+ZX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + aX \\
X_{Y+X_bXXX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + bXXX
\end{aligned}$$

This system is in 4-GNF. We introduce $U_{XX} \stackrel{\text{def}}{=} XX$ and $U_{XX_b} \stackrel{\text{def}}{=} X_bX$ and get

$$\begin{aligned}
X &\stackrel{\text{def}}{=} aX_{Y+ZX} + aXX_b \\
Y &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ \\
Z &\stackrel{\text{def}}{=} a \\
X_b &\stackrel{\text{def}}{=} b \\
X_{Y+ZX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + aX \\
X_{Y+X_bXXX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + bXXX \\
U_{XX} &\stackrel{\text{def}}{=} XX
\end{aligned}$$

$$U_{XX_b} \stackrel{\text{def}}{=} X_b X$$

which then becomes

$$\begin{aligned}
X &\stackrel{\text{def}}{=} aX_{Y+ZX} + aXX_b \\
Y &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ \\
Z &\stackrel{\text{def}}{=} a \\
X_b &\stackrel{\text{def}}{=} b \\
X_{Y+ZX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + aX \\
X_{Y+X_bXXX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + bUX \\
U_{XX} &\stackrel{\text{def}}{=} aX + aXX_bX \\
U_{XX_b} &\stackrel{\text{def}}{=} bX
\end{aligned}$$

finally arriving at Δ', which is

$$\begin{aligned}
X &\stackrel{\text{def}}{=} aX_{Y+ZX} + aXX_b \\
Y &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ \\
Z &\stackrel{\text{def}}{=} a \\
X_b &\stackrel{\text{def}}{=} b \\
X_{Y+ZX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + aX \\
X_{Y+X_bXXX} &\stackrel{\text{def}}{=} aZX_{Y+X_bXXX} + aZ + bUX \\
U_{XX} &\stackrel{\text{def}}{=} aX + aU_{XX_b}X \\
U_{XX_b} &\stackrel{\text{def}}{=} bX
\end{aligned}$$

□

Because of the correspondence with context-free grammars, we immediately see that language equivalence (or completed trace equivalence) is undecidable for normed BPA processes. This follows directly from the result for context-free grammars (see e.g. [HU79]). In our terminology this result reads as follows:

Theorem 2.2.2 *For any normed system Δ of BPA process equations in GNF it is undecidable whether $L(\alpha) = L(\beta)$ for $\alpha, \beta \in Var^*$.*

An easily established consequence is that trace equivalence is undecidable.

Theorem 2.2.3 *For any normed system Δ of BPA process equations in GNF it is undecidable whether $Tr(\alpha) = Tr(\beta)$ for $\alpha, \beta \in Var^*$.*

PROOF: We can reduce language equivalence to trace equivalence, since we have $L(\alpha) = L(\beta)$ iff $Tr(\alpha\surd) = Tr(\beta\surd)$ where $\surd$ is a new action (this is an observation due to Lambert Meertens). □

An important advantage of using GNF is that the states in the transition graph for a process given in this way are elements of Var^*. Moreover, the restriction to variable sequences of length at most 2 guarantees limited growth of these sequences under single transitions. When applying a defining equation to the leftmost variable in a string α the length of the derivative increases by at most 1:

Proposition 2.2.5 *Suppose Δ is in 3-GNF. Then, for any $\alpha \in Var^*$, whenever $\alpha \xrightarrow{a} \alpha'$ we have $length(\alpha') \leq length(\alpha) + 1$.*

PROOF: Suppose $\alpha = X_i\alpha'''$. Then $\alpha \xrightarrow{a} \alpha'$ must be due to $X_i \xrightarrow{a} \alpha''$. This in turn is due to the defining equation $X_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} a_{ij}\alpha_{ij}$ having a summand $a\alpha''$ with $length(\alpha'') \leq 2$. Since $\alpha' = \alpha''\alpha'''$, the result follows. □

Finally, the following simple relationship between lengths and norms for variable sequences becomes particularly useful in Chapter 3.

Proposition 2.2.6 *For $\alpha \in Var^*$ $length(\alpha) \leq |\alpha|$ and $|\alpha| \leq m_\Delta length(\alpha)$.*

2.2.5 Self-bisimulations

For finite-state processes a naive decision procedure for the bisimulation problem $p \sim q$ consists in enumerating all binary relations over the state space and determining if there is a relation among them which is a bisimulation containing (p, q). But since in general bisimulations over normed BPA processes may be infinite - for instance, the least non-empty bisimulation for the transition graph in Example 2.2.1 is the identity - a decision procedure for the bisimulation problem for normed BPA cannot rely on this. However, whenever $\alpha \sim \beta$, our tableau system in Chapter 4 will construct a *self-bisimulation*, a finite relation $R \subseteq Var^* \times Var^*$ whose closure under congruence w.r.t. sequential composition is a bisimulation containing (α, β). The notion of self-bisimulation was introduced by Didier Caucal in [Cau90a] (originally published as [Cau88]). Here the notion of a least congruence is essential.

Definition 2.2.9 *For any binary relation R on Var^*, $\underset{R}{\rightarrow}$ is the least precongruence w.r.t. sequential composition that contains R, $\underset{R}{\longleftrightarrow}$ the symmetric closure of $\underset{R}{\rightarrow}$ and $\underset{R}{\longleftrightarrow}^*$ the reflexive and transitive closure of $\underset{R}{\longleftrightarrow}$ and thus the least congruence w.r.t. sequential composition containing R.*

A self-bisimulation is then simply a bisimulation up to congruence w.r.t. sequential composition.

Definition 2.2.10 *A relation $R \subseteq Var^* \times Var^*$ is called* a self-bisimulation *iff $\alpha R \beta$ implies that*

1. *$\alpha \xrightarrow{a} \alpha'$ implies $\beta \xrightarrow{a} \beta'$ for some β' with $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$*
2. *$\beta \xrightarrow{a} \beta'$ implies $\alpha \xrightarrow{a} \alpha'$ for some β' with $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$*

The following lemma, due to Didier Caucal, shows that a self-bisimulation is a finite witness for bisimilarity[2]:

[2] We reproduce the proof here, since the underlying idea becomes important in Chapter 5, when dealing with the analogous notion for branching bisimulations.

Lemma 2.2.1 [Cau90a] *If R is self-bisimilar then $\underset{R}{\longleftrightarrow}^* \subseteq \sim$.*

Proof: $B = \{(\alpha,\beta) \mid \alpha \underset{R}{\longleftrightarrow}^* \beta\}$ is a bisimulation. Suppose $\alpha \underset{R}{\longleftrightarrow}^* \beta$ and that $\alpha \xrightarrow{a} \alpha'$. We must show that there is a β' such that $\beta \xrightarrow{a} \beta'$ and $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$. We know that $\alpha \underset{R}{\longleftrightarrow}^* \beta$ holds because $\alpha \underset{R}{\longleftrightarrow}{}^k \beta$ for some k. We now proceed by induction in k.

$k = 0$: Trivial, for then $\alpha = \beta$.

$k = 1$: Either $\alpha \underset{R}{\rightarrow} \beta$ or $\beta \underset{R}{\rightarrow} \alpha$. Assume wlog that $\alpha \underset{R}{\rightarrow} \beta$. Then, by the definition of a least precongruence, there exists a $(\alpha_0, \beta_0) \in R$ such that $\alpha = \delta\alpha_0\gamma$ and $\beta = \delta\beta_0\gamma$. If $\delta \neq \epsilon$, $\alpha \xrightarrow{a} \alpha'$ is due to $\delta\alpha_0\gamma \xrightarrow{a} \delta'\alpha_0\gamma$, so our matching transition is $\delta\beta_0\gamma \xrightarrow{a} \delta'\beta_0\gamma$; clearly $\delta'\alpha_0\gamma \underset{R}{\longleftrightarrow}^* \delta'\beta_0\gamma$. If $\delta = \epsilon$, $\alpha \xrightarrow{a} \alpha'$ is $\alpha_0\gamma \xrightarrow{a} \alpha_1\gamma$, due to $\alpha_0 \xrightarrow{a} \alpha_1$. Since $(\alpha_0, \beta_0) \in R$, the latter can be matched by $\beta_0 \xrightarrow{a} \beta_1$ with $\alpha_1 \underset{R}{\longleftrightarrow}^* \beta_1$, so we get the match $\beta_0\gamma \xrightarrow{a} \beta_1\gamma$, and clearly $\alpha_1\gamma \underset{R}{\longleftrightarrow}^* \beta_1\gamma$.

Step, assuming for $k > 1$: Then there is a γ s.t. $\alpha \underset{R}{\longleftrightarrow} \gamma$ and $\gamma \underset{R}{\longleftrightarrow}{}^k \beta$. By induction hypothesis we know that there is a γ' s.t. $\gamma \xrightarrow{a} \gamma'$ and $\alpha' \underset{R}{\longleftrightarrow}^* \gamma'$ and a β' s.t. $\beta \xrightarrow{a} \beta'$ with $\gamma' \underset{R}{\longleftrightarrow}^* \beta'$. But then by transitivity $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$.

The other half of the proof, for $\beta \xrightarrow{a} \beta'$, is identical. □

Corollary 2.2.1 *$\alpha \sim \beta$ iff there is a self-bisimulation R such that $\alpha R \beta$.*

Proof: Clearly, by Proposition 2.2.2, $\sim$ is a self-bisimulation. Conversely, by the above lemma, if R is a self-bisimulation then $\underset{R}{\longleftrightarrow}^*$ is a bisimulation. □

2.2.6 The 'split' lemma

Finally, the following lemma due to [Cau88] is essential in the tableau system of Chapter 4, since it provides us with a way of removing suffixes of bisimilar BPA expressions.

Lemma 2.2.2 *For any normed Δ and $\alpha_1, \alpha_2, \beta \in Var^*$, if $\alpha_1\beta \sim \alpha_2\beta$ then $\alpha_1 \sim \alpha_2$.*

PROOF: Suppose $\alpha_1\beta \sim \alpha_2\beta$. Then $R = \{(\alpha_1, \alpha_2) \mid \alpha_1\beta \sim \alpha_2\beta\}$ is a bisimulation. We have by Proposition 2.2.1 that $|\alpha_1| = |\alpha_2|$. So $\alpha_1 = \epsilon$ iff $\alpha_2 = \epsilon$. Otherwise, we have $\alpha_1\beta \xrightarrow{a} \eta_1$ iff $\alpha_1 \xrightarrow{a} \alpha_1'$ and $\eta_1 = \alpha_1'\beta$. And the matching move $\alpha_2\beta \xrightarrow{a} \eta_2$ with $\eta_1 \sim \eta_2$ must be due to $\alpha_2 \xrightarrow{a} \alpha_2'$ with $\eta_2 = \alpha_2'\beta$ so $(\alpha_1', \alpha_2') \in R$. The other half of the proof is entirely similar. $\square$

Note that the proof relies heavily on normedness; a simple counterexample for the unnormed case is the system of equations $\{X \stackrel{\text{def}}{=} aX, Y \stackrel{\text{def}}{=} a\}$ as $YYX \sim YX$, but clearly $YY \not\sim Y$.

Chapter 3

A modal characterization of SnS

In this chapter we show that a modal mu-calculus that incorporates a notion of counting descendants characterizes the Rabin recognizable tree languages of [Rab69] up to an equivalence of parental information.

In Section 3.1 we outline the syntax and semantics of SnS and the fixed-point calculus, here referred to as CML . We then show in Section 3.2 that SnS is at least as expressive as CML by a translation of CML into SnS . Finally, in Section 3.3 we use Rabin's tree theorem to establish the result by showing how to encode the acceptance condition of a Rabin automaton in CML such that a set of trees equivalent to that recognized by the automaton satisfies the acceptance encoding formula. Our notion of equivalence is a bisimulation on node-labelled trees resembling the observational equivalence of [Mil89] and the tree equivalence of [BCG88].

3.1 Syntax and semantics of SnS and CML

3.1.1 SnS

The second order monadic theory of n successors, SnS , has its set of terms given by the abstract syntax

$$T ::= x \mid \epsilon \mid Ti$$

where $i \in \{0, \ldots, n-1\}$ and x ranges over a set of zero-order (element) variables, Var_0. Its formulae are given by the abstract syntax

$$AF ::= T_1 = T_2 \mid T_1 < T_2 \mid T \in X \mid T \in P_i$$

for atomic formulae. Here X ranges over a set of first-order (set) variables, Var_1 and P_i ranges over a finite set of atomic predicates. For composite formulae the syntax is

$$F ::= AF \mid F_1 \vee F_2 \mid \neg F \mid \forall x.F \mid \forall X.F$$

SnS -structures are of the form $(\{0, \ldots, n-1\}^*, s_0, \ldots, s_{n-1}, <, P_1, \ldots, P_m)$. Here $s_0, \ldots, s_{n-1}$ are the successor functions $s_i(w) = wi$. $<$ is the ancestral ordering on $\{0, \ldots, n-1\}^*$ (Definition 2.1.2) and $P_1, \ldots, P_m$ are subsets of $\{0, \ldots, n-1\}^*$.

The semantics of SnS is defined relative to two variable assigments, $\sigma : Var_0 \to \{0, \ldots, n-1\}^*$ and $\rho : Var_1 \to \mathbf{2}^{\{0,\ldots,n-1\}^*}$. The semantics of a term is an element of $\{0, \ldots, n-1\}^*$ and is defined by

$$\begin{aligned} [\![x]\!]\sigma &= \sigma(x) \\ [\![\epsilon]\!]\sigma &= \epsilon \\ [\![Ti]\!]\sigma &= ([\![T]\!]\sigma)i \end{aligned}$$

The semantics of an SnS -formula F is defined by a relation $\mathcal{M} \models_{\sigma,\rho} F$ where $\mathcal{M}$ is an SnS -structure and σ and ρ are zero- and first-order variable assigments, respectively. The clauses are [1] :

$$
\begin{array}{rcl}
\mathcal{M}\models_{\sigma,\rho} T_1 = T_2 & \Longleftrightarrow & [\![T_1]\!]\sigma = [\![T_2]\!]\sigma \\
\mathcal{M}\models_{\sigma,\rho} T_1 < T_2 & \Longleftrightarrow & [\![T_1]\!]\sigma < [\![T_2]\!]\sigma \\
\mathcal{M}\models_{\sigma,\rho} T \in X & \Longleftrightarrow & [\![T]\!]\sigma \in \rho(X) \\
\mathcal{M}\models_{\sigma,\rho} T \in P & \Longleftrightarrow & [\![T]\!]\sigma \in P \\
\mathcal{M}\models_{\sigma,\rho} \neg F & \Longleftrightarrow & \text{not } \mathcal{M}\models_{\sigma,\rho} F \\
\mathcal{M}\models_{\sigma,\rho} F_1 \vee F_2 & \Longleftrightarrow & \mathcal{M}\models_{\sigma,\rho} F_1 \text{ or } \mathcal{M}\models_{\sigma,\rho} F_2 \\
\mathcal{M}\models_{\sigma,\rho} \forall x.F & \Longleftrightarrow & \text{for all } w \in \{0,\ldots,n-1\}^* : \mathcal{M}\models_{\sigma\{w/x\},\rho} F \\
\mathcal{M}\models_{\sigma,\rho} \forall X.F & \Longleftrightarrow & \text{for all } S \in \mathbf{2}^{\{0,\ldots,n-1\}^*} : \mathcal{M}\models_{\sigma,\rho\{S/X\}} F
\end{array}
$$

When the atomic predicates are explicitly chosen among $P_1, \ldots, P_m$, we shall refer to the language as $SnS_{P_1,\ldots,P_m}$.

3.1.2 *CML*

We now introduce a modal logic, CML , for describing tree properties. CML is a version of the propositional mu-calculus of [Koz83] with label set $\{0, \ldots, n-1\}$. The syntax of its formulae is given by

$$F ::= P \mid F_1 \vee F_2 \mid \neg F \mid Z \mid \nu Z.F \mid \textcircled{\scriptsize i}\, F$$

where $i \in \{0, \ldots, n-1\}$, P ranges over a set of atomic predicates on $\{0, \ldots, n-1\}^*$ and Z ranges over a set of recursion variables, Var_{CML}. We assume that all recursion variables are within the scope of an even number of negations; this will ensure the well-definedness of the semantics of CML given below.

[1] $\sigma\{w/x\}$ is the zero-order assignment that maps x to the node w and otherwise agrees with σ. Similarly, $\rho\{S/X\}$ is the first-order assignment that maps X to the set S and otherwise agrees with ρ.

We interpret CML in the SnS -structures, only we now think of the elements of a structure as possible worlds and the s_i as accessibility relations between possible worlds. The intended semantics of the $ⓘ$ modality in CML is that $ⓘ\, F$ holds if F holds in the ith successor-world.

The semantic function $\| \; \|$ is seen relative to an assignment of the recursion variables, $\rho : Var_{CML} \rightarrow \mathbf{2}^{\{0,\ldots,n-1\}^*}$. It is defined relative to a structure $\mathcal{M}$ by

$$
\begin{array}{rcl}
\|P_j\|^{\mathcal{M}}_{\rho} & = & P_j \\
\|F_1 \vee F_2\|^{\mathcal{M}}_{\rho} & = & \|F_1\|^{\mathcal{M}}_{\rho} \cup \|F_2\|^{\mathcal{M}}_{\rho} \\
\|\neg F\|^{\mathcal{M}}_{\rho} & = & \{0,\ldots,n-1\}^* \setminus (\|F_1\|^{\mathcal{M}}_{\rho}) \\
\|ⓘ\, F\|^{\mathcal{M}}_{\rho} & = & \{w \mid wi \in \|F\|^{\mathcal{M}}_{\rho}\} \\
\|Z\|^{\mathcal{M}}_{\rho} & = & \rho(Z) \\
\|\nu Z.F\|^{\mathcal{M}}_{\rho} & = & \bigcup\{S \mid S \subseteq \|F\|^{\mathcal{M}}_{\rho\{S/Z\}}\}
\end{array}
$$

When the atomic predicates are explicitly chosen from among $P_1, \ldots, P_m$, we shall refer to the mu-calculus as $CML_{P_1,\ldots,P_m}$.

Note that all 'superfluous' logical operators, including $\vee$, $\oplus$ (exclusive or) and $\mu X.F$ (least fixed point), are easily derived. For instance, for the formula $F(Z)$ with free recursion variable Z we have the translation[2] $\mu Z.F = \neg\nu Z.\neg(F(\neg Z/Z))$. Similarly, we can define all temporal operators of CTL [CE81]. As an example, $\forall G$ is given by $\forall G.F = \nu Z.F \wedge \bigwedge_{i=0}^{n-1} ⓘ\, Z$ where Z is a variable not occuring free in P. We shall feel free to use all these derived operators as convenient abbreviations.

3.2 SnS is at least as expressive as CML

Labelled trees and the SnS structures are of course one and the same thing. The only thing we need to observe is that an SnS -structure is an infinite n-ary tree

[2]Here $F(\neg Z/Z)$ denotes the formula $F(Z)$ with all free occurrences of Z replaced by $\neg Z$.

with nodes labelled by the sets of predicates that they satisfy. Thus the structure $\mathcal{M} = (\{0,\ldots,n-1\}^*, s_0,\ldots,s_{n-1},<,P_1,\ldots,P_m)$ becomes $t_{\mathcal{M}} : \{0,\ldots,n-1\}^* \to \mathbf{2}^{\{P_1,\ldots,P_m\}}$ with $P_i \in t_{\mathcal{M}}(w)$ iff $w \in P_i$. Similarly, a labelled tree t labelled by $A = \{a_1 \ldots, a_m\}$ is the structure $\mathcal{M}_t = (\{0,\ldots,n-1\}^*, s_0,\ldots,s_{n-1},<,P_{a_1},\ldots,P_{a_m})$ with $w \in P_{a_j}$ iff $t(w) = a_j$.

We can therefore talk about the tree languages definable in SnS and CML . A tree language L is SnS -definable iff there is an SnS -formula whose models are the trees in L:

Definition 3.2.1 *An A-labelled tree language L is SnS -definable if there is an SnS -formula F with one free first order variable, x, and closed w.r.t. second-order variables such that*

$$L = \{t \mid \mathcal{M}_t \models_{\emptyset\{\epsilon/x\},\emptyset} F(x)\}$$

The single free first order variable is to denote the root of the tree, ϵ. Putting all this slightly differently (with a slight abuse of notation), we can define

$$\|F(x)\| \stackrel{\text{def}}{=} \{t \mid \mathcal{M}_t \models_{\emptyset\{\epsilon/x\},\emptyset} F(x)\}$$

Then L is SnS -definable just in case there is a formula $F(x)$ in SnS such that $L = \|F(x)\|$.

We say that L is CML -definable if there is a formula in CML which is true in the roots of the trees in L and none others:

Definition 3.2.2 *An A-labelled tree language L is CML -definable if there is a closed CML -formula F such that*[3]

$$L = \{t \mid \epsilon \in \|F\|_{\rho}^{\mathcal{M}_t}\}$$

[3]The assignment ρ is inessential, since F is closed, and will henceforth be omitted for closed formulae.

Again, putting all this slightly differently, we can define

$$\|F\| \stackrel{\text{def}}{=} \{t \mid \epsilon \in \|F\|^{\mathcal{M}_t}\}$$

Since the metalanguage used in defining the semantics of CML is not far from SnS , it is easy to see that any tree language definable in CML is also definable in SnS .

Lemma 3.2.1 *For any closed formula F in $CML_{P_1,\ldots,P_m}$ there is a formula $\Theta(F)$ in $SnS_{P_1,\ldots,P_m}$ with $\|F\| = \|\Theta(F)\|$.*

Proof: We exhibit a direct translation Θ from $CML_{P_1,\ldots,P_m}$ to $SnS_{P_1,\ldots,P_m}$ that gives us a formula of $SnS_{P_1,\ldots,P_m}$with one free variable x :

$$\begin{aligned}
\Theta(P_k) &= x \in P_k \\
\Theta(F_1 \wedge F_2) &= \Theta(F_1) \wedge \Theta(F_2) \\
\Theta(\neg F) &= \neg\Theta(F) \\
\Theta(ⓘ F) &= (\Theta(F))[xi/x] \\
\Theta(Z) &= x \in Y_Z \\
\Theta(\nu Z.F) &= \exists S.x \in S \wedge (\forall y.y \in S \Rightarrow \\
&\quad (\Theta(F))[y/x][S/Y_Z]) \wedge \forall T.(\forall z.z \in T \Rightarrow (\Theta(F))[z/x][T/Y_Z] \Rightarrow T \subseteq S)
\end{aligned}$$

($[xi/x]$ denotes a uniform substitution of xi for free occurrences of x – similarly for the other substitutions.) Note that atomic predicates are carried over and that recursion variables become set variables. The translation of fixed-points is just a formulation of Tarski's fixed-point induction principle. A straightforward induction in the structure of CML -formulae shows that this translation gives a formula with one free variable with $\|F\| = \|\Theta(F)\|$. □

Since CML by the above lemma can be embedded in SnS , since the latter is decidable [Rab69] and since the translation is effective we also see that CML is *decidable* (cf. [ES84]).

3.3 CML is as expressive as SnS modulo $\simeq_A$

We now establish that CML is as expressive as SnS up to an 'observational' equivalence of tree languages. We here use the tree theorem of [Rab69].

Theorem 3.3.1 [Rab69] *A tree language is SnS -definable iff it is Rabin recognizable in that*

- *For any formula F in $SnS_{P_1,\ldots,P_m}$ there exists a Rabin automaton $\mathcal{A}_F$ over the alphabet $\mathbf{2}^{P_1,\ldots,P_m}$ such that $t_{\mathcal{M}}$ admits an accepting run on $\mathcal{A}_F$ iff $\mathcal{M}\models F$.*
- *For any Rabin automaton $\mathcal{A}$ over the alphabet $A = \{a_1, \ldots, a_m\}$ there exists a formula $F_{\mathcal{A}}$ in $SnS_{P_{a_1},\ldots,P_{a_m}}$ such that $\mathcal{M}_t\models F_{\mathcal{A}}$ iff t admits an accepting run on $\mathcal{A}$.*

In what follows, the first half of Theorem 3.3.1 will be essential. For a given F of $SnS_{P_1,\ldots,P_m}$ we express the Rabin acceptance condition of the corresponding automaton $\mathcal{A}_F$ in CML . The acceptance condition assumes a knowledge of the states assigned to a node. However, we do not have the state predicates available in our structures, only labelling predicates for the automaton alphabet A, so we must find a way of overcoming this. We do so by coding the product of a tree t and its run r on $\mathcal{A}_F$, $t\hat{\ }r \in T^{\omega}_{A\times Q}$, as a tree where the states assumed in the run can be recovered from the position of the nodes in the encoding using the $\textcircled{\scriptsize i}$-operator. In what follows we assume wlog that $n = 2$.

Definition 3.3.1 *A* computation history (h, S) *for the Rabin automaton* $\mathcal{A} = (Q, q_0, \{\xrightarrow{a} \mid a \in A\}, \Omega)$ *is any function $h : S \to A$ and its domain $S \subseteq (\{0,1\}^*Q)^+$ which is a least set satisfying*

$$q_0 \in S \text{ and } h(q_0) = a \text{ for some } a \in \{a \mid \exists q', q'' : q_0 \xrightarrow{a} (q', q'')\}$$

$$sq \in S \Rightarrow \exists q \xrightarrow{a} (q_1, q_2) : \begin{cases} sq000q0q_1 & \in S \\ sq000q1q_2 & \in S \end{cases} \text{ and } h(sq) = a$$

The definition gives an obvious isomorphism H between the tree/run combinations in $T^{\omega}_{A\times Q}$ and the set of computation histories, so we shall feel free to speak of a computation history associated with a given tree and a run on it. Also, given a history (h,S) it is easy to find the unique run $r_{(h,S)} : \{0,1\}^* \rightarrow Q$ to which it corresponds. By a slight abuse of notation, we let $H(w)$ denote the node in the domain of $H(t\hat{}r)$ that corresponds to the node w in t and r.

We now code all computation histories of $\mathcal{A}$ as full binary trees labelled by $A \cup \{\tau\}$, where τ is a dummy label which signifies that the node in the encoding does not correspond to a node in the original computation history. The coding consists in taking the homomorphic extension K^* of the node-coding given by

$$\begin{aligned} K(q_i) &= 10^i1 \quad 1 \le i \le |Q| \\ K(w) &= w \qquad w \in \{0,1\}^* \end{aligned}$$

and defining the associated labelling $K^*(h,S) : \{0,1\}^* \rightarrow A \cup \{\tau\}$ by

$$K^*(h,S)(w) = \begin{cases} a & \text{if } w = K^*(s) \text{ for some } s \in S \text{ with } h(s) = a \\ \tau & \text{otherwise} \end{cases}$$

The state assumed in a node is thus reflected in the path to its descendants in the coding. Using this fact we can now define the state predicates as formulae $\mathbf{Q_{q_j}}$ of CML , assuming a predicate P_a for every $a \in A \cup \{\tau\}$. A node in a *coded* computation history satisfies the state predicate formula $\mathbf{Q_{q_j}}$ exactly if the run corresponding to the history is labelled by q_j at the corresponding node:

Lemma 3.3.1 *For any tree t and associated computation history (h,S) any node w in the run $r_{(h,S)}$ satisfies $w \in \|Q_{q_j}\|^{\mathcal{M}_r}$ iff $K^*(H(w)) \in \|\mathbf{Q_{q_j}}\|^{\mathcal{M}_{K^*(h,S)}}$ where the $\mathbf{Q_{q_j}}$ are CML -formulae given by*

$$\mathbf{Q_{q_j}} \stackrel{\text{def}}{=} \neg P_\tau \wedge \bigvee_{q_i \in Q} \wedge^1_{k=0} \langle 0\rangle\langle 0\rangle\langle 0\rangle\langle 1\rangle\langle 0\rangle^j\langle 1\rangle \textcircled{k}\, \langle 1\rangle\langle 0\rangle^i\langle 1\rangle(\neg P_\tau) \qquad (3.1)$$

where $\langle i\rangle F \stackrel{\text{def}}{=} \textcircled{i}\,(F \wedge P_\tau) \wedge \bigwedge_{j\neq i} \textcircled{j}\, \forall G.P_\tau$ and $\langle i\rangle^j$ is this iterated j times.

PROOF: The state information in $r_{(h,S)}$ is contained in the shape of the path between successive nodes not labelled by τ. Suppose for a node w in t we have $t(w) = a$ and that $r_{(h,S)}(w) = q_i, r(w0) = q_{i_0}$ and $r(w1) = q_{i_1}$ with $t(w0) = a_0$ and $t(w1) = a_1$. Then for some $sq_i \in S$ we have $h(sq_i) = a$ and $sq_i000q_i0q_{i_0}, sq_i000q_i1q_{i_1} \in S$ with

$$\begin{aligned} h(sq_i000q_i0q_{i_0}) &= a_0 \\ h(sq_i000q_i1q_{i_1}) &= a_1 \end{aligned}$$

In the coded computation history $K^*(sq_i000q_i0q_{i_0}) = K^*(sq_i)00010^i1010^{i_0}1$ and $K^*(sq_i000q_i0q_{i_1}) = K^*(sq_i)00010^i1110^{i_1}1$, so we get

$$\begin{aligned} K^*(h,S)(sq_i000q_i0q_{i_0}) &= a_0 \\ K^*(h,S)(sq_i000q_i1q_{i_1}) &= a_1 \end{aligned}$$

Thus we can find the state assumed in w by recovering the path to the next nodes not labelled by τ. But we must also describe that all nodes on the path between two non-dummy nodes and all subtrees thereof are labelled by τ. We express this by letting $\langle i \rangle F$ denote the formula ⓘ $(F \wedge P_\tau) \wedge \bigwedge_{j \neq i}$ ⓙ $\forall G.P_\tau$ and letting $\langle i \rangle^j$ denote this iterated j times, and we now arrive at (3.1). □

That the coded computation history contains the same ancestral information as the original tree w.r.t. non-τ-labels will be made precise using the equivalence $\simeq_A$ (Definition 2.1.5).

We also need to express a next-time operator w.r.t. coded histories, $X_i.F$, which is to denote that the ith proper descendant has property F. (By a *proper* *descendant* of a node s we mean a descendant not labelled by τ such that all nodes between it and s are labelled by τ, see Definition 2.1.4). This only makes sense in non-dummy nodes, so we get

$$X_i.F \stackrel{\text{def}}{=} \neg P_\tau \quad \wedge \bigvee_{q_{k_1}, q_{k_2} \in Q} \textcircled{0}\,\textcircled{0}\,\textcircled{0}\,\textcircled{1}\,\textcircled{0}^{\,k_1}\textcircled{1}\,\textcircled{i}\,\textcircled{1}\,\textcircled{0}^{\,k_2}\textcircled{1}\,(F \wedge \neg P_\tau)$$

The CTL branching time operators can now be redefined w.r.t. $X_i.F$ so we get e.g. $\forall \mathbf{G}.P \stackrel{\text{def}}{=} \nu Z.P \wedge \bigwedge_{i=0}^{1} X_i.Z$.

We can now describe the behaviour of a Rabin automation by a CML formula.

Theorem 3.3.2 *For any Rabin automaton $\mathcal{A}$ over $A = \{P_{a_1}, \ldots, P_{a_m}\}$ there is a $CML_{P_{a_1},\ldots,P_{a_m},P_\tau}$-formula $Acc_{\mathcal{A}}$ such that for any tree t there is a corresponding computation history (h, S) such that $K^*(q_0) \in \|Acc_{\mathcal{A}}\|^{K^*(h,S)}$ iff $\mathcal{A}$ accepts t.*

Proof: Rabin acceptance is formulated as a conjunction of two CML formulae interpreted over $A \cup \{\tau\}$-labelled trees, namely

$$Acc_{\mathcal{A}} \stackrel{\text{def}}{=} Acc_1 \wedge Acc_2$$

where Acc_1 describes that the $A \cup \{\tau\}$-labelled tree indeed is a coded computation history and Acc_2 describes the Rabin acceptance condition itself.
Acc_1 includes a description of the transition relation of $\mathcal{A}$ and the fact that $K^*(q_0) = 11$, so the coding places the root of a tree/run combination at the node 11:

$$\begin{aligned} Acc_1 \stackrel{\text{def}}{=}\; & P_\tau \wedge \textcircled{0}\,(\forall G.P_\tau) \wedge \textcircled{1}\,\textcircled{0}\,(\forall G.P_\tau) \wedge \\ & \textcircled{1}\,\textcircled{1}\,(\mathbf{Q_{q_0}} \wedge \nu Z.(\bigoplus_{q \in Q} \mathbf{Q_q} \wedge \bigoplus_{(q,a,q_1,q_2) \in \Delta} (\mathbf{Q_q} \wedge P_a \supset \bigwedge_{i=0}^{1} X_i(\mathbf{Q_{q_i}} \wedge Z))) \end{aligned}$$

Acc_2 is given as follows:

$$\begin{aligned} Acc_2 \stackrel{\text{def}}{=}\; & (\forall \mathbf{F}.(\nu Z. \bigvee_{q \in \bigcup U_i} \mathbf{Q_q} \wedge \forall \mathbf{F}.Z)) \wedge \\ & (\bigwedge_{U_i} \bigwedge_{Q_q \in U_i} \neg(\exists \mathbf{F}.\nu Z.(\bigvee_{p \in L_i} \mathbf{Q_p} \wedge \exists \mathbf{F}.(\mathbf{Q_q} \wedge Z)))) \end{aligned}$$

The first conjunct in Acc_2 states that on every path, some state in the U-component of an acceptance pair occurs infinitely often. The second conjunct states that for no acceptance pair is there a path such that both a state in the L-component and a state in the U-component occur infinitely often. $\square$

We now formally state the correspondence between trees and coded computation histories, using the notion of A-bisimulation (Definition 2.1.5). The root of a tree is A-bisimilar to the first node not labelled by τ in the coded computation history, namely the node 11:

Theorem 3.3.3 *For any A-labelled tree t and any of its corresponding coded computation histories $K^*(H(t\hat{}r))$ where r is some run of t we have that*

$$\epsilon_t \simeq_A 11_{K^*(H(t\hat{}r))}$$

PROOF: It is enough to show that the set of pairs of corresponding proper descendants of ϵ_t and $11_{K^*(H(t\hat{}r))}$, i.e. the least set D such that

$$\begin{aligned} D \;=\; & \{\epsilon_t, 11_{K^*(H(t\hat{}r))}\} \cup \\ & \{(w'_1, w''_1) \mid \exists (w', w'') \in D. \exists (w'_2, w''_2). w' \underset{A}{\Rightarrow} (w'_1, w'_2) \wedge w'' \underset{A}{\Rightarrow} (w''_1, w''_2)\} \cup \\ & \{(w'_2, w''_2) \mid \exists (w', w'') \in D. \exists (w'_1, w''_1). w' \underset{A}{\Rightarrow} (w'_1, w'_2) \wedge w'' \underset{A}{\Rightarrow} (w''_1, w''_2)\} \end{aligned}$$

is an A-bisimulation. So take any $(w'_i, w''_i) \in D$. By the isomorphism H and the definition of K^* we have that $t(w) = K^*(H(t\hat{}r))(K^*(H(w)))$; it is immediately seen that the pairs of proper descendants also belong to D. $\square$

If we extend the definition of $\simeq_A$ to tree languages over alphabets including A as

$$L_1 \simeq_A L_2 \;\overset{\text{def}}{\Longleftrightarrow}\; \begin{cases} \forall t \in L_1 \exists t' \in L_2 . t \simeq_A t' \\ \forall t' \in L_2 \exists t \in L_1 . t' \simeq_A t \end{cases}$$

we get from Theorems 3.3.2 and 3.3.3 that

Corollary 3.3.1 *If L is $SnS_{P_{a_1},\dots,P_{a_m}}$-definable, there exists a $CML_{P_{a_1},\dots,P_{a_m},P_\tau}$-definable language L' such that $L \simeq_A L'$.*

Those feeling uneasy about the extra label τ can think of K^* as defining a *partial* labelling function; the labelling predicate P_τ should then be seen as a definedness predicate, and $\simeq_A$ should therefore be seen as a 'Kleene equality' on such partially labelled trees.

Chapter 4

Deciding bisimilarity for normed BPA

In [BBK87b] (and [BBK87a]) Baeten, Bergstra, and Klop prove the remarkable result that bisimulation equivalence is *decidable* for normed BPA processes in 3-GNF. However, their decidability proof relies on isolating a possibly complex periodicity from the transition graphs of these processes. An alternative, more elegant, proof utilizing rewrite techniques is presented by Caucal [Cau90a] (originally published as [Cau88]).

In this chapter we provide a simple and much more direct proof of this decidability result using a *tableau* decision method involving goal-directed rules, a technique closely related to that introduced by Korenjak and Hopcroft in [KH66] for deciding language equivalence for simple grammars. The technique is also related to the tableau systems for local model checking in the modal mu-calculus over finite and infinite state transition systems [SW89,BS90]. A by-product of the tableau system given is a sound and complete equational theory for normed BPA.

In Section 4.1 we outline the existing proofs of the decidability result. In Section 4.2 we give the tableau decision method and in Section 4.3 we present the resulting sound and complete equational theory for normed BPA. Finally, in Section 4.4 we relate our tableau system to the work of [Cau90a] by showing how one can extract what Caucal calles a fundamental relation via a successful tableau and a so-called auxiliary tableau.

4.1 Existing approaches

4.1.1 Baeten, Bergstra, and Klop's proof

The idea behind the proof in [BBK87b] is to view the transition graph of a normed BPA process as a tree and to show that this tree is regular.

The transition graph for a process given by a system of equations Δ in 3-GNF with variables $Var = \{X_1, \ldots, X_n\}$ is unfolded onto a tree whose node set is Var^* (cf. Definition 2.1.1). Here, however, the successors of a node w are $X_1 w, \ldots, X_n w$.

A *translation* is any function Θ_w defined on nodes by $\Theta_w(v) = vw$ and extended to sets in the obvious way. Two sets of nodes V and W are said to be *translation equivalent* if there are translations Θ_v, Θ_w and a set of nodes U such that $\Theta_v(U) = V, \Theta_w(V) = W$. Translation equivalence thus means that two node sets 'have the same shape'.

Example 4.1.1 The tree for $Var = \{X, Y\}$ is depicted in Figure 4–1. The sets $\{YX, XYX, YYX\}$ and $\{YY, XYY, YYY\}$ are translation equivalent because of the set $U = \{Y, XY, YY\}$ and the translations Θ_X and Θ_Y. □

A distance function $d(v, w)$ is defined on pairs of nodes as the least number of edges between v and w. v and w are said to be *far apart* if $d(v, w) > 3$. For instance, the nodes XXX and XXY in Figure 4–1 are far apart since $d(XXX, XYYY) = 6$.

It is then shown how one can decompose a transition graph into slices under the above metric such that the tree of slices yields a regular tree. The decomposition is made relative to some chosen constant d called the *amplitude* of the decomposition.

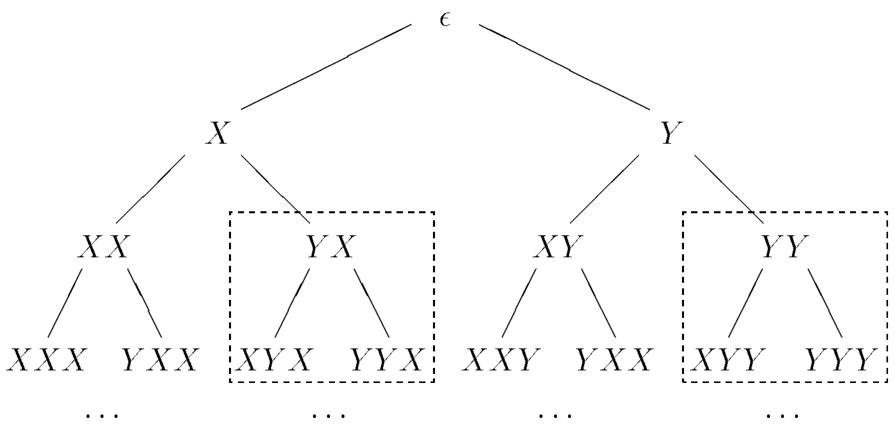

Figure 4–1: Tree with nodes in Var^* where $Var = \{X, Y\}$. The node sets enclosed in dashed rectangles are translation equivalent.

The nth slice in the decomposition contains the nodes α with $nd \leq |\alpha| \leq (n+1)d$ together with those nodes that can be reached in one transition from a node α with $nd < |\alpha| < (n+1)d$. (For an example, see Figure 4–2).

Baeten, Bergstra and Klop then show that there are only finitely many connected fragments up to translation equivalence, and that no such fragment can be decomposed into two fragments that are far apart - this is the desired regular decomposition.

Deciding $\sim$ then consists in checking for the absence of bisimulation errors between transition graphs. A *bisimulation error* in a relation R between transition graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is a triple of nodes $\alpha, \alpha' \in \mathcal{G}_1, \beta \in \mathcal{G}_2$ and a transition $\alpha \xrightarrow{a} \alpha'$ in $\mathcal{G}_1$ such that there is no $\beta \xrightarrow{a} \beta'$ in $\mathcal{G}_2$ with $\alpha' R \beta'$.

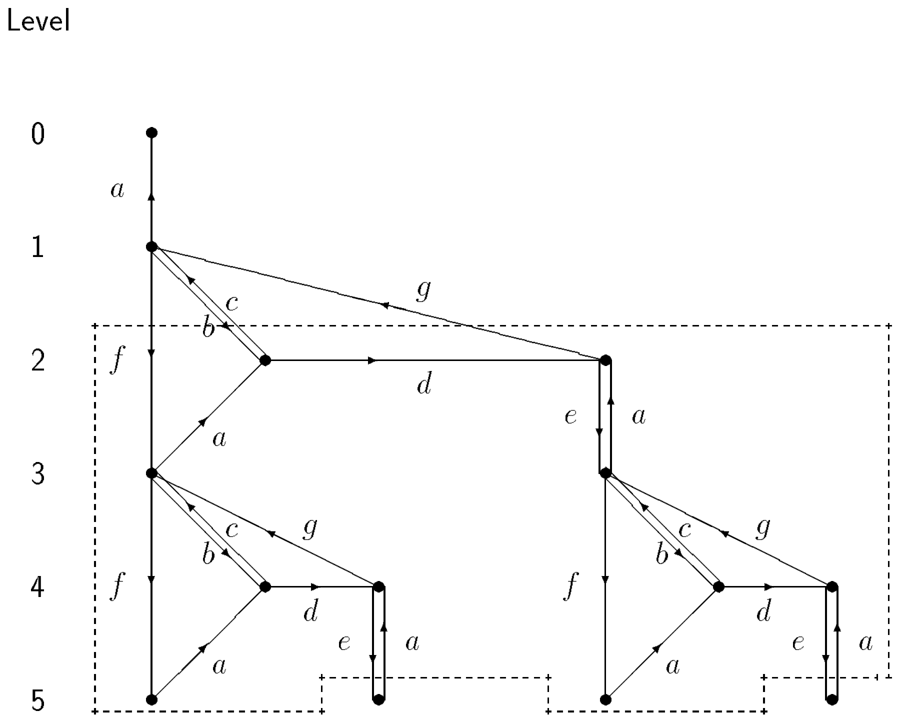

Figure 4–2: The transition graph for $\{X \stackrel{\text{def}}{=} a + bY + fXY;\ Y \stackrel{\text{def}}{=} cX + dZ;\ Z \stackrel{\text{def}}{=} gX + eXZ\}$. Nodes at level n have norm n. The dashed area contains slice 1 of a decomposition with amplitude 2.

Because of the periodicity exhibited by the above regular decomposition, this test can always be done in finite time:

Consider two systems of process equations Δ_1 and Δ_2 in 3-GNF. A *partial bisimulation up to level n* between their transition graphs is a binary relation between nodes with norms $\leq n$ such that it contains no bisimulation errors. Such a partial bisimulation R is called *d-sufficient* with respect to decompositions of amplitude d if for all pairs of slices (Σ_1, Σ_2) related by R (meaning that for some $\alpha \in \Sigma_1, \beta \in \Sigma_2, \alpha R \beta$) one can find translation equivalent copies $(\Sigma_1{}', \Sigma_2{}')$ at least one slice higher up such that the restriction of R to $(\Sigma_1 \times \Sigma_2)$ coincides with the restriction of R to $({\Sigma_1}' \times {\Sigma_2}')$.

It is then shown that one now only has to determine a level $N(\Delta_1, \Delta_2, d)$ and search through all the finitely many binary relations between the nodes in the transition graphs above level $N(\Delta_1, \Delta_2, d)$. $\Delta_1 \sim \Delta_2$ iff one finds a partial bisimulation comparing the roots which is d-sufficient. Decidability follows from the fact that the level $N(\Delta_1, \Delta_2, d)$ can be effectively determined.

4.1.2 Caucal's proof

The proof in [Cau90a] is very different; the idea here is to reduce the bisimilarity problem to a rewriting problem for a relation whose least congruence under sequential composition is decidable. This is done via a characterization of the maximal bisimulation $\sim$ on a transition graph as a Thue congruence, i.e. the least congruence (under sequential composition) generated by a finite relation.

The relation generating $\sim$ is a self-bisimulation (cf. Definition 2.2.10) which is also a *fundamental* relation:

Definition 4.1.1 [Cau90a] *A relation* $R \subseteq Var^+ \times Var^+$ *is called* fundamental *iff*

1. $Dom(R) \subseteq Var$, $Im(R) \subseteq (Var \setminus Dom(R))^+$
2. R *is a function:* $\alpha R \beta$ *and* $\alpha R \gamma$ *implies* $\beta = \gamma$
3. $\alpha R \beta$ *implies* $|\alpha| = |\beta|$

From the first and second conditions above it is immediately seen that fundamental relations are finite and from the third condition one sees that there are finitely many fundamental relations for any normed BPA process (since there are only finitely many elements of Var^* with any given norm). Seen as a rewrite relation, if R is fundamental then it is also canonical, i.e. confluent and well-founded (this follows from the functionality of R and the finiteness of $Dom(R)$), and thus its least congruence is decidable.

Proposition 4.1.1 [Cau90a] *The set of fundamental, self-bisimilar relations for the transition graph for a normed system of BPA equations Δ can be effectively constructed from Δ.*

PROOF: We know that there are only finitely many fundamental relations for the transition graph and that $\underset{R}{\longleftrightarrow}^*$ is well-founded and confluent for any fundamental relation. Thus we can in finite time check for each of these finitely many finite relations whether or not it is a self-bisimulation. $\square$

The main result of [Cau90a] is the following characterization.

Theorem 4.1.1 [Cau90a] *If R is fundamental, self-bisimilar and maximal w.r.t. $\subseteq$ we have $\underset{R}{\longleftrightarrow}^* = \sim$.*

One should note that fundamental, self-bisimilar relations always exist on a transition graph for a normed BPA process. The empty relation $\emptyset$ is clearly fundamental and self-bisimilar. If $\emptyset$ is maximal, $\sim$ is the identity relation. We now get

Corollary 4.1.1 *For any normed system of BPA equations in GNF Δ and $\alpha, \beta \in Var^*$ it is decidable whether $\alpha \sim \beta$.*

PROOF: Use a linear search as outlined in the proof of Proposition 4.1.1 to find R, a fundamental, self-bisimilar relation on $Var \times Var^+$ which is maximal w.r.t. $\subseteq$. Since R is canonical, it is decidable whether or not $\alpha \underset{R}{\longleftrightarrow}^* \beta$. $\square$

4.2 The tableau decision method

The bisimulation checker for normed BPA we now present is a *tableau system*, a goal-directed proof system. The proof technique is similar to the algorithm used in [KH66] to show that language equivalence is decidable for simple grammars.

Assume a fixed system of normed BPA process equations in 3-GNF, $\Delta = \{X_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} a_{ij}\alpha_{ij} \mid 1 \leq i \leq m\}$. We determine whether $X\alpha \sim Y\beta$ (assuming of course that all occurring variables are defined in Δ) by constructing a tableau using the proof rules presented in Table 4–1. A *tableau* for $X\alpha = Y\beta$ is a maximal finite proof tree whose root is labelled $X\alpha = Y\beta$ such that the equations labelling the immediate successors of a node are determined by an application of one of the rules in accordance with the procedure described in this section.

The rules are built around equations $E\alpha = F\beta$ (where α, β could be the empty sequence of variables). Each rule has the form

$$\frac{E\alpha = F\beta}{E_1\alpha_1 = F_1\beta_1 \quad \cdots \quad E_n\alpha_n = F_n\beta_n}$$

(possibly with side conditions). The premise of a rule represents the goal to be achieved (that $E\alpha \sim F\beta$) while the consequents are the subgoals.

The rules are only applied to nodes that are not *terminal*. Terminal nodes are either *successful* or *unsuccessful*.

Definition 4.2.1 *A tableau node is called an* unsuccessful terminal *if it has one of the forms*

1. $\alpha = \beta$ *with* $|\alpha| \neq |\beta|$

2. $a\alpha = b\beta$ *with* $a \neq b$

Clearly, such nodes cannot relate bisimilar processes. In the following subsection we define the notion of successful termination.

4.2.1 Constructing subtableaux

A tableau consists of a number of *eliminating subtableaux* constructed using the rules REC, SUM, and PREFIX of Table 4–1. Each of these rules is *forwards sound* in the sense that if the antecedent is true (the equation relates bisimilar processes) then one can find a set of true consequents. This is expressed in the following three propositions, whose proofs are immediate:

Proposition 4.2.1 (Soundness of REC) *If $X\alpha \sim Y\beta$ and $X \stackrel{\text{def}}{=} E$ and $Y \stackrel{\text{def}}{=} F$, then $E\alpha \sim F\beta$.*

Proposition 4.2.2 (Soundness of SUM) *If $(\sum_{i=1}^{m} a_i\alpha_i)\alpha \sim (\sum_{j=1}^{n} b_j\beta_j)\beta$ then there exist functions $f : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $g : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $a_i\alpha_i\alpha \sim b_{f(i)}\beta_{f(i)}\beta$ for $1 \leq i \leq m$ and $a_{g(j)}\alpha_{g(j)}\alpha \sim b_j\beta_j\beta$ for $1 \leq j \leq n$.*

Proposition 4.2.3 (Soundness of PREFIX) *If $a\alpha \sim a\beta$ then $\alpha \sim \beta$.*

$$\dfrac{\dfrac{X\alpha = Y\beta}{E\alpha = F\beta}\ \mathsf{REC}}{\dfrac{a_1\alpha_1 = a_1\beta_1}{\alpha_1 = \beta_1}\ \mathsf{PREFIX} \qquad \ldots \qquad \dfrac{a_n\alpha_n = a_n\beta_n}{\alpha_n = \beta_n}\ \mathsf{PREFIX}}\ \mathsf{SUM}$$

Figure 4–3: A basic step in the tableau system

A subtableau is built from *basic steps*. See Figure 4–3.

Definition 4.2.2 *A* basic step *for $X\alpha = Y\beta$ consists of an application of* REC *followed by at most one application of* SUM *followed by an application of* PREFIX *to each of its consequents (assuming that no node encountered is an unsuccessful terminal). A* basic node *is any node of the form $\alpha' = \beta'$ where $\alpha', \beta' \in Var^*$.*

Corresponding to a basic step for $X\alpha = Y\beta$ is a set of single transition steps in the operational semantics, as $X\alpha \xrightarrow{a_i} \alpha_i$ and $Y\beta \xrightarrow{a_i} \beta_i$ for any consequent $\alpha_i = \beta_i$. By Proposition 2.2.5 we have that $length(\alpha_i) \leq 1 + length(X\alpha)$ and $length(\beta_i) \leq length(Y\beta) + 1$.

Definition 4.2.3 *Assume that* $k = \min(|X|, |Y|)$. *An* eliminating subtableau *for* $X\alpha = Y\beta$ *iterates the construction of basic steps to depth* k.

See Figure 4–4 for a sketch of an eliminating subtableau in the case when $|X| \leq |Y|$. Notice that if $\alpha' = \beta'$ is a leaf of an eliminating subtableau then $X\alpha \xrightarrow{w} \alpha'$ and $Y\beta \xrightarrow{w} \beta'$ for some w of length k.

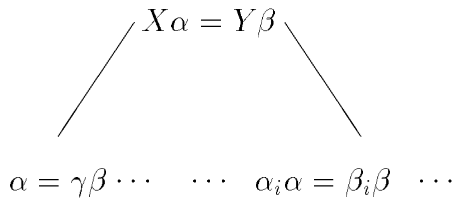

Figure 4–4: An eliminating subtableau for $X\alpha = Y\beta$.

In the case that $|X| \leq |Y|$ each leaf of an eliminating subtableau for $X\alpha = Y\beta$ is either labelled $\alpha = \gamma\beta$, which we call a *residual* of the subtableau, as X has been eliminated, or $\alpha_i\alpha = \beta_i\beta$ where α_i and β_i need not be empty. Since the number of iterations of basic steps is $|X|$ there must be at least one residual and α and β must persist as suffixes throughout the subtableau. For any such subtableau we pick one residual node and call it *the* residual. If instead $|Y| < |X|$ similar remarks would apply except that the residual then has the form $\gamma\alpha = \beta$.

The next step is to apply one of the SUB rules of Table 4–1 to each leaf *other than residuals* of an eliminating subtableau. If the residual is $\alpha = \gamma\beta$ we apply SUBL, and if it is $\gamma\alpha = \beta$ we apply SUBR. So assume $|X| \leq |Y|$; then for each leaf $\alpha_i\alpha = \beta_i\beta$ which is not a residual we obtain

$$\frac{\alpha_i \alpha = \beta_i \beta}{\alpha_i \gamma = \beta_i}\ \mathsf{SUBL} \qquad \text{where } \alpha = \gamma\beta \text{ is the residual}$$

If instead $|Y| < |X|$ so $\gamma\alpha = \beta$ is the residual, SUBR gives us the consequent $\alpha_i = \beta_i\gamma$. The SUB rules are also forwards sound in the following sense:

Proposition 4.2.4 (Soundness of SUBL and SUBR) *If $\alpha_i\alpha \sim \beta_i\beta$ then*

- *if $\alpha \sim \gamma\beta$ then $\alpha_i\gamma \sim \beta_i$, and*
- *if $\gamma\alpha \sim \beta$ then $\alpha_i \sim \beta_i\gamma$*

Proof: By Lemma 2.2.2 a substitution yields $\alpha_i\gamma\beta \sim \beta_i\beta$ and by Proposition 2.2.2 we get $\alpha_i\gamma = \beta_i$. The proof for the other half is entirely similar. □

From the above proof we see that the SUB rules should be thought of as two-step rules consisting of a *substitution* using the residual followed by a *reduction* of the length of the expressions involved according to Proposition 2.2.2. Notice that for any application of SUB we have that

Proposition 4.2.5

1. $|\alpha| < |X\alpha|$ *and* $|\gamma\beta| < |Y\beta|$
2. $length(\alpha_i) + length(\gamma) + length(\beta_i) \leq 3m_\Delta + 1$ *for any application of* SUB.

The latter follows from Proposition 2.2.5. Also notice that the bound obtained here is completely independent of $length(\alpha)$ and $length(\beta)$.

We can now define successful termination.

Definition 4.2.4 *A residual or a consequent of an application of a* SUB *rule is a* successful terminal *if it has one of the forms*

1. *$\alpha = \beta$ where there is a subtableau root above it also labelled $\alpha = \beta$.*

2. *$\alpha = \alpha$*

It should be obvious that a node obeying termination condition 2 in the above relates bisimilar processes. It turns out that this is also true of termination condition 1 in the context of a successful tableau.

When a consequent of SUB or the residual is not a terminal node we build a new eliminating subtableau with it as root as described above, and continue in this fashion. Therefore, a tableau is defined as successions of eliminating subtableaux as sketched in Figure 4–5.

Definition 4.2.5 *A* successful tableau *is a tableau all of whose leaves are successful terminals. If at any point in the construction of an eliminating subtableau we reach an unsuccessful terminal then the resulting tableau is* unsuccessful.

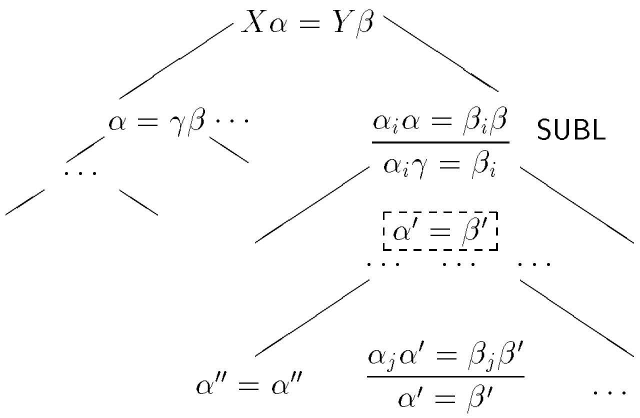

Figure 4–5: A tableau for $X\alpha = Y\beta$; some successful leaves are shown

Example 4.2.1 (Example 2.2.3 continued) Consider again $\Delta = \{X \stackrel{\text{def}}{=} aYX + b, Y \stackrel{\text{def}}{=} bX, A \stackrel{\text{def}}{=} aC + b, C \stackrel{\text{def}}{=} bAA\}$. The tableau in Figure 4–6 is a successful tableau for $X = A$. $\square$

$$
\dfrac{X = A}{\dfrac{aYX + b = aC + b}{\dfrac{aYX = aC}{\dfrac{YX = C}{\dfrac{YX = C}{\dfrac{bXX = bAA}{\dfrac{XX = AA}{\dfrac{(aYX+b)X = (aC+b)A}{\dfrac{aYXX = aCA}{\dfrac{YXX = CA}{YX = C}\,\textsf{SUB}}\,\textsf{PREFIX} \qquad \dfrac{bX = bA}{X = A}\,\textsf{PREFIX}}\,\textsf{SUM}}\,\textsf{REC}}\,\textsf{PREFIX}}\,\textsf{REC}}\,\textsf{SUB}}\,\textsf{PREFIX} \qquad \dfrac{b = b}{\epsilon = \epsilon}\,\textsf{PREFIX}}\,\textsf{SUM}}\,\textsf{REC}
$$

Figure 4–6: A successful tableau for $X = A$ of Example 2.2.3

4.2.2 Decidability, soundness, and completeness

We now give the proof of correctness of the tableau method and give a complexity measure in terms of an upper bound on the length of a tableau path.

Theorem 4.2.1 *Every tableau for $X\alpha = Y\beta$ is finite.*

Proof: If a tableau were infinite then it would have an infinite path. By definition such a path could not contain either successful or unsuccessful terminals. By Proposition 4.2.5(2) such a path can not pass through infinitely many nodes which are consequents of a SUB rule – for since there are only finitely many different equations with total length $\leq 3m_\Delta + 1$ (by Proposition 2.2.6) the path would then contain some successful terminal infinitely often. Otherwise the path must almost always pass through a residual; but this also is impossible as the norm of a residual is strictly less than one directly above it by Proposition 4.2.5(1) and as the norms of the residuals are uniformly bounded by $\max(|\alpha|, |\beta|)$. $\square$

We can give a complexity bound on the tableau method in terms of the longest possible path in any tableau for $X\alpha = Y\beta$. We measure the length of a path in terms of the total number of basic steps, since this gives a measure of the number of transition matches that we need to consider.

Theorem 4.2.2 *Any path in a tableau for $X\alpha = Y\beta$ has a length of at most*

$$m_\Delta \max(|\alpha|, |\beta|, m_\Delta \lceil \frac{3m_\Delta + 1}{2} \rceil \sum_{j=2}^{3m_\Delta+1} (j-1)v^j)$$

basic steps, where v is the cardinality of Var.

Proof: Any SUB consequent has a length of at most $3m_\Delta + 1$, so there can be at most $\sum_{j=2}^{3m_\Delta+1}(j-1)v^j$ distinct SUB consequents along a path. Between any two of these consequents there can be at most $\lceil \frac{3m_\Delta+1}{2} \rceil$ residuals, since the worst that can happen is that the norm on each side decreases by 1 between two consecutive residuals. Thus, any containing SUB consequents has at most $\lceil \frac{3m_\Delta+1}{2} \rceil \sum_{j=2}^{3m_\Delta+1}(j-1)v^j$ subtableau roots. The leftmost path in a tableau contains only residuals, and since their norms are strictly decreasing there can be at most $\max(|\alpha|, |\beta|)$ residuals along this path. Since any subtableau can have a depth of at most m_Δ basic steps, the result follows. $\square$

Corollary 4.2.1 *There are only finitely many tableaux for any $X\alpha = Y\beta$.*

Proof: This follows from the above theorem and the fact that the branching at any basic step in any tableau is uniformly bounded by the maximal number of SUM consequents. This is bounded by $2B_\Delta$ where $B_\Delta = \max\{m \mid \exists X_i \in Var : X_i \stackrel{\text{def}}{=} \sum_{j=1}^{m} a_{ij}\alpha_{ij}\}$. $\square$

The next theorem states soundness and completeness of the tableau method. The proof of soundness relies on the notion of self-bisimulation introduced in section 2.2.5.

Theorem 4.2.3 *$X\alpha \sim Y\beta$ iff there exists a successful tableau for $X\alpha = Y\beta$.*

PROOF: Suppose $X\alpha \sim Y\beta$. Then we can build a tableau for $X\alpha = Y\beta$ which has the property that for each node $\alpha' = \beta'$ we have $\alpha' \sim \beta'$. For by Propositions 4.2.1, 4.2.2, 4.2.3 and 4.2.4 we can at any point in the tableau construction choose true consequents. By Theorem 4.2.1 this tableau construction must terminate and without unsuccessful terminals.

Now assume $\mathbf{T}$ is a successful tableau for $X\alpha = Y\beta$. We now show that $R_{\mathbf{T}} = \{(\alpha, \beta) \mid \alpha = \beta \text{ is a basic node in } \mathbf{T}\}$ is a self-bisimulation. By Lemma 2.2.1 this means that $X\alpha \sim Y\beta$.

So suppose $(\alpha', \beta') \in R_{\mathbf{T}}$. We must then show that $\alpha' \xrightarrow{a} \alpha''$ implies $\exists \beta'' : \beta' \xrightarrow{a} \beta''$ with $\alpha'' \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''$. $\alpha' = \beta'$ can either be a terminal or an internal node.

Suppose $\alpha' = \beta'$ is a terminal. If it is a terminal because of condition 2 we can certainly match within $\underset{R_{\mathbf{T}}}{\longleftrightarrow^*}$, since the least congruence of any relation contains the identity. Otherwise, if $\alpha' = \beta'$ is a terminal is due to condition 1, there is a previous occurrence of $\alpha' = \beta'$ as a subtableau root. Then we have the following basic step in $\mathbf{T}$:

$$\frac{\alpha' = \beta'}{\alpha_1'' = \beta_1'' \quad \cdots \quad \alpha_n = \beta_n''}$$

and $\alpha' \xrightarrow{a_i} \alpha_i''$ is matched by $\beta' \xrightarrow{a_i} \beta_i''$ because $\mathbf{T}$ is successful and $(\alpha_i'', \beta_i'') \in R_{\mathbf{T}}$ for $1 \leq i \leq n$ by definition of $R_{\mathbf{T}}$.

Otherwise $\alpha' = \beta'$ is an internal node. There are now two possibilities: either REC was applied to $\alpha' = \beta'$ or one of the SUB rules was.

Suppose REC was applied. Then we had the basic step

$$\frac{\alpha' = \beta'}{\alpha_1'' = \beta_1'' \quad \cdots \quad \alpha_n = \beta_n''}$$

and just as in the above case, we can match within $R_{\mathbf{T}}$.

Now suppose a SUB rule was applied. Suppose wlog that it was SUBL. Further assume that $\alpha' = \beta'$ is $X_1\alpha_1\alpha_0 = Y_1\beta_1\beta_0$, that $X_1 \stackrel{\text{def}}{=} \sum_{i=1}^{m} a_i\alpha_{2i}$ and $Y_1 \stackrel{\text{def}}{=} \sum_{j=1}^{n} b_j\beta_{2j}$ and that the residual was $\alpha_0 = \gamma_0\beta_0$:

$$\frac{X_1\alpha_1\alpha_0 = Y_1\beta_1\beta_0}{X_1\alpha_1\gamma_0 = Y_1\beta_1}\ \textsf{SUBL}$$

By definition $(\alpha_0, \gamma_0\beta_0) \in R_{\mathbf{T}}$. Either $X_1\alpha_1\gamma_0 = Y_1\beta_1$ is a terminal or a subtableau root. In any case we must have that

$$\forall a_i : X_1\alpha_1\gamma_0 \xrightarrow{a_i} \alpha_{2i}\alpha_1\gamma_0 \ \ \exists b_j : b_j = a_i, Y_1\beta_1 \xrightarrow{b_j} \beta_{2j}\beta_1 \text{ with } \alpha_{2i}\alpha_1\gamma_0 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta_{2j}\beta_1 \quad (4.1)$$

If $X_1\alpha_1\gamma_0 = Y_1\beta_1$ is a terminal, this follows from the same reasoning used in the case where $\alpha' = \beta'$ is a terminal. Otherwise, we have the basic step

$$\dfrac{\dfrac{X_1\alpha_1\gamma_0 = Y_1\beta_1}{(\sum_{i=1}^{m} a_i\alpha_{2i})\alpha_1\gamma_0 = (\sum_{j=1}^{n} b_j\beta_{2j})\beta_1}\ \textsf{REC}}{\dfrac{a_1\alpha_{21}\alpha_1\gamma_0 = a_1\beta_{f(1)}\beta_1}{\alpha_{21}\alpha_1\gamma_0 = \beta_{f(1)}\beta_1}\ \textsf{PREFIX} \quad \cdots \quad \dfrac{a_n\alpha_{g(n)}\alpha_1\gamma_0 = a_n\beta_{2n}\beta_1}{\alpha_{g(n)}\alpha_1\gamma_0 = \beta_{2n}\beta_1}\ \textsf{PREFIX}}\ \textsf{SUM}$$

Now, what are the transitions of $X_1\alpha_1\alpha_0$ and $Y_1\beta_1\beta_0$ and how do we match them ? For $1 \le i \le m$ we have that $X_1\alpha_1\alpha_0 \xrightarrow{a_i} \alpha_{2i}\alpha_1\alpha_0$ and for $1 \le j \le n$, $Y_1\beta_1\beta_0 \xrightarrow{b_j} \beta_{2j}\beta_1\beta_0$. For any $X_1\alpha_1\alpha_0 \xrightarrow{a_i} \alpha_{2i}\alpha_1\alpha_0$ there is (4.1) a $Y_1\beta_1\beta_0 \xrightarrow{b_{f(i)}} \beta_{2f(i)}\beta_1\beta_0$ such that $\alpha_{2i}\alpha_1\gamma_0 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta_{2f(i)}\beta_1$. We now have the match, $\alpha_{2i}\alpha_1\alpha_0 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta_{2f(i)}\beta_1\beta_0$. For since $\alpha_{2i}\alpha_1\gamma_0 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta_{2f(i)}\beta_1$, also $\alpha_{2i}\alpha_1\gamma_0\beta_0 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta_{2f(i)}\beta_1\beta_0$. Since $(\alpha_0, \gamma_0\beta_0) \in R_{\mathbf{T}}$ we then have $\alpha_{2i}\alpha_1\alpha_0 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta_{2f(i)}\beta_1\beta_0$, as was to be shown. Finding a match for any $Y_1\beta_1\beta_0 \xrightarrow{b_j} \beta_{2j}\beta_1\beta_0$ is entirely similar. □

Corollary 4.2.2 *For any normed system of BPA equations in GNF* Δ *and* $\alpha, \beta \in Var^*$ *it is decidable whether* $\alpha \sim \beta$.

PROOF: A decision procedure goes as follows: Enumerate all the finitely many tableaux for $\alpha = \beta$ (this is possible by Corollary 4.2.1). By Theorem 4.2.3 $\alpha \sim \beta$ iff we find a successful tableau. □

4.3 An equational theory

Besides yielding a straightforward decision procedure, the tableau technique can also be used to build a (weakly) sound and complete sequent-style equational theory for bisimulation equivalence of normed BPA processes given in 3-GNF. For all that is required is a family of sound rules that permit one to derive the roots of successful tableaux. The proof system presented in this section is due to Colin Stirling.

The equational theory is somewhat non-standard in the arena of process algebras. As it depends on assumptions, it is different in style both from Milner's elegant equational theory for regular processes with an explicit fixed point operator μ [Mil84] and the version in [BK88] without μ.

Since the theory is based on the tableau system from the previous section, we restrict our attention to normed systems of process equations in 3-GNF. Let Δ be such a system. The proof system appeals to *assumptions* of the form $X\alpha = Y\beta$. The basic sequent of the system has the form $\Gamma \vdash_\Delta E = F$ where Γ is a set of assumptions and E, F range over BPA expressions. A sequent is interpreted as follows:

Definition 4.3.1 *We write* $\Gamma \models_\Delta E = F$ *when it is the case that if the relation* $\{(X\alpha, Y\beta) \mid X\alpha = Y\beta \in \Gamma\} \cup \{(X_i, E_i) \mid X_i \stackrel{\text{def}}{=} E_i \in \Delta\}$ *is part of a bisimulation then* $E \sim F$.

Thus, the special case $\emptyset \models_\Delta E = F$ states that $E \sim F$ (relative to the system of process equations Δ).

The proof system is given in Table 4–2. Equivalence and congruence rules are R1-5. The rules R6-10 correspond to the BPA laws A1-A5 of Table 2–1. R11 and R12 deal with recursion and have been dictated by the tableau method. R11 is an assumption *introduction* rule, justified by the interpretation of sequents described above. R12 is an assumption *elimination* or discharge rule, which at the same time is a version of fixed point induction. Notice that the rule is contextual in character, involving the BPA contexts $[\,]\alpha$ and $[\,]\beta$ where $[\,]$ is a 'hole'.

Definition 4.3.2 *A* proof *of* $\Gamma \vdash_\Delta E = F$ *is a finite proof tree with the root labelled by* $\Gamma \vdash_\Delta E = F$*, with leaves that are instances of the axioms* R1,R6-10 *or* R11 *and such that the parent of a set of nodes is determined by an application of one of the rules* R2-5 *or* R12*. If* $\Gamma \vdash_\Delta E = F$ *has a proof we simply write* $\Gamma \vdash_\Delta E = F$*.*

In our proof that the equational theory is weakly sound and complete it turns out to be easiest to prove that it is in fact *strongly* sound. In our proof we need to appeal to the following standard characterization of the maximal strong bisimulation as a limit:

Definition 4.3.3 *For any transition graph* $\mathcal{G} = (Pr, Act, \{\xrightarrow{a}\})$ *define the family of binary relations* $\{\sim_n\}_{n=0}^{\omega}$ *over* Pr *inductively as follows.*

- $p \sim_0 q$ *for all* $p, q \in Pr$*,*
- $p \sim_{n+1} q$ *iff*
  - *if* $p \xrightarrow{a} p'$ *then* $\exists q'$ *with* $q \xrightarrow{a} q'$ *and* $p' \sim_n q'$ *and*
  - *if* $q \xrightarrow{a} q'$ *then* $\exists p'$ *with* $p \xrightarrow{a} p'$ *and* $p' \sim_n q'$*.*

Theorem 4.3.1 [Mil89] *For any image-finite transition graph we have*

$$\sim \;=\; \bigcap_{n=0}^{\omega} \sim_n$$

Theorem 4.3.2 *If* $\Gamma \vdash_\Delta X\alpha = Y\beta$ *then* $\Gamma \models_\Delta X\alpha = Y\beta$

Proof: Contraposition. Assume that we have a proof of $\Gamma \vdash_\Delta X\alpha = Y\beta$ but that $\Gamma \not\models_\Delta X\alpha = Y\beta$. Then $\{(X\alpha, Y\beta) \mid X\alpha = Y\beta \in \Gamma\} \cup \{(X_i, E_i) \mid X_i \stackrel{\text{def}}{=} E_i \in \Delta\}$ is part of a bisimulation but $X\alpha \not\sim Y\beta$; consequently, as Δ defines an image-finite transition graph, Theorem 4.3.1 says that $X\alpha \not\sim_n Y\beta$ for some n.

Observe that if we ignore the hypotheses, then R2-5,R12 preserve $\sim_n$ and in all other rules except R11 the conclusion is true for all $\sim_n$. For instance, for R12 we have that if $E\alpha \sim_n F\beta$ then $X\alpha \sim_n Y\beta$ because $X \stackrel{\text{def}}{=} E \in \Delta$ and $Y \stackrel{\text{def}}{=} F \in \Delta$. Similarly, for R3 $E \sim_n F$ and $F \sim_n G$ imply that $E \sim_n G$. Now significantly, in the case of R5 we can strengthen this to say that $E_1 \sim_n F_1$ and $E_2 \sim_{n-1} F_2$ imply that $E_1E_2 \sim_n F_1F_2$ (when $|E_1| > 0$.)

Now consider the proof tree for $\Gamma \vdash_\Delta X\alpha = Y\beta$. Since $X\alpha \not\sim_n Y\beta$, by the above observations, there is a path π to some leaf in the proof tree such that for every node $\Gamma_i \vdash_\Delta \alpha_i = \beta_i (1 \leq i \leq m)$ along π we have $\alpha_i \not\sim_{k_i} \beta_i$ for some k_i. For each i choose k_i such that it is the least number with this property.

The leaf of π cannot be an assumption in Γ, since Γ is part of a bisimulation. Nor can it be an identity. The only other possibility is that the leaf at the end of the path is an instance of R11 of the form $\Gamma', X'\alpha' = Y'\beta' \vdash_\Delta X'\alpha' = Y'\beta'$ and such that $X'\alpha' \not\sim_{k_m} Y'\beta'$ where k_m is the least number with this property. Assume that $X' \stackrel{\text{def}}{=} E' \in \Delta$ and $Y' \stackrel{\text{def}}{=} F' \in \Delta$.

As $X'\alpha' = Y'\beta'$ has been eliminated as a hypothesis in the course of the proof, there must be an application of R12 on π with premise $\Gamma_i \vdash_\Delta E'\alpha' = F'\beta'$ somewhere on the path π. On the subpath between this premise and the leaf $\Gamma, X'\alpha' = Y'\beta' \vdash_\Delta X'\alpha' = Y'\beta'$ there must be at least one application of the congruence rule R5 in order to build up the expressions E' and F'; this can be shown, once we take into account the easily proven fact that in any sentence a of proof, with conclusion $E = F$ either both E and F are guarded or neither is. By the above observation on the soundness of R5 w.r.t. $\sim_n$ every node $\alpha_i = \beta_i$ on the subpath must have $\alpha_i \sim_k \beta_i$ for all $k < k_m$. Since E' and F' are guarded, in at least one application of $\mathsf{R5}$ an equation derived from $X'\alpha' = Y'\beta'$ must be the right-hand premise (possibly simply to introduce action prefixes). Thus we must in fact have that $E'\alpha' \sim_k F'\beta'$ for some $k \geq k_m$. But this implies that $X'\alpha' \sim_k Y'\beta'$ for some $k \geq k_m$, a contradiction of our assumption that $X'\alpha' \not\sim_{k_m} Y'\beta'$. $\square$

The completeness proof depends on simulating the tableau construction using the proof rules. We first show that the thinning rule usually found in sequent-based proof systems is a derived rule in ours.

Lemma 4.3.1 (Thinning) *If $\Gamma \vdash_\Delta E = F$ then $\Gamma, \Gamma' \vdash_\Delta E = F$ for any Γ'.*

PROOF: Consider a proof of $\Gamma \vdash_\Delta E = F$. Clearly, any leaf $\Gamma_i \vdash_\Delta E_i = F_i$ can be replaced by $\Gamma_i, \Gamma' \vdash_\Delta E_i = F_i$ yielding a proof tree with nodes of the form $\Gamma_{ij}, \Gamma' \vdash_\Delta E_{ij} = F_{ij}$ whenever the original proof had $\Gamma_{ij} \vdash_\Delta E_{ij} = F_{ij}$. $\square$

The completeness proof rests on a number of lemmas and definitions which tell us how to determine our sets of hypotheses throughout a proof of $X\alpha \sim Y\beta$ from a successful tableau for $X\alpha \sim Y\beta$.

Definition 4.3.4 *In a successful tableau $\mathbf{T}$, we define the set of* companion nodes *$Com(E\alpha' = F\beta')$ for a node $E\alpha' = F\beta'$ as the set of nodes along the path to the root of $\mathbf{T}$ that correspond to an instance of a successful terminal for termination condition 1.*

For any subtableau $\mathbf{T}'$ of $\mathbf{T}$ the set $Basic_{\mathbf{T}'}(E\alpha' = F\beta')$ for a node $E\alpha' = F\beta'$ in $\mathbf{T}'$ is the set of basic nodes on the path starting above $E\alpha' = F\beta'$ and ending at the root of $\mathbf{T}'$.

Proposition 4.3.1 *For any node $E\alpha' = F\beta'$ in a successful tableau $\mathbf{T}$ we have*

$$Com(E\alpha' = F\beta') \subseteq Basic_{\mathbf{T}}(E\alpha' = F\beta')$$

Lemma 4.3.2 *Let $\mathbf{T}'$ be a subtableau of a successful tableau $\mathbf{T}$ such that $\mathbf{T}'$ is built using only basic steps, has root $X'\alpha' = Y'\beta'$ and leaves $\alpha_1 = \beta_1, \ldots, \alpha_n = \beta_n$. If for some Γ we have $\Gamma \vdash_\Delta \alpha_i = \beta_i$ for $1 \leq i \leq n$ then $\Gamma \vdash_\Delta X'\alpha' = Y'\beta'$ with a proof tree with nodes of the form $\Gamma, Basic_{\mathbf{T}'}(E''\alpha'' = F''\beta'') \vdash_\Delta E''\alpha'' = F''\beta''$ for any node $E''\alpha'' = F''\beta''$ in $\mathbf{T}'$.*

Proof: Induction in d, the depth w.r.t. basic steps of $\mathbf{T}'$.

$d = 1$: $\mathbf{T}'$ consists of one basic step:

$$\dfrac{\dfrac{X'\alpha' = Y'\beta'}{(\sum_{i=1}^{m} a_i\alpha_i)\alpha' = (\sum_{j=1}^{n} b_j\beta_j)\beta'}\ \mathsf{REC}}{\dfrac{a_1\alpha_1\alpha' = b_{f(1)}\beta_{f(1)}\beta'}{\alpha_1\alpha' = \beta_{f(1)}\beta'}\ \mathsf{PREFIX} \quad \cdots \quad \dfrac{a_{g(n)}\alpha_{g(n)}\alpha' = b_n\beta_n\beta'}{\alpha_{g(n)}\alpha' = \beta_n\beta'}}\ \mathsf{SUM}\ \ \mathsf{PREFIX}$$

where $\Gamma \vdash_\Delta \alpha_i\alpha' = \beta_{f(i)}\beta'$ for $1 \leq i \leq m$ and $\Gamma \vdash_\Delta \alpha_{g(j)}\alpha' = \beta_j\beta'$ for $1 \leq j \leq n$. Since we have that

$$Basic_{\mathbf{T}'}(\alpha_i\alpha' = \beta_i\beta') = Basic_{\mathbf{T}'}(\alpha_j\alpha' = \beta_j\beta') = \{X'\alpha' = Y'\beta'\}$$

for any consequents, Lemma 4.3.1 tells us that

$$\Gamma, X'\alpha' = Y'\beta' \vdash_\Delta \alpha_i\alpha' = \beta_{f(i)}\beta' \text{ for } 1 \leq i \leq m$$

and

$$\Gamma, X'\alpha' = Y'\beta' \vdash_\Delta \alpha_{g(j)}\alpha' = \beta_j\beta' \text{ for } 1 \leq j \leq n$$

Repeated use of $\mathsf{R5}$ followed by repeated use of $\mathsf{R4}$ gives us

$$\Gamma, X'\alpha' = Y'\beta' \vdash_\Delta (\sum_{i=1}^{m} a_i\alpha_i)\alpha' = (\sum_{j=1}^{m} b_j\beta_j)\beta'$$

Finally, by R12 we get $\Gamma \vdash_\Delta X'\alpha' = Y'\beta$.

Step (assuming for d): The first basic step of the subtableau is as in the base case. By induction hypothesis we have that

$$\Gamma \vdash_\Delta \alpha_i \alpha' = \beta_{f(i)} \beta' \text{ for } 1 \leq i \leq m$$

and

$$\Gamma \vdash_\Delta \alpha_{g(j)} \alpha' = \beta_j \beta' \text{ for } 1 \leq j \leq n$$

And by Lemma 4.3.1 we get that

$$\Gamma, X'\alpha' = Y'\beta' \vdash_\Delta \alpha_i \alpha' = \beta_{f(i)} \beta' \text{ for } 1 \leq i \leq m$$

and

$$\Gamma, X'\alpha' = Y'\beta' \vdash_\Delta \alpha_{g(j)} \alpha' = \beta_j \beta' \text{ for } 1 \leq j \leq n$$

The proof now proceeds as for the base case. □

Lemma 4.3.3 *Given a successful tableau* $\mathbf{T}$*, for any* $X\alpha = Y\beta$ *that is a terminal or the root of an eliminating subtableau we have* $Com(X\alpha = Y\beta) \vdash_\Delta X\alpha = Y\beta$*.*

Proof: Induction in the structure of $\mathbf{T}$.

Base case - $X\alpha = Y\beta$ *is a terminal:* $X\alpha = Y\beta$ is either a terminal due to termination condition 2 or termination condition 1. In the former case, R1 immediately gives us $Com(X\alpha = Y\beta) \vdash_\Delta X\alpha = Y\beta$. In the latter case, $X\alpha = Y\beta \in Com(X\alpha = Y\beta)$ so we get desired result by R11.

Step: Now $X\alpha = Y\beta$ is root of the subtableau $\mathbf{T}'$:

$$X\alpha = Y\beta$$

$$\alpha = \gamma\beta \cdots \quad \cdots \frac{\alpha_i\alpha = \beta_i\beta}{\alpha_i\gamma = \beta_i}\,\mathsf{SUBL} \cdots$$

By induction hypothesis, we have $Com(\alpha = \gamma\beta) \vdash_\Delta \alpha = \gamma\beta$ and for any SUB consequent (assume wlog that it is SUBL) $Com(\alpha_i\gamma = \beta_i) \vdash_\Delta \alpha_i\gamma = \beta_i$. But since there are no terminals within $\mathbf{T}'$ we have that $Com(\alpha = \gamma\beta) = Com(\alpha_i\gamma = \beta_i)$. By R1, we get $Com(\alpha = \gamma\beta) \vdash_\Delta \beta = \beta$ and by R5 $Com(\alpha = \gamma\beta) \vdash_\Delta \alpha_i\gamma\beta = \beta_i\beta$. By R3 this implies $Com(\alpha = \gamma\beta) \vdash_\Delta \alpha_i\alpha = \beta_i\beta$. By Lemma 4.3.2 we then get $Com(X\alpha = Y\beta) \vdash_\Delta X\alpha = Y\beta$ as desired. Note also that by Lemma 4.3.2 that for any node $E''\alpha'' = F''\beta''$ in $\mathbf{T}'$ we have $Basic_{\mathbf{T}'}(E''\alpha'' = F''\beta'') \vdash_\Delta E''\alpha'' = F''\beta''$.
□

Theorem 4.3.3 *If $X\alpha \sim Y\beta$ (with respect to Δ) then $\emptyset \vdash_\Delta X\alpha = Y\beta$*

Proof: By Theorem 4.2.3 we know that $X\alpha = Y\beta$ has a successful tableau $\mathbf{T}$. For each node $E''\alpha'' = F''\beta$ in $\mathbf{T}$ we have that $Basic_{\mathbf{T}}(E''\alpha'' = F''\beta'') \vdash E''\alpha'' = F''\beta$. If $E''\alpha'' = F''\beta$ is a subtableau root or a terminal, this follows from Lemma 4.3.3, Proposition 4.3.1 and Lemma 4.3.1. If $E''\alpha'' = F''\beta$ is a node in an eliminating subtableau, it follows from the remarks at the end of the proof of Lemma 4.3.3 and Lemma 4.3.1. Since $Basic_{\mathbf{T}}(X\alpha = Y\beta) = \emptyset$, the result follows. □

4.4 Extracting fundamental relations

In Section 4.2 we have seen that the tableau system presented generates a self-bisimulation in case of successful termination. In this section we show another relationship with the work of Caucal [Cau90a] in that we give an *auxiliary tableau* *system* for extracting a *fundamental* relation R from a successful tableau for $X\alpha = Y\beta$ with the property that $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$.

One can think of the least congruence $\underset{R}{\longleftrightarrow}^*$ of a relation R as the set of equations provable within equational logic (with added congruence rules) using R as axioms. Thus, we can we view a fundamental relation with the above property as constituting a 'local axiomatization' of $\sim$, relative to Δ and the root equation $X\alpha = Y\beta$.

Throughout the following we shall assume the existence of a successful tableau $\mathbf{T}$ for $X\alpha = Y\beta$.

The fundamental[1] observation is that for the eliminating subtableau for $X\alpha = Y\beta$ we must, when $\alpha = \gamma\beta$ is the residual, have $Y\beta \sim X\gamma\beta$ and thus by Proposition 2.2.2 $Y \sim X\gamma$; assume now wlog that X and Y are not the same variable. Since $|Y| = |X\gamma|$ we know that Y does not occur in $X\gamma$, so $(Y, X\gamma)$ is a fundamental relation. Clearly, if we let $R = \{(\alpha, \gamma\beta), (Y, X\gamma)\}$ we have $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$. The auxiliary tableau system now gradually modifies and extends R until it becomes a fundamental relation with this property. While doing this we may need to introduce new goals.

The auxiliary tableau system is built around sequents of the form $R \vdash_{\mathbf{T}} \Gamma$ where R is a finite subset of $Var \times Var^+$ and Γ is a finite set of equations over Var^*. Since the relations R constructed are all fundamental (by Proposition 4.4.2 below), they are all confluent and strongly normalizing, so for any α its unique normal form $\alpha \downarrow R$ is known to exist.

At all times during the auxiliary tableau construction we rewrite as much of Γ as much as possible using R. We may then need to introduce new goals or extend R. There are in general three possible situations possibly at any point where this can happen:

[1] Pun intended.

- If an equation $X\alpha = Y\beta$ has the residual $\alpha = \gamma\beta$ and $R \cup \{(Y, X\gamma)\}$ is fundamental we simply extend R with the pair $(Y, X\gamma)$. This justifies the rule EXTEND.

- If an equation $X\alpha = Y\beta$ has the residual $\alpha = \gamma\beta$ but $R \cup \{(Y, X\gamma)\}$ is not fundamental because $Y \in Dom(R)$ with $(Y, X_1\gamma_1) \in R$ for some $X_1\gamma_1$. Then we must also compare $X_1\gamma_1$ and $X\gamma$. This gives rise to the rule COMPARE.

- If an equation $X\alpha = Y\beta$ has the residual $\alpha = \gamma\beta$ but $R \cup \{(Y, X\gamma)\}$ is not fundamental even though $Y \notin Dom(R)$ because some variable $Z \in Dom(R)$ occurs in $X\gamma$. We must rewrite $X\gamma$ and can then add $(Y, (X\gamma) \downarrow R)$ to R. This is the basis of the rule UPDATE.

The rule REWRITE tells us that we must rewrite using R whenever possible. And finally there are the rules CONGL and CONGR whose purpose is to remove identical heads and tails from equations. CONGR is strictly speaking not necessary but has been included for reasons of symmetry.

The rules have the following *priority*: CONGL must always be used whenever possible to remove identical leftmost variables. Next in priority is REWRITE. Finally, the other rules have equal priority. Thus we see that REWRITE will only be used between updatings of R, as desired.

The rules in Table 4–3 are sound w.r.t. $\sim$.

Definition 4.4.1 *A relation $R \subseteq Var \times Var^+$ is said to be* $\sim$ -consistent *if $R \subseteq \sim$. Similarly, a set of equations Γ is called* $\sim$ -consistent *if for all $\alpha' = \beta' \in \Gamma'$ we have $\alpha' \sim \beta'$.*

Proposition 4.4.1 (Soundness under $\sim$) *If $R' \vdash_{\mathrm{T}} \Gamma'$ has R' and Γ' $\sim$ -consistent then for the consequent $R'' \vdash_{\mathrm{T}} \Gamma''$ of any rule application to $R' \vdash_{\mathrm{T}} \Gamma'$, R'' and Γ'' are $\sim$ -consistent.*

PROOF: The soundness of EXTEND, COMPARE, UPDATE and REWRITE follows from Proposition 2.2.2. The soundness of CONGR follows from Lemma 2.2.2. The only mildly interesting case is CONGL. But if $\alpha\delta_1 \sim \alpha\delta_2$ we have for any $w \in Act$ such that $\alpha \stackrel{w}{\to} \epsilon$ that $\alpha\delta_1 \stackrel{w}{\to} \delta_1$ which is matched by some $\alpha\delta_2 \stackrel{w}{\to} \rho\delta_2$ with $\delta_1 \sim \rho\delta_2$. Since by Proposition 2.2.1 $|\delta_1| = |\delta_2|$ and $|\delta_1| = |\rho\delta_2|$ we get $|\rho| = 0$, which implies that $\rho = \epsilon$. □

The rules also preserve fundamentality:

Proposition 4.4.2 *If in $R' \vdash_{\mathbf{T}} \Gamma'$ we have that R' is fundamental and $\sim$ -consistent and Γ is $\sim$ -consistent, then for the consequent $R'' \vdash_{\mathbf{T}} \Gamma''$ of any rule application to $R' \vdash_{\mathbf{T}} \Gamma'$ we have that R'' is fundamental.*

PROOF: For EXTEND, preservation of fundamentality is required by the side condition. In the case of COMPARE, REWRITE, CONGL and CONGR, R' is not changed. In UPDATE we know that $Y \notin Dom(R)$ so $R' \cup \{(Y, (X\gamma) \downarrow R')\}$ is also functional. Since R' was assumed fundamental, no variable in $(X\gamma) \downarrow R'$ occurs in $Dom(R')$. Finally, by Proposition 4.4.1, $Y \sim (X\gamma) \downarrow R'$ so $|Y| = |(X\gamma) \downarrow R'|$. □

We can now be precise about the notion of an auxiliary tableau.

Definition 4.4.2 *An* auxiliary tableau *for $R \vdash_{\mathbf{T}} \Gamma$ is a maximal sequence of sequents $R_0 \vdash_{\mathbf{T}} \Gamma_0, R_1 \vdash_{\mathbf{T}} \Gamma_1, \ldots, R_{n-1} \vdash_{\mathbf{T}} \Gamma_{n-1}, R_n \vdash_{\mathbf{T}} \Gamma_n$ where $R \vdash_{\mathbf{T}} \Gamma = R_0 \vdash_{\mathbf{T}} \Gamma_0$ and for all $i \geq 0$ $R_{i+1} \vdash_{\mathbf{T}} \Gamma_{i+1}$ is the consequent of using a rule in Table 4–3 with $R_i \vdash_{\mathbf{T}} \Gamma_i$ as premise. An auxiliary tableau is finite if for some n all equations in Γ_n are identities (i.e. of the form $\alpha' = \alpha'$ for some α').*

Lemma 4.4.1 *If Γ is $\sim$* -consistent *all auxiliary tableaux for $\emptyset \vdash_{\mathbf{T}} \Gamma$ are finite.*

PROOF: Every time an equation $\alpha' = \beta'$ is replaced in a rule application, it is replaced by equations whose norms are all $\leq |\alpha'| = |\beta'|$. At least one of these new equations has norm $< |\alpha'|$. Thus we must eventually reach a situation where all equations in a sequent are identities, possibly of the form $\epsilon = \epsilon$. $\square$

Theorem 4.4.1 *If R_0 is fundamental and $\sim$ -consistent and Γ_0 is $\sim$ -consistent, then for any finite auxiliary tableau $R_0 \vdash_{\mathbf{T}} \Gamma_0, \ldots, R_n \vdash_{\mathbf{T}} \Gamma_n$ we have $\alpha \underset{R}{\longleftrightarrow}^* \beta$ with $R = R_n$ for any $\alpha = \beta \in \Gamma_0$.*

PROOF: We proceed by induction in n.

$n = 1$: The auxiliary tableau is $R_0 \vdash_{\mathbf{T}} \Gamma, R \vdash_{\mathbf{T}} \Gamma'$, and every equation $\alpha = \beta$ in Γ' is an identity. Thus, obviously $\alpha \underset{R}{\longleftrightarrow}^* \beta$. We now proceed by case analysis, looking at the rule used on an $X\alpha = Y\beta \in \Gamma$.

If the rule was EXTEND, we have $\alpha \underset{R}{\longleftrightarrow}^* \gamma\beta$ and $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$ for any $\alpha' = \beta' \in \Gamma'$. Therefore, $X\alpha \underset{R}{\longleftrightarrow}^* X\gamma\beta$ and $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$ as desired.

Had the rule been UPDATE, we would have $\alpha \downarrow R \underset{R}{\longleftrightarrow}^* (\gamma\beta) \downarrow R$ so $\alpha \underset{R}{\longleftrightarrow}^* \gamma\beta$ and $Y \underset{R}{\longleftrightarrow}^* X\gamma$, so again $X\alpha \underset{R}{\longleftrightarrow}^* X\gamma\beta$ implying $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$.

If the rule was REWRITE, we would have $(X\alpha) \downarrow R \underset{R}{\longleftrightarrow}^* (Y\beta) \downarrow R$ implying $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$.

And if the rule used had been COMPARE, we had $X_1\gamma_1 \underset{R}{\longleftrightarrow}^* X\gamma$ and $\alpha \underset{R}{\longleftrightarrow}^* \gamma\beta$ and $Y \underset{R}{\longleftrightarrow}^* X_1\gamma_1$. But then $Y\beta \underset{R}{\longleftrightarrow}^* X_1\gamma_1\beta$ and $X_1\gamma_1\beta \underset{R}{\longleftrightarrow}^* X\gamma\beta$, which implies that $Y\beta \underset{R}{\longleftrightarrow}^* X\gamma\beta$ and again $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$.

CONGL and CONGR are immediate.

Step: The auxiliary tableau is now $R_0 \vdash_{\mathbf{T}} \Gamma_0, R_1 \vdash_{\mathbf{T}} \Gamma_1 \ldots, R_n \vdash_{\mathbf{T}} \Gamma_n$ and $R_1 \vdash_{\mathbf{T}} \Gamma_1, \ldots, R_n \vdash_{\mathbf{T}} \Gamma_n$ is an auxiliary tableau for $R_1 \vdash_{\mathbf{T}} \Gamma_1$, so for every equation $\alpha' = \beta' \in \Gamma_1$ by induction hypothesis $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$. The proof proceeds exactly as in the base case. $\square$

From the above theorem and Lemma 4.4.1 we now get the desired result:

Corollary 4.4.1 *If $X\alpha \sim Y\beta$ by the successful tableau $\mathbf{T}$, for any auxiliary tableau $\emptyset \vdash_{\mathbf{T}} X\alpha = Y\beta, \ldots, R_n \vdash_{\mathbf{T}} \Gamma_n$ we have $X\alpha \underset{R}{\longleftrightarrow}^* Y\beta$ where $R = R_n$.*

Rules within subtableaux

REC $\dfrac{X\alpha = Y\beta}{E\alpha = F\beta}$ where $X \stackrel{\mathrm{def}}{=} E$ and $Y \stackrel{\mathrm{def}}{=} F$

PREFIX $\dfrac{a\alpha = a\beta}{\alpha = \beta}$

SUM $\dfrac{(\sum_{i=1}^{m} a_i\alpha_i)\alpha = (\sum_{j=1}^{n} b_j\beta_j)\beta}{\{a_i\alpha_i\alpha = b_{f(i)}\beta_{f(i)}\beta\}_{i=1}^{m} \ \{a_{g(j)}\alpha_{g(j)}\alpha = b_j\beta_j\beta\}_{j=1}^{n}}$

$$\textit{where} \quad \begin{array}{l} f : \{1, \ldots, m\} \to \{1, \ldots, n\} \\ g : \{1, \ldots, n\} \to \{1, \ldots, m\} \\ \textit{with } m, n \geq 1 \end{array}$$

Rules for new subtableaux

SUBL $\dfrac{\alpha_i\alpha = \beta_i\beta}{\alpha_i\gamma = \beta_i}$ where $\alpha = \gamma\beta$ is the residual

SUBR $\dfrac{\alpha_i\alpha = \beta_i\beta}{\alpha_i = \beta_i\gamma}$ where $\gamma\alpha = \beta$ is the residual

Table 4–1: The tableau rules

Equivalence

R1 $$\Gamma \vdash_\Delta E = E$$

R2 $$\frac{\Gamma \vdash_\Delta E = F}{\Gamma \vdash_\Delta F = E}$$

R3 $$\frac{\Gamma \vdash_\Delta E = F \quad \Gamma \vdash_\Delta F = G}{\Gamma \vdash_\Delta E = G}$$

Congruence

R4 $$\frac{\Gamma \vdash_\Delta E_1 = F_1 \quad \Gamma \vdash_\Delta E_2 = F_2}{\Gamma \vdash_\Delta E_1 + E_2 = F_1 + F_2}$$

R5 $$\frac{\Gamma \vdash_\Delta E_1 = F_1 \quad \Gamma \vdash_\Delta E_2 = F_2}{\Gamma \vdash_\Delta E_1 E_2 = F_1 F_2}$$

BPA axioms

R6 $$\Gamma \vdash_\Delta E + F = F + E$$

R7 $$\Gamma \vdash_\Delta (E + F) + G = E + (F + G)$$

R8 $$\Gamma \vdash_\Delta E + E = E$$

R9 $$\Gamma \vdash_\Delta (E + F)G = EG + FG$$

R10 $$\Gamma \vdash_\Delta E(FG) = (EF)G$$

Recursion

R11 $$\Gamma, X\alpha = Y\beta \vdash_\Delta X\alpha = Y\beta$$

R12 $$\frac{\Gamma, X\alpha = Y\beta \vdash_\Delta E\alpha = F\beta}{\Gamma \vdash_\Delta X\alpha = Y\beta} \qquad X \stackrel{\mathrm{def}}{=} E,\ Y \stackrel{\mathrm{def}}{=} F \in \Delta$$

Table 4–2: Rules of inference in the equational theory

EXTEND	$\dfrac{R \vdash_{\mathbf{T}} X\alpha = Y\beta, \Gamma}{R, Y = X\gamma \vdash_{\mathbf{T}} \alpha = \gamma\beta, \Gamma}$	if $\alpha = \gamma\beta$ is the residual of $X\alpha = Y\beta$ and $R, Y = X\gamma$ is fundamental
COMPARE	$\dfrac{R \vdash_{\mathbf{T}} X\alpha = Y\beta, \Gamma}{R \vdash_{\mathbf{T}} X_1\gamma_1 = X\gamma, \alpha = \gamma\beta, \Gamma}$	if $Y \in Dom(R)$ and $\alpha = \gamma\beta$ is the residual for $X\alpha = Y\beta$ but $(Y, X_1\gamma_1) \in R$
UPDATE	$\dfrac{R \vdash_{\mathbf{T}} X\alpha = Y\beta, \Gamma}{R, Y = (X\gamma)\downarrow R \vdash_{\mathbf{T}} (\alpha)\downarrow R = (\beta)\downarrow R, \Gamma}$	if $Y \notin Dom(R)$ and $\alpha = \gamma\beta$ is the residual for $X\alpha = Y\beta$ but some variable $Z \in Dom(R)$ occurs in $X\gamma$
REWRITE	$\dfrac{R \vdash_{\mathbf{T}} X\alpha = Y\beta, \Gamma}{R \vdash_{\mathbf{T}} (X\alpha)\downarrow R = (Y\beta)\downarrow R, \Gamma}$	if $(X\alpha)\downarrow R \neq X\alpha$ or $(Y\beta)\downarrow R \neq Y\beta$
CONGL	$\dfrac{R \vdash_{\mathbf{T}} \alpha\delta_1 = \alpha\delta_2, \Gamma}{R \vdash_{\mathbf{T}} \delta_1 = \delta_2, \Gamma}$	
CONGR	$\dfrac{R \vdash_{\mathbf{T}} \alpha_1\delta = \alpha_2\delta, \Gamma}{R \vdash_{\mathbf{T}} \alpha_1 = \alpha_2, \Gamma}$	

Table 4–3: Rules of the auxiliary tableau system

Chapter 5

Introducing silent actions

In this chapter we extend the decidability result obtained in the previous chapter by using a similar tableau method to show that the branching bisimilation equivalence introduced by van Glabbeek and Weijland in [vGW89b] is decidable for the class of normed recursively defined BPA processes with silent actions.

Section 5.1 introduces the notion of branching bisimilarity. Section 5.2 introduces the class of normed $\mathrm{BPA}^{\tau}_{\mathrm{rec}}$ processes. In Section 5.3 we describe the tableau system, prove its soundness and completeness, give a complexity measure and establish the decidability result for branching bisimilarity.

5.1 Branching bisimilarity

The processes that we will be looking at have their behavioural semantics given by transition graphs with silent actions. For comparison we first describe the notion of weak bisimulation equivalence, introduced by Milner [Mil80,Mil89]. This equivalence is essentially bisimulation equivalence defined on the derived *weak* transition relations that disregard silent actions.

Definition 5.1.1 *For a transition graph $\mathcal{G} = (Pr, Act \cup \{\tau\}, \rightarrow)$ with silent action τ, the* weak transition relations $\{\stackrel{s}{\Longrightarrow} | s \in Act \cup \{\epsilon\}\}$ *are given by* $\stackrel{a}{\Longrightarrow} \; = \; \stackrel{\tau}{\rightarrow}^{*} \stackrel{a}{\rightarrow} \stackrel{\tau}{\rightarrow}^{*}$

for $a \in Act$ *and* $\stackrel{\epsilon}{\Longrightarrow} \; = \; \stackrel{\tau}{\rightarrow}^{*}$

In the definition below, we use the 'observational' mapping $\phi : (Act \cup \{\tau\})^* \rightarrow Act^*$ which is the homomorphic extension of the function defined by $\phi(a) = a$ for $a \in Act$ and $\phi(\tau) = \epsilon$.

Definition 5.1.2 [Mil89] *A* weak bisimulation *on $\mathcal{G}$ is a symmetric relation $R \subseteq Pr \times Pr$ such that whenever pRq for any $a \in Act \cup \{\tau\}$ we have that $p \stackrel{a}{\rightarrow} p'$ implies that there exists a q' such that $q \stackrel{\phi(a)}{\Longrightarrow} q'$ with $p'Rq'$. We define $\approx$ by*

$$\approx \; = \{(p, q) \mid pRq \textit{ for some weak bisimulation } R\}$$

If $p \approx q$ we say that p and q are weakly bisimilar.

The notion of branching bisimilarity was put forward by van Glabbeek and Weijland in [vGW89b] as an alternative to weak bisimulation.

Definition 5.1.3 [vGW89b] *A* branching bisimulation (bb) *on $\mathcal{G}$ is a symmetric relation $R \subseteq Pr \times Pr$ such that whenever pRq for any $a \in Act \cup \{\tau\}$ we have that $p \stackrel{a}{\rightarrow} p'$ implies*

- *$a = \tau$ and $p'Rq$ or*
- *there exist q_1', q' such that $q \stackrel{\epsilon}{\Longrightarrow} q_1' \stackrel{a}{\rightarrow} q'$ with $pRq_1', p'Rq'$*

We define $\approx_b$ by

$$\approx_b \; = \{(p, q) \mid pRq \textit{ for some bb } R\}$$

If $p \approx_b q$ we say that p and q are branching bisimilar.

Unlike weak bisimulation equivalence, changes in branching properties caused by individual τ-transitions must always be taken into account in branching bisimulation. (Example 5.2.2 provides an example of the importance of this, namely two processes that are weakly bisimilar but not branching bisimilar). An equivalent definition which reflects this *stuttering property* better is the one below which we will be using in the tableau system presented in Section 5.3.

Proposition 5.1.1 *A branching bisimulation on $\mathcal{G}$ is a symmetric relation $R \subseteq Pr \times Pr$ such that whenever pRq for any $a \in Act \cup \{\tau\}$ we have that if $p \xrightarrow{a} p'$ then either*

- *$a = \tau$ and $p'Rq$ or*
- *there exist $q'_0, \ldots, q'_n, q'$ such that $q = q'_0 \xrightarrow{\tau} q'_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} q'_n \xrightarrow{a} q'$ with pRq'_i for $0 \leq i \leq n$ and $p'Rq'$.*

van Glabbeek proves that the above notion of branching bisimilarity is indeed equivalent to that of Definition 5.1.3 by introducing a notion of what he calls a semi-branching bisimulation where the conditions for matching $p \xrightarrow{\tau} p'$ have been relaxed to allow matches of the form $q \stackrel{\epsilon}{\Longrightarrow} q'$ with $p'Rq'$. He then proceeds to show that the maximal semi-bb satisfies the stuttering property of Definition 5.1.1. For the details of the proof, see [vG90a].

5.2 Normed $\mathbf{BPA}^{\tau}_{\mathrm{rec}}$

Extending BPA with silent actions gives us the class of processes $\mathrm{BPA}^{\tau}_{\mathrm{rec}}$ [BK88]. Again, these are processes given by systems of defining equations $\Delta = \{X_i \stackrel{\mathrm{def}}{=} E_i \mid 1 \leq i \leq m\}$. The process expressions E_i are now given by the syntax

$$E ::= a \mid \tau \mid E_1 + E_2 \mid E_1 E_2 \mid X$$

where τ is a new, silent action not in Act. As in the previous chapter, elements of Var^* will be denoted by Greek letters: $\alpha, \beta, \ldots$ The operational semantics given by the transition relations $\{ \xrightarrow{a} \mid a \in Act \cup \{\tau\}\}$ is as given in Definition 2.2.3.

We restrict our attention to *weakly normed* systems of equations.

Definition 5.2.1 *The* weak norm *of any $X \in Var$ is given by*

$$||X|| = \min\{length(w) \mid X \stackrel{w}{\Longrightarrow} \epsilon, w \in Act^*\}$$

A system of defining equations Δ *is* weakly normed *if for any* $X \in Var$ $0 < ||X|| < \infty$. *The maximal norm of any variable in* Δ *is defined by* $M_\Delta = \max\{||X|| | X \in Var\}$.

Since norms must be strictly positive, all variables must eventually perform an observable action and processes can therefore *not* terminate silently. While this is an important restriction, it is also in analogy with the requirement that a grammar has no empty productions.

In Section 5.3 we shall also need the notion of norm from the previous chapter; to avoid confusion we shall refer to $|X|$ as the *strong norm* of X. Clearly, if Δ is weakly normed it is also strongly normed.

Finally, we again restrict our attention to systems of defining equations given in 3-Greibach Normal Form (3-GNF). As before, this is actually no real restriction, since by Theorem 2.2.1 any system of equations Δ in $\mathrm{BPA}^\tau_{\mathrm{rec}}$ can effectively be rewritten to a Δ' which is *strongly* bisimilar to Δ and therefore normed iff Δ is [BBK87a]. This once again leaves us with transition graphs whose states are strings of process variables; the further restriction to variable sequences of length at most 2 guarantees limited growth when determining single transitions: Proposition 2.2.5 again applies.

Because weak norms are assumed strictly positive, we retain the simple relationship between lengths and norms of Proposition 2.2.6:

Proposition 5.2.1 *For* $\alpha \in Var^*$ $length(\alpha) \leq ||\alpha||$ *and* $||\alpha|| \leq M_\Delta length(\alpha)$.

The weak norm is additive under sequential composition:

Proposition 5.2.2 *For* $\alpha, \beta \in Var^*$ $||\alpha\beta|| = ||\alpha|| + ||\beta||$.

PROOF: As for Proposition 2.2.1. $\square$

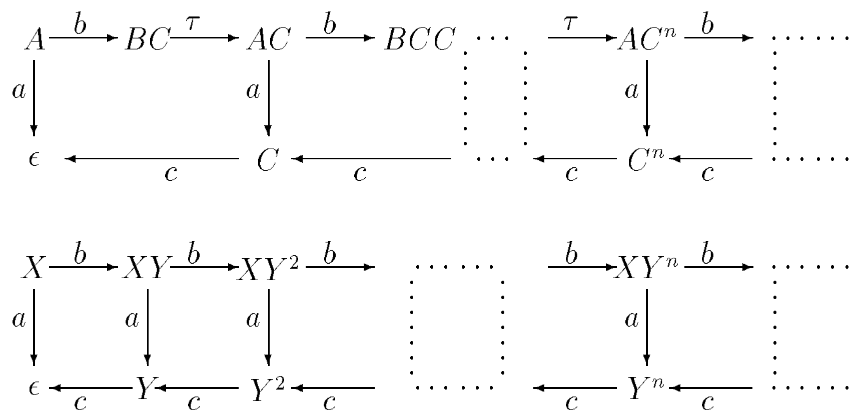

Figure 5–1: Transition graphs for $\Delta_1 = \{A \stackrel{\mathrm{def}}{=} a + bBC;\ B \stackrel{\mathrm{def}}{=} \tau A;\ C \stackrel{\mathrm{def}}{=} c\}$ (top) and $\Delta_2 = \{X \stackrel{\mathrm{def}}{=} a + bXY;\ Y \stackrel{\mathrm{def}}{=} c\}$ (bottom)

Definition 5.2.2 *The* observational language $L_{obs}(X)$ *accepted by a variable* X *is defined by* $L_{obs}(X) = \{w \mid X \stackrel{obs}{\Longrightarrow} \epsilon\}$.

We have

Proposition 5.2.3 [vG90a] *If* $\alpha \approx_b \beta$ *then* $L_{obs}(\alpha) = L_{obs}(\beta)$.

Note that for weakly normed systems this implies

Proposition 5.2.4 $\alpha \approx_b \beta$ *implies that* $||\alpha|| = ||\beta||$.

Example 5.2.1 Consider $\Delta_1 = \{A \stackrel{\mathrm{def}}{=} a + bBC;\ B \stackrel{\mathrm{def}}{=} \tau A;\ C \stackrel{\mathrm{def}}{=} c\}$ and $\Delta_2 = \{X \stackrel{\mathrm{def}}{=} a + bXY;\ Y \stackrel{\mathrm{def}}{=} c\}$ (cf. Example 2.2.1). The transition graphs are shown in Figure 5–1. For Δ_1 we have $L_{obs}(A) = \{b^n ac^n \mid n \geq 0\}$, $L_{obs}(B) = L_{obs}(A)$ and $L_{obs}(C) = \{c\}$. $X \approx_b A$ because of the branching bisimulation

$$\{(XY^n, AC^n) \mid n \geq 0\} \cup \{(XY^n, BC^n) \mid n \geq 1\} \cup \{(Y^n, C^n) \mid n \geq 1\} \cup \{(\epsilon, \epsilon)\}$$

$\square$

For the tableau system we need the counterparts of Proposition 2.2.2 and Lemma 2.2.2. (The proofs that follow utilize the notion of bb from Definition 5.1.3.) Firstly, $\approx_b$ is a congruence w.r.t. sequential composition:

Proposition 5.2.5 *If $\alpha_1 \approx_b \beta_1$ and $\alpha_2 \approx_b \beta_2$ then $\alpha_1\alpha_2 \approx_b \beta_1\beta_2$*

PROOF: $R = \{(\alpha_1\alpha_2, \beta_1\beta_2) \mid \alpha_1 \approx_b \beta_1, \alpha_2 \approx_b \beta_2\}$ is a bb. It is obvious that R is symmetric. Suppose $(\alpha_1\alpha_2, \beta_1\beta_2) \in R$. If $\alpha_1\alpha_2 \xrightarrow{a} \alpha'$ this must be due to either $\alpha_1 \xrightarrow{a} \alpha_1'$ or $\alpha_1 = \epsilon$ and $\alpha_2 \xrightarrow{a} \alpha_2'$. In the former case $\alpha_1\alpha_2 \xrightarrow{a} \alpha_1'\alpha_2$, in the latter $\alpha_1\alpha_2 \xrightarrow{a} \alpha_2'$. If $\alpha_1 \xrightarrow{a} \alpha_1'$ there are two possibilities. Either $a = \tau$ and $\alpha_1' \approx_b \beta_1$, but then $(\alpha_1'\alpha_2, \beta_1\beta_2) \in R$. Or there exist β_1', β_1'' with $\beta \stackrel{\epsilon}{\Longrightarrow} \beta_1' \xrightarrow{a} \beta_1''$ such that $\alpha_1 \approx_b \beta_1'$ and $\alpha_1' \approx_b \beta_1''$. Then $\beta_1\beta_2 \stackrel{\epsilon}{\Longrightarrow} \beta_1'\beta_2 \xrightarrow{a} \beta_1''\beta_2$ and $(\alpha_1\alpha_2, \beta_1'\beta_2) \in R$, $(\alpha_1'\alpha_2, \beta_1''\beta_2) \in R$. If $\alpha_1 = \epsilon$ we must have $\beta_1 = \epsilon$ and thus $\beta_2 \stackrel{\epsilon}{\Longrightarrow} \beta_2' \xrightarrow{a} \beta_2''$ with $\alpha_2 \approx_b \beta_2'$ and $\alpha_2' \approx_b \beta_2''$, so $(\alpha_1\alpha_2, \beta_1\beta_2') \in R$ and $(\alpha_1\alpha_2', \beta_1\beta_2'') \in R$. □

The other result is a new version of the 'split' lemma.

Lemma 5.2.1 *If $\alpha_1\alpha \approx_b \alpha_2\alpha$ then $\alpha_1 \approx_b \alpha_2$.*

PROOF: By Proposition 5.2.2 $||\alpha_1|| = ||\alpha_2||$, and since all variables are normed, $\alpha_1 = \epsilon$ iff $\alpha_2 = \epsilon$. Now $R = \{(\alpha_1, \alpha_2) \mid \alpha_1\alpha \approx_b \alpha_2\alpha\}$ is a bb. It is obvious that R is symmetric. Suppose $(\alpha_1, \alpha_2) \in R$ and $\alpha_1 \neq \epsilon$. Then also $\alpha_2 \neq \epsilon$, since $||\alpha_1|| = ||\alpha_2||$. So if $\alpha_1 \xrightarrow{a} \alpha_1'$ then $\alpha_1\alpha \xrightarrow{a} \alpha_1'\alpha$ can either be matched by $a = \tau$ and $\alpha_1'\alpha \approx_b \alpha_2\alpha$, implying $\alpha_1' R \alpha_2$, or by $\alpha_2\alpha \stackrel{\epsilon}{\Longrightarrow} \alpha_2'\alpha \xrightarrow{a} \alpha_2''\alpha$ with $\alpha_2'\alpha \approx_b \alpha_1\alpha$ and $\alpha_2''\alpha \approx_b \alpha_1'\alpha$. The latter holds, since weak norms are positive, so $\alpha_2\alpha \stackrel{\epsilon}{\Longrightarrow} \xrightarrow{a}$ because $\alpha_2 \stackrel{\epsilon}{\Longrightarrow} \xrightarrow{a}$. □

It is important to note that this does *not* hold for weak bisimulation. The following counterexample arose in a discussion with Kim Larsen and is due to him.

Example 5.2.2 Consider $\Delta = \{X = aY, Y = a + \tau X, A = a + aB, B = a\}$. As $||X|| = 2, ||Y|| = 1, ||A|| = 1$ and $||B|| = 1$, Δ clearly obeys all requirements stated above. It is easily seen that $X \approx BY$ and that $A \not\approx B$. However, we have $AY \approx BY$, since $\{(AY, BY), (BY, X), (Y, Y), (\epsilon, \epsilon), (X, X)\}$ is a weak bisimulation. The problem lies in the fact that weak bisimilarity does not require the results of intermediate steps in weak transitions to be related. In particular, $AY \xrightarrow{a} BY$ is matched by $BY \overset{a}{\Longrightarrow} X$. The latter is due to $BY \xrightarrow{a} Y \xrightarrow{\tau} X$, where we clearly have that $AY \not\approx Y$. □

5.3 A tableau system for branching bisimulation

5.3.1 Building tableaux

A tableau for determining branching bisimilarity is a maximal proof tree built using the proof rules in Table 5–1. Tableaux consist of a number of subtableaux. These are built from successive applications of the STEP rule, which corresponds to the notion of a basic step in Chapter 4 (Definition 4.2.2).

STEP is applicable iff there is a possibility of matching transitions. A *possible match* is any set of equations whose sides are the results of successful matching transitions according to the definition of branching bisimilarity in Proposition 5.1.1:

Definition 5.3.1 *A set of equations M is a* possible match *for $\alpha = \beta$ if for any $a \in Act$ we have that if $\alpha \xrightarrow{a} \alpha'$ then either*

- *$a = \tau$ and $\alpha' = \beta \in M$ or*
- *there exist $\beta_0' = \beta, \ldots, \beta_n', \beta'$ s.t. $\beta_0 \xrightarrow{\tau} \beta_1' \xrightarrow{\tau} \cdots \xrightarrow{\tau} \beta_n' \xrightarrow{a} \beta'$ with $\alpha = \beta_i' \in M$ for $0 \leq i \leq n$ and $\alpha' = \beta' \in M$.*

and similarly for any $\beta \xrightarrow{a} \beta'$.

This definition appears to allow *infinitely* many possible matches, since there seems to be no bound on the length n of a matching transition sequence. However, this is not the case. Firstly, we have

Proposition 5.3.1 *If $\alpha \approx_b \beta$ we can find a possible match M for $\alpha = \beta$ such that whenever $\alpha \xrightarrow{a} \alpha'$ is matched by $\beta_0' = \beta, \ldots, \beta_n', \beta'$ such that $\beta_0 \xrightarrow{\tau} \beta_1' \xrightarrow{\tau} \cdots \xrightarrow{\tau} \beta_n' \xrightarrow{a} \beta'$ with $\alpha = \beta_i' \in M$ for $0 \leq i \leq n$ and $\alpha' = \beta' \in M$ all β_i' $(0 \leq i \leq n)$ are distinct.*

PROOF: If some state β_j occurred twice in $\beta_0 \xrightarrow{\tau} \beta_1' \xrightarrow{\tau} \cdots \xrightarrow{\tau} \beta_n' \xrightarrow{a} \beta'$ we could remove any states between the two occurrences of β_j and still have a matching sequence. □

Secondly, we have

Proposition 5.3.2 *If $X\alpha \approx_b Y\beta$ and $X\alpha \xrightarrow{a} \alpha'\alpha$ is matched by $Y\beta \xRightarrow{\epsilon} \beta_1'\beta \xrightarrow{a} \beta'\beta$ any intermediate state β'' in $Y\beta \xRightarrow{\epsilon} \beta_1'\beta$ has $length(\beta'') \leq M_\Delta + length(\beta)$. Furthermore, $length(\beta'\beta) \leq M_\Delta + length(\beta) + 1$*

PROOF: For any single transition step $\beta_1 \xrightarrow{\tau} \beta_2$ in $Y\beta \xRightarrow{\epsilon} \beta_1'\beta$ Proposition 2.2.5 applies, so $length(\beta_2) \leq 1 + length(\beta_1)$. Moreover, we must have $\beta_2 \approx_b X\alpha$ so by Proposition 5.2.4 we have $||\beta_2|| = ||X\alpha||$. Thus, in the worst case, where $||Y|| = M_\Delta$, we would have replaced Y by a sequence of M_Δ variables each having weak norm 1. □

The outbranching is a multiple of the bound $B_{X,Y}$ on the number of single transition steps for $X\alpha = Y\beta$; this factor only depends on the leftmost variables.

Definition 5.3.2 *$B_{X,Y}$ is the cardinality of*

$$\{\alpha' \mid X \xrightarrow{a} \alpha', a \in Act \cup \{\tau\}\} \cup \{\beta' \mid Y\beta \xrightarrow{a} \beta', a \in Act \cup \{\tau\}\}$$

Proposition 5.3.3 *Let v be the cardinality of Var. If $X\alpha \approx_b Y\beta$, there is a possible match for $X\alpha = Y\beta$ with at most $B_{X,Y} \sum_{j=2}^{2K} (j-1)v^j$ equations in a match, where $K = M_\Delta + 1 + \max(length(\alpha), length(\beta))$.*

PROOF: By Proposition 5.3.2, we see that there is a possible match for $X\alpha = Y\beta$ with expressions with lengths bounded by $K = M_\Delta + 1 + \max(length(\alpha), length(\beta))$.

So the total length on an expression is at most $2K$, meaning that there are at most $\sum_{j=2}^{2K}(j-1)v^j$ different equations for any $X\alpha \xrightarrow{a} \alpha'\alpha$ in a possible match. □

Clearly, STEP is forwards sound in the following sense:

Proposition 5.3.4 (Forwards soundness of STEP) *If $\alpha \approx_b \beta$, then there is a possible match M such that whenever $\alpha' = \beta' \in M$ we have $\alpha' \approx_b \beta'$.*

The tableau construction procedure is analogous to that presented in Section 4.2. An eliminating subtableau for $X\alpha = Y\beta$ consists of attempted matches to the depth where an equation of the form $\alpha = \gamma\beta$ is reached. When $|X| \leq |Y|$ each non-residual leaf of an eliminating subtableau for $X\alpha = Y\beta$ is either labelled $\alpha = \gamma\beta$ (a *residual* of the subtableau), or $\alpha_i\alpha = \beta_i\beta$. Because the number of successive attempted matches is $|X|$ there is at least one residual and since all norms are strictly positive, α and β must persist as suffixes throughout the subtableau. As before, for any such subtableau we pick one residual node and call it *the* residual. If instead $|Y| < |X|$ the same holds, only now the residual is $\gamma\alpha = \beta$.

Unless a subtableau leaf is a successful terminal (Definition 5.3.4 below) it is used as the basis of a new subtableau. However, before a new subtableau is constructed, for every leaf one of the SUB rules is used to trim the length of the expressions in the new subtableau root. The SUB rules work just as the SUB rules in the previous chapter, and are forwards sound in the same sense:

Proposition 5.3.5 (Soundness of SUBL and SUBR) *If $\alpha_i\alpha \approx_b \beta_i\beta$ and $\alpha \approx_b \gamma\beta$ then $\alpha_i\gamma \approx_b \beta_i$. If $\gamma\alpha_i \approx_b \beta_i$ then $\alpha_i \approx_b \beta_i\gamma$*

Proof: Exactly as in the proof of Proposition 4.2.4, only now using Propositions 5.2.5 and 5.2.1. □

The rules are only applied to nodes that are not *terminal*. Terminal nodes can either be *successful* or *unsuccessful*.

Definition 5.3.3 *A tableau node is an* unsuccessful terminal *if it has one of the forms*

1. $\alpha = \beta$ *with* $||\alpha|| \neq ||\beta||$

2. $\alpha = \beta$ *with* $\alpha \neq \epsilon, \beta \neq \epsilon$ *and no possible match exists (i.e.* STEP *is inapplicable).*

In both of these cases it is obvious that the expressions compared are not branching bisimilar. Thus, whenever we see an unsuccessful terminal the whole tableau construction aborts.

The nodes that can be successful terminals are those that are potential roots of eliminating subtableaux:

Definition 5.3.4 *A residual or consequent of an application of a* SUB *rule is a* successful terminal *if it has one of the forms*

1. $\alpha = \beta$ *where there is another subtableau root above it on the path from the root also labelled* $\alpha = \beta$

2. $\alpha = \alpha$

$$\dfrac{X = A}{\epsilon = \epsilon \qquad \dfrac{\dfrac{XY = BC}{XY = BC}\,\mathsf{SUBL}}{\dfrac{XY = AC}{X = A}\,\mathsf{SUBL} \quad \dfrac{XYY = BCC}{XY = BC}\,\mathsf{SUBL} \quad \dfrac{Y = C}{\epsilon = \epsilon}\,\mathsf{STEP}}\,\mathsf{STEP}}\,\mathsf{STEP}$$

Figure 5–2: A successful tableau for $X = A$

Example 5.3.1 (Example 5.2.1 cont.) The tableau in Figure 5–2 is a successful tableau for $X = A$. □

5.3.2 Termination, completeness, and soundness

It is important for our decidability result that all tableaux are finite. This follows from reasoning entirely similar to that for the tableau system for strong bisimulation.

Theorem 5.3.1 *For any equation $\alpha = \beta$ all tableaux are finite.*

PROOF: Since our tableaux are finitely branching by Proposition 5.3.3, by König's Lemma an infinite tableau would have an infinite path. This would then be caused by the combined absence of unsuccessful termination and the successful termination condition 1 along that path. Since we have assumed 3-GNF and normedness, there is a uniform bound on the total length of the consequent of a SUB rule. Assume wlog that we have a subtableau with root $X\alpha = Y\beta$ and that a SUBL rule was applied to a subtableau leaf:

$$\frac{\alpha_1\alpha = \beta_1\beta}{\alpha_1\gamma = \beta_1} \quad \mathsf{SUBL}$$

Since the depth of the subtableau is at most m_Δ, repeated applications of Proposition 5.3.2 tell us that $length(\alpha_1) \leq m_\Delta(M_\Delta + 1)$, $length(\beta_1) \leq m_\Delta(M_\Delta + 1)$ and $length(\gamma) \leq m_\Delta(M_\Delta + 1)$. This implies a uniform bound on the length of SUB consequents of $3m_\Delta(M_\Delta + 1)$, so there can be no infinite path through infinitely many SUB applications since there are of course only finitely many different equations of any given length. Nor can an infinite path pass through infinitely many residuals. For if a residual $\alpha_0 = \beta_0$ is above the residual $\alpha_1 = \beta_1$ we have that $||\alpha_0|| = ||\beta_0|| < ||\alpha_1|| = ||\beta_1||$. By Proposition 5.2.1, any subsequence of residuals therefore has a uniform bound on the total lengths of expressions compared, again ensuring termination. $\square$

It is easily seen that the tableau system is complete:

Theorem 5.3.2 *If $\alpha \approx_b \beta$, $\alpha = \beta$ has a successful tableau.*

PROOF: By the forwards soundness of the STEP and SUB rules (Propositions 5.3.4 and 5.3.5) we can use the tableau rules in such a way that only valid consequents arise. Clearly this must give rise to a finite, successful tableau. □

Finally we must show soundness of the tableau system, namely that the existence of a successful tableau for $\alpha = \beta$ indicates that $\alpha \approx_b \beta$. This follows from the fact that the tableau system tries to construct a 'self-branching bisimulation', which, if a successful tableau is reached, consists of the symmetric closure of the set of nodes in the successful tableau. This notion is the counterpart of the notion of a self-bisimulation defined in Section 2.2.5 and used in the proof of Theorem 4.2.3. In order to define the corresponding notion for branching bisimulation, we need a simple rephrasing of Proposition 5.1.1:

Proposition 5.3.6 *A branching bisimulation on a transition graph $\mathcal{G}$ is a symmetric relation $R \subseteq Pr \times Pr$ such that whenever pRq for any $a \in Act \cup \{\tau\}$ we have that if $p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \cdots p_m \xrightarrow{a} p'$ then there exist $q_0, q_1, \ldots, q_m, q'$ such that $q_0 = q$ and $p_i R q_i$ for $1 \leq i \leq m$, $p' R q'$ and for $i < m$*

- - *$q_i = q_{i+1}$ or*
  - *there exist $q_{i_1}, \ldots, q_{i_{n(i)}}$ such that $q_i \xrightarrow{\tau} q_{i_1} \xrightarrow{\tau} \cdots q_{i_{n(i)}} \xrightarrow{\tau} q_{i+1}$ with $p_i R q_{i_j}$ for $1 \leq j \leq n(i)$*

and either

- *$a = \tau$ and $q_m = q'$ or*
- *there exist $q_{m_1}, \ldots, q_{m_{n(m)}}$ such that $q_m \xrightarrow{\tau} q_{m_1} \xrightarrow{\tau} \cdots q_{m_{n(m)}} \xrightarrow{a} q'$*

PROOF: Clearly, any relation that satisfies the conditions of the proposition is a bb (let $m = 0$). For the reverse direction, suppose $p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \cdots p_m \xrightarrow{a} p'$. One then uses a straightforward induction in m.

Base case - $m = 0$: This is immediate, since here the definitions coincide.

Step - assuming for $m = k$: Here we know by induction hypothesis that the transition sequence $p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \cdots p_m$ can be matched according to the conditions in the proposition. We then extend the match to cover $p_m \xrightarrow{a} p'$ by appealing to Proposition 5.1.1.

$\square$

Recall that $\underset{R}{\longleftrightarrow}^*$ is the least congruence under sequential composition containing R (Definition 2.2.9).

Definition 5.3.5 *A* branching bisimulation up to sequential congruence (sbb) *is a symmetric relation $R \subseteq Var^* \times Var^*$ such that whenever $\alpha R \beta$ $\alpha = \epsilon$ iff $\beta = \epsilon$ and for any $a \in Act \cup \{\tau\}$ we have that if $\alpha = \alpha_0 \xrightarrow{\tau} \alpha_1 \xrightarrow{\tau} \cdots \alpha_m \xrightarrow{a} \alpha'$ then there exist $\beta_0, \beta_1, \ldots, \beta_m, \beta'$ such that $\beta_0 = \beta$ and $\alpha_i \underset{R}{\longleftrightarrow}^* \beta_i$ for $1 \leq i \leq m$, $\alpha' \underset{R}{\longleftrightarrow}^* \beta'$ and for $i < m$*

-
  - *$\beta_i = \beta_{i+1}$ or*
  - *there exist $\beta_{i_1}, \ldots, \beta_{i_{n(i)}}$ s.t. $\beta_i \xrightarrow{\tau} \beta_{i_1} \xrightarrow{\tau} \cdots \beta_{i_{n(i)}} \xrightarrow{\tau} \beta_{i+1}$ with $\alpha_i \underset{R}{\longleftrightarrow}^* \beta_{ij}$ for $1 \leq j \leq n(i)$*

and either

- *$a = \tau$ and $\beta_m = \beta'$ or*
- *there exist $\beta_{m_1}, \ldots, \beta_{m_{n(i)}}$ s.t. $\beta_m \xrightarrow{\tau} \beta_{m_1} \xrightarrow{\tau} \cdots \beta_{m_{n(m)}} \xrightarrow{a} \beta'$ with $\alpha_m \underset{R}{\longleftrightarrow}^* \beta_{mj}$ for $1 \leq j \leq n(m)$.*

The reason why a bisimulation up to sequential congruence can be said to be an essential part of a bisimulation lies in the following result, which is a generalization of Lemma 2.2.1.

Lemma 5.3.1 *If R is an sbb then $\underset{R}{\longleftrightarrow}^*$ is a bb.*

Proof: It is clear that $\underset{R}{\longleftrightarrow}^*$ is symmetric. Now suppose $\alpha \underset{R}{\longleftrightarrow}^* \beta$; we must show that we can match transitions within $\underset{R}{\longleftrightarrow}^*$ as required by Definition 5.3.5. $\alpha \underset{R}{\longleftrightarrow}^* \beta$ must be due to $\alpha \underset{R}{\longleftrightarrow}^n \beta$ for some n, and the proof now proceeds by induction in n:

Base case 1, $n = 0$ is obvious, since $\alpha = \beta$ in this case.

Base case 2, $n = 1$ must have $\alpha = \sigma\alpha_0\chi, \beta = \sigma\beta_0\chi$ with $\alpha_0 R \beta_0$. If $\alpha_0 = \beta_0 = \epsilon$, we are done. Otherwise, if $\sigma \neq \epsilon$ any transition sequence

$$\alpha \xrightarrow{\tau} \alpha_1 \xrightarrow{\tau} \cdots \alpha_m \xrightarrow{a} \alpha' \tag{5.1}$$

is due to

$$\sigma \xrightarrow{\tau} \sigma_1 \xrightarrow{\tau} \cdots \sigma_m \xrightarrow{a} \sigma'$$

with $\alpha_i = \sigma_i\alpha_0\chi$ for $1 \leq i \leq m$ and $\alpha' = \sigma'\alpha_0\chi$. It is easy to see that this transition sequence can be matched by

$$\sigma\beta_0\chi \xrightarrow{\tau} \sigma_1\beta_0\chi \xrightarrow{\tau} \cdots \sigma_m\beta_0\chi \xrightarrow{a} \sigma'\beta_0\chi$$

with $\sigma_i\alpha_0\chi \underset{R}{\longleftrightarrow}^* \sigma_i\beta_0\chi$ for all $1 \leq i \leq m$ and $\sigma'\alpha_0\chi \underset{R}{\longleftrightarrow}^* \sigma'\beta_0\chi$.

If $\sigma = \epsilon$, (5.1) becomes

$$\alpha_0\chi \xrightarrow{\tau} \alpha_{01}\chi \xrightarrow{\tau} \cdots \alpha_{0m}\chi \xrightarrow{a} \alpha_0'\chi \tag{5.2}$$

due to

$$\alpha_0 \xrightarrow{\tau} \alpha_{01} \xrightarrow{\tau} \cdots \alpha_{0m} \xrightarrow{a} \alpha_0'$$

There exist $\beta_{00}, \ldots, \beta_{0m}, \beta_0'$ where for all i either $\beta_{0i} = \beta_{0i+1}$ for $i \geq 1$ or there exist $\beta_{0i1}', \ldots, \beta_{0in(i)}'$ such that $\beta_0 = \beta_{00} \xrightarrow{\tau} \cdots \beta_{00n(0)} \xrightarrow{\tau} \beta_{01} \xrightarrow{\tau} \cdots \beta_{0m} \xrightarrow{\tau} \cdots \beta_{0mn(m)} \xrightarrow{a} \beta_0'$ is a matching transition sequence with $\alpha_{0i} \underset{R}{\longleftrightarrow}^* \beta_{0ij}$ for $1 \leq i \leq m$ and $1 \leq j \leq n(i)$ and $\alpha_0' \underset{R}{\longleftrightarrow}^* \beta_0'$. If $a = \tau$, possibly $\beta_{0m}' = \beta_0'$. Here (5.2) is matched by

$$\beta_0\chi \xrightarrow{\tau} \cdots \beta_{01}\chi \xrightarrow{\tau} \cdots \beta_{0m}\chi \xrightarrow{\tau} \cdots \xrightarrow{a} \beta'_0\chi$$

where possibly $\beta'_{0m}\chi = \beta'_0\chi$.

Step, $n = k + 1$ assuming for k, where $k \geq 1$: Here there is an ρ such that $\alpha \underset{R}{\leftrightarrow}^k \rho \underset{R}{\leftrightarrow} \beta$. By induction hypothesis, we must have that (5.1) is matched using $\rho_0 = \rho, \rho_1, \ldots, \rho_m, \rho'$ where $\alpha_i \underset{R}{\leftrightarrow}^* \rho_i$ and either $\rho_i = \rho_{i+1}$ or there exist $\rho_{i1}, \ldots, \rho_{in(i)}$ such that

$$\rho_i \xrightarrow{\tau} \rho_{i1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} \rho_{in(i)} \xrightarrow{\tau} \rho_{i+1} \tag{5.3}$$

with $\alpha_i \underset{R}{\leftrightarrow}^* \rho_{ij}$ for $1 \leq j \leq n(i)$ and either $\rho_m = \rho'$ with $a = \tau$ or there exist $\rho_{m1}, \ldots, \rho_{mn(m)}$ such that

$$\rho_m \xrightarrow{\tau} \rho_{m1} \xrightarrow{\tau} \cdots \xrightarrow{\tau} \rho_{mn(n)} \xrightarrow{\tau} \rho'$$

Each transition sequence of the form (5.3) can be matched by β_i, β_{i+1} such that either $\beta_i = \beta_{i+1}$ or there exist $\beta_{i1}, \ldots, \beta_{ik(i)}$ such that

$$\beta_i \xrightarrow{\tau} \beta_{i1} \xrightarrow{\tau} \beta_{i2} \cdots \beta_{ik_i} \xrightarrow{\tau} \beta_{i+1}$$

such that for all $1 \leq i \leq m$ we have $\rho_i \underset{R}{\leftrightarrow}^* \beta_i$ and $\rho_i \underset{R}{\leftrightarrow}^* \beta_{ij}$ for $1 \leq j \leq n(i)$. And if $\rho_m \neq \rho'$ we can match similarly with $\beta_m \xrightarrow{\tau} \cdots \xrightarrow{a} \beta'$. By transitivity of $\underset{R}{\leftrightarrow}^*$ we see that the concatenation of the matching sequences

$$\beta_0 \xrightarrow{\tau} \cdots \beta_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} \beta_m \xrightarrow{\tau} \cdots \xrightarrow{a} \beta'$$

constitutes the desired matching sequence of transitions for (5.1). □

Corollary 5.3.1 *$\alpha \approx_b \beta$ iff there is an sbb R such that $\alpha R \beta$.*

PROOF: From the above and from the fact that any bb is an sbb. □

We now have

Theorem 5.3.3 *If $\alpha = \beta$ has a successful tableau* $\mathbf{T}$ *then*

$$R_{\mathbf{T}} = \{(\alpha', \beta') \mid \alpha' = \beta' \textit{ or } \beta' = \alpha' \textit{ is an equation in } \mathbf{T}\}$$

is an sbb.

PROOF: It is obvious that $R_{\mathbf{T}}$ is symmetric. Since all variables have positive norms, we see that $\alpha' = \epsilon$ iff $\beta' = \epsilon$. We must now show that for any $(\alpha', \beta') \in R_{\mathbf{T}}$ any transition sequence

$$\alpha' \xrightarrow{\tau} \alpha'_1 \xrightarrow{\tau} \cdots \alpha'_m \xrightarrow{a} \alpha'' \tag{5.4}$$

can match within $\underset{R_{\mathbf{T}}}{\longleftrightarrow}^*$. We now proceed by induction in the length m of the τ-transition.

Base case, $m = 0$: Here (5.4) is $\alpha' \xrightarrow{a} \alpha'_1$, and there are now four cases to be considered, depending on where in $\mathbf{T}$ $\alpha' = \beta'$ is found.

- If $\alpha' = \beta'$ is a successful terminal due to condition 2, it is obvious that we can match within $\underset{R_{\mathbf{T}}}{\longleftrightarrow}^*$, since any least congruence contains the identity.

- If $\alpha' = \beta'$ is a successful terminal due to condition 1 this means that there is a subtableau root above it also labelled $\alpha' = \beta'$. This node must be the premise of a STEP application, and because $\mathbf{T}$ is successful, a possible match exists so we can find a matching sequence

$$\beta' \xrightarrow{\tau} \beta''_{11} \xrightarrow{\tau} \cdots \xrightarrow{\tau} \beta''_{1n(1)} \xrightarrow{a} \beta''$$

  where $(\alpha', \beta''_{1j}) \in R_{\mathbf{T}}$ for $1 \leq j \leq n(1)$ and $(\alpha'', \beta'') \in R_{\mathbf{T}}$.

- If $\alpha' = \beta'$ is an internal node and the premise of a STEP, we proceed exactly as in the previous case for termination condition 1.

- If $\alpha' = \beta'$ is an internal node and the premise of a SUB, assume wlog that the rule applied was SUBL. Further suppose $\alpha' = \beta'$ is $X_1\alpha_1\alpha = Y_1\beta_1\beta$ and that $\alpha = \gamma\beta$ is the residual. Then we have

$$\frac{X_1\alpha_1\alpha = Y_1\beta_1\beta}{X_1\alpha_1\gamma = Y_1\beta_1}\quad \mathsf{SUBL}$$

If $X_1\alpha_1\gamma = Y_1\beta$ is a terminal node it can either be an identity (condition 2) or a repeated occurrence (condition 1). In the former case we know that the transition $X_1\alpha_1\gamma \xrightarrow{a} \alpha''_a\alpha_1\gamma$ can be matched by $Y_1\beta \xrightarrow{a} \beta''_a\beta_1$ where $\alpha''_a\alpha_1\gamma = \beta''_a\beta_1$, so clearly $\alpha''_a\alpha_1\gamma \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''_a\beta_1$.

In the latter case the situation is the same as when STEP is applied to the SUB consequent. Here the transition $X_1\alpha_1\gamma \xrightarrow{a} \alpha''_a\alpha_1\gamma$ will be matched either by $Y_1\beta_1$ (if $a = \tau$) with $\alpha''_a\alpha_1\gamma R_{\mathbf{T}} Y_1\beta_1$ or by some

$$Y_1\beta_1 \xrightarrow{\tau} \beta''_{11}\beta_1 \xrightarrow{\tau} \cdots \xrightarrow{a} \beta''_{1n(1)}\beta_1 \xrightarrow{a} \beta''_a\beta_1$$

with $(\alpha''_a\alpha_1\gamma, \beta''_{1j}\beta_1) \in R_{\mathbf{T}}$ for $1 \leq j \leq n(1)$ and $(\alpha''_a\alpha_1\gamma, \beta''_a\beta_1) \in R_{\mathbf{T}}$. Clearly $X_1\alpha_1\alpha \xrightarrow{a}$ iff $X_1\alpha_1\gamma \xrightarrow{a}$ and thus (5.4) is of the form $X_1\alpha_1\alpha \xrightarrow{a} \alpha''_a\alpha_1\alpha$ which can be matched either by $Y_1\beta_1\beta$ if $a = \tau$ or by

$$Y_1\beta_1\beta \xrightarrow{\tau} \beta''_{11}\beta_1\beta \cdots \xrightarrow{a} \beta_{1n}\beta_1\beta \xrightarrow{a} \beta''_a\beta_1\beta$$

For then $X_1\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''_{1j}\beta_1\beta$ for $1 \leq j \leq n(1)$ and $\alpha''_a\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''_a\beta_1\beta$. This holds, since $\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \gamma\beta$ and $X_1\alpha_1\gamma R_{\mathbf{T}} \beta''_{1j}\beta_1$ implies $X_1\alpha_1\gamma\beta \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''_{1j}\beta_1\beta$ giving $X_1\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''_{1j}\beta_1\beta$ and $\alpha''_a\alpha_1\gamma\beta R_{\mathbf{T}} \beta''_a\beta_1\beta$, implying $\alpha''_a\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta''_a\beta_1\beta$. Similarly, $\alpha'_a\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} Y_1\beta_1\beta$.

Step: Suppose $\alpha' \xrightarrow{\tau} \alpha'_1 \xrightarrow{\tau} \cdots \alpha'_m \xrightarrow{a} \alpha''$. If $\alpha' = \beta'$ is a terminal because of condition 2, it is obvious that we can match within $\underset{R_{\mathbf{T}}}{\longleftrightarrow^*}$. Otherwise, by the base case we know that $\alpha' \xrightarrow{\tau} \alpha'_1$ can be matched either by $\alpha'_1 \underset{R_{\mathbf{T}}}{\longleftrightarrow^*} \beta'$ or by

$$\beta' \xrightarrow{\tau} \beta'_{11} \xrightarrow{\tau} \cdots \xrightarrow{\tau} \beta''_{1n(1)} \xrightarrow{a} \beta'_1$$

where $\alpha' \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta''_{1j}$ for $1 \leq j \leq n(1)$ and $\alpha'_1 \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta'_1$. Now if $(\alpha'_1, \beta'_1) \in R_{\mathbf{T}}$ (or $(\alpha'_1, \beta') \in R_{\mathbf{T}}$) we know by induction hypothesis that we can match the transition sequence $\alpha'_1 \xrightarrow{\tau} \cdots \alpha'_m \xrightarrow{a} \alpha''$ within $\underset{R_{\mathbf{T}}}{\longleftrightarrow}^*$ so our match for (5.4) simply consists in combining these two matches. From the base case we see that the only case where it might *not* be the case that $(\alpha'_1, \beta'_1) \in R_{\mathbf{T}}$ or $(\alpha'_1, \beta') \in R_{\mathbf{T}}$ is when $\alpha' = \beta'$ is the premise of a SUB rule. So assume wlog that the rule applied was SUBL and that $\alpha' = \beta'$ is $X_1\alpha_1\alpha = Y_1\beta_1\beta$ with $\alpha = \gamma\beta$ as the residual. Then we have

$$\frac{X_1\alpha_1\alpha = Y_1\beta_1\beta}{X_1\alpha_1\gamma = Y_1\beta_1} \quad \mathsf{SUBL}$$

So here $X_1\alpha_1\alpha \xrightarrow{\tau} X_2\alpha_2\alpha_1\alpha$, so consequently $\alpha'_1 = \beta'_1$ is of the form $X_2\alpha_2\alpha_1\alpha = Y_2\beta_2\beta_1\beta$ with either $(X_2\alpha_2\alpha_1\gamma, Y_2\beta_2\beta_1) \in R_{\mathbf{T}}$ or $X_2\alpha_2\alpha_1\gamma = Y_2\beta_2\beta_1$, if we are dealing with a terminal of type 2. A closer look at the transition sequence $\alpha'_1 \xrightarrow{\tau} \cdots \alpha'_m \xrightarrow{a} \alpha''$ reveals that it is of the form

$$X_2\alpha_2\alpha_1\alpha \xrightarrow{\tau} \alpha_3\alpha_1\alpha \cdots \xrightarrow{\tau} \alpha_m\alpha_1\alpha \xrightarrow{a} \alpha''_1\alpha_1\alpha \tag{5.5}$$

which is due to

$$X_2\alpha_2\alpha_1\gamma \xrightarrow{\tau} \alpha_3\alpha_1\gamma \cdots \xrightarrow{\tau} \alpha_m\alpha_1\gamma \xrightarrow{a} \alpha''_1\alpha_1\gamma$$

Both in the cases when $X_2\alpha_2\alpha_1\gamma = Y_2\beta_2\beta_1$ is a terminal of type 2 and when we have $(X_2\alpha_2\alpha_1\gamma, Y_2\beta_2\beta_1) \in R_{\mathbf{T}}$, we know (in the latter case by induction hypothesis) that there must exist $\beta_i\beta_1$ for $3 \leq i \leq m$ and $\beta''_1\beta_1$ such that either

$$\beta_2\beta_1 \xrightarrow{\tau} \beta_3\beta_1 \cdots \beta_m\beta_1 \cdots \xrightarrow{a} \beta''_1\beta_1$$

or, if $a = \tau$

$$\beta_2\beta_1 \xrightarrow{\tau} \beta_3\beta_1 \cdots \beta_m\beta_1 = \beta_1''\beta_1$$

with $\alpha_i\alpha_1\gamma \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_i\beta_1$ and either $\beta_i\beta_1 = \beta_{i+1}\beta_1$ or there are $\beta_{ij}\beta_1$ for $1 \leq j \leq n(i)$ such that $\beta_i\beta_1 \xrightarrow{\tau} \beta_{i1}\beta_1 \xrightarrow{\tau} \cdots \beta_{i+1}\beta_1$ with $\alpha_i\alpha_1\gamma \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_{ij}\beta_1$ and $\alpha_1''\alpha_1\gamma \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_1''\beta_1$. If $a \neq \tau$, we can also have $\beta_{mj}\beta_1$ for $1 \leq j \leq n(m)$ with $\beta_m\beta_1 \xrightarrow{\tau} \beta_{m1}\beta_1 \xrightarrow{\tau} \cdots \xrightarrow{a} \beta_1''\beta_1$ and $\alpha_m\alpha_1\gamma \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_{mj}\beta_1$ and $\alpha_1''\alpha_1\gamma \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_m\beta_1$. At any rate, our match for the sequence $\alpha' \xrightarrow{\tau} \alpha_1' \xrightarrow{\tau} \cdots \alpha_m' \xrightarrow{a} \alpha''$ consists of the concatenation of the match for $\alpha' \xrightarrow{\tau} \alpha_1'$ and that for (5.5), which is either

$$\beta_2\beta_1\beta \xrightarrow{\tau} \beta_3\beta_1\beta \cdots \beta_m\beta_1\beta \cdots \xrightarrow{a} \beta_1''\beta_1\beta$$

or, if $a = \tau$

$$\beta_2\beta_1\beta \xrightarrow{\tau} \beta_3\beta_1\beta \cdots \beta_m\beta_1\beta = \beta_1''\beta_1\beta$$

with $\beta_i\beta_1\beta$ for $3 \leq i \leq m$ and if $\beta_i\beta_1\beta \neq \beta_{i+1}\beta_1\beta$ the additional $\beta_{ij}\beta_1\beta$ for $1 \leq j \leq n(i)$. Clearly $\alpha_i\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_i\beta_1\beta$, $\alpha_i\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_{ij}\beta_1\beta$, $\alpha_1''\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_1''\beta_1\beta$ and $\alpha_m\alpha_1\alpha \underset{R_{\mathbf{T}}}{\longleftrightarrow}^* \beta_{mj}\beta_1\beta$, the latter if $\beta_m \neq \beta_1''$. □

So we now get the soundness of the tableau system as

Corollary 5.3.2 *If $\alpha = \beta$ has a successful tableau then $\alpha \approx_b \beta$.*

5.3.3 Complexity of the tableau system and decidability

The complexity of the tableau system can be measured in terms of the maximal depth of a tableau, i.e. the length w.r.t. STEP applications of the longest possible path in a successful tableau for an equation $X\alpha = Y\beta$. Let v be the cardinality of Var. By the discussion in the proof of Theorem 5.3.1 we have that any SUB consequent has a length of at most $3m_\Delta(M_\Delta + 1)$, so an upper bound on the number of distinct SUB consequents along any tableau path is $\sum_{j=2}^{3m_\Delta(M_\Delta+1)}(j-1)v^j$.

Between any two SUB consequents there are at most $\lceil\frac{3m_\Delta(M_\Delta+1)}{2}\rceil$ residuals, since the worst that can happen is that the total norm decreases by 2 in every subtableau along the way. Thus, any path that contains SUB consequents can have at most $\lceil\frac{3m_\Delta(M_\Delta+1)}{2}\rceil\sum_{j=2}^{3m_\Delta M_\Delta+3}(j-1)v^j$ subtableau roots. As for the leftmost path, all of whose subtableau roots are residuals, there can be at most $\max(||\alpha||,||\beta||)$ residuals, since the norm of the residuals is strictly decreasing. So, since a subtableau can have a depth w.r.t. STEP applications of at most m_Δ, any path can have a length of at most

$$m_\Delta\ \max(||\alpha||,||\beta||,\lceil\frac{3m_\Delta(M_\Delta+1)}{2}\rceil\sum_{j=2}^{3m_\Delta(M_\Delta+1)}(j-1)v^j) \tag{5.6}$$

STEPs.

We also have an upper bound on the outbranching of any tableau for $X\alpha=Y\beta$. This follows from the fact that there is a uniform upper bound on the total length of any subtableau root in any tableau for $X\alpha=Y\beta$. Any subtableau root along the leftmost path is a residual and has its total length bounded by $2\max(||\alpha||,||\beta||)$. Since, as we saw, any SUB consequent has its length bounded by $3m_\Delta(M_\Delta+1)$, and thus also has a norm of at most $3m_\Delta(M_\Delta+1)$, any of its following residuals must also have a length of at most $3m_\Delta(M_\Delta+1)$. So the length of any subtableau root is bounded by

$$L=\max(2\max(||\alpha||,||\beta||),3m_\Delta(M_\Delta+1))$$

By repeated applications of Proposition 5.3.2 we see that any node in a subtableau has a length of at most $2m_\Delta(M_\Delta+1)+L$. By Proposition 5.3.3 this means that there is a uniform upper bound on the number of STEP consequents at any point in any tableau for $X\alpha=Y\beta$ of

$$\max\{B_{X,Y}\mid X,Y\in Var\}\sum_{j=2}^{2m_\Delta(M_\Delta+1)+L}(j-1)v^j \tag{5.7}$$

Theorem 5.3.4 *For any $X\alpha = Y\beta$ there are finitely many possible tableaux.*

PROOF: From (5.6) and (5.7). $\square$

Theorem 5.3.5 *For any weakly normed Δ it is decidable whether or not $\alpha \approx_b \beta$ for $\alpha, \beta \in Var^*$.*

PROOF: A naive decision procedure for $\approx_b$ constructs all the finitely many tableaux for $\alpha = \beta$, answering 'yes' if a successful tableau occurs and 'no' otherwise. $\square$

Just as the system in Chapter 4, this tableau system has exponential complexity in terms of the longest possible path of a generated tableau. But again, in the case of a successful tableau we get additional information in the form of a finite relation whose congruence closure w.r.t. sequential composition is a bisimulation containing the initial equation.

Rule within a subtableau

STEP $$\frac{\alpha = \beta}{\alpha_1 = \beta_1 \ldots \alpha_k = \beta_k}$$ where $\{\alpha_1 = \beta_1 \ldots \alpha_k = \beta_k\}$

is a possible match for $\alpha = \beta$

Rules for new subtableaux

SUBL $$\frac{\alpha_i \alpha = \beta_i \beta}{\alpha_i \gamma = \beta_i}$$ where $\alpha = \gamma\beta$ is the residual

SUBR $$\frac{\alpha_i \alpha = \beta_i \beta}{\alpha_i = \beta_i \gamma}$$ where $\gamma\alpha = \beta$ is the residual

Table 5–1: The tableau rules

Chapter 6

Negative results

In [HT90] Huynh and Tian have shown that the readiness and failures equivalences are undecidable for BPA processes. In this chapter we give an alternative account of their proof and examine all other equivalences in Figure 1–1, showing that *none* of them are decidable for normed BPA processes and that a number of preorders are also undecidable. This we do by reducing language containment for deterministic processes to the various preorders and reducing language equivalence and in one case trace equivalence for normed BPA processes to the equivalences.

6.1 Deterministic BPA processes

In what follows the undecidability of language inclusion for simple grammars, a result established by Friedman in [Fri76], is crucial.

Definition 6.1.1 [KH66] *A* simple grammar *is a context-free grammar in GNF such that there are no two distinct productions $A \to a\alpha_1, A \to a\alpha_2$ for any nonterminal A.*

Thus, we see that simple grammars correspond to *deterministic* systems of BPA equations in GNF. For simple context-free grammars the language equivalence problem was proved to be decidable by Korenjak and Hopcroft [KH66]. However, as was later shown by Friedman, the language containment problem is *undecidable*. The proof of this consists in a reduction from the halting problem via the Post correspondence problem: Given a Turing machine M and an input w we can effectively construct a Post system and from it two simple grammars such that language inclusion holds iff the Post system has a match iff M halts on w. Reformulated in the BPA framework, Friedman's theorem reads:

Theorem 6.1.1 *Let Δ be a normed and deterministic system of BPA equations. It is undecidable whether $L(\alpha) \subseteq L(\beta)$ for $\alpha, \beta \in Var^*$.*

However, it is important to note that for deterministic BPA processes trace equivalence and bisimulation equivalence coincide:

Proposition 6.1.1 *If Δ is a deterministic normed system of BPA equations, $Tr(\alpha) = Tr(\beta)$ iff $\alpha \sim \beta$ for any $\alpha, \beta \in Var^*$.*

PROOF: $\{(\alpha, \beta) \mid Tr(\alpha) = Tr(\beta)\}$ is a bisimulation. Since Δ is deterministic, there is exactly one α' such that $\alpha \xrightarrow{a} \alpha'$. Since $Tr(\alpha) = Tr(\beta)$ we know that there also is exactly one β' such that $\beta \xrightarrow{a} \beta'$ and that $Tr(\alpha') = Tr(\beta')$. The other half of the proof is similar. □

Consequently, in the deterministic case the linear/branching time hierarchy collapses (see also [Eng85] and [vG90a]), and since language equivalence is decidable, *all* equivalences are *decidable* in this case. However, we already know from Theorems 2.2.2 and 2.2.3 that the language and trace equivalences are undecidable for the full BPA, so there are bound to be differences in the general case.

6.2 Simulation equivalence

We start off with simulation and simulation equivalence.

Definition 6.2.1 *A binary relation R between processes is a* simulation *iff whenever pRq then for each $a \in Act$ $p \xrightarrow{a} p' \;\Rightarrow\; \exists q : q \xrightarrow{a} q' \wedge p'Rq'$. A process p is* simulated *by a process q, notation $p \subsetneq q$, iff there is a simulation relation R with pRq. Two processes p and q are* simulation equivalent*, written $p \leftrightarrow q$, iff $p \subsetneq q$ and $q \subsetneq p$.*

We first show that that simulation is undecidable for deterministic normed BPA processes. This is a direct corollary of Theorem 6.1.1; this was pointed out to me by Didier Caucal.

Theorem 6.2.1 *Simulation is undecidable for deterministic normed BPA processes.*

PROOF: Let Δ define a normed deterministic BPA process and let $\alpha, \beta \in Var^*$. Now let $\surd$ be a new action not in Act. We then have

$$L(\alpha) \subseteq L(\beta) \quad \text{iff} \quad \alpha\surd \subsetneq \beta\surd$$

as α and β are deterministic. With this observation, we reduce language containment for deterministic normed BPA processes to the simulation preorder for deterministic normed BPA processes. □

Theorem 6.2.2 *Simulation equivalence is undecidable for normed BPA processes.*

PROOF: We can reduce simulation to simulation equivalence by the following observation, which was pointed out to me by Jan Friso Groote:

$$\alpha \subsetneq \beta \quad \text{iff} \quad \alpha + \beta \leftrightarrow \beta.$$

□

6.3 n-nested simulation equivalence

The notion of n-nested simulation equivalence was introduced by Groote and Vaandrager [GV89] in their study of the *tyft/tyxt*-format for structured operational semantics because 2-nested simulation equivalence is the completed trace congruence for this format.

Definition 6.3.1 *For all* $n \in \mathbf{N}$*,* n-nested simulation*, written* $\underrightarrow{\subset}^n$*, is inductively defined by*

- $p \underrightarrow{\subset}^0 q$ *for all processes* p *and* q*,*
- $p \underrightarrow{\subset}^{n+1} q$ *iff there is a simulation relation* $R \subseteq (\underrightarrow{\subset}^n)^{-1}$ *with* pRq*.*

Two processes p *and* q *are* n-nested simulation equivalent*, written* $p \leftrightarrow^n q$*, iff* $p \underrightarrow{\subset}^n q$ *and* $q \underrightarrow{\subset}^n p$*.*

Note that 1-nested simulation is just simulation and that therefore 1-nested simulation equivalence is simulation equivalence.

Lemma 6.3.1 *For all* $n \in \mathbf{N}$*,* n*-nested simulation is a precongruence under action prefixing and* $+$*.*

PROOF: Induction in n.

$n = 1$: This simply states that $\underrightarrow{\subset}$ is a precongruence under action prefixing and $+$. Clearly if $p \underrightarrow{\subset} q$ and $p \underrightarrow{\subset} q'$ we have that $ap \underrightarrow{\subset} aq$ and $p + p' \underrightarrow{\subset} q + q'$.

Step - assuming for n*:* Here if $p \underrightarrow{\subset}^{n+1} q$ we have that there is a simulation $R \subseteq (\underrightarrow{\subset}^n)^{-1}$ such that pRq. But then, since $q \underrightarrow{\subset}^n p$ we must by induction hypothesis have $aq \underrightarrow{\subset}^n ap$ and thus $R \cup \{(ap, aq)\}$ is a simulation with $R \cup \{(ap, aq)\} \subseteq (\underrightarrow{\subset}^n)^{-1}$. The proof for $+$ is entirely similar.

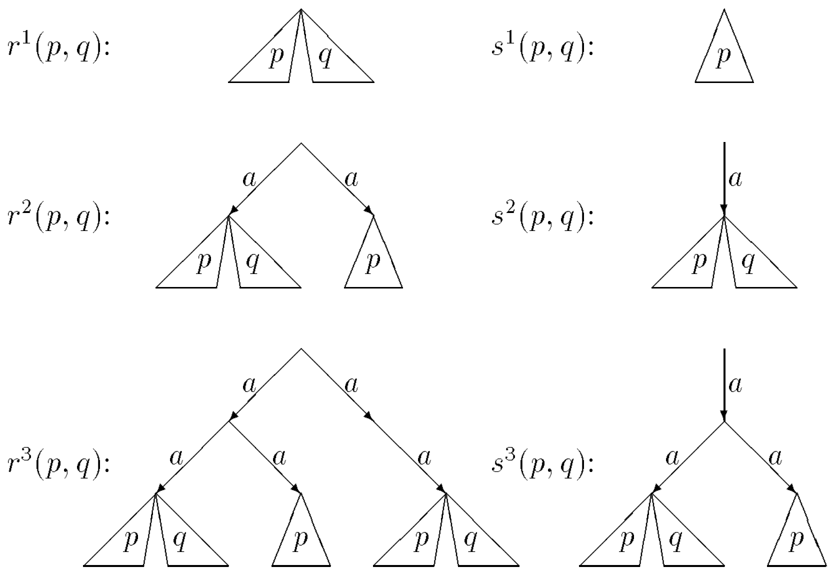

Figure 6–1: $r^n(p,q)$ and $s^n(p,q)$ for $n = 1, 2, 3$

□

The undecidability proof that follows is due to Jan Friso Groote. The class of processes defined in the following can be used to reduce simulation to both n-nested simulation and n-nested simulation equivalence.

Some of these processes are depicted in Figure 6–1.

Definition 6.3.2 *Let p and q be processes and let a be an action. The processes $r^n(p,q)$ and $s^n(p,q)$ for $n > 0$ are inductively defined by:*

$$r^1(p,q) = p + q, \qquad s^1(p,q) = p,$$

$$r^{n+1}(p,q) = a\, r^n(p,q) + a\, s^n(p,q), \qquad s^{n+1}(p,q) = a\, r^n(p,q).$$

Observe that if p and q are normed BPA processes, then so are $r^n(p,q)$ and $s^n(p,q)$.

Lemma 6.3.2 *Let p and q be processes. For all $n > 0$ it holds that*

1. $s^n(p,q) \underset{\rightarrow}{\subset}^n r^n(p,q)$,

2. $r^n(p,q) \subsetneq^n s^n(p,q)$ *iff* $q \subsetneq p$.

PROOF: Both proofs proceed by induction. For 1 we have

$n = 1:$ The lemma then reduces to $p \subsetneq p + q$ which obviously holds.

Step - assuming for n: Define for processes p and q the simulation

$$R = \{(ar^n(p,q), ar^n(p,q) + as^n(p,q))\} \cup Id$$

where Id is the identity relation. $R \subseteq (\subsetneq^n)^{-1}$, as we have $r^n(p,q) \subsetneq^n r^n(p,q)$ and $s^n(p,q) \subsetneq^n r^n(p,q)$ by induction hypothesis, which by Lemma 6.3.1 gives us $ar^n(p,q) + as^n(p,q) \subsetneq^n ar^n(p,q)$.

For 2 the proof is

$n = 1:$ The lemma here reduces to $p + q \subsetneq p$ iff $q \subsetneq p$. If there is a simulation R with qRp then $\{(p+q,p)\} \cup R \cup Id$ is a simulation. And if $p + qRp$ for a simulation R, then $\{(q,p)\} \cup R$ is a simulation.

Step - assuming for n: For the 'if' direction suppose for processes p and q that $q \subsetneq p$ and $r^n(p,q) \subsetneq^n s^n(p,q)$. Then define

$$R = \{(ar^n(p,q) + as^n(p,q), ar^n(p,q))\} \cup (\subsetneq^n)^{-1}$$

$R \subseteq (\subsetneq^n)^{-1}$ since Lemma 6.3.1 gives us $ar^n(p,q) \subsetneq^n ar^n(p,q) + as^n(p,q)$, so $r^{n+1}(p,q) \subsetneq^{n+1} s^{n+1}(p,q)$. For the 'only if' direction suppose $q \not\subsetneq p$. By induction hypothesis $r^n(p,q) \not\subsetneq^n s^n(p,q)$. So also have $r^{n+1}(p,q) \not\subsetneq^{n+1} s^{n+1}(p,q)$, for any candidate simulation would be one containing the pair $(a\,r^n(p,q) + a\,s^n(p,q), a\,r^n(p,q))$. As $r^{n+1}(p,q) \xrightarrow{a} s^n(p,q)$ can only be matched by the transition $s^{n+1}(p,q) \xrightarrow{a} r^n(p,q)$ it must be that $(s^n(p,q), r^n(p,q)) \in R$. But since $R \subseteq (\subsetneq^n)^{-1}$ we would have $r^n(p,q) \subsetneq^n s^n(p,q)$,

□

Theorem 6.3.1 *For $n > 0$ n-nested simulation and n-nested simulation equivalence are undecidable for normed BPA processes.*

PROOF: We reduce simulation to n-nested simulation using the following observation:

$$q \subsetneq\!\!\!\rightarrow p \quad \text{iff} \quad r^n(p,q) \subsetneq\!\!\!\rightarrow^n s^n(p,q).$$

We reduce simulation to n-nested simulation equivalence using:

$$q \subsetneq\!\!\!\rightarrow p \quad \text{iff} \quad r^n(p,q) \leftrightarrow^n s^n(p,q).$$

Because $n > 0$ both facts follow directly from Lemma 6.3.2. As simulation is undecidable, n-nested simulation and n-nested simulation equivalence cannot be decidable. □

One should notice here that the limit of the n-nested simulation equivalences as $n \to \omega$ is strong bisimulation equivalence:

Theorem 6.3.2 [GV89] *For any finitely branching labelled transition graph we have*

$$\sim \; = \bigcap_{n=0}^{\omega} \leftrightarrow^n$$

So we here have the odd situation, because of Theorem 6.3.2 and the results of Chapter 4, that while $\sim$ is decidable, it is the limit of a series of undecidable approximations.

6.4 n-bounded-tr-bisimulation

We now consider n–bounded-tr-bisimulation. This equivalence is a generalization of trace equivalence and the *possible futures equivalence* of [RB81], in that 1-bounded-tr-bisimulation corresponds to the former and 2-bounded-tr-bisimulation to the latter.

Definition 6.4.1 *We define* n-bounded-tr-bisimulation*, written* $\sim_{tr}^n$*, inductively as follows.*

- $p \sim_{tr}^0 q$ *for all processes* p *and* q*,*
- $p \sim_{tr}^{n+1} q$ *iff*
  - *if* $p \stackrel{w}{\to} p'$ *then* $\exists q'$ *with* $q \stackrel{w}{\to} q'$ *and* $p' \sim_{tr}^n q'$ *and*
  - *if* $q \stackrel{w}{\to} q'$ *then* $\exists p'$ *with* $p \stackrel{w}{\to} p'$ *and* $p' \sim_{tr}^n q'$*.*

This notion of equivalence also arises naturally as the consecutive approximations of bisimulation equivalence [Mil80,Mil89]. For finitely branching transition graphs, and therefore for BPA processes, the limit of the n-bounded-tr-bisimulations for $n \to \omega$ coincides with bisimulation equivalence:

Theorem 6.4.1 [Mil89] *For any finitely branching labelled transition graph we have*

$$\sim \; = \bigcap_{n=0}^{\omega} \sim_{tr}^n$$

The following proof uses the same reduction that was employed in [KS90] to show that n-tr-bisimulation for regular processes is PSPACE-complete. The following easy lemma is crucial:

Lemma 6.4.1 [KS90] *Let* p *and* q *be processes. For all* $n > 0$ *it holds that*

$$p \sim_{tr}^n q \textit{ iff } p + q \sim_{tr}^n p \textit{ and } p + q \sim_{tr}^n q$$

Lemma 6.4.2 [KS90] *Let* p *and* q *be processes. For all* $n > 0$ *it holds that*

$$p \sim_{tr}^n q \textit{ iff } a(p+q) \sim_{tr}^{n+1} ap + aq$$

PROOF: For the 'if' half we prove the contrapositive, stating that $p \not\sim_{tr}^n q$ implies that $a(p+q) \not\sim_{tr}^{n+1} ap + aq$. Assume $p \not\sim_{tr}^n q$. But then $a(p+q) \stackrel{a}{\to} p + q$ whereas

$ap + aq \xrightarrow{a} p$ and $ap + aq \xrightarrow{a} q$. Since $p \not\sim_{tr}^{n} q$ we have by the previous lemma that either $p + q \not\sim_{tr}^{n} p$ or $p + q \not\sim_{tr}^{n} q$, so clearly $a(p + q) \not\sim_{tr}^{n+1} ap + aq$. The 'only if' half of the proof also proceeds by contraposition, showing that $a(p + q) \not\sim_{tr}^{n+1} ap + aq$ implies $p \not\sim_{tr}^{n} q$. Assuming $a(p+q) \not\sim_{tr}^{n+1} ap+aq$, the action string that distinguishes $a(p + q)$ and $ap + aq$ must be a, since for any longer string w the w-derivatives are identical. Thus, either $p + q \not\sim_{tr}^{n} p$ or $p + q \not\sim_{tr}^{n} q$, and again by the previous lemma $p \not\sim_{tr}^{n} q$. □

Theorem 6.4.2 *For $n > 0$ n-bounded-tr-bisimulation is undecidable for normed BPA processes.*

PROOF: We reduce n-bounded-tr-bisimulation to $n + 1$-bounded-tr-bisimulation using Lemma 6.4.2. Since 1-bounded-tr-bisimulation is trace equivalence, which is undecidable (Theorem 2.2.3), the result follows. □

So the consequence of the above result, seen in conjunction with Theorem 6.4.1 and the results in Chapter 4, is again the rather odd one that $\sim$ is decidable while none of these non-trivial approximations are!

6.5 Failures, readiness, failure-trace and ready-trace equivalences

In their paper [HT90] Huynh and Tian show that failure equivalence [BHR84] and readiness equivalence [BKO88,OH86] are undecidable for normed BPA processes. Here we give an alternative account of their technique, using a simpler transformation to show that ready trace and failure trace equivalence are undecidable.

Definition 6.5.1 *For any process p, define*

$$failures(p) = \{(w, X) \mid \exists p' : p \xrightarrow{w} p', \forall a \in X : p' \not\xrightarrow{a} \},$$

$$readies(p) = \{(w, X) \mid \exists p' : p \xrightarrow{w} p', p' \xrightarrow{a} \iff a \in X\}.$$

Processes p and q are failures equivalent*, written* $p \sim_f q$*, iff* $failures(p) = failures(q)$*. Processes p and q are* readiness equivalent*, written* $p \sim_r q$*, iff* $readies(p) = readies(q)$*.*

The proof technique of Huynh and Tian involves defining a class of processes, called *locally unary* processes, for which failures and readiness equivalence coincide with completed trace equivalence.

Definition 6.5.2 [HT90] *A process p is* locally unary *iff for each* p' *with* $p \xrightarrow{w} p'$ *there is at most one* $a \in Act$ *such that* $p' \xrightarrow{a}$ *.*

Lemma 6.5.1 [HT90] *If p and q are locally unary normed processes then*

$$p \sim_r q \quad \text{iff} \quad p \sim_f q \quad \text{iff} \quad L(p) = L(q).$$

PROOF:[HT90] It is sufficient to show that $L(p) = L(q)$ implies $p \sim_r q$. Suppose $L(p) = L(q)$ and $(w, X) \in readies(p)$. If $X = \emptyset$ we have $w \in L(p)$ and hence $(w, \emptyset) \in readies(q)$. Otherwise, since q is locally unary we have $X = \{a\}$ for some $a \in Act$. Since p is normed, there must be a $w' \in Act^*$ such that $waw' \in L(q)$. Since $L(p) = L(q)$ and q is locally unary and normed, we must therefore also have that $(w, \{a\}) \in readies(q)$. □

The idea is now, given a Δ to construct a locally unary Δ' containing the variables of Δ such that $L(\alpha) = L(\beta)$ in Δ if and only if $L(\alpha) = L(\beta)$ in Δ'. The following construction accomplishes this. The idea is simply to precede any action by a # that indicates that a nondeterministic choice has been made.

Definition 6.5.3 *Given a system of equations* Δ *in GNF let* Δ' *have the action set* $Act' = Act \cup \{\#\}$ *(where # is a new action) and process variables* $Var' = Var \cup \{X_a \mid a \in Act\}$*. For every process equation in* Δ

$$X_i \stackrel{\text{def}}{=} \sum a_j \alpha_j$$

create the equations

$$X_i \stackrel{\text{def}}{=} \sum \# X_{a_j} \alpha_j$$
$$X_{a_j} \stackrel{\text{def}}{=} a_j$$

in the new system Δ'.

It is obvious that Δ' is normed iff Δ is (in fact if $|X| = k$ in Δ then $|X| = 2k$ in Δ). We now have

Proposition 6.5.1 *Δ' is locally unary.*

PROOF: If a state in the transition graph for Δ' is of the form $X\gamma$ with $X \in Var$ we must have $X\gamma \stackrel{\#}{\rightarrow}$ and nothing else. Otherwise, if it is of the form $X_a\gamma$ we can only have $X_a\gamma \stackrel{a}{\rightarrow}$ □

The following is now obvious from the definition of Δ'.

Proposition 6.5.2 *For $\alpha \in Var^*$ we have $\alpha \stackrel{a}{\rightarrow} \alpha'\alpha''$ in Δ iff $\alpha \stackrel{\#}{\rightarrow} X_a\alpha'\alpha'' \stackrel{a}{\rightarrow} \alpha'\alpha''$ in Δ'.*

We therefore also see that

Proposition 6.5.3 *For $\alpha \in Var^*$ $b_1b_2...b_n \in L(\alpha)$ relative to Δ iff $\#b_1\#b_2...\#b_n \in L(\alpha)$ relative to Δ'.*

Theorem 6.5.1 [HT90] *Failures and readiness equivalence are undecidable for normed BPA processes.*

PROOF: From Proposition 6.5.3 we can reduce language equivalence for normed BPA processes to language equivalence for locally unary processes and the theorem now follows from Lemma 6.5.1. □

The above ideas can also be used to prove that failure trace and ready trace equivalence are undecidable. For normed BPA, failure trace equivalence [vG90b] coincides with the notion of refusal testing [Phi87].

Definition 6.5.4 *The* refusal relation $\stackrel{A}{\rightarrow}$ *for* $A \subseteq Act$ *is defined for any processes* p, q *by* $p \stackrel{A}{\rightarrow} q$ *iff* $p = q$ *and whenever* $a \in A$, $p \stackrel{a}{\not\rightarrow}$ *. The* failure trace *relations* $\stackrel{u}{\rightarrow}$ *for* $u \in (Act \cup \mathcal{P}(Act))^*$ *are defined as the reflexive and transitive closure of the refusal and transition relations. Define*

$$\textit{failure-traces}(p) = \{u \in (Act \cup \mathcal{P}(Act))^* \mid \exists p' : p \stackrel{u}{\rightarrow} p'\}.$$

Processes p *and* q *are* failure-trace equivalent*, written* $p \sim_{ftr} q$ *iff* $\textit{failure-traces}(p) = \textit{failure-traces}(q)$*.*

The definition of ready trace equivalence that we use here is a characterisation presented in [vG90b].

Definition 6.5.5 *Define*

$$\textit{ready-trace}(p) = \{A_0 a_1 A_1 \ldots a_n A_n \mid$$

$$\exists p_0, \ldots, p_n : p = p_0 \stackrel{a_1}{\rightarrow} p_1 \cdots \stackrel{a_n}{\rightarrow} p_n,\ p_i \stackrel{a}{\rightarrow} \iff a \in A_i, 0 \leq i \leq n\}.$$

Processes p *and* q *are* ready trace equivalent*, written* $p \sim_{rtr} q$*, iff* $\textit{ready-trace}(p) = \textit{ready-trace}(q)$*.*

Lemma 6.5.2 *If* p *and* q *are locally unary normed processes then* $p \sim_{ftr} q$ *iff* $L(p) = L(q)$ *iff* $p \sim_{rtr} q$*.*

PROOF: It is enough to show that $L(p) = L(q)$ implies $p \sim_{ftr} q$ and $p \sim_{rtr} q$. Define $h : (Act \cup \mathcal{P}(Act))^* \rightarrow Act^*$ as the homomorphic extension of

$$h(u) = \begin{cases} \epsilon & \text{if } u \in \mathcal{P}(Act) \\ u & \text{otherwise.} \end{cases}$$

Then, if $u \in \textit{failure-traces}(p)$ clearly for some $v \in Act^*$, $h(u)v \in L(p)$. Thus, since $L(p) = L(q)$ we must have $h(u)v \in L(q)$ and since q is locally unary and normed, it is now easy to see that $u \in \textit{failure-traces}(q)$. For the other part of the proof, note that if $A_0 a_1 A_1 a_2 \ldots a_n A_n$ is a ready trace for a locally unary process, all A_i $(0 \leq i < n)$ are singleton sets and A_n is the empty set or a singleton set. Then if $u \in \textit{ready-trace}(p)$ we have a $v \in Act^*$ such that $h(u)v \in L(p)$. Since $L(p) = L(q)$ we must have $h(u)v \in L(q)$, and since q is locally unary and normed we get $u \in \textit{ready-trace}(q)$. □

Corollary 6.5.1 *Failure trace equivalence and ready trace equivalence are undecidable for normed BPA processes.*

6.6 Ready-simulation or 2/3-bisimulation

The notion of ready simulation (or 2/3-bisimulation) originated in work by Bloom, Istrail and Meyer [BIM90] and Larsen and Skou [LS90]. It is the completed trace congruence induced by the GSOS-format [BIM90].

Definition 6.6.1 *A relation R between processes is a* ready simulation *iff it is a simulation and whenever pRq then for each $a \in Act$ we have $p \xrightarrow{a}$ if $q \xrightarrow{a}$. We say that q ready simulates p, written $p \subsetneq\!\!\!\rightarrow_r q$, iff there is a ready simulation R with pRq. Two processes p and q are* ready simulation equivalent, *written $p \leftrightarrow_r q$, iff $p \subsetneq\!\!\!\rightarrow_r q$ and $q \subsetneq\!\!\!\rightarrow_r p$.*

The idea behind the proof in this section is to find a class of processes where the ready simulation and simulation preorders coincide. The following class of processes, introduced by Jan Friso Groote, is essential here:

Definition 6.6.2 *A system of process equations Δ in GNF is said to be* two-step deterministic *iff whenever $\alpha \xrightarrow{a} \alpha_1 \xrightarrow{b}$ and $\alpha \xrightarrow{a} \alpha_2 \xrightarrow{b}$ then $\alpha_1 = \alpha_2$.*

Note that the notion of being two-step deterministic is strictly weaker than that of being deterministic.

We now show that for locally unary, two step deterministic processes language inclusion coincides with ready simulation. We use the construction of Definition 6.5.3 to show that language inclusion for deterministic processes can be reduced to language inclusion for unary locally, two-step deterministic processes. This enables us to show that ready simulation is undecidable for locally unary, two step deterministic processes.

The following two results follow immediately from Propositions 6.5.2 and 6.5.3. Let Δ and Δ' be given as in Definition 6.5.3.

Proposition 6.6.1 *If Δ is deterministic, then Δ' is two-step deterministic.*

Lemma 6.6.1 *For $\alpha, \beta \in Var^*$ $L(\alpha) \subseteq L(\beta)$ in Δ iff $L(\alpha) \subseteq L(\beta)$ in Δ'.*

PROOF: Directly from Proposition 6.5.3. □

The next lemma is due to Jan Friso Groote.

Lemma 6.6.2 *Let Δ' define a normed BPA process in GNF. If Δ' is locally unary and two-step deterministic, then for any $\alpha, \beta \in Var^*$ we have $L(\alpha) \subseteq L(\beta)$ iff $\alpha\surd \subsetneq_r \beta\surd$ (where $\surd$ is a new action not occurring in Act').*

PROOF: The 'if' half is obvious, so it suffices to prove the 'only if' half. We define the relation

$$R = \{(\alpha\surd, \beta\surd) \mid L(\alpha) \subseteq L(\beta)\}$$

and show that it is a ready simulation. This is easy for the pair $(\surd, \surd)$. So we only consider pairs $(\alpha\surd, \beta\surd)$ where $\alpha, \beta \neq \epsilon$.

First we show that if $\beta\surd \xrightarrow{a}$ then $\alpha\surd \xrightarrow{a}$. So, assume $\beta\surd \xrightarrow{a}$. First observe, as α is normed, that $\alpha\surd \xrightarrow{b_1 \cdots b_n \surd}$ for some $n > 0$. As $L(\alpha) \subseteq L(\beta)$, $\beta\surd \xrightarrow{b_1 \cdots b_n \surd}$. As β is locally unary, it must be that $a = b_1$. Hence, $\alpha\surd \xrightarrow{a}$.

Now we show that R is a simulation relation. Assume $(\alpha\surd, \beta\surd) \in R$ and $\alpha\surd \xrightarrow{a} \alpha'$. There is exactly one action b such that $\alpha' \xrightarrow{b}$. If $b = \surd$ then $\alpha' = \surd$. Moreover, there is some β' such that $\beta\surd \xrightarrow{a} \beta' \xrightarrow{\surd}$. Clearly $\beta' = \surd$, so $(\alpha', \beta') \in R$. If $b \neq \surd$ then $\alpha' = \alpha''\surd$ for some α'' and, as Δ' is two-step deterministic

$$L(\alpha''\surd) = \{bw \mid abw \in L(\alpha\surd), w \in (Act^*)\surd\}.$$

As Δ' is two-step deterministic, there is exactly one β' such that $\beta\surd \xrightarrow{a} \beta' \xrightarrow{b}$, $\beta' = \beta''\surd$ and $L(\beta''\surd) = \{bw \mid abw \in L(\beta\surd)\}$. As clearly $L(\alpha''\surd) \subseteq L(\beta''\surd)$ it follows that $(\alpha''\surd, \beta''\surd) \in R$. □

We now have the following:

Theorem 6.6.1 *Ready simulation is undecidable for normed BPA processes.*

Proof: We reduce language containment for deterministic processes to ready simulation for locally unary, two step deterministic processes. Given a deterministic Δ, let $\alpha, \beta \in Var^*$. We now have the following (strongly relying upon Lemmas 6.6.1 and 6.6.2), where $\surd$ is a new action:

$$\begin{array}{rll} L(\alpha) \subseteq L(\beta) \text{ in } \Delta & \text{iff} & L(\alpha) \subseteq L(\beta) \text{ in } \Delta' \\ & \text{iff} & \alpha\surd \subsetneq_r \beta\surd \text{ in } \Delta' \end{array}$$

□

Theorem 6.6.2 *Ready simulation equivalence is undecidable for locally unary normed BPA processes.*

Proof: As in the proof of Theorem 6.2.2, we can reduce simulation to ready simulation equivalence by the following observation, first made by Jan Friso Groote:

$$\alpha \subsetneq_r \beta \quad \text{iff} \quad \alpha + \beta \leftrightarrow_r \beta.$$

□

Chapter 7

Conclusion

In this final chapter we summarize the results of the previous chapters, describe open problems and outline directions for further work on these.

7.1 Summary of the main results

We have shown (in **Chapter 3**) that a modal mu-calculus with labels $\{0, \ldots, n-1\}$ can define the SnS -definable n-ary tree languages up to an observational equivalence. The main idea in the proof is an application of Rabin's theorem, which states that the SnS -definable languages correspond to the n-ary Rabin-recognizable tree languages. For any Rabin-recognizable tree language L we can, by extending our alphabet with a 'silent' label, by using an encoding scheme for the Rabin automaton $\mathcal{A}_L$ accepting L, express the transition relation and acceptance condition for $\mathcal{A}_L$ within the mu-calculus such that a set of trees equivalent to L is defined. In **Chapter 4** we have given an alternative and much simpler proof of the decidability of bisimulation equivalence for normed BPA processes, first proved by Baeten, Bergstra and Klop [BBK87b,BBK87a] and later by Caucal [Cau88,Cau90a]. Our decidability proof uses a tableau system closely related to the branching algorithms employed in the study of equivalence problems in language theory [KH66,Cou83]. If a successful tableau for an equation $\alpha = \beta$ exists, the tableau provides us with a finite witness for a bisimulation containing (α, β). This witness is a self-bisimulation in the sense of [Cau88,Cau90a], which means that we by taking its congruence closure w.r.t. sequential composition get a (potentially infinite) bisimulation containing (α, β). The length of the longest possible path in any tableau for a given equation over variables in a system of process equations Δ has been found. We have presented a sequent-based equational theory for bisimilarity over normed BPA processes in 3-GNF, a result due to Colin Stirling. Its existence follows directly from the tableau system; the theory is shown to be strongly sound and weakly complete. Finally, we have shown how to extract a fundamental relation R (as in the work of [Cau88,Cau90a]) from a successful tableau for $\alpha = \beta$ such that $\alpha \underset{R}{\longleftrightarrow}^* \beta$. This is done via another so-called auxiliary tableau system. Then, in **Chapter 5** we introduced silent actions into normed BPA, considering a class of BPA processes with the restriction that process termination must involve performing an observable action. We have then shown how the decidability method of Chapter 4 could be modified to apply to van Glabbeek's branching bisimulation equivalence. In **Chapter 6** we completed the picture by showing that *all* equivalences below bisimulation in the linear/branching time hierarchy are undecidable for normed BPA processes in 3-GNF and thus that they are undecidable for BPA processes in general. The proofs involve reductions to the language inclusion problem for simple grammars of [Fri76], and the language and trace equivalence problems for normed BPA processes. The rest of the chapter consists of a discussion of open problems that relate to the work in this thesis and how these may be approached.

7.2 Various kinds of infinite transition graphs

A main theme of this thesis has been the exploration of infinite-state transition graphs, in particular their logics and equivalences. Several questions remain open here.

7.2.1 SnS-definable tree languages

It is indeed possible that the mu-calculus with label set $\{0, \ldots, n-1\}$ and SnS are entirely equi-expressive, not just modulo some equivalence. However, it is very unlikely that we will ever find a direct translation from SnS to the mu-calculus. For if we do, this translation must have a hideous complexity, since the mu-calculus is elementary and SnS is not. The encoding scheme employed in Chapter 3 does not contradict this; the extraction of a Rabin automaton from a formula in SnS can introduce an exponential blowup of the number of states every time a negation is considered [Rab69].

7.2.2 Context-free graphs

An interesting question that should be asked in the combined light of our equi-expressiveness result and the results in Chapters 4 to 6 is what happens when we look at the so-called *context-free graphs*, of which the transition graphs of normed BPA form a subclass.

Context-free graphs correspond to the transition graphs for pushdown automata. From the point of view of bisimulation equivalence, pushdown automata are more expressive than context-free grammars: Caucal and Monfort have shown [CM90] that there exist pushdown automata whose transition graphs are not bisimilar to any graph for a context-free grammar/normed BPA process.

Muller and Schupp have shown [MS85] that the counterpart of SnS for context-free graphs, the monadic second-order theory of context-free graphs, is decidable. One may now wonder how the mu-calculus with label set A is related to the monadic second-order theory of context-free graphs over A. A more general question is how the monadic second-order theory of (finitely branching) transition graphs is related to the mu-calculus. It is not immediately clear exactly how one would approach this problem; the result for SnS relied on the correspondence with Rabin automata. Rabin automata were originally introduced because they provide one with a way of reducing the satisfiability of an SnS formula to the question of whether the language accepted by an automaton is empty. In [MS85] Muller and Schupp use a tiling technique for approaching the decidability problem. Whether this could be of use in establishing an equi-expressiveness result is not obvious.

Since the transition graphs of normed BPA processes form a proper subclass of the context-free graphs, one may also wonder whether the decidability result for bisimilarity extends to pushdown automata.

The problem lies with the 'split' lemma, since it relies on the fact that the states in the transition graph of a normed BPA process given in GNF are sequences of process variables - any part of a string of variables is therefore itself a state. This is not the case when we look at pushdown automata. Here a state in the transition graph is of the form qw where q is the state of the automaton and w is a string representing the contents of the stack. So we will have to use a different approach.

Here the various characterizations showing the regularity of context-free graphs may be of interest. The decomposition result used by Baeten, Bergstra and Klop, stating that a transition graph for a normed BPA process has finitely many connected fragments up to translation equivalence, is a special case of a result established by Muller and Schupp [MS85]. In the formulation of Caucal [Cau90b] this result says that the context-free graphs are exactly those generated by some deterministic graph grammar.

7.2.3 Unnormed BPA

We have seen that almost all equivalences are undecidable for BPA, but for bisimulation equivalence the case is certainly not closed, for we have only dealt with normed BPA. Since an important application of process calculi is the modelling of systems that do not terminate, the case where processes can have infinite norms is relevant indeed. A result due to Baeten and Bergstra shows that in the case where the system of equations is guarded and *none* of the process variables have a finite norm the solutions to the defining equations are regular processes and thus have finite transition graphs. Intuitively, this should not surprise us since any process term that follows an unnormed variable can never give rise to a transition – any summand aX_1X_2 in a GNF process equation where $|X_1| = \infty$ will reduce to aX_1. We therefore see that the only interesting case is the case where some process variables are normed and others are not. The main problem that prevents us from a direct extension of our result is again the 'split' lemma (Lemma 2.2.2), here since it no longer holds in the unnormed case. What this tells us, though, is that if we want to use the tableau method, we need to find another way to limit the growth of expressions. A decidability result for the case where all processes involved are *deterministic* has been dealt with by Caucal in [Cau86] in which it was shown that language equivalence for simple context-free grammars is decidable. As was noted in Chapter 6, simple grammars correspond to deterministic BPA processes, so here bisimulation and language equivalence coincide. The approach used in [Cau86] is somewhat similar to that of [Cau88,Cau90a], in that it is shown that $\sim$ is a Thue congruence. Here, however, one cannot rely on properties of the usual norm. Instead a prefix norm $|\ |_f$ given as the homomorphic extension on $(\mathbf{N}, +)$ of

$$|X|_f = \begin{cases} |X| & \text{if } |X| < \infty \\ 0 & \text{if } |X| = \infty \end{cases}$$

is used. Caucal's decision procedure consists of two semi-decision procedures, one searching for the finite relation generating $\sim$ and another searching for a bisimulation error. It is not obvious how or if we could use this to find a tableau decision method similar to that used in this thesis. Moreover, Caucal's method does not provide us with a way of extracting a complete axiomatization as was done in Chapter 4.

7.2.4 Beyond BPA

A natural and obvious extension of BPA would be to allow a notion of parallelism. However, as we introduce more operators, we will eventually reach a process language where bisimilarity is undecidable. But it might be that one could isolate a class of static operators such that strong bisimilarity (and possibly also weak bisimilarity) would remain decidable; here the view of static constructs put forward in e.g. [Hüt88] may have something to offer. In particular one may wonder if the parallel should include some form of communication or simply be the merge operator. The case for weak equivalence is of course especially interesting here if synchronization becomes a possibility.

One of the many problems with going beyond sequential composition is that we in general no longer have finitely many process expressions with a given norm. If we are to use a tableau method similar to that used in Chapters 4 and 5, we will probably need a very different tableau system.

7.3 Complexity bounds

Even though we have established that $\sim$ is decidable for normed BPA, it remains to be seen if the decidability is realistic. In other words: what is the lower complexity bound ? The proof due to Baeten, Bergstra and Klop gives no idea as to the complexity bound of the decision method that they employ. It would be interesting to have a more precise estimate of the value of $N(\Delta_1, \Delta_2, d)$, the level above which a periodic bisimulation must exist, and compare this with our complexity bound in terms of the length of a longest path. Caucal's proof has a decision algorithm which enumerates all the finitely many fundamental relations on $Var \times Var^+$, filtering out the ones that are self-bisimulations and ordering them under $\subseteq$. This algorithm must be exponential in the number of variables but a more careful analysis is required.

7.4 Weak equivalences

The results of Chapter 6 show that almost all the weak counterparts of the equivalences in the linear/branching time hierarchy are undecidable, but there are still several open questions for the weak versions of bisimulation equivalence. For branching bisimulation over $\mathrm{BPA}^{\tau}_{\mathrm{rec}}$ as considered in Chapter 5, the restriction to processes with strictly positive norms is rather strong, as it rules out the possibility of a process terminating silently. A problem with having nullary norms in the tableau system is that we no longer are guaranteed that α and β persist throughout an eliminating subtableau for $X\alpha = Y\beta$, since a match for $X\alpha \xrightarrow{a}$ may require access to observable actions inside β. So the natural question is whether there is a way of introducing nullary norms. Moreover, we would of course also want to get rid of the restriction of normedness altogether. However, since this problem also needs to be tackled for strong bisimulation equivalence, it seems that progress must first be made here before we can give any answer for the branching bisimulation case. Last, but not least the questions for weak bisimilarity all remain open. As we saw, Lemma 5.2.1 does not hold for this equivalence so a different approach must be used in that case.

7.5 Equational theories

It would be nice to give another formulation of the theory presented in Section 4.3 that did not have to use the simultaneous fixed-point induction rule $\mathsf{R12}$. In particular, we would want a formulation that used explicit fixed points in process expressions and extended to *all* BPA processes. The importance of such an equational theory for the full BPA is that it would generalize the axiomatization for regular processes [Mil84]. However, the crucial difficulty in extending Milner's theory centres on the appropriate fixed-point induction rule. A case to ponder on is that $(\mu X.aX)E \sim (\mu X.aXXF)$ for *any* E and F.

Finally, it would be interesting if we could give a syntax-directed version of our tableau system for branching bisimulation since this could give us an equational theory of $\approx_b$ over normed $\mathrm{BPA}^{\tau}_{\mathrm{rec}}$ along the same lines. A naive approach would be to add the τ-laws for branching bisimulation to our equational theory for strong bisimilarity. However, this theory would *not* be able to simulate the tableau construction, in particular the STEP rule. $\mathsf{R12}$ is the rule that corresponds to the REC rule in the tableau system in Chapter 4, and in order to determine the transitions in a STEP application to $\alpha = \beta$ we may need to use the defining equations a different number of times on each side of this equation – think of the case where we are comparing a process with silent actions to one without.

Bibliography

[Abr87] S. Abramsky. Observational equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[BBK87a] 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.

[BBK87b] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *LNCS 259*, pages 93–114. Springer-Verlag, 1987.

[BCG88] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing Kripke structures in temporal logic. *Theoretical Computer Science*, 51:115–131, 1988.

[BHR84] S.D. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.

[BIM90] B. Bloom, S. Istrail, and A. Meyer. Bisimulation can't be traced. Technical Report TR 90-1150, Cornell University, August 1990.

[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 J.W. de Bakker, W.P de Roever, and G. Rozenberg, editors, *LNCS 354*, pages 50–122. Springer-Verlag, 1988.

[BKO88] J.A. Bergstra, J.W. Klop, and E.-R. Olderog. Readies and failures in the algebra of communicating processes. *SIAM Journal on Computing*, 17:1134–1177, 1988.

[BS90] J. Bradfield and C. Stirling. Verifying temporal properties of processes. In *LNCS 458*, pages 115–125. Springer-Verlag, 1990.

[Büc60] J.E. Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology, and Philosophy of Science*, pages 1–11, Stanford, 1960.

[Cau86] D. Caucal. Décidabilité de l'egalité des langages algébriques infinitaires simples. In *Proceedings of STACS 86, LNCS 210*, pages 37–48. Springer-Verlag, 1986.

[Cau88] D. Caucal. Graphes canoniques de graphes algébriques. Rapport de Recherche 872, INRIA, Juillet 1988.

[Cau90a] D. Caucal. Graphes canoniques de graphes algébriques. *Informatique théorique et Applications (RAIRO)*, 24(4):339–352, 1990.

[Cau90b] D. Caucal. On the regular structure of prefix rewriting. In *Proceedings of CAAP 90, LNCS 431*. Springer-Verlag, 1990.

[CE81] E.M. Clarke and E.A. Emerson. Using branching time temporal logic to synthesize synchronization skeletons. In *LNCS 131*, pages 52–71. Springer-Verlag, 1981.

[CM90] D. Caucal and R. Monfort. On the transition graphs of automata and grammars. Technical report, IRISA, 1990.

[Cou83] B. Courcelle. An axiomatic approach to the Korenjak-Hopcroft algorithms. *Mathematical Systems Theory*, 16:191–231, 1983.

[Dam90] Mads Dam. Translating CTL^* into the modal μ-calculus. Technical Report ECS-LFCS-90-123, Laboratory for Foundations of Computer Science, November 1990.

[DNMV90] R. De Nicola, U. Montanari, and F.W. Vaandrager. Back and forth bisimulations. In J. Bergstra and J.W. Klop, editors, *CONCUR 90, LNCS 458*, pages 152–165. Springer-Verlag, August 1990.

[DNV90] R. De Nicola and F.W. Vaandrager. Three logics for branching bisimulation. In *Proceedings of 5th Annual Symposium on Logic in Computer Science (LICS 90)*. IEEE, Computer Society Press, 1990.

[Eng85] J. Engelfriet. Determinacy $\rightarrow$ (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36(1):21–25, 1985.

[ES84] E.A. Emerson and R.S. Street. The propositional mu-calculus is elementary. In *Proceedings of 11th ICALP, LNCS 172*, pages 465–472. Springer-Verlag, 1984.

[Fri76] E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1:297–316, 1976.

[GH91] J.F. Groote and Hüttel. Undecidable equivalences for basic process algebra. Technical Report ECS-LFCS-91-169, Department of Computer Science, University of Edinburgh, August 1991.

[Gro89] J.F. Groote. Transition system specifications with negative premises. Report CS-R8950, CWI, 1989. An extended abstract appeared in J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 90*, Amsterdam, *LNCS 458*, pages 332–341. Springer-Verlag, 1990.

[GV89] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence (extended abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Del la Rocca, editors, *Proceedings of 16th ICALP, LNCS 372*, pages 423–438. Springer-Verlag, 1989. Full version to appear in *Information and Computation*.

[Hen89] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1989.

[HM85] M. Hennessy and A.J.R.G Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[Hoa88] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1988.

[HS91] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of 6th Annual Symposium on Logic in Computer Science (LICS 91)*, pages 376–386. IEEE Computer Society Press, 1991.

[HT87] T. Hafer and W. Thomas. Computation tree logic CTL^* and path quantifiers in the monadic theory of the binary tree. In *Proceedings of 11th ICALP, LNCS 267*, pages 269–279, 1987.

[HT90] Dung T. Huynh and Lu Tian. On deciding readiness and failure equivalences for processes. Technical Report UTDCS-31-90, University of Texas at Dallas, September 1990.

[HU79] J. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Hüt88] H. Hüttel. Operational and denotational properties of a modal process logic. Master's thesis, Aalborg University Centre, 1988.

[Hüt90] H. Hüttel. *SnS* can be modally characterized. *Theoretical Computer Science*, 74:239–248, 1990.

[Hüt91] H. Hüttel. Silence is golden: Branching bisimilarity is decidable for context-free processes. In *Proceedings of CAV91, LNCS 575*, pages 2–12. Springer-Verlag, 1991.

[KH66] A.J. Korenjak and J.E. Hopcroft. Simple deterministic languages. In *Proceedings of Seventh Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KS90] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[LS90] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *16th Symp. Principles of Programming Languages*, pages 344–352. ACM, January 1990.

[Mey75] A.R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In *Proceedings of Boston Univ. Logic Colloquium, LNCM 453*, pages 132–154. Springer-Verlag, 1975.

[Mil80] Robin Milner. *A Calculus of Communicating Systems, LNCS 92*. Springer-Verlag, 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 International, 1989.

[MS85] D. Muller and P. Schupp. The theory of ends, pushdown automata and second order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[Niw88] D. Niwinski. Fixed points vs. infinite generation. In *Proceedings of 3rd Annual Symposium on Logic in Computer Science (LICS 88)*, pages 402–409, Edinburgh, 1988. IEEE Computer Society Press.

[OH86] E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings of 5th GI Conference LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[Phi87] I.C.C. Philips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987.

[Plo81] Gordon Plotkin. A structural approach to operational semantics. Technical Report FN-19, Computer Science Department, Aarhus University, 1981.

[Rab69] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.

[Rab77] M.O. Rabin. Decidable theories. In Barwise, editor, *Handbook of Mathematical Logic*, pages 595–629. North-Holland, 1977.

[RB81] W.S. Rounds and S.D. Brookes. Possible futures, acceptances, refusals and communicating processes. In *Proc. 22nd Annual Symposium on Foundations of Computer Science*, pages 140–149, New York, 1981. IEEE.

[Sti87] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, 49:311–347, 1987.

[Sti91] C. Stirling. Modal and temporal logics. In Abramsky, editor, *Handbook of Logic in Computer Science*. Oxford University Press, 1991. To appear.

[SW89] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *LNCS 351*, pages 369–383. Springer-Verlag, 1989.

[Tho90] W. Thomas. Automata on infinite objects. In van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 133–191. North-Holland, 1990.

[vG90a] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, CWI/Vrije Universiteit, 1990.

[vG90b] R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR 90,* Amsterdam*, LNCS 458*, pages 278–297. Springer-Verlag, 1990.

[vGW89a] R.J. van Glabbeek and P.W. Weijland. Refinement in branching time semantics. Report CS-R8922, CWI, Amsterdam, 1989. Also appeared in: Proceedings of AMAST, May 1989, Iowa, USA, pp. 197–201.

[vGW89b] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989.

[VWS83] M. Vardi, P. Wolper, and A.P. Sistla. Reasoning about infinite computation paths. In *Proceedings of 24th IEEE FOCS*, pages 185–194, 1983.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.