# Clocks, Trees and Stars

# in Process Theory

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam,
op gezag van de Rector Magnificus
prof. dr. P.W.M. de Meijer
ten overstaan van een door het college van dekanen
ingestelde commissie in het openbaar te verdedigen
in de Aula der Universiteit
(Oude Lutherse Kerk, ingang Singel 411, hoek Spui),
op donderdag 1 december 1994 te 9.00 uur

door
Willem Jan Fokkink
geboren te Oegstgeest

1994

| | |
|---|---|
| $1^e$ Promotor: | prof. dr. J.A. Bergstra |
| $2^e$ Promotor: | prof. dr. J.C.M. Baeten |
| Faculteit: | Wiskunde en Informatica |

The work in this thesis has been carried out at the CWI in Amsterdam, in the context of:

- ESPRIT basic research action no. 3006, Theories of Concurrency: unification and extension (CONCUR),
- RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS),
- RACE project no. 2076, Broadband Object-Oriented Service Technology: a tools environment for advanced telecommunication services (BOOST).

Printed by CopyPrint 2000, Enschede



CWI, P.O. Box 94079, 1090 GB Amsterdam, wan@cwi.nl

ISBN: 90-74795-11-0

# Contents

# Acknowledgements

First of all I want to thank my first promotor Jan Bergstra, omnipresent, full of energy, never short of an original opinion or a fresh scheme.

Second promotor Jos Baeten was my supervisor in the first year, and arranged my appointments in three international research projects.

It has been a great pleasure to live in the friendly and stimulating working climate of the CWI, these last four years. Frits Vaandrager has been an ideal supervisor, always willing to discuss any topic or to read any draft and provide good comments. I am very grateful to Henri Korver, Steven Klusener, Doeko Bosscher and David Griffioen for creating a special AP2 atmosphere, being a unique mixture of small talk, weird parties and devoted science.

Chris Verhoef played a key role in my getting the job at the CWI, and he posed me a question which resulted in writing the paper on tree rules. Special thanks also go to Jos van Wamel, companion in the ORFIS project, who has become such a good friend, and to ski and skate partner Alban Ponse, whose canteen question on iteration provided me half a year of research.

Hans Zantema is the co-author of Chapter 3, on the Kleene star. If ever you have a problem in termination of a term rewriting system, I strongly recommend that you send it to `hansz@cs.ruu.nl`. Steven Klusener is the co-author of Chapter 7, on real time process algebra. We have had endless technical debates on this topic, which, I must confess, were mostly won by him.

I thank the members of the reading committee Jaco de Bakker, Peter van Emde Boas, Catuscia Palamidessi and Frits Vaandrager, for their careful review of this thesis, and Paul Klint for his willingness to take part in the opposition.

Finally, I want to thank for various reasons, Praethuys companion Arie van Deursen, secretary Mieke Bruné, Rob van Glabbeek for his fine comments on the paper on tree rules, Catuscia Palamidessi who initiated the generalization of the unification theorem, Jan Friso Groote and Bas van Vlijmen from the NS project, ex-room-mate Joris Hillebrand, cleaner Mustapha Rouhou for his Ramadan meals, Joop Fokkink for the enormous joint effort spent on renovating Bankastraat 5 $^{\text{II}}$, chess-mate Niek Narings who comes back from Spain in order to take part in the ceremony, and many many others for entertaining conversations near the cappuccino machine.

# 1

# Preface

Process algebra, or process theory, constitutes an attempt to reason about 'behaviours of systems' in a mathematical framework. Starting from a syntax, each syntactic object is supplied with some kind of behaviour, and a semantic equivalence says which behaviours are to be identified. Three well-known semantic equivalences are trace equivalence, strong bisimulation and branching bisimulation. Process algebra expresses such equivalences in axioms, or equational laws. We require that a set of axioms is sound (i.e. if two behaviours can be equated, then they are semantically equivalent), and we desire that it is complete (i.e. if two behaviours are semantically equivalent, then they can be equated).

Process algebra can be applied to prove correctness of system behaviour. It enables to express (un)desirable properties of the behaviour of a system in an abstract way, and to deduce by mathematical manipulations whether or not the behaviour satisfies such a property.

Process algebra has links with a surprising number of other fields in theoretical computer science, such as term rewriting, abstract data types, formal languages, operational semantics, dynamic logic and modal logic. This thesis shows clear signs of such connections: Chapters 2 and 3 consider an operator from formal languages, and make heavy use of term rewriting, Chapter 4 deals with a format in operational semantics, and Chapter 5 generalizes a folk result from logic programming to infinite sets.

The diversity of topics in this thesis is caused by my main scientific interest, which is trying to solve open questions. This characteristic has the virtue that the problems that are solved in this thesis are in general of a complicated nature, but it has the drawback that the introductions of the chapters do not excel in providing a deep motivation for these problems.

The contents of this thesis can be divided into three distinct parts, namely clocks, trees and stars. Actually, the only obvious link between these three parts is that they are in the realm of process algebra. The title of the thesis reflects the chronological order in which the papers on these topics have been produced. In the thesis itself this order has been reversed.

The chapters in this thesis are self-contained, so that they can be read separately.

In such a set-up, repetition of basic definitions is hard to avoid. Especially the last three chapters contain quite some overlap in their preliminaries.

## Stars

During the early days of computer science, about forty years ago, Kleene introduced a binary operator $x^*y$, called iteration or Kleene star. It describes the behaviour of the program `while` $b$ `do` $x$ `od` ; $y$, that is, $x^*y$ can choose to execute either $x$, after which it evolves into $x^*y$ again, or $y$, after which it terminates. Two years later, a unary version $x^*$ of iteration was proposed, which has been studied extensively ever since. In 1964, Redko proved that there does not exist a complete finite set of (unconditional) axioms for unary iteration with respect to trace equivalence.

One and a half year ago, Bergstra, Bethke and Ponse proposed eight axioms for Basic Process Algebra (BPA) extended with the binary Kleene star; the five standard axioms from BPA together with three extra axioms for iteration.

$$\begin{aligned} x \cdot x^*y + y &= x^*y \\ x^*y \cdot z &= x^*(yz) \\ x^*(y \cdot (x+y)^*z + z) &= (x+y)^*z \end{aligned}$$

They conjectured that these axioms are complete with respect to strong bisimulation (in the absence of the constants 'empty process' and 'deadlock'). Chapter 3 contains a proof of this conjecture.

The usual strategy for proving such a completeness result is to produce a Term Rewriting System (TRS) from the axioms as follows.

1. Turn the axioms $s = t$ into rewrite rules $s \to t$.

2. Add extra rewrite rules in order to make the TRS weakly confluent, which means that if there are one-step reductions from a term $p$ to terms $p'$ and $p''$, then both $p'$ and $p''$ can be reduced to a term $q$.

3. Check that the resulting TRS is terminating, which means that there are no infinite reductions.

Then Newman's Lemma ensures that each term reduces to a unique normal form, which does not reduce any further. The last step in the completeness proof is to show that normal forms with equivalent behaviour are syntactically the same.

In this case, the extra rewrite rules that are needed in order to make the TRS weakly confluent are very complicated. As a consequence, it became a major problem to prove termination. Luckily, this property could be deduced by means of semantic labelling, a technique for proving termination which has been developed by Hans Zantema quite recently. Finally, a hairy proof showed that strongly bisimilar normal forms are indeed syntactically the same (modulo AC).

With this result at hand, it turned out to be quite easy to come up with a complete set of axioms for prefix iteration $a^*x$, where the left argument is restricted

to atomic actions. Moreover, the constant deadlock could be added to this syntax without severe complications. In Chapter 2 it is proved that basic CCS extended with prefix iteration is axiomatized completely by the four standard axioms for basic CCS together with two extra axioms for prefix iteration.

$$\begin{aligned} a \cdot a^*x + x &= a^*x \\ a^*(a^*x) &= a^*x \end{aligned}$$

The completeness proof for prefix iteration in Chapter 2 is in fact a highly simplified version of the completeness proof for iteration, and it makes a perfect introduction before reading Chapter 3.

## Trees

In order to describe a semantic equivalence by means of axioms, it is essential that such an equivalence is a congruence. This means that if $p_i$ and $q_i$ are equivalent behaviours for $i = 1, ..., n$, and if $f$ is a function with $n$ arguments, then $f(p_1, ..., p_n)$ and $f(q_1, ..., q_n)$ are equivalent behaviours. Unfortunately, proofs of such congruence properties are invariantly long, technical and boring. Therefore, until recently, such congruence proofs were usually skipped, either with the claim 'trivial' or with the one-liner 'left to the reader'.

A popular way to supply terms with behaviour is by means of transition rules à la Plotkin. De Simone was the first to define a format for transition rules which ensures that behaviour defined by rules in this format is always a congruence for strong bisimulation. This format was generalized by Groote and Vaandrager to the so-called tyft/tyxt format. They proved that behaviours generated by well-founded tyft/tyxt rules are always a congruence for strong bisimulation. They showed that all the syntactic restrictions of the tyft/tyxt format are essential for this congruence result, but it was unclear whether the well-foundedness restriction is vital. This restriction was needed in the proof, but no counter-examples were found to show that the congruence theorem breaks down without it.

Chapter 4 provides the answer to this open question, namely, it shows that the well-foundedness restriction can be omitted. This follows from a stronger result, which says that for each collection of transitions rules in tyft/tyxt format, there is an equivalent collection of transition rules in the more restrictive tree format. Tree rules are well-founded, so the congruence theorem of Groote and Vaandrager applies to this format.

A key lemma in the proof that tyft/tyxt reduces to tree turned out to be a weaker version of a well-known result in unification theory. This result uses the following definitions:

1. A substitution $\sigma$ is idempotent if $\sigma\sigma = \sigma$.

2. A substitution $\sigma$ is a unifier of a collection $E$ of equations if $(s)\sigma = (t)\sigma$ for all equations $s = t$ in $E$.

3. A unifier $\Theta$ of $E$ is called most general if each unifier of $E$ is of the form $\Theta\sigma$ for some substitution $\sigma$.

A classical theorem says that if a finite set of equations allows a unifier, then it allows an idempotent most general unifier. The lemma in Chapter 4 however also applies to infinite sets of equations. This led to the generalization of this unification theorem to infinite sets, which is presented in Chapter 5.

## Clocks

Each field of science is familiar with the phenomenon fashion. Suddenly, a certain special topic $A$ becomes a main point of interest, many scientists express their own views on $A$, conferences are dedicated to $A$, journals are flooded by papers on $A$. At the time of my appointment at the CWI, time clearly was the fashion in process algebra. While reasoning about behaviours, time often is a crucial and complicating factor. Hence, many process algebras were extended with some notion of time.

Jos Baeten and Jan Bergstra defined an extension of the Algebra of Communicating Processes (ACP) with real time. They introduced the advanced notion of integration, which allows to express time dependencies, i.e. the behaviour of a process may depend on the moment in time when some previous action was executed. Steven Klusener introduced quite a number of new concepts for this algebra, such as prefix integration and conditional terms, and he defined an axiomatization for conditional terms. According to my job description, I started to work in the algebra ACP$\rho I$ that was proposed by Steven.

My first feat, which is presented in Chapter 7, was to show that strong bisimulation equivalence for ACP$\rho I$ is decidable, i.e. for each pair of terms in ACP$\rho I$ it can be decided whether or not they are bisimilar. The decision algorithm is based on Steven's axiomatization for conditional terms.

In Chapter 6 it is investigated whether the merge $\|$ can be eliminated from regular processes in ACP$\rho I$ with recursion, which means that for each pair of regular processes $p$ and $q$, their merge $p\|q$ is regular too. This natural question leads to a remarkable result, namely, the answer is 'no' for the full algebra of regular processes, but it is 'yes' for a certain subalgebra. That is, for each pair of processes in this algebra, their merge is bisimilar to a process in this algebra. This subclass is very specific, because if it is enlarged or restricted in any obvious way, then the elimination result is lost. The discovered algebra is equal to the class of timed automata of Alur and Dill, which is a popular extension of automata with time.

Steven defined branching bisimulation in the presence of time. Chapter 8 presents a complete axiomatization for Basic Process Algebra extended with the constant 'silent step', denoted by $\tau$, and deadlock and recursion and time, but without integration, modulo branching bisimulation. It turns out that it is much easier to deal with the silent step together with recursion in the presence of time. Namely, the untimed recursive equation $X = \tau \cdot X$ has infinitely many solutions $\tau \cdot p$, while its timed variant allows only one solution.

# Origins of the chapters

Chapter 2. W.J. Fokkink. A complete equational axiomatization for prefix iteration. *Information Processing Letters*, 52(6):333–337, 1994.

Elsevier is acknowledged for their permission to print this paper.

Chapter 3. W.J. Fokkink and H. Zantema. Basic process algebra with iteration: completeness of its equational axioms. *The Computer Journal*, 37(4):259–267, 1994.

Chapter 4. W.J. Fokkink. The tyft/tyxt format reduces to tree rules. In M. Hagiya and J.C. Mitchell, editors, *Proceedings 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, Sendai, Japan, *LNCS 789*, pages 440–453. Springer-Verlag, 1994.

Chapter 5. W.J. Fokkink. Idempotent most general unifiers for infinite sets. Report CS-R9442, CWI, Amsterdam, 1994.

Chapter 6. W.J. Fokkink. An elimination theorem for regular behaviours with integration. In E. Best, editor, *Proceedings 4th International Conference on Concurrency Theory (CONCUR'93)*, Hildesheim, *LNCS 715*, pages 432–446. Springer-Verlag, 1993.

Chapter 7. W.J. Fokkink and A.S. Klusener. An effective axiomatization for real time ACP. To appear in *Information and Computation*.

Chapter 8. W.J. Fokkink. Regular processes with rational time and silent steps. Report CS-R9231, CWI, Amsterdam, 1992.

# 2

# A Complete Equational Axiomatization for Prefix Iteration

Wan Fokkink

Prefix iteration $a^*x$ is added to *Minimal Process Algebra* ($\mathrm{MPA}_\delta$), which is a subalgebra of $\mathrm{BPA}_\delta$ equivalent to Milner's basic CCS. We present a finite equational axiomatization for $\mathrm{MPA}^*_\delta$, and prove that this axiomatization is complete with respect to strong bisimulation equivalence. To obtain this result, we set up a term rewriting system, based on the axioms, and show that bisimilar terms have the same normal form.

## 2.1 Introduction

Kleene [5] defined a binary operator $\_^*\_$ in the context of finite automata, called *Kleene star* or *iteration*. Intuitively, the expression $p^*q$ yields a solution for the recursive equation $X = p \cdot X + q$. In other words, $p^*q$ can choose to execute either $p$, after which it evolves into $p^*q$ again, or $q$, after which it terminates.

Milner [9] studied the unary version $p^*$ of the Kleene star in the setting of (strong) bisimulation equivalence, and raised the question whether there exists a complete axiomatization for it. Bergstra, Bethke and Ponse [1] incorporated the binary Kleene star in Basic Process Algebra (BPA) [2], and they suggested three equational axioms for iteration. In Chapter 3 it is proved that these three axioms, together with the five standard axioms for BPA, are a complete axiomatization for $\mathrm{BPA}^*$ modulo bisimulation.

In this chapter, we add the deadlock $\delta$ to the syntax. Sewell [12] proved that there does not exist a complete finite equational axiomatization for $\mathrm{BPA}^*_\delta$. In order to prove a completeness result, nevertheless, we restrict the binary sequential composition $x \cdot y$ to its unary prefix version $a \cdot x$, to obtain *Minimal Process Algebra* $\mathrm{MPA}_\delta$, equivalent to basic CCS [8]. Likewise, we add prefix iteration $a^*x$ to the syntax, resulting in the algebra $\mathrm{MPA}^*_\delta$. This algebra is less expressive than $\mathrm{BPA}^*_\delta$. For instance, it cannot express a simple process such as $(a + b)^*c$. On the other

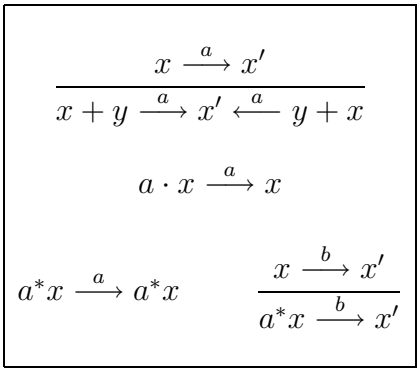


Table 2.1: Action rules for $\mathrm{MPA}^*_\delta$

hand, it contains processes which can be expressed neither in $\mathrm{BPA}^*$ nor in $\mathrm{BPA}_\delta$, such as $a^*\delta$.

We propose two simple equational axioms for iteration, which are actually instantiations of the first and the third axiom for the binary Kleene star. We prove that these two axioms, together with the four standard axioms of $\mathrm{MPA}_\delta$, are a complete axiomatization for $\mathrm{MPA}^*_\delta$ with respect to bisimulation. The proof consists of producing a term rewriting system from the axioms, and showing that bisimilar normal forms are equal modulo AC. This method yields an algorithm to decide whether or not two terms are bisimilar.

**Acknowledgements.** Jan Bergstra initiated this research, and Jos van Wamel provided helpful comments.

## 2.2 Minimal Process Algebra with Iteration

We assume an alphabet $A$ of atomic actions. The signature of the algebra $\mathrm{MPA}^*_\delta(A)$, or $\mathrm{MPA}^*_\delta$ for short, consists of a constant $\delta$, which represents deadlock, together with the binary alternative composition $x+y$, and the unary prefix sequential composition $a \cdot x$ and prefix iteration $a^*x$, for $a \in A$. Table 2.1 presents an operational semantics for $\mathrm{MPA}^*_\delta$ in Plotkin style [11]. Prefix iteration $a^*x$ can choose to execute either $a$, after which it evolves into $a^*x$ again, or $x$.

Our model for $\mathrm{MPA}^*_\delta$ consists of all the closed terms that can be constructed from deadlock and the three operators. That is, the BNF grammar for the collection of process terms is as follows, where $a \in A$:

$$p \quad ::= \quad \delta \mid p + p \mid a \cdot p \mid a^*p.$$

As binding convention, $^*$ binds stronger than $\cdot$, which in turn binds stronger than $+$.

Process terms are considered modulo *(strong) bisimulation equivalence* [10]. Intuitively, two process terms are bisimilar if they have the same branching structure.

| | | | |
|---|---|---|---|
| A1 | $x+y$ | $=$ | $y+x$ |
| A2 | $(x+y)+z$ | $=$ | $x+(y+z)$ |
| A3 | $x+x$ | $=$ | $x$ |
| A6 | $x+\delta$ | $=$ | $x$ |
| MI1 | $a\cdot a^*x+x$ | $=$ | $a^*x$ |
| MI3 | $a^*(a^*x)$ | $=$ | $a^*x$ |

Table 2.2: Axioms for $\mathrm{MPA}^*_\delta$

**Definition 2.1** *Two processes $p_0$ and $q_0$ are called* bisimilar*, denoted by $p_0 \leftrightarrows q_0$, if there exists a symmetric relation $\mathcal{B}$ on processes such that $p_0 \mathcal{B} q_0$, and if $p \xrightarrow{a} p'$ and $p\mathcal{B}q$, then there is a transition $q \xrightarrow{a} q'$ with $p'\mathcal{B}q'$.*

The action rules in Table 2.1 are in the *tyft/tyxt* format of Groote and Vaandrager [4]. Hence, bisimulation equivalence is a congruence with respect to all the operators, i.e. if $p \leftrightarrows p'$ and $q \leftrightarrows q'$, then $p+q \leftrightarrows p'+q'$ and $a\cdot p \leftrightarrows a\cdot p'$ and $a^*p \leftrightarrows a^*p'$. See [4] for the definition of the tyft/tyxt format, and for a proof of this congruence result. (This proof uses the extra assumption that the rules are *well-founded*. In Chapter 4 it is shown that this requirement can be dropped.)

Furthermore, the three rules for $\mathrm{MPA}_\delta$ are *pure*, and the two rules for iteration incorporate the Kleene star in the left-hand side of their conclusions. Hence, $\mathrm{MPA}^*_\delta$ is an operationally conservative extension of $\mathrm{MPA}_\delta$, i.e. the action rules for iteration do not influence the transition systems of $\mathrm{MPA}_\delta$ terms. See [4] for the definitions, and for a proof of this conservativity result.

Table 2.2 contains an axiom system for $\mathrm{MPA}^*_\delta$, which consists of the four axioms from $\mathrm{MPA}_\delta$ together with two axioms for iteration. In the sequel, $p=q$ will mean that the equality can be derived from these axioms. The axiomatization for $\mathrm{MPA}^*_\delta$ is sound with respect to bisimulation equivalence, i.e. if $p=q$ then $p \leftrightarrows q$. Since bisimulation is a congruence, this can be verified by checking soundness for each axiom separately, which is left to the reader. In this chapter it is proved that the axiomatization is complete with respect to bisimulation, i.e. if $p \leftrightarrows q$ then $p=q$.

## 2.3 A Term Rewriting System

Our aim is to prove that the axioms in Table 2.2 are complete for our model of $\mathrm{MPA}^*_\delta$ modulo bisimulation. A standard scheme for such a proof is to set up a Term Rewriting System (TRS) from the axioms as follows.

1. Turn the axioms into rewrite rules.

2. Apply the Knuth-Bendix completion algorithm [7], which yields extra rewrite

rules to make the TRS *weakly confluent*. That is, if a term $p$ has one-step reductions $p'$ and $p''$, then both terms can be reduced to a term $q$.

3. Check that the resulting TRS is *terminating*, which means that there are no infinite reductions.

If a TRS is weakly confluent and terminating, then Newman's Lemma says that it reduces each term to a unique normal form, which does not reduce any further. The construction of the TRS ensures that all its rules can be deduced from the axioms. The final step in the completeness proof is to show that bisimilar normal forms are syntactically equal.

See [3, 6] for an overview of the field of term rewriting.

### 2.3.1 Proper iteration

We want to define a TRS for process terms that reduces bisimilar terms to the same normal form. However, it is not so easy to construct such a TRS for $\mathrm{MPA}^*_\delta$. Namely, the terms $a^*x + x$ and $a^*x$ are bisimilar, so they should reduce to the same normal form. A rule $a^*x \longrightarrow a^*x + x$ does not terminate, so we need the rule

$$a^*x + x \;\longrightarrow\; a^*x.$$

This rule is not yet sufficient, because it does not deal with the case $a^*(b^*x) + x \leftrightarrows a^*(b^*x)$. Hence, for this case we must introduce an extra rewrite rule. But this rule does not cover the case $a^*(b^*(c^*x)) + x \leftrightarrows a^*(b^*(c^*x))$, etc. So in order to obtain unique normal forms modulo bisimulation for $\mathrm{MPA}^*_\delta$, apparently we need an infinite number of rewrite rules.

To avoid this complication, we replace iteration by an equivalent operator $a^\oplus x$, called *proper* prefix iteration, which represents the behaviour of $a \cdot a^*x$.[1] The operational semantics and the axiomatization for proper iteration are given in Table 2.3. They are obtained from the action rules and axioms for $\mathrm{MPA}^*_\delta$, using the equivalences $a^*x \leftrightarrows a^\oplus x + x$ and $a^\oplus x \leftrightarrows a \cdot a^*x$. Note that

$$\begin{array}{lcll} \mathrm{MPA}^*_\delta & + & (a^\oplus x = a \cdot a^*x) & \vdash \ \mathrm{PMI1,3}, \\ \mathrm{MPA}^\oplus_\delta & + & (a^*x = a^\oplus x + x) & \vdash \ \mathrm{MI1,3}. \end{array}$$

So we find that the axiomatization in Table 2.3 is complete for $\mathrm{MPA}^\oplus_\delta$ if and only if the axiomatization in Table 2.2 is complete for $\mathrm{MPA}^*_\delta$.

### 2.3.2 The TRS for $\mathrm{MPA}^\oplus_\delta$

We want to find a TRS for $\mathrm{MPA}^\oplus_\delta$ that reduces bisimilar terms to the same normal form. In particular, the TRS should be terminating. Axioms A1,2 obstruct this property, so from now on process terms are considered modulo AC (that is, modulo

[1] The standard notation for this construct would be $a^+x$, but we want to avoid ambiguous use of the $+$.

| $a^{\oplus}x \xrightarrow{a} a^{\oplus}x + x$ | | |
|---|---|---|
| PMI1 | $a \cdot (a^{\oplus}x + x)$ | $= a^{\oplus}x$ |
| PMI3 | $a^{\oplus}(a^{\oplus}x + x)$ | $= a^{\oplus}x$ |

Table 2.3: Semantics and axioms for proper iteration

| | | | |
|---|---|---|---|
| 1. | $x + x$ | $\longrightarrow$ | $x$ |
| 2. | $x + \delta$ | $\longrightarrow$ | $x$ |
| 3. | $a \cdot (a^{\oplus}x + x)$ | $\longrightarrow$ | $a^{\oplus}x$ |
| 4. | $a^{\oplus}(a^{\oplus}x + x)$ | $\longrightarrow$ | $a^{\oplus}x$ |
| 5. | $a \cdot (a^{\oplus}\delta)$ | $\longrightarrow$ | $a^{\oplus}\delta$ |
| 6. | $a^{\oplus}(a^{\oplus}\delta)$ | $\longrightarrow$ | $a^{\oplus}\delta$ |

Table 2.4: Rewrite rules for $\mathrm{MPA}^{\oplus}_{\delta}$

associativity and commutativity of the $+$). This equivalence is denoted by $p =_{\mathrm{AC}} q$, and we say that $p$ and $q$ are of the same form.

Table 2.4 contains a TRS for $\mathrm{MPA}^{\oplus}_{\delta}$, which is obtained in two steps. First, axioms A3,6 and MI1,3 are turned into rewrite rules, aiming from left to right. Next, the Knuth-Bendix completion algorithm is applied, which yields Rules 5 and 6. The resulting TRS in Table 2.4 is weakly confluent, and all its rules can be deduced from the axioms for $\mathrm{MPA}^{\oplus}_{\delta}$. Furthermore, in each rule the term at the left-hand side contains more symbols than the term at the right-hand side, so clearly the TRS is terminating. Thus, Newman's Lemma ensures that the TRS reduces each term to a unique normal form, modulo AC.

## 2.4 Normal Forms Decide Bisimilarity

We have developed a TRS for $\mathrm{MPA}^{\oplus}_{\delta}$ that reduces terms to a unique normal form. Its rules can all be deduced from the axioms of $\mathrm{MPA}^{\oplus}_{\delta}$. Therefore, all the rules are sound with respect to bisimulation equivalence, so each term is bisimilar with its normal form. Hence, in order to determine completeness of the axiomatization for $\mathrm{MPA}^{\oplus}_{\delta}$ with respect to bisimulation, it is sufficient to prove that if two normal forms are bisimilar, then they are equal modulo AC.

The proof of the completeness theorem is in fact a simplified version of the

completeness proof in Chapter 3, with some minor extra cases to deal with deadlock. We apply induction on the following weight function on terms:

$$\begin{aligned} g(\delta) &= 0 \\ g(p+q) &= \max\{g(p), g(q)\} \\ g(a \cdot p) &= g(p) + 1 \\ g(a^{\oplus} p) &= g(p) + 1. \end{aligned}$$

Clearly, each process term $p$ is a sum of terms of the form $\delta$ and $a \cdot q$ and $a^{\oplus} q$, which are called the *summands* of $p$.

**Theorem 2.2** *If two normal forms $p$ and $q$ are bisimilar, then $p =_{\mathrm{AC}} q$.*

**Proof.** We apply induction on $g(p) + g(q)$. If $g(p) + g(q) = 0$, then both $p$ and $q$ must be sums of $\delta$. Since $p$ and $q$ are normal forms, Rule 1 ensures that both $p$ and $q$ are of the form $\delta$, so $p =_{\mathrm{AC}} q$.

Now assume that we have already proved the theorem for bisimilar normal forms $p$ and $q$ with $g(p) + g(q) < n$, for some $n \geq 1$. We prove it for $g(p) + g(q) = n$, by showing that the separate bisimilar summands of $p$ and $q$ are of the same form. Since $g(p)+g(q) > 0$, clearly $p$ and $q$ are not bisimilar to $\delta$. Then Rule 2 ensures that they do not contain any summands $\delta$. This leaves the following three possibilities.

1. First, suppose that summands $a \cdot r$ of $p$ and $a \cdot s$ of $q$ are bisimilar, so $r \leftrightarrow s$. Since $g(r) + g(s) < n$, the induction hypothesis yields $r =_{\mathrm{AC}} s$.

2. Next, let summands $a \cdot r$ and $a^{\oplus} s$ be bisimilar, so $r \leftrightarrow a^{\oplus} s + s$. We deduce a contradiction.

   If $s \neq_{\mathrm{AC}} \delta$, then $a^{\oplus} s + s$ is a normal form, because we cannot apply Rule 1 or 2 to $a^{\oplus} s + s$, and $a^{\oplus} s$ and $s$ are normal forms. Moreover, $g(r) + g(a^{\oplus} s + s) < n$, so the induction hypothesis yields $r =_{\mathrm{AC}} a^{\oplus} s + s$. Then we can apply Rule 3 to $a \cdot r =_{\mathrm{AC}} a \cdot (a^{\oplus} s + s)$, so $a \cdot r$ is not a normal form. Contradiction.

   If $s =_{\mathrm{AC}} \delta$, then $r \leftrightarrow a^{\oplus}\delta$, and $g(r)+g(a^{\oplus}\delta) < n$, so induction yields $r =_{\mathrm{AC}} a^{\oplus}\delta$. Then we can apply Rule 5 to $a \cdot r =_{\mathrm{AC}} a \cdot (a^{\oplus}\delta)$. Again, contradiction.

3. Finally, assume that summands $a^{\oplus} r$ and $a^{\oplus} s$ are bisimilar, so $a^{\oplus} r + r \leftrightarrow a^{\oplus} s + s$. We prove $r =_{\mathrm{AC}} s$.

   If $r$ and $s$ do not contain summands that are bisimilar with $a^{\oplus} s$ and $a^{\oplus} r$ respectively, then $a^{\oplus} r + r \leftrightarrow a^{\oplus} s + s$ implies $r \leftrightarrow s$. Since $g(r) + g(s) < n$, induction yields $r =_{\mathrm{AC}} s$, and we are done.

   So suppose that either $r$ contains a summand bisimilar to $a^{\oplus} s$, or $s$ contains a summand bisimilar to $a^{\oplus} r$. We deduce a contradiction.

   By symmetry, it is sufficient to deduce a contradiction for the first case only, where $r$ contains a summand bisimilar to $a^{\oplus} s$. Induction yields that this summand of $r$ is of the form $a^{\oplus} s$. According to Rule 1, $r$ can contain only one subterm of the form $a^{\oplus} s$. Hence, either $r =_{\mathrm{AC}} a^{\oplus} s$, or $r =_{\mathrm{AC}} a^{\oplus} s + r'$ where

the summands of $r'$ are not bisimilar to $a^{\oplus}s$. Then $a^{\oplus}r + r \leftrightarrow a^{\oplus}s + s$ implies that the summands of $r'$ are bisimilar to summands of $s$.

The term $s$ does not contain any summands bisimilar to $a^{\oplus}s$ or $a^{\oplus}r$. For else, induction would yield that this summand is of the form $a^{\oplus}s$ or $a^{\oplus}r$ respectively, which would imply that $s$ contains more symbols than $s$ or $r$ respectively. However, clearly $s$ cannot contain more symbols than itself, and since $r$ has a summand $a^{\oplus}s$, it follows that $r$ contains more symbols than $s$.

Recall that $r$ is either of the form $a^{\oplus}s + r'$ or $a^{\oplus}s$, and if $r'$ occurs, then all its summands are bisimilar to summands of $s$. Conversely, since $a^{\oplus}r + r \leftrightarrow a^{\oplus}s + s$, and since the summands of $s$ are not bisimilar to $a^{\oplus}s$ or $a^{\oplus}r$, it follows that they must all be bisimilar to summands of $r'$, or to $\delta$. Hence, either $s \leftrightarrow r'$ if $r'$ occurs, or $s \leftrightarrow \delta$ otherwise. We distinguish the two possibilities.

- $r =_{\mathrm{AC}} a^{\oplus}s + r'$ and $s \leftrightarrow r'$. Then induction implies $s =_{\mathrm{AC}} r'$, so we can apply Rule 4 to $a^{\oplus}r =_{\mathrm{AC}} a^{\oplus}(a^{\oplus}s + s)$. Contradiction.
- $r =_{\mathrm{AC}} a^{\oplus}s$ and $s \leftrightarrow \delta$. Then induction implies $s =_{\mathrm{AC}} \delta$, so we can apply Rule 6 to $a^{\oplus}r =_{\mathrm{AC}} a^{\oplus}(a^{\oplus}\delta)$. Again, contradiction.

Hence, we may conclude that $p$ and $q$ contain exactly the same summands. Rule 1 ensures that both $p$ and $q$ contain each summand only once, so $p =_{\mathrm{AC}} q$. □

**Corollary 2.3** *The axiomatization* A1,2,3,6 + MI1,3 *for* $\mathrm{MPA}^*_\delta$ *is complete with respect to bisimulation equivalence.*

**Proof.** If two terms in $\mathrm{MPA}^{\oplus}_\delta$ are bisimilar, then according to Theorem 2.2 their normal forms are of the same form. Since all the rewrite rules can be deduced from A1,2,3,6 + PMI1,3, it follows that this is a complete axiom system for $\mathrm{MPA}^{\oplus}_\delta$. Then A1,2,3,6 + MI1,3 is a complete axiomatization for $\mathrm{MPA}^*_\delta$. □

# References

[1] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[2] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[3] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics*, pages 243–320. Elsevier, 1990.

[4] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[5] S.C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[6] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume I, Background: Computational Structures*, pages 1–116. Oxford University Press, 1992.

[7] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970. Reprinted in *Automation of Reasoning 2*, pages 342–376. Springer-Verlag, 1983.

[8] R. Milner. *A Calculus of Communicating Systems. LNCS 92.* Springer-Verlag, 1980.

[9] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[10] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *5th GI Conference*, *LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[11] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

[12] P. Sewell. Bisimulation is not finitely (first order) equationally axiomatisable. In *Proceedings 9th IEEE Symposium on Logic in Computer Science (LICS'94),* Paris, pages 62–70. IEEE Computer Society Press, 1994.

# 3

# Basic Process Algebra with Iteration: Completeness of its Equational Axioms

Wan Fokkink & Hans Zantema

Bergstra, Bethke and Ponse proposed an axiomatization for Basic Process Algebra extended with (binary) iteration. In this chapter, we prove that this axiomatization is complete with respect to strong bisimulation equivalence. To obtain this result, we set up a term rewriting system, based on the axioms, and prove that this term rewriting system is terminating, and that bisimilar normal forms are equal modulo AC.

## 3.1 Introduction

Kleene [7] defined a binary operator $x^*y$ in the context of finite automata, which denotes the iterate of $x$ and $y$. Intuitively, the expression $x^*y$ can choose to execute either $x$, after which it evolves into $x^*y$ again, or $y$, after which it terminates. Kleene formulated some algebraic laws for this operator, notably (in our notation) $x^*y = x \cdot x^*y + y$. Copi, Elgot and Wright [6] proposed a simplification of Kleene's setting, e.g. they defined a unary version of the Kleene star in the presence of an empty word. The unary Kleene star has been studied extensively ever since.

Redko [12] (see also [5]) proved for the unary Kleene star that a complete finite axiomatization for language equality does not exist. Salomaa [13] presented a complete finite axiomatization which incorporates one conditional axiom, namely (in our notation) $x = y \cdot x + z$ implies $x = y^*z$ if $y$ does not incorporate the empty word. According to Kozen [8] this last property is not algebraic, in the sense that it is not preserved under substitution of terms for actions. He proposed two alternative conditional axioms which do not have this drawback. These axioms however are not sound in the setting of (strong) bisimulation equivalence.[1]

[1]For example, one of Kozen's axioms is $x + y \cdot x + z = x \Longrightarrow x + y^*z = x$, which induces $(a + b)^*c + a^*c = (a + b)^*c$.

Milner [9] studied the Kleene star in the setting of bisimulation equivalence, and raised the question whether there exists a complete axiomatization for it. Bergstra, Bethke and Ponse [3] incorporated the binary Kleene star in Basic Process Algebra (BPA). They suggested three axioms BKS1-3 for BPA$^*$, where axiom BKS1 is the defining axiom from Kleene, while their most advanced axiom BKS3 originates from Troeger [15]:

$$x^*(y \cdot (x+y)^* z + z) \;=\; (x+y)^* z$$

In this chapter we prove that BKS1-3, together with the five standard axioms for BPA, form a complete axiomatization for BPA$^*$ with respect to bisimulation equivalence. For this purpose, we will replace iteration by proper iteration $x^{\oplus}y$. This construct executes $x$ at least one time, or in other words, $x^{\oplus}y$ is equivalent to $x \cdot x^* y$. The axioms BKS1-3 are adapted to this new setting, and we will define a term rewriting system based on the axioms of BPA$^{\oplus}$. Deducing termination of this TRS is a key step in the completeness proof; we will apply the strategy of semantic labelling from one of the authors [17]. Finally, we will show that bisimilar normal forms are syntactically equal modulo AC. These results together imply that the axiomatization for BPA$^*$ is complete with respect to bisimulation equivalence. Moreover, the applied method yields an efficient algorithm to decide whether or not two terms are bisimilar.

Sewell [14] proved that if the deadlock $\delta$ is added to BPA$^*$, then a complete finite equational axiomatization does not exist. In Chapter 2 it is shown that if sequential composition and iteration are replaced by their prefix counterparts, then six simple equational axioms are complete for this algebra.

**Acknowledgements.** Jan Bergstra is thanked for his enthusiastic support, and Jos van Wamel for many stimulating discussions.

## 3.2 BPA with Binary Kleene Star

This section introduces the basic notions. We assume an alphabet $A$ of atomic actions, together with three binary operators: alternative composition $+$, sequential composition $\cdot$, and binary Kleene star $^*$. Table 3.1 presents an operational semantics for BPA$^*$ in Plotkin style [11]. The special symbol $\surd$ represents (successful) termination.

Our model for BPA$^*$ consists of all the closed terms that can be constructed from the atomic actions and the three binary operators. That is, the BNF grammar for the collection of process terms is as follows, where $a \in A$:

$$p \;::=\; a \mid p+p \mid p \cdot p \mid p^* p.$$

In the sequel the operator $\cdot$ will often be omitted, so $pq$ denotes $p \cdot q$. As binding convention, $^*$ binds stronger than $\cdot$, which in turn binds stronger than $+$.

Process terms are considered modulo (strong) bisimulation equivalence [10]. Intuitively, two process terms are bisimilar if they have the same branching structure.

$$a \xrightarrow{a} \surd$$

$$\frac{x \xrightarrow{a} \surd}{x + y \xrightarrow{a} \surd \xleftarrow{a} y + x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x' \xleftarrow{a} y + x}$$

$$\frac{x \xrightarrow{a} \surd}{x \cdot y \xrightarrow{a} y} \qquad \frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}$$

$$\frac{x \xrightarrow{a} \surd}{x^* y \xrightarrow{a} x^* y} \qquad \frac{x \xrightarrow{a} x'}{x^* y \xrightarrow{a} x' \cdot x^* y}$$

$$\frac{y \xrightarrow{a} \surd}{x^* y \xrightarrow{a} \surd} \qquad \frac{y \xrightarrow{a} y'}{x^* y \xrightarrow{a} y'}$$

Table 3.1: Action rules for BPA$^*$

**Definition 3.1** *Two processes $p_0$ and $q_0$ are called* bisimilar*, denoted by $p_0 \leftrightarrows q_0$, if there exists a symmetric relation $\mathcal{B}$ on processes such that:*

- *$p_0 \mathcal{B} q_0$,*
- *if $p \mathcal{B} q$ and $p \xrightarrow{a} p'$, then there is a transition $q \xrightarrow{a} q'$ such that $p' \mathcal{B} q'$,*
- *if $p \mathcal{B} q$ and $p \xrightarrow{a} \surd$, then $q \xrightarrow{a} \surd$.*

The action rules in Table 3.1 are in the *path* format of Baeten and Verhoef [2]. Hence, bisimulation equivalence is a congruence with respect to all the operators, which means that if $p \leftrightarrows p'$ and $q \leftrightarrows q'$, then $p + q \leftrightarrows p' + q'$ and $pq \leftrightarrows p'q'$ and $p^*q \leftrightarrows p'^*q'$. See [2] for the definition of the path format, and for a proof of this congruence result. (This proof uses the extra assumption that the rules are *well-founded*. In Chapter 4 it is shown that this requirement can be dropped.)

Furthermore, the action rules for BPA are *pure*, and the two rules for iteration incorporate the Kleene star in the left-hand side of their conclusions. Hence, BPA$^*$ is an operationally conservative extension of BPA, i.e. the action rules for iteration do not influence the transition systems of BPA terms. See Verhoef [16] for the definitions, and for a proof of this conservativity result.

Table 3.2 contains an axiom system for BPA$^*$. It consists of the standard axioms A1-5 for BPA, together with three axioms BKS1-3 for iteration. In the sequel, $p = q$ will mean that the equality can be derived from these axioms.

The axiomatization for BPA$^*$ is sound with respect to bisimulation equivalence, i.e. if $p = q$ then $p \leftrightarrows q$. Since bisimulation equivalence is a congruence, this can be verified by checking soundness for each axiom separately, which is left to the reader. The purpose of this chapter is to prove that the axiomatization is complete with respect to bisimulation, i.e. if $p \leftrightarrows q$ then $p = q$.

| | | | |
|---|---|---|---|
| A1 | $x+y$ | $=$ | $y+x$ |
| A2 | $(x+y)+z$ | $=$ | $x+(y+z)$ |
| A3 | $x+x$ | $=$ | $x$ |
| A4 | $(x+y)z$ | $=$ | $xz+yz$ |
| A5 | $(xy)z$ | $=$ | $x(yz)$ |
| BKS1 | $x\cdot x^*y+y$ | $=$ | $x^*y$ |
| BKS2 | $x^*y\cdot z$ | $=$ | $x^*(yz)$ |
| BKS3 | $x^*(y\cdot(x+y)^*z+z)$ | $=$ | $(x+y)^*z$ |

Table 3.2: Axioms for BPA$^*$

## 3.3 A Conditional Term Rewriting System

Our aim is to define a Term Rewriting System (TRS) for process terms in BPA$^*$ that reduces each term to a unique normal form, such that if two terms are bisimilar, then they have the same normal form. However, we shall see that one cannot hope to find such a TRS for iteration. Therefore, we will replace it by a new, equivalent operator $p^{\oplus}q$, representing the behaviour of $p\cdot p^*q$, and we will develop a TRS for the algebra BPA$^{\oplus}$.

We want our TRS to be terminating, so we cannot add the axioms A1,2 as rewrite rules. Therefore, process terms are considered modulo AC, that is, modulo associativity and commutativity of the $+$.

### 3.3.1 Turning round two rules for BPA

The axiom A3 yields the expected rewrite rule

$$\mathbf{x}+\mathbf{x} \longrightarrow \mathbf{x}.$$

Usually, in BPA, the axiom A4 as a rewrite rule aims from left to right. However, in BPA$^*$ we need this rewrite rule in the opposite direction. For example, in order to reduce the term $a\cdot(a+b)^*c+b\cdot(a+b)^*c+c$ to the term $(a+b)^*c$, we need the reduction

$$a\cdot(a+b)^*c+b\cdot(a+b)^*c \longrightarrow (a+b)\cdot(a+b)^*c.$$

Hence, we define the rewrite rule for A4 the other way round.

$$\mathbf{xz}+\mathbf{yz} \longrightarrow (\mathbf{x}+\mathbf{y})\mathbf{z}.$$

In BPA the axiom A5 aims from left to right too, but since we have reversed A4, we must do the same for A5, otherwise the TRS would not be confluent. For example, the term $(ab)d+(ac)d$ would have two different normal forms:

$$a(bd)+a(cd) \quad \text{and} \quad (ab+ac)d.$$

So we opt for the rule

$$\mathbf{x(yz)} \longrightarrow \mathbf{(xy)z}.$$

### 3.3.2 Proper iteration

Although we have already defined part of a TRS that should reduce terms that are bisimilar to the same normal form, we shall see now that such a TRS does not exist at all.

Since $x^*y+z \leftrightarrows x^*y$ if $y+z \leftrightarrows y$, such terms should have the same normal form. Therefore, one would expect a rule

$$x^*y + z \longrightarrow x^*y \qquad \text{if } y + z \longrightarrow\!\!\!\!\!\rightarrow y.$$

However, this rule does not yield unique normal forms, because we have reversed the rule for A4. For example, the term $a^*(b+ce)+ce+de$ would have two different normal forms:

$$a^*(b + ce) + de \quad \text{and} \quad a^*(b + ce) + (c + d)e.$$

To avoid this complication, we replace iteration by an operator $x^{\oplus}y$, called *proper* iteration, which displays the behaviour of $x \cdot x^*y$.[2] The operational semantics and the axiomatization for proper iteration are given in Tables 3.3 and 3.4. They are obtained from the action rules and axioms for iteration, using the equivalences $x^*y \leftrightarrows x^{\oplus}y + y$ and $x^{\oplus}y \leftrightarrows x \cdot x^*y$. Note that

$$\begin{array}{lcll} \mathrm{BPA}^* & + & (x^{\oplus}y = x \cdot x^*y) & \vdash \text{ PI1-3,} \\ \mathrm{BPA}^{\oplus} & + & (x^*y = x^{\oplus}y + y) & \vdash \text{ BKS1-3.} \end{array}$$

So we find that the axiomatization in Table 3.4 is complete for $\mathrm{BPA}^{\oplus}$ if and only if the axiomatization in Table 3.2 is complete for $\mathrm{BPA}^*$.

$$\frac{x \xrightarrow{a} x'}{x^{\oplus}y \xrightarrow{a} x'(x^{\oplus}y + y)} \qquad \frac{x \xrightarrow{a} \surd}{x^{\oplus}y \xrightarrow{a} x^{\oplus}y + y}$$

Table 3.3: Action rules for proper iteration

[2] The standard notation for this construct would be $x^+y$, but we want to avoid ambiguous use of the $+$.

| | | | |
|---|---|---|---|
| PI1 | $x(x^{\oplus}y+y)$ | $=$ | $x^{\oplus}y$ |
| PI2 | $(x^{\oplus}y)z$ | $=$ | $x^{\oplus}(yz)$ |
| PI3 | $x^{\oplus}(y((x+y)^{\oplus}z+z)+z)$ | $=$ | $x((x+y)^{\oplus}z+z)$ |

Table 3.4: Axioms for proper iteration

### 3.3.3 One rule for axiom PI2

Now that we have replaced iteration by proper iteration, we can continue to define rewrite rules for this new operator. We start with the one for axiom PI2. The question is whether it should rewrite from left to right or vice versa. If it would rewrite from left to right, it would clash with the rule for A4. For example, then the term $a^{\oplus}b \cdot c + dc$ would have two different normal forms:

$$a^{\oplus}(bc) + dc \quad \text{and} \quad (a^{\oplus}b + d)c.$$

Hence, PI2 yields the rule

$$\mathbf{x}^{\oplus}(\mathbf{yz}) \longrightarrow (\mathbf{x}^{\oplus}\mathbf{y})\mathbf{z}.$$

### 3.3.4 Four rules for axiom PI1

The next rule stems from axiom PI1:

$$\mathbf{x}(\mathbf{x}^{\oplus}\mathbf{y}+\mathbf{y}) \longrightarrow \mathbf{x}^{\oplus}\mathbf{y}.$$

This rewrite rule causes serious complications concerning confluence; it turns out that we need three extra rules to obtain this property.

1. A term $x(y^{\oplus}z+z)+y(y^{\oplus}z+z)$ has two different reductions:

   $$x(y^{\oplus}z+z)+y^{\oplus}z \quad \text{and} \quad (x+y)(y^{\oplus}z+z).$$

   So for the sake of confluence, one of these two reducts should reduce to the other. If we would add the rule $(x+y)(y^{\oplus}z+z) \longrightarrow x(y^{\oplus}z+z)+y^{\oplus}z$ to the TRS, then the term $(ac+bc)((bc)^{\oplus}d+d)$ would have two different normal forms:

   $$(ac)((bc)^{\oplus}d+d)+(bc)^{\oplus}d \quad \text{and} \quad ((a+b)c)((bc)^{\oplus}d+d).$$

   Hence, we opt for the rule

   $$\mathbf{x}(\mathbf{y}^{\oplus}\mathbf{z}+\mathbf{z})+\mathbf{y}^{\oplus}\mathbf{z} \longrightarrow (\mathbf{x}+\mathbf{y})(\mathbf{y}^{\oplus}\mathbf{z}+\mathbf{z}).$$

2. A term $x(y(y^{\oplus}z + z))$ has two different reductions:

   $$x(y^{\oplus}z) \quad \text{and} \quad (xy)(y^{\oplus}z + z).$$

   A rule $(xy)(y^{\oplus}z + z) \longrightarrow x(y^{\oplus}z)$ clashes with the rule for A5, because then the term $(a(bc))((bc)^{\oplus}d + d))$ would get two different normal forms:

   $$a((bc)^{\oplus}d) \quad \text{and} \quad ((ab)c)((bc)^{\oplus}d + d)).$$

   Therefore, we define

   $$\mathbf{x}(\mathbf{y}^{\oplus}\mathbf{z}) \longrightarrow (\mathbf{xy})(\mathbf{y}^{\oplus}\mathbf{z} + \mathbf{z}).$$

3. Finally, a term $x^{\oplus}(y(y^{\oplus}z + z))$ has two different reductions:

   $$x^{\oplus}(y^{\oplus}z) \quad \text{and} \quad (x^{\oplus}y)(y^{\oplus}z + z).$$

   Since a rule $(x^{\oplus}y)(y^{\oplus}z + z) \longrightarrow x^{\oplus}(y^{\oplus}z)$ would clash with the rule for PI2, we opt for

   $$\mathbf{x}^{\oplus}(\mathbf{y}^{\oplus}\mathbf{z}) \longrightarrow (\mathbf{x}^{\oplus}\mathbf{y})(\mathbf{y}^{\oplus}\mathbf{z} + \mathbf{z}).$$

### 3.3.5 Two conditional rules for axiom PI3

The obvious interpretation of axiom PI3 as a rewrite rule,

$$x^{\oplus}(x'((x + x')^{\oplus}z + z) + z) \longrightarrow x((x + x')^{\oplus}z + z),$$

obstructs confluence. For if $x$ and $x'$ are normal forms, while the expression $x + x'$ is not, then after reducing $x + x'$ we can no longer apply this rule. Therefore, we translate PI3 to a conditional rule:

$$\mathbf{x}^{\oplus}(\mathbf{x}'(\mathbf{y}^{\oplus}\mathbf{z} + \mathbf{z}) + \mathbf{z}) \longrightarrow \mathbf{x}(\mathbf{y}^{\oplus}\mathbf{z} + \mathbf{z}) \quad \textbf{if } \mathbf{x} + \mathbf{x}' \longrightarrow\!\!\!\!\rightarrow \mathbf{y}.$$

Again, this rule leads to a TRS that is not confluent, because a term $x^{\oplus}(y(y^{\oplus}z+z)+z)$ with $x + y \longrightarrow\!\!\!\!\rightarrow y$ has two reductions:

$$x^{\oplus}(y^{\oplus}z + z) \quad \text{and} \quad x(y^{\oplus}z + z).$$

So in order to obtain confluence, we add one last conditional rule to the TRS:

$$\mathbf{x}^{\oplus}(\mathbf{y}^{\oplus}\mathbf{z} + \mathbf{z}) \longrightarrow \mathbf{x}(\mathbf{y}^{\oplus}\mathbf{z} + \mathbf{z}) \quad \textbf{if } \mathbf{x} + \mathbf{y} \longrightarrow\!\!\!\!\rightarrow \mathbf{y}.$$

| | | | |
|---|---|---|---|
| 1. | $x+x$ | $\longrightarrow$ | $x$ |
| 2. | $xz+yz$ | $\longrightarrow$ | $(x+y)z$ |
| 3. | $x(yz)$ | $\longrightarrow$ | $(xy)z$ |
| 4. | $x^{\oplus}(yz)$ | $\longrightarrow$ | $(x^{\oplus}y)z$ |
| 5. | $x(x^{\oplus}y+y)$ | $\longrightarrow$ | $x^{\oplus}y$ |
| 6. | $x(y^{\oplus}z+z)+y^{\oplus}z$ | $\longrightarrow$ | $(x+y)(y^{\oplus}z+z)$ |
| 7. | $x(y^{\oplus}z)$ | $\longrightarrow$ | $(xy)(y^{\oplus}z+z)$ |
| 8. | $x^{\oplus}(y^{\oplus}z)$ | $\longrightarrow$ | $(x^{\oplus}y)(y^{\oplus}z+z)$ |
| 9. | $x^{\oplus}(x'(y^{\oplus}z+z)+z)$ | $\longrightarrow$ | $x(y^{\oplus}z+z)$ if $x+x' \longrightarrow\!\!\!\!\!\rightarrow y$ |
| 10. | $x^{\oplus}(y^{\oplus}z+z)$ | $\longrightarrow$ | $x(y^{\oplus}z+z)$ if $x+y \longrightarrow\!\!\!\!\!\rightarrow y$ |

Table 3.5: Rewrite rules for $\mathrm{BPA}^{\oplus}$

### 3.3.6 The entire TRS

The entire TRS is given once again in Table 3.5. The rules are to be interpreted modulo AC. It is easy to see that all rules can be deduced from $\mathrm{BPA}^{\oplus}$.

The usual strategy for deducing that each term has a unique normal form, is to prove that the TRS is both *weakly confluent*, (i.e. if a term $p$ has reductions $p' \longleftarrow p \longrightarrow p''$, then there exists a $q$ such that $p' \longrightarrow\!\!\!\!\!\rightarrow q \longleftarrow\!\!\!\!\!\leftarrow p''$), and *terminating* (i.e. there are no infinite reductions). Newman's Lemma says that such a TRS reduces each term to a unique normal form, which does not reduce any further.

Although our choice of rewrite rules has been motivated by the wish for a weakly confluent TRS, it is not so easy to deduce this property yet, due to the presence of conditional rules. The next example shows that the usual method for checking weak confluence of a TRS, namely verifying this property for all overlapping redexes, does not work in a conditional setting.

**Example 3.2** *Consider the TRS which consists of the rules*

$$\begin{array}{rcll} f(x) & \longrightarrow & b & \text{if } x \longrightarrow\!\!\!\!\!\rightarrow a, \\ a & \longrightarrow & c. & \end{array}$$

*There are no overlapping redexes, but this TRS is not weakly confluent:* $f(c) \longleftarrow f(a) \longrightarrow b$.

However, it will turn out that the confluence property is not needed in the proof of the main theorem, which states that bisimilar normal forms are equal modulo AC. Hence, confluence will simply be a consequence of this theorem.

### 3.3.7 Termination

Proving termination of the TRS in Table 3.5, modulo AC, is a complicated matter. This is mainly due to the presence of Rule 7, in which the left-hand side can be obtained from the right-hand side by the removal of subterms. A powerful technique for proving termination of TRSs that incorporate such rules is *semantic labelling* [17], where operation symbols that occur in the rewrite rules are supplied with labels, which depend on the semantics of the arguments. Then two TRSs are involved: the original system and the labelled system. The main theorem of [17] states that the labelled system terminates if and only if the original system terminates.

The theory of semantic labelling has been developed for unconditional TRSs. Therefore, we adapt the TRS in Table 3.5 to an unconditional TRS $R$, simply by removing the conditions from the last two rules. We shall prove that $R$ is terminating, which immediately implies termination of the conditional TRS in Table 3.5.

**Proposition 3.3** *The TRS $R$ is terminating.*

**Proof.** The method from [17] starts with choosing a model, which consists of a set $\mathcal{M}$, and for each function symbol $f$ in the original signature with arity $n$ a mapping $f_{\mathcal{M}} : \mathcal{M}^n \to \mathcal{M}$, such that for every rewrite rule, and for all possible values for its variables in the model, the left-hand side and the right-hand side are equal in the model. Here we choose the model to be the positive natural numbers. Each process $p$ is interpreted by its *norm* $|p|$, being the least number of steps in which it can terminate. This norm can be defined inductively as follows:

$$\begin{aligned} |a| &= 1 \\ |p + q| &= \min\{|p|, |q|\} \\ |pq| &= |p| + |q| \\ |p^{\oplus}q| &= |p| + |q|. \end{aligned}$$

Note that norm is associative and commutative with respect to the choice operator, which is essential in order to obtain the termination result modulo AC. Clearly norm is preserved under bisimulation equivalence. Since the Rules 1-8 of $R$ are sound with respect to bisimulation, it follows that norm is preserved under application of these rewrite rules. And it is easy to verify that Rules 9 and 10 of $R$, which are not sound because they lack their original conditions, preserve norm too.

Next, we select labels for the function symbols. As labels for the operators sequential composition and proper iteration we choose the positive natural numbers, while the atoms and the choice operator remain unchanged. In each ground term, the occurrences of sequential composition and proper iteration are labelled as follows: we replace $p \cdot q$ by $p\langle|q|\rangle q$ and $p^{\oplus}q$ by $p[|q|]q$.

Finally, for each rule in the TRS we construct a collection of labelled rules. This is done by replacing the variables in the original rule by all possible values in the model, and computing the resulting labels for the operators. This results in the

following TRS $\bar{R}$, where the rules are defined for positive natural numbers $i$ and $j$.

$$\begin{array}{rcl}
x+x & \longrightarrow & x \\
x\langle i\rangle z + y\langle i\rangle z & \longrightarrow & (x+y)\langle i\rangle z \\
x\langle i+j\rangle(y\langle j\rangle z) & \longrightarrow & (x\langle i\rangle y)\langle j\rangle z \\
 & & \\
x[i+j](y\langle j\rangle z) & \longrightarrow & (x[i]y)\langle j\rangle z \\
 & & \\
x\langle i\rangle(x[i]y+y) & \longrightarrow & x[i]y \\
x\langle i\rangle(y[i]z+z)+y[i]z & \longrightarrow & (x+y)\langle i\rangle(y[i]z+z) \\
x\langle i+j\rangle(y[j]z) & \longrightarrow & (x\langle i\rangle y)\langle j\rangle(y[j]z+z) \\
x[i+j](y[j]z) & \longrightarrow & (x[i]y)\langle j\rangle(y[j]z+z) \\
 & & \\
x[i](x'\langle i\rangle(y[i]z+z)+z) & \longrightarrow & x\langle i\rangle(y[i]z+z) \\
x[i](y[i]z+z) & \longrightarrow & x\langle i\rangle(y[i]z+z)
\end{array}$$

Suppose that $R$ admits an infinite reduction. Replace the variables in this reduction by a constant $a$ to obtain an infinite ground reduction in $R$. For each symbol '$\cdot$' and '$\oplus$' that occurs in this reduction, compute its corresponding label. This way the infinite ground reduction in $R$ transforms into an infinite ground reduction in $\bar{R}$. Hence, termination of $\bar{R}$ implies termination of $R$.

It remains to prove termination of $\bar{R}$. Although $\bar{R}$ is a TRS with infinitely many rules, this is much easier than proving termination of $R$. Define a weight function $w$:

$$\begin{array}{rcl}
w(a) & = & 1 \\
w(p+q) & = & w(p)+w(q) \\
w(p\langle i\rangle q) & = & w(p)+iw(q) \\
w(p[i]q) & = & w(p)+(i+1)w(q)
\end{array}$$

It is easy to verify that for any choice of values for variables in any rule, the weight of the left-hand side is strictly greater than the weight of the right-hand side. For example, in the case of Rule 7 these weights are

$$\begin{array}{rcl}
 & w(x) + (i+j)w(y) + (i+j)(j+1)w(z) \\
\text{and} & w(x) + (i+j)w(y) + j(j+2)w(z)
\end{array}$$

respectively. And $(i+j)(j+1) > j(j+2)$ for $i, j \geq 1$.

Due to the strict monotonic behaviour of $w$ (here it is essential that $i > 0$) we conclude that each reduction step yields a strict decrease of weight. Hence the system $\bar{R}$ is terminating, and so $R$ is terminating. □

## 3.4 Normal Forms Decide Bisimilarity

In the previous section we have developed a TRS for BPA$^{\oplus}$ that reduces terms to a normal form. Since all rewrite rules are sound with respect to bisimulation equivalence, it follows that each term is bisimilar with its normal forms. So in order to

determine completeness of the axiomatization for $\mathrm{BPA}^{\oplus}$ with respect to bisimulation equivalence, it is sufficient to prove that if two normal forms are bisimilar, then they are equal modulo AC.

### 3.4.1 An ordering on process terms

As induction base in the proof of our main theorem, we will need a well-founded ordering on process terms that should preferably have the following properties:

1. $p \leq p+q \qquad p < pq \qquad p < p^{\oplus}q$
   $q \leq p+q \qquad q < pq \qquad q < p^{\oplus}q.$

2. The ordering is preserved under bisimulation.

However, an ordering combining these properties is never well-founded, because for such an ordering we have

$$p^{\oplus}q \leq p^{\oplus}q + q < p(p^{\oplus}q + q)$$

Since $p(p^{\oplus}q+q) \leftrightarrows p^{\oplus}q$, it follows that $p^{\oplus}q < p^{\oplus}q$.

The norm, indicating the least number of steps a process must make before it can terminate, induces an ordering that *almost* satisfies all desired properties. The only serious drawback of this ordering is that $|p| \geq |p+q|$. Therefore we adapt it to an ordering induced by $L$-value, which is defined as follows:

$$L(p) = \max\{|p'| \mid p' \text{ is a proper substate of } p\}$$

where 'proper substate' means that $p$ can evolve into $p'$ by one or more transitions. Since norm is preserved under bisimulation equivalence, the same holds for $L$.

**Lemma 3.4** *If $p \leftrightarrows q$, then $L(p) = L(q)$.*

**Proof.** If $p'$ is a proper substate of $p$, then bisimilarity of $p$ and $q$ implies that there is a proper substate $q'$ of $q$ such that $p' \leftrightarrows q'$, and so $|p'| = |q'|$. Hence, $L(p) \leq L(q)$, and by symmetry $L(q) \leq L(p)$. □

We deduce the inductive definition for $L$-value. $L(p+q)$ is the maximum of the collection

$$\begin{array}{ll} & \{|p'| \mid p' \text{ proper substate of } p\} \\ \cup & \{|q'| \mid q' \text{ proper substate of } q\}, \end{array}$$

so $L(p+q) = \max\{L(p), L(q)\}$. Next, $L(pq)$ is the maximum of the collection

$$\begin{array}{lll} & \{|p'q| \mid p' \text{ proper substate of } p\} & \\ \cup & \{|q|\} \quad \cup & \{|q'| \mid q' \text{ proper substate of } q\}, \end{array}$$

so $L(pq) = \max\{L(p)+|q|, L(q)\}$. Finally, $L(p^{\oplus}q)$ is the maximum of the collection

$$\begin{array}{lll} & \{|p'(p^{\oplus}q+q)| \mid p' \text{ proper substate of } p\} & \\ \cup & \{|p^{\oplus}q+q|\} \quad \cup & \{|q'| \mid q' \text{ proper substate of } q\}, \end{array}$$

so $|p^{\oplus}q| = \max\{L(p) + |q|, L(q)\}$. Recapitulating, we have found:

$$\begin{aligned} L(a) &= 0 \\ L(p+q) &= \max\{L(p), L(q)\} \\ L(pq) &= \max\{L(p) + |q|, L(q)\} \\ L(p^{\oplus}q) &= \max\{L(p) + |q|, L(q)\}. \end{aligned}$$

Hence, $L$-value too satisfies almost all the requirements formulated above; only, we have inequalities $L(q) \leq L(pq)$ and $L(q) \leq L(p^{\oplus}q)$, instead of the desired strict inequalities. Therefore, we introduce a second weight function $g$ on process terms, defined by:

$$\begin{aligned} g(a) &= 0 \\ g(p+q) &= \max\{g(p), g(q)\} \\ g(pq) &= g(q) + 1 \\ g(p^{\oplus}q) &= g(q) + 1. \end{aligned}$$

Note that $g$-value is not preserved under bisimulation equivalence. However, the following lemma holds.

**Lemma 3.5** *If $p \longrightarrow q$, then $g(p) \geq g(q)$.*

**Proof.** For each rewrite rule it is easily checked that the $g$-value of the left-hand side is greater than or equal than the $g$-value of the right-hand side. Since the functions that are used in the definition of $g$ are weakly monotonous in their coordinates, we may conclude that $g$-value is never increased by a rewrite step. □

In the proof of the main theorem we will apply induction on a lexicographical combination of $L$-value and $g$-value.

### 3.4.2 Some lemmas

We deduce three lemmas that will be used in the proof of the main theorem. The first lemma is typical for normed processes [1], i.e. for processes that are able to terminate in finitely many transitions. This lemma originates from Caucal [4].

**Lemma 3.6** *If $pr \leftrightarrow qr$, then $p \leftrightarrow q$.*

**Proof.** A transition $p'r \xrightarrow{a} p''r$ in $pr$ cannot be mimicked by a transition $q'r \xrightarrow{a} r$ in $qr$, because $|p''r| > |r|$. Hence, each transition $p'r \xrightarrow{a} p''r$ is mimicked by a transition $q'r \xrightarrow{a} q''r$, and vice versa. This induces a bisimulation relation between $p$ and $q$; the transition $p' \xrightarrow{a} p''$ in $p$ is mimicked by the transition $q' \xrightarrow{a} q''$ in $q$, and vice versa. □

**Definition 3.7** *We say that two process terms $p$ and $q$ have* behaviour in common *if there are $p'$ and $q'$ such that $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ and $p' \leftrightarrow q'$.*

**Lemma 3.8** *If two terms $pq$ and $rs$ have behaviour in common, and $|q| \geq |s|$, then either $q \leftrightarrow ts$ for some $t$, or $q \leftrightarrow s$.*

**Proof.** If $pq \xrightarrow{a} q$ and $rs \xrightarrow{a} r's$ with $q \leftrightarrows r's$, or if $pq \xrightarrow{a} q$ and $rs \xrightarrow{a} s$ with $q \leftrightarrows s$, then we are done. And $pq \xrightarrow{a} p'q$ and $rs \xrightarrow{a} s$ with $p'q \leftrightarrows s$ would contradict $|q| \geq |s|$. Thus, the only interesting case is if $pq \xrightarrow{a} p'q$ and $rs \xrightarrow{a} r's$ with $p'q \leftrightarrows r's$. The inequality $|q| \geq |s|$ then yields $|p'| \leq |r'|$.

We show, with induction on $|p'|$, that $p'q \leftrightarrows r's$ together with $|p'| \leq |r'|$ indicate either $q \leftrightarrows ts$ for some $t$ or $q \leftrightarrows s$. If $|p'| = 1$, then $p' \xrightarrow{a} \surd$, and so $p'q \xrightarrow{a} q$. Since $p'q \leftrightarrows r's$, this transition can be mimicked by a transition $r's \xrightarrow{a} r''s$ or $r's \xrightarrow{a} s$, and so $q \leftrightarrows r''s$ or $q \leftrightarrows s$ respectively.

Next, let $|p'| = n+1$. Clearly, there is a transition $p' \xrightarrow{a} p''$ with $|p''| = n$. Since $p'q \leftrightarrows r's$, and $p'q \xrightarrow{a} p''q$, there must be a transition $r's \xrightarrow{a} r''s$ with $p''q \leftrightarrows r''s$. Since $|r'| \geq |p'| = n+1$ implies $|r''| \geq n = |p''|$, the induction hypothesis learns that either $q \leftrightarrows ts$ for some $t$, or $q \leftrightarrows s$. □

**Lemma 3.9** *If a term $rs$ has normal form $q$, then $pq$ or $p^{\oplus}q$ is* not *a normal form.*

**Proof.** Suppose that $q$ is a normal form of a term $rs$. Each rule in Table 3.5 that applies to a term of the form $tu$ or $t^{\oplus}u$, reduces it to one of either forms again. So $q$ must be in one of either forms. But Rules 3, 4, 7 and 8 reduce $p(tu)$ and $p^{\oplus}(tu)$ and $p(t^{\oplus}u)$ and $p^{\oplus}(t^{\oplus}u)$ respectively. Hence, $pq$ and $p^{\oplus}q$ are not in normal form. □

### 3.4.3 The main theorem

Process terms are considered modulo AC. From now on, this equivalence is denoted by $p =_{\mathrm{AC}} q$, and we say that $p$ and $q$ are of the same form. Clearly, each process term $p$ is a sum of terms of the form $a$ and $qr$ and $q^{\oplus}r$, which are called the *summands* of $p$.

**Theorem 3.10** *If two normal forms $p$ and $q$ are bisimilar, then $p =_{\mathrm{AC}} q$.*

**Proof.** In order to prove the theorem, we prove three extra statements in parallel.

*A.* If two normal forms $p =_{\mathrm{AC}} rs$ and $q =_{\mathrm{AC}} tu$ have common behaviour, then $s =_{\mathrm{AC}} u$.

*B.* If two normal forms $p =_{\mathrm{AC}} rs$ and $q =_{\mathrm{AC}} t^{\oplus}u$ have common behaviour, then $s =_{\mathrm{AC}} t^{\oplus}u + u$.

*C.* If two normal forms $p =_{\mathrm{AC}} r^{\oplus}s$ and $q =_{\mathrm{AC}} t^{\oplus}u$ have common behaviour, then $r^{\oplus}s =_{\mathrm{AC}} t^{\oplus}u$.

The statement in the main theorem is labelled *D*.

If $L(p) = L(q) = 0$, then both $p$ and $q$ must be sums of atoms. So in this case *A* and *B* and *C* are empty statements. And *D* holds too, because bisimilarity of $p$ and $q$ indicates that they contain exactly the same atoms, and Rule 1 ensures that both terms contain each of these atoms only once.

Next, fix an $m > 0$ and assume that we have already proved the four statements if $L(p)$ and $L(q)$ are smaller than $m$. We will prove it for the case that they are

equal to $m$. Let $A_n$ and $B_n$ and $C_n$ and $D_n$ denote the assertions for pairs $p, q$ with $\max\{L(p), L(q)\} \leq m$ and $g(p) + g(q) \leq n$. They are proved by induction on $n$.

The case $n = 0$ corresponds with the case $L(p) = L(q) = 0$, because if $g(p) + g(q) = 0$, then both $p$ and $q$ must be sums of atoms. As induction hypothesis we now assume $A_n, B_n, C_n$ and $D_n$, and we shall prove $A_{n+1}, B_{n+1}, C_{n+1}$ and $D_{n+1}$.

1. $A_{n+1}$ is true.

Let normal forms $rs$ and $tu$ have behaviour in common, with $L(rs) \leq m$ and $L(tu) \leq m$ and $g(rs) + g(tu) = n + 1$. We want to prove $s =_{\mathrm{AC}} u$. By symmetry we may assume $|s| \geq |u|$, so Lemma 3.8 offers two possibilities.

1.1 $s \leftrightarrow u$.

$L(s) \leq L(rs) \leq m$ and $L(u) \leq L(tu) \leq m$ and $g(s) + g(u) < g(rs) + g(tu) = n + 1$. Hence, $D_n$ yields $s =_{\mathrm{AC}} u$.

1.2 $s \leftrightarrow vu$ for some $v$.

Let $w$ be a normal form of $vu$. According to Lemma 3.5 $g(w) \leq g(vu)$, so $g(s) + g(w) < g(rs) + g(vu) = n + 1$. Further, since $s \leftrightarrow w$, $L(w) = L(s) \leq m$. Hence, $D_n$ yields $s =_{\mathrm{AC}} w$. However, Lemma 3.9 says that $s$ cannot be a normal form of a term $vu$. Contradiction.

2. $B_{n+1}$ is true.

According to the previous point we may assume $A_{n+1}$. Let normal forms $rs$ and $t^{\oplus}u$ have behaviour in common, with $L(rs) \leq m$ and $L(t^{\oplus}u) \leq m$ and $g(rs) + g(t^{\oplus}u) = n + 1$. We want to prove $s =_{\mathrm{AC}} t^{\oplus}u + u$. Since $t^{\oplus}u \leftrightarrow t(t^{\oplus}u + u)$, Lemma 3.8 offers three possibilities.

2.1 $s \leftrightarrow t^{\oplus}u + u$.

The term $t^{\oplus}u + u$ is a normal form, because we cannot apply Rules 1,2 or 6 to it. Moreover, $g(s) + g(t^{\oplus}u + u) = g(s) + g(t^{\oplus}u) = n$, so $D_n$ yields $s =_{\mathrm{AC}} t^{\oplus}u + u$.

2.2 $vs \leftrightarrow t^{\oplus}u + u$ for some $v$.

This implies $v's \leftrightarrow u$ for some $v'$. As in 1.2, we can deduce that then $u$ is a normal form of $v's$, which is a contradiction according to Lemma 3.9.

2.3 $s \leftrightarrow v(t^{\oplus}u + u)$ for some $v$.

Note that $g(s) + g(v(t^{\oplus}u + u)) = n + 1$, so we cannot yet apply $D_n$.

Let $v$ be a normal form. If $v =_{\mathrm{AC}} t$ then $s \leftrightarrow t^{\oplus}u$, so that $D_n$ yields $s =_{\mathrm{AC}} t^{\oplus}u$. Then Rule 7 reduces $rs$, which is a contradiction. So apparently $v$ cannot be of the form $t$. Thus, Rule 5 cannot be applied to $v(t^{\oplus}u + u)$, so this term is a normal form.

First, consider a summand $\alpha\beta$ of $s$. This term and $v(t^{\oplus}u + u)$ have behaviour in common, so $A_{n+1}$ yields $\beta =_{\mathrm{AC}} t^{\oplus}u + u$.

Next, consider a summand $\alpha^{\oplus}\beta$ of $s$. This term and $v(t^{\oplus}u + u)$ have behaviour in common. Since $\alpha^{\oplus}\beta \leftrightarrow \alpha(\alpha^{\oplus}\beta + \beta)$, Lemma 3.8 offers three possibilities.

- $\alpha^{\oplus}\beta + \beta \underline{\leftrightarrow} t^{\oplus}u + u$.

  $g(\alpha^{\oplus}\beta+\beta)+g(t^{\oplus}u+u) \leq g(s)+g(t^{\oplus}u) = n$, so $D_n$ implies $\alpha^{\oplus}\beta+\beta =_{\mathrm{AC}} t^{\oplus}u+u$. Since the summands of $\alpha^{\oplus}\beta + \beta$ and $t^{\oplus}u + u$ with greatest size are $\alpha^{\oplus}\beta$ and $t^{\oplus}u$ respectively, it follows that $\alpha^{\oplus}\beta =_{\mathrm{AC}} t^{\oplus}u$.

- $w(\alpha^{\oplus}\beta + \beta) \underline{\leftrightarrow} t^{\oplus}u + u$ for some $w$.

  Then $w'(\alpha^{\oplus}\beta + \beta) \underline{\leftrightarrow} u$ for some $w'$, and we obtain a contradiction as in 1.2.

- $\alpha^{\oplus}\beta + \beta \underline{\leftrightarrow} w(t^{\oplus}u + u)$ for some $w$.

  Then $\beta \underline{\leftrightarrow} w'(t^{\oplus}u + u)$ for some $w'$, and we obtain a contradiction as in 1.2.

So we may conclude $\alpha^{\oplus}\beta =_{\mathrm{AC}} t^{\oplus}u$.

If $s$ contains several summands of the form $\alpha(t^{\oplus}u + u)$ or $t^{\oplus}u$, then we can apply Rule 1,2 or 6 to $s$. However, $s$ is in normal form, so apparently it consists of a single term $\alpha(t^{\oplus}u + u)$ or $t^{\oplus}u$. Then apply Rule 3 or 7 applies to $rs$, which again is a contradiction, because $rs$ is in normal form.

3. $C_{n+1}$ is true.

Assume normal forms $r^{\oplus}s$ and $t^{\oplus}u$ that have behaviour in common, with $L(r^{\oplus}s) \leq m$ and $L(t^{\oplus}u) \leq m$ and $g(r^{\oplus}s) + g(t^{\oplus}u) = n + 1$. We want to prove $r^{\oplus}s =_{\mathrm{AC}} t^{\oplus}u$. By symmetry we may assume $|r^{\oplus}s| \geq |t^{\oplus}u|$, so Lemma 3.8 offers two possibilities.

3.1 $r^{\oplus}s + s \underline{\leftrightarrow} v(t^{\oplus}u + u)$ for some $v$.

Then $s \underline{\leftrightarrow} v'(t^{\oplus}u + u)$ for some $v'$. This leads to a contradiction as in 2.3.

3.2 $r^{\oplus}s + s \underline{\leftrightarrow} t^{\oplus}u + u$.

First, suppose that $s$ and $u$ have no behaviour in common with $t^{\oplus}u$ and $r^{\oplus}s$ respectively, so that $s \underline{\leftrightarrow} u$ and $r^{\oplus}s \underline{\leftrightarrow} t^{\oplus}u$. Since $D_n$ applies to the first equivalence, we get $s =_{\mathrm{AC}} u$. And the second equivalence yields $r(r^{\oplus}s + s) \underline{\leftrightarrow} t(t^{\oplus}u + u) \underline{\leftrightarrow} t(r^{\oplus}s + s)$, so Lemma 3.6 implies $r \underline{\leftrightarrow} t$. Since $L(r) = L(t) < m$, statement $D$ then implies $r =_{\mathrm{AC}} t$, and we are done.

So we can suppose that either $s$ and $t^{\oplus}u$ have behaviour in common, or $u$ and $r^{\oplus}s$ have behaviour in common. We deduce a contradiction.

By symmetry it is sufficient to deduce a contradiction for the first case only, where $s$ and $t^{\oplus}u$ have behaviour in common. If a summand $\alpha\beta$ or $\alpha^{\oplus}\beta$ of $s$ has behaviour in common with $t^{\oplus}u$, then $B_n$ or $C_n$ implies $\beta =_{\mathrm{AC}} t^{\oplus}u+u$ or $\alpha^{\oplus}\beta =_{\mathrm{AC}} t^{\oplus}u$ respectively. If $s$ contains several summands of the form $\alpha(t^{\oplus}u + u)$ or $t^{\oplus}u$, then Rules 1,2 or 6 can be applied to it. However, $s$ is a normal form, so apparently it contains exactly one such summand.

If $u$ and $r^{\oplus}s$ have behaviour in common too, then similarly we can deduce that $u$ has a summand of the form $\beta(r^{\oplus}s + s)$ or $r^{\oplus}s$, which indicates that $u$ has a size greater than $s$. On the other hand, $s$ has a summand $\alpha(t^{\oplus}u + u)$ or $t^{\oplus}u$, so $s$ has a size greater than $u$. This cannot be, so $u$ and $r^{\oplus}s$ have no behaviour in common.

And if $u$ has behaviour in common with the summand $\alpha(t^{\oplus}u + u)$ or $t^{\oplus}u$ of $s$, then it follows from $A_n$ or $B_n$ or $C_n$ that $u$ has a summand of the form $\beta(t^{\oplus}u + u)$ or $t^{\oplus}u$. Again we establish a contradiction; $u$ has greater size than itself.

Hence, we have found that

- $r^{\oplus}s + s \leftrightarrows t^{\oplus}u + u$,
- $s$ has a summand $\alpha(t^{\oplus}u + u)$ or $t^{\oplus}u$, and all other summands of $s$ have no behaviour in common with $t^{\oplus}u$,
- $u$ has no behaviour in common with $r^{\oplus}s$, nor with the summand $\alpha(t^{\oplus}u+u)$ or $t^{\oplus}u$ of $s$.

From these facts it follows that

- $s =_{\mathrm{AC}} \alpha(t^{\oplus}u + u) + s'$ or $s =_{\mathrm{AC}} t^{\oplus}u + s'$,
- $s' \leftrightarrows u$,
- $r^{\oplus}s + \alpha(t^{\oplus}u + u)$ or $r^{\oplus}s + t^{\oplus}u$ is bisimilar to $t^{\oplus}u$.

Since $s' \leftrightarrows u$, $D_n$ yields $s' =_{\mathrm{AC}} u$. We distinguish the two possible forms of $s$.

- $s =_{\mathrm{AC}} \alpha(t^{\oplus}u + u) + u$.

  Then $r \oplus s + \alpha(t \oplus u + u) \leftrightarrows t \oplus u$. Since $r \oplus s + s \leftrightarrows t \oplus u + u$, this yields $(r + \alpha)(t \oplus u + u) \leftrightarrows t(t \oplus u + u)$. Lemma 3.6 implies $r + \alpha \leftrightarrows t$, so since $L(r + \alpha) = L(t) < m$, we obtain $r + \alpha \longrightarrow\!\!\!\!\!\rightarrow t$. Then Rule 9 can be applied to $r^{\oplus}s =_{\mathrm{AC}} r^{\oplus}(\alpha(t^{\oplus}u+u)+u)$. Since $r^{\oplus}s$ is a normal form, this is a contradiction.

- $s =_{\mathrm{AC}} t^{\oplus}u + u$.

  Then $r^{\oplus}s+t^{\oplus}u \leftrightarrows t^{\oplus}u$. Since $r^{\oplus}s+s \leftrightarrows t^{\oplus}u+u$, this yields $(r+t)(t^{\oplus}u+u) \leftrightarrows t(t^{\oplus}u + u)$. Lemma 3.6 implies $r + t \leftrightarrows t$, so since $L(r + t) = L(t) < m$, we obtain $r + t \longrightarrow\!\!\!\!\!\rightarrow t$. Then Rule 10 can be applied to $r^{\oplus}s =_{\mathrm{AC}} r^{\oplus}(t^{\oplus}u + u)$, and once more we have found a contradiction.

4. $D_{n+1}$ is true.

We may assume $A_{n+1}$ and $B_{n+1}$ and $C_{n+1}$. Let $p$ and $q$ be bisimilar normal forms with $L(p) = L(q) = m$ and $g(p) + g(q) = n + 1$. We want to prove $p =_{\mathrm{AC}} q$.

First, we show that each summand of $p$ is bisimilar to a summand of $q$, and vice versa. Clearly, each atomic summand $a$ of $p$ corresponds with a summand $a$ of $q$. We show that each non-atomic summand of $p$ also corresponds to a summand of $q$.

Suppose that a summand $rs$ of $p$ has behaviour in common with two summands of $q$. If these summands are of the form $tu$ and $t'u'$, then $A_{n+1}$ implies $u =_{\mathrm{AC}} s =_{\mathrm{AC}} u'$, so that Rule 2 reduces this pair. If they are of the form $tu$ and $t'^{\oplus}u'$, then $A_{n+1}$ and $B_{n+1}$ give $u =_{\mathrm{AC}} s =_{\mathrm{AC}} t'^{\oplus}u' + u'$, so that Rule 6 reduces this pair. Finally, if they

are of the form $t^{\oplus}u$ and $t'^{\oplus}u'$, then $B_{n+1}$ implies $t^{\oplus}u + u =_{\rm AC} s =_{\rm AC} t'^{\oplus}u' + u'$. This means $t^{\oplus}u =_{\rm AC} t'^{\oplus}u'$, so Rule 1 reduces this pair.

Similarly, if a summand $r^{\oplus}s$ of $p$ has behaviour in common with two summands of $q$, we find using $B_{n+1}$ and $C_{n+1}$ that Rule 1, 2 or 6 can be applied to this pair.

So, since $q$ is a normal form, the assumption of a non-atomic summand of $p$ having behaviour in common with two summands of $q$ leads to a contradiction. By symmetry, each non-atomic summand of $q$ too can have behaviour in common with only one summand of $p$. So apparently, each non-atomic summand of $p$ is bisimilar to a non-atomic summand of $q$ and vice versa.

- Suppose that summands $rs$ and $tu$ are bisimilar. Then $A_{n+1}$ implies $s =_{\rm AC} u$, so according to Lemma 3.6 $r \leftrightarrow t$. Since $L(r) = L(t) < m$, we obtain $r =_{\rm AC} t$.

- If summands $rs$ and $t^{\oplus}u$ are bisimilar, then $B_{n+1}$ implies $s =_{\rm AC} t^{\oplus}u + u$. Then $r(t^{\oplus}u + u) =_{\rm AC} rs \leftrightarrow t^{\oplus}u \leftrightarrow t(t^{\oplus}u + u)$, so Lemma 3.6 implies $r \leftrightarrow t$. Since $L(r) = L(t) < m$, this yields $r =_{\rm AC} t$. Hence $rs =_{\rm AC} t(t^{\oplus}u + u)$, so we can apply Rule 5 to $rs$. Contradiction.

- Finally, if summands $r^{\oplus}s$ and $t^{\oplus}u$ are bisimilar, then $C_{n+1}$ says that they are of the same form.

Hence, $p$ and $q$ contain exactly the same summands. Rule 1 indicates that each of these summands occurs only once in both $p$ and $q$, so $p =_{\rm AC} q$. □

**Corollary 3.11** *The TRS in Table 3.5 is confluent.*

**Corollary 3.12** *The axioms A1-5 + BKS1-3 for BPA* are complete with respect to bisimulation equivalence.*

**Proof.** If two terms in BPA$^{\oplus}$ are bisimilar, then according to Theorem 3.10 their normal forms are of the same form. Since all the rewrite rules can be deduced from A1-5 + PI1-3, it follows that this is a complete axiom system for BPA$^{\oplus}$. Then A1-5 + BKS1-3 is a complete axiomatization for BPA$^*$. □

# References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993.

[2] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, *Proceedings CONCUR'93,* Hildesheim, *LNCS 715*, pages 477–492. Springer-Verlag, 1993.

[3] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[4] D. Caucal. Graphes canoniques et graphes algébriques. *Theoretical Informatics and Applications*, 24(4):339–352, 1990.

[5] J.H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.

[6] I.M. Copi, C.C. Elgot, and J.B. Wright. Realization of events by logical nets. *Journal of the ACM*, 5:181–196, 1958.

[7] S.C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[8] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[9] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[10] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *5th GI Conference*, *LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[11] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

[12] V.N. Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964. In Russian.

[13] A. Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the ACM*, 13(1):158–169, 1966.

[14] P. Sewell. Bisimulation is not finitely (first order) equationally axiomatisable. In *Proceedings 9th IEEE Symposium on Logic in Computer Science (LICS'94),* Paris, pages 62–70. IEEE Computer Society Press, 1994.

[15] D.R. Troeger. Step bisimulation is pomset equivalence on a parallel language without explicit internal choice. *Mathematical Structures in Computer Science*, 3:25–62, 1993.

[16] C. Verhoef. A general conservative extension theorem in process algebra. In E.-R. Olderog, editor, *Proceedings IFIP Conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*, San Miniato, *IFIP Transactions* A-56, pages 149–168. Elsevier, 1994.

[17] H. Zantema. Termination of term rewriting by semantic labelling. Report RUU-CS-92-38, Utrecht University, 1992. Revised version RUU-CS-93-24. To appear in *Fundamenta Informaticae* (special issue on term rewriting systems).

# 4

# The Tyft/Tyxt Format Reduces to Tree Rules

Wan Fokkink

Groote and Vaandrager introduced the *tyft/tyxt format* for Transition System Specifications (TSSs), and established that for each TSS in this format that is *well-founded*, the bisimulation equivalence it induces is a congruence. In this chapter, we construct for each TSS in tyft/tyxt format an equivalent TSS that consists of *tree rules* only. As a corollary we can give an affirmative answer to an open question, namely whether the well-foundedness condition in the congruence theorem for tyft/tyxt can be dropped. These results extend to tyft/tyxt with negative premises and predicates.

## 4.1 Introduction

A current method to provide process algebras and specification languages with an operational semantics is based on the use of transition systems, advocated by Plotkin [13]. Given a set of states, the transitions between these states are obtained inductively from a Transition System Specification (TSS), which consists of transition rules. Such a rule, together with a number of transitions, may imply the validity of another transition.

We will consider a specific type of transition systems, in which states are the closed terms generated by a single-sorted signature, and transitions are supplied with labels. A great deal of the operational semantics of formal languages in Plotkin style that have been defined over the years, are within the scope of this format.

To distinguish such labelled transition systems, many different equivalences have been defined, the finest of which is the strong bisimulation equivalence of Park [12]. In general, this equivalence is not a congruence, i.e. the equivalence class of a term $f(p_1, ..., p_m)$ modulo strong bisimulation is not always determined by the equivalence classes of the terms $p_i$. However, congruence is an essential property, for instance, to fit the equivalence into an axiomatic framework.

Several formats have been developed which ensure that the bisimulation equivalence induced by a TSS in such a format is always a congruence. A first proposal was made by De Simone [14], which was generalized by Bloom, Istrail and Meyer [3] to the GSOS format. Next, Groote and Vaandrager [10] introduced the tyft/tyxt format, and proved a congruence theorem for TSSs in this format that satisfy a well-foundedness criterion.

Up to now, it has been an open question whether or not well-foundedness is an essential ingredient of this congruence theorem. The requirement popped up in the proof, but no counter-example was found to show that the theorem breaks down if well-foundedness were omitted from it. In this chapter, we prove that the congruence theorem does hold for general TSSs in tyft/tyxt format, i.e. that the requirement of well-foundedness can be omitted.

In fact, we will establish a stronger result, namely that for each TSS in tyft/tyxt format, there is an equivalent TSS consisting of 'tree rules' only. A tree rule is a well-founded rule of the form

$$\frac{\{z_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_m) \xrightarrow{a} t}$$

where the $y_i$ and the $x_j$ are distinct variables and are the only variables that occur in the rule, the $z_i$ are variables, $f$ is a function symbol, and $t$ is any term. Using terminology from [10], we can say that a tree rule is a pure and well-founded xyft rule. Since tree rules are well-founded, the reduction of tyft/tyxt format to tree format immediately implies that the congruence theorem concerning the tyft/tyxt format can do without well-foundedness.

The major advantage of the main theorem is that it facilitates reasoning about the tyft/tyxt format. Because often it is much easier to prove a theorem for TSSs in tree format than for TSSs in tyft/tyxt format. For example, this is the case with the congruence theorem itself. Another striking example consists of Theorems 8.6.6 and 8.9.1 in [10]. With our result at hand, the complicated proof of the second theorem can be skipped, because now the second theorem follows from the first one.

Furthermore, the removal of well-foundedness from the congruence theorem for tyft/tyxt increases the convenience of applying this theorem, since the user no longer has to recall and check the complicated well-foundedness criterion.

Rob van Glabbeek independently proved the same result, which he announced in [7]. His proof is along the same lines as the one presented in this chapter.

The proof of the main theorem makes heavy use of a standard result from unification theory, which says that for each set of equations that is unifiable, there exists an idempotent most general unifier. In unification theory, this result is proved for finite sets of equations, and for substitutions that have a finite domain. However, we will need the result in a setting which does not satisfy these finiteness constraints. See Chapter 5 for a proof of the unification result in the infinite case.

Groote [9] added negative premises to tyft/tyxt, resulting in the ntyft/ntyxt format, and proved that the congruence theorem extends to well-founded TSSs in

ntyft/ntyxt format. We will show that the reduction of tyft/tyxt rules to tree rules can be lifted to the positive part of rules in ntyft/ntyxt format, but a simple example learns that this reduction cannot be applied to the negative premises. Again, we will find that the congruence theorem concerning the ntyft/ntyxt format can do without well-foundedness.

Verhoef [15] defined the panth format, which adds predicates to ntyft/ntyxt, and proved that the congruence theorem holds for well-founded TSSs in panth format. We will show that our results extend to the panth format too.

**Acknowledgements.** Catuscia Palamidessi and Fer-Jan de Vries noted the link with unification, and Chris Verhoef provided useful comments. Special thanks go to Rob van Glabbeek and Frits Vaandrager for suggesting some substantial improvements.

## 4.2 Preliminaries

This section contains the basic definitions.

### 4.2.1 The signature

In the sequel we assume the existence of an infinite set of variables $V$.[1]

**Definition 4.1** *A (single-sorted) signature $\Sigma$ consists of a set of function symbols, disjoint with $V$, together with their arities.*

*The collection $\mathbb{T}(\Sigma)$ of* (open) terms *over $\Sigma$ is defined as the least set satisfying:*

- *each variable from $V$ is in $\mathbb{T}(\Sigma)$,*
- *if $f \in \Sigma$ has arity $n$, and $t_1, ..., t_n \in \mathbb{T}(\Sigma)$, then $f(t_1, ..., t_n) \in \mathbb{T}(\Sigma)$.*

*A term is called* closed *if it does not contain any variables.*

A *substitution* is a mapping $\sigma : V \to \mathbb{T}(\Sigma)$. Each substitution is extended to a mapping from terms to terms in the standard way.

### 4.2.2 Transition system specifications

In the sequel we assume the existence of a set of labels $A$.

**Definition 4.2** *For each label $a$, the expression $\xrightarrow{a}$ denotes a binary relation on terms. A pair $t \xrightarrow{a} t'$ is called a* transition. *A transition is* closed *if it involves closed terms.*

[1] In several constructions we will assume the existence of 'fresh' variables, i.e. variables that have not yet been used in the construction. Some caution is needed to ensure the existence of such fresh variables at any time, but clearly this technical problem is not of a serious nature.

**Definition 4.3** *A* (transition) rule *is an expression of the form $H/c$, with $H$ a collection of transitions, called the* premises *(or the* hypotheses*), and $c$ a transition, called the* conclusion *of the rule.*

*A* Transition System Specification (TSS) *is a collection of transition rules.*

The notion of substitution extends to transitions and rules as expected.

**Definition 4.4** *A* proof *from a TSS $R$ of a rule $H/c$ consists of an upwardly branching tree in which all upward paths are finite. Moreover, the nodes of the tree are labelled by transitions, such that:*

- *the root has label $c$,*
- *if some node has label $d$, and $I$ is the set of labels of nodes directly above this node, then*
  1. *either $I = \emptyset$, and $d \in H$,*
  2. *or $I/d$ is a substitution instance of a rule in $R$.*

*We say that a transition $t \xrightarrow{a} t'$ is* provable *from $R$, if the rule with no premises and conclusion $t \xrightarrow{a} t'$ has a proof from $R$.*

**Definition 4.5** *Two TSSs are* (transition) equivalent *if exactly the same closed transitions are provable from both.*

We will say that a rule $r$ together with a substitution $\sigma$ *deduces* a transition $t \xrightarrow{a} t'$ from $R$ if all the premises of $r$ under $\sigma$ are provable from $R$, and the conclusion of $r$ under $\sigma$ results to $t \xrightarrow{a} t'$.

### 4.2.3 Strong bisimulation

**Definition 4.6** *Assume a TSS $R$. Two closed terms $p_0, q_0$ are* $R$-bisimilar*, notation $p_0 \leftrightarrows_R q_0$, if there exists a symmetric binary relation $\mathcal{B}$ on closed terms such that*

- *$p_0 \mathcal{B} q_0$,*
- *if $p \mathcal{B} q$, and $p \xrightarrow{a} p'$ is provable from $R$, then there is a closed term $q'$ such that $q \xrightarrow{a} q'$ is provable from $R$, and $p' \mathcal{B} q'$.*

### 4.2.4 The tyft/tyxt format

In general, bisimulation equivalence it is not a *congruence*, i.e. it may be the case that $p_i \leftrightarrows_R q_i$ for $i = 1, ..., n$, but $f(p_1, ..., p_n)$ and $f(q_1, ..., q_n)$ are not $R$-bisimilar. Therefore, Groote and Vaandrager [10] have introduced the *tyft/tyxt format*. If a TSS is in this format, and it satisfies a well-foundedness criterion, then the bisimulation it induces is a congruence.

**Definition 4.7** *A transition rule is a* tyft *rule if it is of the form*

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_m) \xrightarrow{a} t}$$

*where the $x_k$ and the $y_i$ are distinct variables (and $I$ is some, not necessarily finite, index set). Similarly, a* tyxt *rule is of the form*

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{x \xrightarrow{a} t}$$

*where $x$ and the $y_i$ are distinct variables. A TSS is said to be in* tyft/tyxt *format if it consists of tyft and tyxt rules only.*

**Definition 4.8** *Assume a set $\{t_i \xrightarrow{a_i} t'_i \mid i \in I\}$ of transitions. Its 'dependency graph' is a directed graph, with the collection of variables $V$ as vertices, and with as edges the collection*

$$\{\langle x, y\rangle \mid x \text{ and } y \text{ occur in } t_i \text{ and } t'_i \text{ respectively, for some } i \in I\}.$$

*A set of transitions is called* well-founded *if any backward chain of edges in its dependency graph is finite. A transition rule is well-founded if its collection of premises is so, and a TSS is well-founded if all its rules are so.*

**Example 4.9** *Examples of sets of transitions that are not well-founded are:*

- $\{y \xrightarrow{a} y\}$,
- $\{y_1 \xrightarrow{a} y_2,\ y_2 \xrightarrow{b} y_1\}$,
- $\{y_{i+1} \xrightarrow{a} y_i \mid i = 0, 1, 2, ...\}$.

The following congruence theorem originates from [10].

**Theorem 4.10** *If a TSS $R$ is well-founded and in tyft/tyxt format, then $\underline{\leftrightarrow}_R$ is a congruence.*

In Section 4.4 we will see that the requirement of well-foundedness in this theorem can be dropped.

## 4.3 Unification

A standard result from logic programming says that if a finite collection $E$ of equations between terms is *unifiable*, then there exists a *unifier* $\bar{\rho}$ for $E$ such that each unifier for $E$ is also a unifier for $\bar{\rho}$. This result follows from the well-known Martelli-Montanari algorithm [11]. See [1] for the basic definitions and for an introduction to the field of logic programming and unification.

In Chapter 5, this theorem is generalized to the case where $E$ may be infinite. The first property in Lemma 4.12, which will be vital in the proof of the main theorem, is a corollary of this unification result. However, we present a full proof of the lemma, because we will need two extra properties of the unifier $\bar{\rho}$, which follow most easily from its construction. Also, the proof of this lemma is much simpler than the proof of the stronger unification result in Chapter 5.

**Definition 4.11** *A substitution $\sigma$ is a* unifier *for a substitution $\rho$ if $\sigma\rho = \sigma$. In this case, $\rho$ is called* unifiable*.*

**Lemma 4.12** *If a substitution $\rho$ is unifiable, then there exists a unifier $\bar{\rho}$ for $\rho$ with the following properties:*

1. *Each unifier for $\rho$ is also a unifier for $\bar{\rho}$.*
2. *If $\rho(x) = x$, then $\bar{\rho}(x) = x$.*
3. *If $\rho^n(x)$ is a variable for all $n \geq 0$, then $\bar{\rho}(x)$ is a variable.*

**Proof.** Let $W$ denote the collection of variables $x$ for which $\rho^n(x)$ is a variable for all $n \geq 0$. First, we define the restriction $\bar{\rho}_0$ of $\bar{\rho}$ to $W$.

Define a binary relation $\sim$ on $W$ by $x \sim x'$ if $\rho^m(x) = \rho^n(x')$ for certain $m$ and $n$. Note that $\sim$ is an equivalence relation. Under $\bar{\rho}_0$, we contract the elements of each equivalence class $C \subseteq W$ to one variable from this class as follows.

- If $\rho(x_0) = x_0$ for some $x_0 \in C$, then for all $x \in C$ $\rho^n(x) = x_0$ for some $n$. This implies $\rho(x) \neq x$ for $x \in C \backslash \{x_0\}$, so $x_0$ is determined uniquely. Put $\bar{\rho}_0(x) = x_0$ for $x \in C$.

- If $\rho(x) \neq x$ for all $x \in C$, then just pick some $x_0 \in C$ and put $\bar{\rho}_0(x) = x_0$ for $x \in C$.

Put $\bar{\rho}_0(y) = y$ for $y \notin W$.

We construct $\bar{\rho}(y)$ as follows. By assumption, $\rho$ allows a unifier $\sigma$. Since $\sigma\rho = \sigma$, it follows that $\sigma\rho^n = \sigma$ for $n \geq 0$. Clearly, the *size* of each $\rho^n(y)$ (that is, the number of function symbols it contains) is smaller or equal than the size of $\sigma\rho^n(y) = \sigma(y)$. Moreover, each term $\rho^{n+1}(y)$ has at least the size of $\rho^n(y)$. So from a certain natural $N(y)$ onwards, the terms $\rho^n(y)$ all have the same size. Hence, for $n \geq N(y)$, $\rho^{n+1}(y)$

is obtained from $\rho^n(y)$ by replacing variables by variables. This means that all variables in $\rho^{N(y)}(y)$ are in $W$. Put

$$\bar{\rho}(y) = \bar{\rho}_0\rho^{N(y)}(y).$$

Note that $N(x) = 0$ if $x \in W$, so $\bar{\rho}$ equals $\bar{\rho}_0$ on $W$. We check the required properties for $\bar{\rho}$.

- $\bar{\rho}$ is a unifier for $\rho$.

  First, consider a variable $x \in W$. Since $\rho(x) \sim x$, and $\bar{\rho}_0$ contracts variables in the same equivalence class, we have $\bar{\rho}_0\rho(x) = \bar{\rho}_0(x)$. Since $\bar{\rho}$ equals $\bar{\rho}_0$ on $W$, this implies $\bar{\rho}\rho(x) = \bar{\rho}(x)$.

  Next, consider a variable $y \notin W$. Then clearly $N(y) = N(\rho(y)) + 1$, so

  $$\bar{\rho}\rho(y) = \bar{\rho}_0\rho^{N(\rho(y))}\rho(y) = \bar{\rho}_0\rho^{N(y)}(y) = \bar{\rho}(y).$$

- Each unifier $\sigma$ for $\rho$ is a unifier for $\bar{\rho}$.

  First, consider a variable $x \in W$. Since $\bar{\rho}_0(x) \sim x$, there are $m$ and $n$ such that $\rho^m\bar{\rho}_0(x) = \rho^n(x)$. After applying $\sigma$ to both sides we get $\sigma\bar{\rho}_0(x) = \sigma(x)$. Since $\bar{\rho}_0(y) = y$ for variables $y \notin W$, it follows that $\sigma\rho_0 = \sigma$.

  So for each variable $y$ we have

  $$\sigma\bar{\rho}(y) = \sigma\bar{\rho}_0\rho^{N(y)}(y) = \sigma\rho^{N(y)}(y) = \sigma(y).$$

- If $\rho(x) = x$, then $\bar{\rho}(x) = x$.

  Clearly $x \in W$, so $\bar{\rho}(x) = \bar{\rho}_0(x)$. Since $\rho(x) = x$, the construction of $\bar{\rho}_0$ ensures that $\rho_0(x) = x$.

- If $\rho^n(x)$ is a variable for all $n \geq 0$, then $\bar{\rho}(x)$ is a variable.

  By definition $x \in W$, so $\bar{\rho}(x) = \bar{\rho}_0(x)$. From the construction of $\bar{\rho}_0$ it follows that its image contains variables only. □

## 4.4 Tyft/Tyxt Reduces to Tree

This section contains the proof of the main theorem, which says that for each TSS in tyft/tyxt format there exists an equivalent TSS in the more restrictive tree format.

### 4.4.1 Tyft/tyxt reduces to tyft

The following lemma from [10] indicates that we can refrain from tyxt rules.

**Lemma 4.13** *Each TSS $R$ in tyft/tyxt format is equivalent to a TSS in tyft format.*

**Proof.** Replace each tyxt rule $r$ in $R$ by a collection of tyft rules $\{r_f | f \in \Sigma\}$, where each $r_f$ is obtained by substituting $f(x_1, ..., x_n)$ for $x$ in $r$, with $x_1, ..., x_n$ variables that do not yet occur in $r$. Let $R'$ denote the collection of tyft rules that is thus obtained. Clearly, for each proof from $R$ of a certain closed transition, there is a proof from $R'$ of the same transition, and vice versa. Hence, $R$ and $R'$ are equivalent. □

### 4.4.2 Tyft reduces to xyft

In this section, we prove that the tyft format reduces to xyft rules, which will be an intricate affair.

**Definition 4.14** *A tyft rule is said to be a* xyft *rule if the left-hand sides of its premises are all single variables.*

**Theorem 4.15** *Each TSS $R$ in tyft format is equivalent to a TSS in xyft format.*

**Proof.** We shall prove $R$ equivalent with the TSS $S$ of xyft rules that are provable from $R$. Since all rules in $S$ are provable from $R$, clearly the transitions provable from $S$ are provable from $R$. We now show that each closed transition $p \xrightarrow{a} p'$ provable from $R$ is provable from $S$, using ordinal induction on the length of a proof $P$ of $p \xrightarrow{a} p'$ from $R$.

Let $P$ have length $\alpha$, and suppose that we have proved the case for closed transitions that have a proof shorter than $\alpha$ from $R$. We will construct from $P$ a sequence of proofs $Q_n$ from $R$ of *tyft* rules $r_n$ that, together with a substitution $\sigma_n$, deduce $p \xrightarrow{a} p'$ from $S$. Each $Q_n$ will be a sub-tree of $P$, where its nodes are furnished with new labels, which under $\sigma_n$ yield the original labels of $P$. The 'limit' of the $Q_n$ will be a proof $Q$ from $R$ of a *xyft* rule $r$ that deduces $p \xrightarrow{a} p'$ from $S$.

Let $r_0 \in R$ together with a substitution $\sigma_0$ constitute the last step in $P$. The premises of $r_0$ under $\sigma_0$ are all provable from $R$ by a strict sub-proof of $P$, so according to the induction hypothesis these transitions are provable from $S$. Hence, $r_0$ together with $\sigma_0$ deduces $p \xrightarrow{a} p'$ from $S$. The proof $Q_0$ of $r_0$ from $R$ consists simply of a bottom node labelled by the conclusion of $r_0$ and upper nodes labelled by the premises of $r_0$.

Next, suppose that we have constructed a proof $Q_{n-1}$ from $R$ of a tyft rule $r_{n-1}$, which together with a $\sigma_{n-1}$ deduces $p \xrightarrow{a} p'$ from $S$. By assumption, $Q_{n-1}$ is a sub-tree of $P$, and the labels of $Q_{n-1}$ under $\sigma_{n-1}$ yield the original labels of $P$. Let $r_{n-1}$ be of the form

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_m) \xrightarrow{a} t}$$

Let $I_0 \subseteq I$ be the subset of $i$'s for which the term $t_i$ is not a single variable, but of the form $g_i(u_{i1}, ..., u_{im_i})$.

The premises of $r_{n-1}$ are labels of upper nodes in $Q_{n-1}$. Since $Q_{n-1}$ is a sub-tree of $P$, the premises correspond with nodes in $P$, which have labels $\sigma_{n-1}(t_i \xrightarrow{a_i} y_i)$. For $i \in I_0$, let $s_i \in R$ and $\tau_i$ together constitute the step in $P$ to the node with label $\sigma_{n-1}(t_i \xrightarrow{a_i} y_i)$. Ordinal induction implies that the premises of $s_i$ under $\tau_i$ are provable from $S$. To obtain $Q_n$, the rules $s_i$ will be imported into $Q_{n-1}$, so assume that each $s_i$ contains only fresh variables, to avoid name clashes.

Since the conclusion of $s_i$ under $\tau_i$ yields $\sigma_{n-1}(t_i \xrightarrow{a_i} y_i)$, and $t_i = g_i(u_{i1}, ..., u_{im_i})$, it follows for $i \in I_0$ that $s_i$ is of the form

$$\frac{\{t_j \xrightarrow{b_j} y_j \mid j \in J_i\}}{g_i(x_{i1}, ..., x_{im_i}) \xrightarrow{a_i} v_i}$$

with $\tau_i(x_{ik}) = \sigma_{n-1}(u_{ik})$ and $\tau_i(v_i) = \sigma_{n-1}(y_i)$.

Let $\sigma_n$ be a substitution equal to $\sigma_{n-1}$ for variables in $Q_{n-1}$, and equal to the $\tau_i$ for variables in the $s_i$ for $i \in I_0$. Moreover, define a substitution $\rho_n$ by:

$$\begin{array}{rcll} \rho_n(x_{ik}) & = & u_{ik} & \text{for } i \in I_0 \text{ and } k = 1, ..., m_i, \\ \rho_n(y_i) & = & v_i & \text{for } i \in I_0, \\ \rho_n(x) & = & x & \text{otherwise.} \end{array}$$

Note that $\sigma_n$ is a unifier for $\rho_n$, or in other words, $\sigma_n\rho_n = \sigma_n$:

$$\begin{array}{rcccccccc} \sigma_n\rho_n(x_{ik}) & = & \sigma_n(u_{ik}) & = & \sigma_{n-1}(u_{ik}) & = & \tau_i(x_{ik}) & = & \sigma_n(x_{ik}), \\ \sigma_n\rho_n(y_i) & = & \sigma_n(v_i) & = & \tau_i(v_i) & = & \sigma_{n-1}(y_i) & = & \sigma_n(y_i). \end{array}$$

So Lemma 4.12 indicates the existence of a unifier $\bar{\rho}_n$ for $\rho_n$ with the following properties.

1. Each unifier for $\rho_n$ is a unifier for $\bar{\rho}_n$, which implies: $\bar{\rho}_n\bar{\rho}_n = \bar{\rho}_n$, $\sigma_n\bar{\rho}_n = \sigma_n$.
2. If $\rho_n(x) = x$, then $\bar{\rho}_n(x) = x$.

Since $\bar{\rho}_n$ is a unifier for $\rho_n$, it follows that the conclusion of $\bar{\rho}_n(s_i)$ equals $\bar{\rho}_n(t_i \xrightarrow{a_i} y_i)$ for $i \in I_0$:

$$\begin{array}{rcl} \bar{\rho}_n(g_i(x_{i1}, ..., x_{im_i}) \xrightarrow{a_i} v_i) & = & \bar{\rho}_n(g_i(x_{i1}, ..., x_{im_i}) \xrightarrow{a_i} \rho_n(y_i)) \\ = \quad \bar{\rho}_n(\rho_n(g_i(x_{i1}, ..., x_{im_i})) \xrightarrow{a_i} y_i) & = & \bar{\rho}_n(g_i(u_{i1}, ..., u_{im_i}) \xrightarrow{a_i} y_i). \end{array}$$

Hence, we can extend $\bar{\rho}_n(Q_{n-1})$ to a proof $Q_n$ from $R$. Namely, extend $\bar{\rho}_n(Q_{n-1})$ above nodes labelled by $t_i \xrightarrow{a_i} y_i$ for $i \in I_0$ with new nodes that have as labels the premises of $\bar{\rho}_n(s_i)$, i.e. $\bar{\rho}_n(t_j \xrightarrow{b_j} y_j)$ for $j \in J_i$. Since the new steps in $Q_n$ match with the rules $\bar{\rho}_n(s_i)$ for $i \in I_0$, it follows that $Q_n$ constitutes a proof from $R$ of some rule $r_n$.

Due to property 2 of $\bar{\rho}_n$, the rule $r_n$ has conclusion $f(x_1,...,x_m) \xrightarrow{a} \bar{\rho}_n(t)$, and premises $\bar{\rho}_n(t_i) \xrightarrow{a_i} y_i$ for $i \in I \backslash I_0$ and $\bar{\rho}_n(t_j) \xrightarrow{b_j} y_j$ for $i \in I_0$ and $j \in J_i$. Hence, $r_n$ is a tyft rule. Property 1, $\sigma_n\bar{\rho}_n = \sigma_n$, implies that $r_n$ together with $\sigma_n$ deduces $p \xrightarrow{a} p'$ from $S$.

Finally, $\sigma_n\bar{\rho}_n = \sigma_n$ ensures that $\sigma_n$ applied to $Q_n$ yields the original labels of $P$.

In general, $r_n$ is not yet a xyft rule, because although we have removed from $r_n$ all the premises of $r_{n-1}$ that do not have a single variable as left-hand side, we may have introduced other premises in $r_n$ that are of this form. Therefore, we repeat the construction above again and again, to obtain sequences $\{Q_n\}_{n=0}^{\infty}$ and $\{r_n\}_{n=0}^{\infty}$ and $\{\sigma_n\}_{n=0}^{\infty}$ and $\{\bar{\rho}_n\}_{n=1}^{\infty}$, where $Q_n$ is a proof from $R$ of $r_n$, and $r_n$ together with $\sigma_n$ deduces $p \xrightarrow{a} p'$ from $S$.

We construct the limit $Q$ of the proofs $Q_n$. The tree structure of $Q$ is simply the limit of the trees $Q_n$; this is well-defined, because $Q_n$ incorporates $Q_{n-1}$. However, the labels of the nodes in $Q$ cannot be determined so easily, because the labels in the $Q_n$ are not consistent; if a certain node in $Q_{n-1}$ has label $l$, then in $Q_n$ it is renamed to $\bar{\rho}_n(l)$. To resolve this complication, we need some extra machinery.

If $\bar{\rho}_n(x) \neq x$, then it follows from property 1 of $\bar{\rho}_n$, namely idempotence, that $x$ cannot occur in any term $\bar{\rho}_n(y)$. To obtain $Q_n$, we have applied $\bar{\rho}_n$ to all its labels, so $x$ does not occur in $Q_n$. This implies $\bar{\rho}_m(x) = x$ for $m > n$. Hence, we can define a substitution $\bar{\rho}$ as follows:

$$\begin{array}{lcll} \bar{\rho}(x) & = & \bar{\rho}_n(x) & \text{if } \bar{\rho}_n(x) \neq x \text{ for some } n, \\ \bar{\rho}(x) & = & x & \text{otherwise.} \end{array}$$

Furthermore, let $\sigma$ be a substitution that equals $\sigma_n$ for variables in $Q_n$ for all $n$. Since $\sigma_n\bar{\rho}_n = \sigma_n$ for all $n$, we have $\sigma\bar{\rho} = \sigma$. So according to Lemma 4.12 there exists a unifier $\hat{\rho}$ for $\bar{\rho}$ with the following properties:

1. $\sigma\hat{\rho} = \sigma$.
2. If $\bar{\rho}(x) = x$, then $\hat{\rho}(x) = x$.
3. if $\bar{\rho}^k(x)$ is a variable for $k \geq 0$, then $\hat{\rho}(x)$ is a variable.

Since $\hat{\rho}$ is a unifier for $\bar{\rho}$, it follows that $\hat{\rho}$ is a unifier for all substitutions $\bar{\rho}_n$.

Now we can determine the labels of $Q$. If a node has label $l$ in $Q_{n-1}$, then in $Q$ we furnish it with the label $\hat{\rho}(l)$. This definition does not depend on the choice of $n$, because although in $Q_n$ the label is adapted to $\bar{\rho}_n(l)$, the equality $\hat{\rho}\bar{\rho}_n = \hat{\rho}$ ensures that the resulting label in $Q$ would remain the same.

Since $Q$ is a sub-tree of $P$, each upward path in $Q$ must be finite. And if a step in $P$ matches with a rule $s \in R$ together with a $\tau$, then the same step in $Q$ matches with $s$ together with $\hat{\rho}\tau$. Hence, $Q$ is a proof from $R$ of a rule $r$.

We check that $r$ is xyft. First, consider a premise of $r$. It has been introduced in some $r_n$, and was maintained in all subsequent $r_m$, so apparently in $r_n$ it had the form $z \xrightarrow{b} y$, and $\bar{\rho}^k(z)$ is a variable for all $k \geq 0$. So according to property 3, $\hat{\rho}(z)$ is a variable. Moreover, $\bar{\rho}_n(y) = y$ for all $n$, so due to property 2 $\hat{\rho}(y) = y$.

Summarizing, the premise in $r$ has the form $\hat{\rho}(z) \xrightarrow{b} y$ with $\hat{\rho}(z)$ a variable. Clearly, the conclusion of $r$ equals $f(x_1, ..., x_m) \xrightarrow{a} \hat{\rho}(t)$ (where $t$ is the right-hand side of the conclusion of $r_0$). So $r$ is xyft.

Since $r$ is provable from $R$ and xyft, by definition $r \in S$. Since $\sigma\hat{\rho} = \sigma$, the conclusion of $r$ under $\sigma$ results to $p \xrightarrow{a} p'$, and the premises of $r$ under $\sigma$ are all provable from $S$. So $r$ proves $p \xrightarrow{a} p'$ from $S$. Then clearly $p \xrightarrow{a} p'$ is provable from $S$. □

Although according to Theorem 4.15 the tyft/tyxt format reduces to the more restrictive xyft format, this is by no means an argument to abandon the tyft/tyxt format. Because a simple TSS in tyft/tyxt format may take a much more complicated form if it is described in xyft format. This is demonstrated by the following example.

**Example 4.16** *Assume two functions $a, b$ of arity zero, a function $f$ of arity one, and a label $l$. Consider the following TSS in tyft format.*

$$a \xrightarrow{l} a \qquad\qquad \frac{a \xrightarrow{l} y}{a \xrightarrow{l} f(y)}$$

*To describe this TSS in xyft format, we need an infinite number of rules: $a \xrightarrow{l} f^n(a)$ for $n = 0, 1, 2, ...$ (The auxiliary function symbol $b$ is present to avoid that the TSS can be described by the single rule $a \xrightarrow{l} x$.)*

### 4.4.3 Xyft reduces to tree

The following terminology originates from [10].

**Definition 4.17** *A variable is called* free *in a rule if it does not occur at the right-hand side of the premises, nor at the left-hand side of the conclusion of the rule. A rule is called* pure *if it does not contain any free variables.*

**Definition 4.18** *A* tree *rule is a pure and well-founded xyft rule.*

**Theorem 4.19** *Each TSS $R$ in xyft format is equivalent to a TSS in tree format.*

**Proof.** We prove $R$ equivalent with the TSS $S$ of tree rules that can be proved from $R$. Since all rules in $S$ can be proved from $R$, clearly each transition provable from $S$ is also provable from $R$. We check the converse, namely that a closed transition $p \xrightarrow{a} p'$ provable from $R$ is provable from $S$.

Fix a rule $r$ in $R$ that together with a closed substitution $\sigma$ deduces $p \xrightarrow{a} p'$ from $R$. Let $r$ be of the form

$$\frac{\{z_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_m) \xrightarrow{a} t}$$

Using ordinal induction, it follows that $\sigma(z_i \xrightarrow{a_i} y_i)$ is provable from $S$ for $i \in I$.

We construct from $r$ a rule $r'$ in $S$ that deduces $p \xrightarrow{a} p'$ from $S$ as follows. If there is no backward path in the dependency graph of $r$ from a vertice $y_i$ to a vertice $x_j$, then replace the variables $z_i$ and $y_i$ in $r$ by $\sigma(z_i)$ and $\sigma(y_i)$ respectively. Moreover, replace free variables $z$ in $t$ by $\sigma(z)$. The resulting rule $r''$ is a substitution instance of $r$, so $r''$ is provable from $R$. Remove each premise $\sigma(z_i \xrightarrow{a_i} y_i)$ from $r''$. Since such transitions are provable from $R$, the resulting rule $r'$ is provable from $R$.

Clearly, $r'$ is pure and xyft. Moreover, $r'$ is well-founded, because for each premise $z_i \xrightarrow{a_i} y_i$ in $r'$, the backward path from the vertice $y_i$ in the dependency graph of $r'$ terminates at a vertice $x_j$. Hence, $r'$ is a tree rule, so $r' \in S$. Since $r'$ together with $\sigma$ deduces $p \xrightarrow{a} p'$ from $S$, it follows that $p \xrightarrow{a} p'$ is provable from $S$. □

So, we have found that for each TSS in tyft/tyxt format there exists an equivalent TSS in tree format. Since tree rules are well-founded tyft rules, this result implies that the congruence theorem for tyft/tyxt can do without well-foundedness.

**Corollary 4.20** *If a TSS $R$ is in tyft/tyxt format, then $\underline{\leftrightarrow}_R$ is a congruence.*

## 4.5 Extensions to Other Formats

### 4.5.1 The ntyft/ntyxt format

Groote [9] extended the tyft/tyxt format to the *ntyft/ntyxt* format, which as extra feature allows transition rules to contain negative premises, i.e. expressions of the form $t \xrightarrow{a}\!\!\!\!\!\!/\;$. In a setting with negative premises, the definition for transitions that can be proved from a TSS has to be adapted, in order to deal with rules such as

$$\frac{t \xrightarrow{a}\!\!\!\!\!\!/\;}{t \xrightarrow{a} t'}$$

The most general way to associate transitions with a TSS that incorporates negative premises is through the notion of a *stable model*, which was introduced by Gelfond and Lifschitz [6] in logic programming. Bol and Groote [4], who adapted this notion for TSSs, showed that there exist TSSs in ntyft/ntyxt format with a unique stable model for which bisimulation is not a congruence. Therefore, they defined the (complicated) notion *positive after reduction*, inspired by the work of Van Gelder, Ross and Schlipf [5] in logic programming, and proved a congruence theorem for well-founded TSSs in the ntyft/ntyxt format that are positive after reduction.

Van Glabbeek [8] proposes the notion *complete (with positive support)*. Basically, a TSS is complete if for each closed transition $p \xrightarrow{a} p'$, the TSS can prove either $p \xrightarrow{a} p'$ or its negation $p \xrightarrow{a}\!\!\!\!\!\!/\; p'$. Van Glabbeek proposes a notion of provability that allows to derive negative transitions. Van Glabbeek shows that the notions *complete* and *positive after reduction* are equivalent.

Without any further complications, we can repeat the construction from the previous section to show that each complete TSS in ntyft/ntyxt format is equivalent

to a complete TSS in an extension of the tree format, which allows rules to have premises of the form $t \stackrel{a}{\not\longrightarrow}$. Again, TSSs in the latter format are well-founded, so as a corollary we see that the well-foundedness condition in the congruence theorem for the ntyft/ntyxt format can be dropped.

**Corollary 4.21** *If a complete TSS $R$ is in ntyft/ntyxt format, then $\underline{\leftrightarrow}_R$ is a congruence.*

### 4.5.2 The panth format

Baeten and Verhoef [2] extended the tyft/tyxt format with predicates, i.e. not only relations $t \stackrel{a}{\longrightarrow} t'$, but also predicates such as $t \stackrel{a}{\longrightarrow} \surd$ are allowed to occur in transition rules. The definition of strong bisimulation, Definition 4.6, is adapted accordingly by adding a third condition:

- if $p\mathcal{B}q$, and $p \stackrel{a}{\longrightarrow} \surd$ is provable from $R$, then $q \stackrel{a}{\longrightarrow} \surd$ is provable from $R$.

Next, Verhoef [15] extended the path format with negative premises. A congruence theorem holds for well-founded complete TSSs that are in the so-called *panth* format, which is essentially the natural extension of ntyft/ntyxt with predicates.

Without any further complications, we can repeat the construction from the previous section to show that each complete TSS in panth format is equivalent to a complete TSS in an extension of the tree format, which allows rules to have premises of the form $z \stackrel{a}{\longrightarrow} \surd$ and $t \stackrel{a}{\not\longrightarrow}$ and $t \stackrel{a}{\not\longrightarrow} \surd$, and a conclusion of the form $f(x_1, ..., x_m) \stackrel{a}{\longrightarrow} \surd$. As a corollary, we see that the well-foundedness condition in the congruence theorem for the panth format can be dropped.

**Corollary 4.22** *If a complete TSS $R$ is in panth format, then $\underline{\leftrightarrow}_R$ is a congruence.*

### 4.5.3 Panth does not reduce to negative tree

We show that in general terms in negative premises cannot be reduced to variables. The *negative* tree format allows complete TSSs which consist of pure and well-founded panth rules, where the variables of all the premises (so also of the negative premises) are variables. We present a complete TSS in panth format for which there does not exist an equivalent TSS in negative tree format.

Our counter-example is presented in the setting of Basic Process Algebra (BPA). This formalism assumes an alphabet $A$, representing functions with arity zero, and the functions $+$ and $\cdot$, both of arity two, denoting alternative and sequential composition respectively. BPA assumes relation $\stackrel{a}{\longrightarrow}$ and predicates $\stackrel{a}{\longrightarrow} \surd$ for $a \in A$. The action rules of BPA are as follows.

Add two functions $f$ and $g$ with arity one and a predicate $\stackrel{ok}{\longrightarrow} \surd$ to BPA, and extend the operational semantics by the following two transition rules, to obtain the TSS $R$.

$$\frac{x \stackrel{a}{\longrightarrow} y_1 \quad y_1 \stackrel{a}{\longrightarrow} \surd}{g(x) \stackrel{ok}{\longrightarrow} \surd} \qquad \frac{g(x)\neg \stackrel{ok}{\longrightarrow} \surd}{f(x) \stackrel{ok}{\longrightarrow} \surd}$$

$$a \xrightarrow{a} \surd$$

$$\frac{x \xrightarrow{a} \surd}{x + y \xrightarrow{a} \surd \quad y + x \xrightarrow{a} \surd} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x' \quad y + x \xrightarrow{a} x'}$$

$$\frac{x \xrightarrow{a} \surd}{x \cdot y \xrightarrow{a} y} \qquad \frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}$$

The TSS $R$ is complete and in panth format. The premise $g(x)\neg \xrightarrow{ok} \surd$ cannot be reduced. An obvious attempt to delete this negative premise would be to replace the second rule by the following two rules.

$$\frac{x \xrightarrow{a}\!\!\!\!\!/\;\;}{f(x) \xrightarrow{ok} \surd} \qquad \frac{x \xrightarrow{a} y \quad y \xrightarrow{a}\!\!\!\!\!/\;\;}{f(x) \xrightarrow{ok} \surd}$$

However, this adapted TSS is not equivalent to $R$. For example, $f(aa + ab) \xrightarrow{ok} \surd$ holds in the new TSS, but not in $R$.

**Lemma 4.23** *Let $T$ be a TSS in negative tree format and $p_0$ and $p_1$ two closed terms, such that:*

1. *if $T$ proves $p_0 \xrightarrow{a} q$, then $T$ proves $p_1 \xrightarrow{a} q$,*
2. *if $T$ proves $p_0 \xrightarrow{a} \surd$, then $T$ proves $p_1 \xrightarrow{a} \surd$,*
3. *if $T$ proves $p_0\neg \xrightarrow{a}$, then $T$ proves $p_1\neg \xrightarrow{a}$,*
4. *if $T$ proves $p_0\neg \xrightarrow{a} \surd$, then $T$ proves $p_1\neg \xrightarrow{a} \surd$.*

*If $T$ proves $f(p_0) \xrightarrow{ok} \surd$, then $T$ proves $f(p_1) \xrightarrow{ok} \surd$.*

**Proof.** Suppose that $r \in T$ together with a substitution $\sigma$ deduces $f(p_0) \xrightarrow{a} \surd$ from $T$. Since $r$ is in negative tree format, it has a conclusion of the form $f(x) \xrightarrow{ok} \surd$, and $\sigma(x) = p_0$.

Define a substitution $\sigma'$ by $\sigma'(x) = p_1$ and $\sigma'(x) = \sigma(y)$ for $y \neq x$. Since $r$ is in negative tree format, properties 1-4 of the transition systems of $p_0$ and $p_1$ ensure that $r$ together with $\sigma'$ deduces $f(p_1) \xrightarrow{ok} \surd$ from $T$. □

The TSS $R$ that was defined before proves $f(aa) \xrightarrow{ok} \surd$, but not $f(aa+ab) \xrightarrow{ok} \surd$. Hence, Lemma 4.23 ensures that $R$ cannot be equivalent to a TSS in negative tree format.

# References

[1] K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics*, pages 493–574. Elsevier, 1990.

[2] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, *Proceedings 4th Conference on Concurrency Theory (CONCUR'93),* Hildesheim, *LNCS 715*, pages 477–492. Springer-Verlag, 1993.

[3] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Proceedings 15th ACM Symposium on Principles of Programming Languages,* San Diego, California, pages 229–239, 1988. To appear in *Journal of the ACM.*

[4] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Proceedings 18th International Colloquium on Automata, Languages and Programming (ICALP'91),* Madrid, *LNCS 510*, pages 481–494. Springer-Verlag, 1991.

[5] A. van Gelder, K. Ross and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings 7th Symposium on Principles of Database Systems*, pages 221–230. ACM SIGACT-SIGMOD, 1988.

[6] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings 5th Conference on Logic Programming*, pages 1070–1080. MIT press, 1988.

[7] R.J. van Glabbeek. Full abstraction in structural operational semantics. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Proceedings 3rd Conference on Algebraic Methodology and Software Technology (AMAST'93),* Twente, The Netherlands, Workshops in Computing, pages 77–84. Springer-Verlag, 1993.

[8] R.J. van Glabbeek. Unpublished manuscript on the meaning of negative premises in transition system specifications.

[9] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

[10] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[11] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[12] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *5th GI Conference*, *LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[13] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

[14] R. de Simone. Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[15] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In B. Jonsson and J. Parrow, editors, *Proceedings 5th Conference on Concurrency Theory (CONCUR'94),* Uppsala, *LNCS 836*, pages 433–448. Springer-Verlag, 1994.

# 5

# Idempotent Most General Unifiers for Infinite Sets

Wan Fokkink

A standard result from unification theory says that if a finite set $E$ of equations between terms is unifiable, then there exists an idempotent most general unifier for $E$. In this chapter, the theorem is generalized, in first-order logic, to the case where $E$ may be infinite.

## 5.1 Introduction

The unification problem is to determine, given an equation $s = t$ in some logic, whether there exists a substitution $\sigma$ such that $(s)\sigma = (t)\sigma$. The substitution $\sigma$ is a 'unifier', and $s = t$ is called 'unifiable'. For an introduction into the field of unification theory, see [1, 7].

A first algorithm, which solves the unification problem in first-order logic, stems from Herbrand's thesis [4] (see [3]). This algorithm was rediscovered by Prawitz [11], and its full significance was recognized only after Robinson [12] had employed it in his resolution principle for automatic theorem-proving. Robinson was the first to define the basic concepts for unification. His algorithm decides whether an equation is unifiable or not, and if so, then it produces a unifier which is idempotent and 'most general', which means that all other unifiers for the equation can be derived from it. More efficient unification algorithms, for finite sets of equations, were proposed by Paterson and Wegman [9] and Martelli and Montanari [8] and Eder [2].

Pietrzykowski [10] defined an algorithm to detect whether or not an equation is unifiable in second-order logic. Huet [5] showed that such an algorithm does not exist in third-order logic. In [6] however, he introduced an algorithm in $\omega$-order logic which, given a unifiable equation, computes a unifier for this equation. For non-unifiable equations, this algorithm may not terminate.

In this chapter, we prove that each *infinite* unifiable collection of equations, in first-order logic, allows an idempotent most general unifier. Since the collection of

equations is infinite, our construction involves limit procedures.

For an application of this result, in the setting of operational semantics à la Plotkin, see Chapter 4 of this thesis.

**Acknowledgements.** This research was initiated by discussions with Catuscia Palamidessi.

## 5.2 Preliminaries

In the sequel we assume an alphabet, which consists of the disjoint union of an infinite set of variables and a set of function symbols. Each function symbol $f$ is provided with an arity $ar(f)$, being a natural number $\geq 0$. The collection of *terms* over the alphabet is defined inductively as follows:

- each variable is a term,
- for $f$ is a function symbol, and $t_1, ..., t_{ar(f)}$ terms, $f(t_1, ..., t_{ar(f)})$ is a term.

The number of function symbols in a term is called the *size* of the term. Syntactic equivalence between terms is denoted by $\_ = \_$.

A *substitution* is a mapping from variables to terms. The notation $\sigma = \tau$ means that $(x)\sigma = (x)\tau$ for all variables $x$. Each substitution is extended to a mapping from terms to terms in the standard way.

**Definition 5.1** *The* domain *of a substitution $\sigma$ is the collection of variables $x$ for which $(x)\sigma \neq x$.*

Substitutions are allowed to have an infinite domain.

**Definition 5.2** *A substitution $\sigma$ is* idempotent *if $\sigma\sigma = \sigma$.*

In the sequel, $E$ denotes a set of equations $s = t$ between terms.

**Definition 5.3** *A substitution $\sigma$ is a* unifier *for $E$ if for all $s = t \in E$ we have $(s)\sigma = (t)\sigma$. The set $E$ is called* unifiable *if it allows a unifier.*

*A substitution $\sigma$ is a* unifier *for a substitution $\tau$ if $\tau\sigma = \sigma$.*

**Definition 5.4** *A unifier $\Theta$ for $E$ is called* most general *if for each unifier $\sigma$ for $E$ there exists a substitution $\sigma'$ such that $\Theta\sigma' = \sigma$.*

## 5.3 The Main Theorem

A standard result from unification theory says that if a *finite* set $E$ of equations between terms is unifiable, then there exists an idempotent most general unifier for $E$. In this chapter, the theorem is generalized to the case where $E$ may be infinite. First, we rephrase the theorem.

**Proposition 5.5** *The following two statements for a substitution $\Theta$ are equivalent.*

1. *$\Theta$ is an idempotent most general unifier for $E$.*
2. *$\Theta$ is a unifier for $E$, and each unifier for $E$ is a unifier for $\Theta$.*

**Proof.** ($\Rightarrow$) Let $\sigma$ unify $E$. Since $\Theta$ is most general, there is a substitution $\sigma'$ such that $\Theta\sigma' = \sigma$. Furthermore, $\Theta$ is idempotent, so

$$\Theta\sigma = \Theta\Theta\sigma' = \Theta\sigma' = \sigma.$$

($\Leftarrow$) Each unifier $\sigma$ for $E$ unifies $\Theta$, which means that $\Theta\sigma = \sigma$. So $\Theta$ is most general. $\Theta$ is a unifier for $E$, so in particular $\Theta$ unifies itself. Hence, $\Theta\Theta = \Theta$. □

An equation $s = t$ will be called a *proper sub-equation* of equations $C[s] = C[t]$ for non-trivial contexts $C[]$.

**Theorem 5.6** *If $E$ is unifiable, then there exists a unifier $\Theta$ for $E$ such that each unifier for $E$ is also a unifier for $\Theta$.*

**Proof.** Let $\tau_0$ denote the identity substitution, and define $E_0^e = \{e\}$ for each $e \in E$. We define a construction which produces from a substitution $\tau_{n-1}$ and unifiable sets of equations $E_{n-1}^e$, a substitution $\tau_n$ and unifiable sets of equations $E_n^e$.

- If $E_{n-1}^e$ contains an equality $f(s_1, ..., s_{ar(f)}) = g(t_1, ..., t_{ar(g)})$, then $f \equiv g$, because $E_{n-1}^e$ is unifiable. Replace each such equation in $E_{n-1}^e$ by its proper sub-equations $s_i = t_i$ for $i = 1, ..., ar(f)$. Denote the resulting set by $F_n^e$. Note that a substitution unifies $E_{n-1}^e$ if and only if it unifies $F_n^e$.
- Suppose that a variable $x$ is not in the domain of $\tau_{n-1}$, and that $\cup_{e\in E} F_n^e$ contains (one or more) equations of the form $x = t$ or $t = x$, where $t$ is not a single variable. Then choose one of these equations $x = t$ or $t = x$, and put $(x)\tau_n = t$. Put $(y)\tau_n = (y)\tau_{n-1}$ for all other variables $y$. In particular, $\tau_n$ equals $\tau_{n-1}$ on the domain of $\tau_{n-1}$.
- Put $E_n^e = (F_n^e)\tau_n$.

From the following Property 1, and from the fact that $\tau_0$ and $E$ are unifiable, it follows immediately that all $E_n^e$ are unifiable.

1. Each unifier $\sigma$ for $\tau_{n-1}$ and $\cup_{e\in E} E_{n-1}^e$, is also a unifier for $\tau_n$ and $\cup_{e\in E} E_n^e$.

*Proof.* Since $\sigma$ unifies $E_{n-1}^e$, it also unifies $F_n^e$, for each $e \in E$.

If $(x)\tau_n = (x)\tau_{n-1}$ for a variable $x$, then $(x)\tau_n\sigma = (x)\tau_{n-1}\sigma = (x)\sigma$, because $\sigma$ unifies $\tau_{n-1}$. Otherwise, if $(x)\tau_n \neq (x)\tau_{n-1}$, then it follows from the construction of $\tau_n$ that $(x)\tau_n = x$ (or its reverse) is in $\cup_{e\in E} F_n^e$. Since $\sigma$ unifies all $F_n^e$, it follows that $(x)\tau_n\sigma = (x)\sigma$. Hence, $\sigma$ unifies $\tau_n$.

$(E_n^e)\sigma = (F_n^e)\tau_n\sigma = (F_n^e)\sigma$, because $\sigma$ unifies $\tau_n$. Since $\sigma$ unifies $F_n^e$, it follows that $\sigma$ unifies $E_n^e$, for each $e \in E$. □

2. Each unifier $\sigma$ for $\tau_n$ and $E^e_n$, is also a unifier for $\tau_{n-1}$ and $E^e_{n-1}$.

*Proof.* $\tau_{n-1}$ equals $\tau_n$ on its domain, and $\sigma$ unifies $\tau_n$, so $\sigma$ also unifies $\tau_{n-1}$.

$(F^e_n)\sigma = (F^e_n)\tau_n\sigma = (E^e_n)\sigma$, and $\sigma$ unifies $E^e_n$, so $\sigma$ unifies $F^e_n$. Then $\sigma$ unifies $E^e_{n-1}$. □

Since $\tau_n$ equals $\tau_{n-1}$ on the domain of $\tau_{n-1}$, we can define the 'union' $\tau$ of the substitutions $\tau_n$:

$$(x)\tau = \begin{cases} (x)\tau_n & \text{if } (x)\tau_n \neq x \text{ for some } n, \\ x & \text{otherwise.} \end{cases}$$

3. For each variable $x$, either $(x)\tau = x$, or $(x)\tau$ is not a variable.

*Proof.* If $(x)\tau \neq x$, then $(x)\tau = (x)\tau_n$ for some $n > 0$. Let $n$ be the smallest natural number for which this equality holds, so that $(x)\tau_n \neq (x)\tau_{n-1}$. Then it follows from the construction of $\tau_n$ that there is an equation $x = t$ or $t = x$ in $\cup_{e\in E} F^e_n$, where $t$ is not a variable, and $(x)\tau_n = t$. Hence, $(x)\tau = t$ is not a variable. □

4. For each variable $x$, there is a natural $M(x)$ such that $(x)\tau^{M(x)+1} = (x)\tau^{M(x)}$.

*Proof.* Fix a unifier $\sigma$ for $E$. Since $\sigma$ also unifies the identity $\tau_0$, Property 1 implies that $\sigma$ is a unifier for all $\tau_n$. So $\sigma$ is a unifier for their union $\tau$, which means that $(x)\tau^m\sigma = (x)\sigma$ for all $m$. Thus, the size of the terms $(x)\tau^m$ cannot grow beyond the size of $(x)\sigma$. The term $(x)\tau^{m+1}$ is obtained from $(x)\tau^m$ by application of $\tau$, so the size of $(x)\tau^{m+1}$ is at least the size of $(x)\tau^m$. Hence, there is a natural $M(x)$ such that for $m \geq M(x)$, all terms $(x)\tau^m$ have the same size. Then Property 3 of $\tau$ implies $(x)\tau^{m+1} = (x)\tau^m$ for $m \geq M(x)$. □

We define the 'limit' $\bar{\tau}$ of $\tau$ by

$$(x)\bar{\tau} = (x)\tau^{M(x)}.$$

Property 4 implies that $\tau\bar{\tau} = \bar{\tau}$. So, since $\tau$ is the union of all $\tau_n$, $\tau_n\bar{\tau} = \bar{\tau}$ for all $n$.

5. For each $e \in E$, there is a natural $N(e)$ such that $E^e_{N(e)}$ contains only equations of the form $x = y$, where $x$ and $y$ are not in the domain of $\bar{\tau}$.

*Proof.* Fix an $e \in E$, and consider the sequence $\{(E^e_n)\bar{\tau}\}^\infty_{n=0}$.

Each equation in $E^e_{n-1}$ is either maintained, or replaced by proper sub-equations in $F^e_n$. Hence, each equation in $(E^e_{n-1})\bar{\tau}$ is either maintained, or replaced by proper sub-equations in $(F^e_n)\bar{\tau} = (F^e_n)\tau_n\bar{\tau} = (E^e_n)\bar{\tau}$. Since proper sub-equations have a size smaller than the original equation, each chain of subsequent proper sub-equations in the subsequent $(E^e_n)\bar{\tau}$ is finite. So, by König's Lemma, there is some $N(e)$ such that all equations in $(E^e_{n-1})\bar{\tau}$ are maintained in $(E^e_n)\bar{\tau}$ for each $n > N(e)$.

Consider an equation $s = t$ in $E^e_{n-1}$ for some $n > N(e)$. Since $(s = t)\bar{\tau} \in (E^e_{n-1})\bar{\tau}$ is maintained in $(E^e_n)\bar{\tau}$, $s = t \in E^e_{n-1}$ is maintained in $F^e_n$. So $s = t$ cannot have any proper sub-equations, or in other words, $s$ or $t$ must be a variable, say, $s = x$.

Now suppose that $t$ is *not* a variable. We deduce a contradiction. First, we show that then $(x)\tau_n$ is not a variable. Distinguish two cases.

- $(x)\tau_{n-1} \neq x$. Since $\tau_n$ and $\tau_{n-1}$ coincide on the domain of $\tau_{n-1}$, we have $(x)\tau_n = (x)\tau_{n-1} \neq x$. Then Property 3 yields that $(x)\tau_n$ is not a variable.

- $(x)\tau_{n-1} = x$. Then $x$ is not in the domain of $\tau_{n-1}$. Furthermore, $x = t \in F^e_n$ where $t$ is not a variable. So from the construction of $\tau_n$ we see that $(x)\tau_n$ is not a variable.

The equation $x = t \in F^e_n$ takes the form $(x = t)\tau_n$ in $E^e_n$. Since $(x)\tau_n$ and $(t)\tau_n$ are not variables, this equation is replaced by proper sub-equations in $F^e_{n+1}$. But this contradicts the fact that equations in $(E^e_n)\bar{\tau}$ are maintained in $(E^e_{n+1})\bar{\tau}$. So apparently, $t$ must be a variable.

Thus, each equation in $E^e_{n-1}$ for $n > N(e)$ is of the form $x = y$. In $E^e_n$, such an equation takes the form $(x = y)\tau_n$, so $(x)\tau_n$ and $(y)\tau_n$ are variables too. Then Property 3 yields $(x)\tau_n = x$ and $(y)\tau_n = y$. Hence, $x$ and $y$ are not in the domain of $\tau_n$ for any $n > N(e)$, so they are not in the domain of their union $\tau$. Then $x$ and $y$ are not in the domain of $\bar{\tau}$. □

We define the 'limit' $\bar{E}$ of $E$ by

$$\bar{E} \;=\; \bigcup_{e \in E} E^e_{N(e)}.$$

Construct a unifier $\rho$ for $\bar{E}$ as follows. Two variables are said to be 'equivalent' if they can be equated by equations in $\bar{E}$. We define $\rho$ to contract the elements of each equivalence class $C$ to one variable in this class. That is, just pick some $x_0 \in C$, and put $(x)\rho = x_0$ for $x \in C$.

Finally, we define the desired unifier $\Theta$ for $E$ such that each unifier for $E$ is also a unifier for $\Theta$:

$$\Theta = \bar{\tau}\rho.$$

6. $\Theta$ is a unifier for $E$.

*Proof.* Since $\tau_n\bar{\tau} = \bar{\tau}$, also $\tau_n\Theta = \tau_n\bar{\tau}\rho = \bar{\tau}\rho = \Theta$. So $\Theta$ unifies $\tau_n$ for all $n$.

Consider an equation $x = y \in \bar{E}$. Property 5 ensures that $x$ and $y$ are not in the domain of $\bar{\tau}$, so $(x = y)\Theta = (x = y)\bar{\tau}\rho = (x = y)\rho$. Since $x$ and $y$ can be equated in $\bar{E}$, $\rho$ maps $x$ and $y$ to the same variable. So indeed this last equality holds, and thus $(x = y)\Theta$ holds. So $\Theta$ unifies $\bar{E}$.

Since $\Theta$ unifies all $\tau_n$ and $\bar{E}$, in particular it unifies $\tau_{N(e)}$ and $E^e_{N(e)}$ for each $e \in E$. Then Property 2 yields that $\Theta$ unifies $E^e_0 = \{e\}$, for each $e \in E$. □

7. Each unifier for $E$ is a unifier for $\Theta$.

*Proof.* Let $\sigma$ unify $E$. Then according to Property 1, $\sigma$ unifies $\tau_n$ and $E^e_n$ for all naturals $n$ and $e \in E$.

$\sigma$ unifies all $\tau_n$, so it unifies their union $\tau$. Since $(x)\bar{\tau} = (x)\tau^{M(x)}$, $\sigma$ unifies $\bar{\tau}$.

$\sigma$ unifies all $E^e_{N(e)}$, so it unifies their union $\bar{E}$. By definition of $\rho$, $(x)\rho$ and $x$ can be equated in $\bar{E}$ for each $x$. Hence, $(x)\rho\sigma = (x)\sigma$ for each $x$.

So, $\Theta\sigma = \bar{\tau}\rho\sigma = \bar{\tau}\sigma = \sigma$. □

# References

[1] K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B, Formal Methods and Semantics*, pages 493–574. Elsevier, 1990.

[2] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1(1):31–46, 1985.

[3] J. van Heijenoort, editor. *Jacques Herbrand: Écrits Logiques*. Presses Universitaires de France, 1968. English translation: W.D. Goldfarb, editor. *Jacques Herbrand: Logical Writings*. Reidel, 1971.

[4] J. Herbrand. *Recherches sur la Théorie de la Démonstration*. PhD thesis, Université de Paris, 1930.

[5] G.P. Huet. The undecidability of unification in third order logic. *Information and Control*, 22(3):257–267, 1973.

[6] G.P. Huet. A unification algorithm for typed $\bar{\lambda}$-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[7] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming. Volume II: Deductive Systems*. Addison-Wesley, 1990.

[8] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[9] M. Paterson and M. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, 1978.

[10] T. Pietrzykowski. A complete mechanization of second-order type theory. *Journal of the ACM*, 20(2):333–364, 1973.

[11] D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.

[12] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

# 6

# An Elimination Theorem for Regular Behaviours with Integration

Wan Fokkink

This chapter deals with an extension of the process algebra ACP with rational time and integration. We determine a proper subdomain of the regular processes for which an elimination theorem holds, namely, for each pair of processes $p_0, p_1$ in this class there is a process $q$ in this class such that $p_0 \| p_1$ and $q$ are bisimilar. Some simple examples show that if this subdomain is enlarged, then the elimination result is lost. The subdomain is equivalent to the model of timed automata from Alur and Dill.

## 6.1 Introduction

In recent years, process algebras such as CCS, CSP and ACP, have been extended with constructs that mean to describe some notion of either discrete or dense time. This chapter is based on the approach of Baeten and Bergstra [3], which extends ACP with real time. They introduced the notion of integration, which expresses the possibility that an action occurs somewhere within a time interval. The construct $\int_{v \in V} p$ executes the process $p$, where the behaviour of $p$ may depend on the value of $v$ in the time interval $V$. In this chapter, we restrict to prefix integration, and integration is parametrized by conditions, which consist of inequalities between linear expressions of variables. These notions originate from Klusener [9].

This chapter concerns regular processes. Traditionally, a process is regular if it consists of a finite number of states. However, here such a definition would not work, due to the presence of the integral construct, which causes even finite processes to have an infinite number of transitions. Therefore, a regular process is defined to be the solution of a linear specification, which is motivated by the fact that regular processes in the untimed case are exactly the solutions of linear specifications.

For the sake of verification in process algebra, it is important to have an elimination theorem which says that the parallel composition of two regular processes

is again a regular process. Namely, in general a verification deals with a process $\partial_H(p_1 \| \cdots \| p_k)$, where $p_1, ..., p_k$ are regular. Elimination theorems have been deduced for regular processes in untimed ACP (see [4]), and in timed ACP without integration [6]. In this chapter, we set out to deduce an elimination theorem for regular processes in timed ACP with integration. A simple example will show that in general, the merge can *not* be eliminated from processes in this algebra. We will determine a subclass of 'strongly' regular processes, for which an elimination theorem does hold. Furthermore, some more examples will show that the elimination result is lost if the subalgebra is enlarged in any obvious way.

At first sight, the syntactic restrictions for the subdomain of strongly regular processes may seem quite arbitrary. However, if one studies the examples more closely, then it turns out that linear specifications which do not satisfy these restrictions tend to describe different kinds of irregular behaviour, such as accelerations and oscillations. The subdomain seems to be sufficiently wide for practical purposes, see e.g. Hillebrand [8].

The subdomain of strongly regular processes can be linked to the class of timed automata from Alur and Dill [2]. It is not possible to obtain a precise translation between the processes in our subdomain and timed automata, due to the requirement of non-Zeno behaviour and the presence of fairness restrictions for languages accepted by timed automata. If these restrictions are discarded, then the classes of strongly regular processes and of timed automata turn out to be equivalent. Hence, the operations from ACP can be used to compose smaller timed automata into larger ones. This compositionality is missing in existing timed automata work.

The proof of the elimination theorem turns out to be deplorably complicated. In order to keep the exposition as simple as possible, the left merge $\between$, the communication operator $|$ and the encapsulation operator $\partial_H$ will be excluded from to the syntax. The elimination result extends to the syntax which does incorporate these constructs without any complications.

**Acknowledgements.** Steven Klusener and an anonymous referee provided helpful comments, and Frits Vaandrager suggested the link with timed automata.

## 6.2 The Syntax and Semantics

This section contains a description of the syntax and semantics for ACP with relative time and integration, denoted by ACP$rI$, together with recursion. For the elimination result it is essential that we restrict the time domain to the rational numbers. For example, if $X = a[1] \cdot X$ and $Y = b[\sqrt{2}] \cdot Y$, then the merge cannot be eliminated from $X \| Y$.

### 6.2.1 Bounds and conditions

In the sequel, we assume a countably infinite set of time variables $TVar$.

**Definition 6.1** *The set of* bounds*, with typical element $b$, is defined as follows, where $r \in \mathbb{Q}$ and $v \in TVar$.*

$$b \ ::= \ r \ | \ v \ | \ b + b \ | \ r \cdot b.$$

The expression $b_0 - b_1$ abbreviates $b_0 + (-1) \cdot b_1$.

**Definition 6.2** *The set of* conditions*, with typical element $\phi$, is defined by*

$$\phi \ ::= \ b_0 \leq b_1 \ | \ \phi \wedge \phi \ | \ \neg\phi.$$

In the sequel, we use the abbreviations $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$, and $tt$ for $\neg(1 \leq 0)$, and $ff$ for $1 \leq 0$, and $b_0 < b_1$ for $\neg(b_1 \leq b_0)$, and $b_0 = b_1$ for $b_0 \leq b_1 \wedge b_1 \leq b_0$, and $b_0 < v < b_1$ for $b_0 < v \wedge v < b_1$.

### 6.2.2 Process terms

We assume a countable alphabet $A$ of atomic actions, together with a special constant $\delta$, representing deadlock. In the sequel, $a$ and $\alpha$ denote elements of $A$ and $A \cup \{\delta\}$ respectively. Furthermore, we assume a communication function $| : A \times A \to A$, which is commutative and associative and has $\delta$ as zero element.

*Integration* enables to express that the behaviour of a process may depend on the value of a time variable. If a process $p$ depends on the value of $v$ between 1 and 2, then we write

$$\int_{1<v<2} p.$$

Here, integration is parametrized by conditions, and we deal with *prefix* integration $\int_\phi \alpha[v] \cdot p$. If $\alpha \neq \delta$, then this process can execute the action $\alpha[r]$ under the condition that $\phi[r/v]$ is true, after which the process results to $p[r/v]$.

In ACP$rI$, process terms are constructed from prefix integration, the alternative composition $x + y$, the merge $x \| y$ and the time shift $(b)x$, where $b$ is a bound. The time shift is an auxiliary operator that is needed in the operational semantics of the merge; the process $(b)p$ denotes the process $p$ that is shifted forward $b$ time units in time. As binding convention, merge, integration and time shift bind stronger than alternative composition.

Assume a finite set $V_E$ of recursion variables, where each variable $X$ in $V_E$ is supplied with an arity $ar(X)$. The recursive specification $E$ consists of a collection

$$\{X(v_1, ..., v_{ar(X)}) = t_X \mid X \in V_E\},$$

where $v_1, ..., v_{ar(X)}$ are distinct time variables, and $t_X$ is a process term constructed from integration, the alternative composition, the merge, the time shift, and expressions of the form $Y(b_1, ..., b_{ar(Y)})$, where $Y \in V_E$ and $b_1, ..., b_{ar(Y)}$ bounds.

**Definition 6.3** *$E$ is called* well-defined *if each term $t_X$ for $X \in V_E$ contains only the time variables $v_1, ..., v_{ar(X)}$.*

The set of process terms, with typical element $p$, is defined by

$$p \ ::= \ \int_\phi \alpha[v] \ \mid \ \int_\phi \alpha[v] \cdot p \ \mid \ p + p \ \mid \ p \| p \ \mid \ (b)p \ \mid \ \langle X(b_1, ..., b_{ar(X)}) | E \rangle.$$

where $E$ is well-defined.

In the sequel, $\int_{v=b} \alpha[v]$ is often abbreviated to $\alpha[b]$, and $\langle X(b_1, ..., b_{ar(X)}) | E \rangle$ to $X(b_1, ..., b_{ar(X)})$.

### 6.2.3 Free variables and substitutions

In general, one cannot attach a transition system to a process term that contains time variables which are not bound by an integral sign. Therefore, we need the notion of a time-closed process. In the term $\int_\phi \alpha[v]$, occurrences of $v$ in $\phi$ are bound, and in $\int_\phi \alpha[v] \cdot p$, occurrences of $v$ in $\phi$ and in $p$ are bound. A time variable in a process term is called free if it is not bound by any integral signs. The collection of free variables in a term $p$ is denoted by $fvar(p)$. A term $p$ is called *time-closed* if $fvar(p) = \emptyset$. As a model we take the collection $\mathcal{T}^{cl}$ of time-closed process terms, modulo bisimulation.

A substitution is a mapping from time variables to bounds. For $b$ a bound and $\sigma$ a substitution, $\sigma(b)$ denotes the bound that results from substituting $\sigma(v)$ for time variables $v$ in $b$. Substitutions extend to conditions as expected. A substitution $\sigma : TVar \to \mathbb{Q}_{\geq 0}$ is called a *valuation*.

For a process term $p$, $\sigma(p)$ is obtained by replacing free occurrences of time variables $v$ by $\sigma(v)$. In this definition however, there is one serious complication. Namely, if a free occurrence of $v$ in $p$ has been replaced by $\sigma(v)$, then variables $w$ that occur in $\sigma(v)$ may suddenly be bound in subterms $\int_\phi \alpha[w] \cdot q$ of $p$. A solution for this problem, which originates from the $\lambda$-calculus, is to allow unrestricted substitution by renaming bound variables. In the sequel, process terms are considered modulo $\alpha$-conversion, and when a substitution is applied, bound variables are renamed. Stoughton [11] presented a simple treatment of this technique.

If for a substitution $\sigma$ there are finitely many variables $v_1, ..., v_n$ such that $\sigma(v_i) \neq v_i$, then often we will use the standard notation $p[\sigma(v_1)/v_1, ..., \sigma(v_n)/v_n]$ for $\sigma(p)$.

### 6.2.4 Operational semantics

Table 6.1 contains an operational semantics for time-closed processes, taken from [9]. We focus on relative time, i.e. we assume that an expression $a[r]$ denotes an action $a$ that is executed exactly $r$ time units after the previous action has been executed. Time starts at zero, which means that actions with negative time stamps do not display any behaviour.

The timed deadlock $\delta[r]$ idles until time $r$. For example, the process $\int_{v<1} a[v] + \delta[3]$ either executes the $a$ before time 1, or idles until time 3. On the other hand, the process $\int_{v<1} a[v] + \delta[1]$ will always execute the $a$ before time 1. In order to distinguish processes that only differ in their deadlock behaviour, we introduce the

delay predicate $U_r(p)$, which holds if $p$ can idle beyond time $r$. Processes that only differ in their deadlock behaviour have distinct delays. For example, $U_2(\int_{v<1} a[v] + \delta[3])$, but not $U_2(\int_{v<1} a[v] + \delta[1])$.

The rules which define the communication operators are such that the merge does not result in arbitrary interleavings. For this would result in transitions such as $a[1] \parallel b[2] \xrightarrow{b[2]} (-2)a[1]$, where $(-2)a[1]$ does not display any behaviour. Such situations are avoided by a side condition. Namely, if a process $p$ can execute an action $a[r]$, then $p\|q$ can execute $a[r]$ only if $U_r(q)$.

In the action rules for recursion, the construct $\langle t_X[r_1/v_1, ..., r_{ar(X)}/v_{ar(X)}] \mid E\rangle$ denotes $t_X[r_1/v_1, ..., r_{ar(X)}/v_{ar(X)}]$ with each occurrence of expressions $Y(b_1, ..., b_{ar(Y)})$ replaced by $\langle Y(b_1, ..., b_{ar(Y)})|E\rangle$.

### 6.2.5 Bisimulation

Time-closed process terms are considered modulo *(strong) bisimulation*.

**Definition 6.4** *Two time-closed process terms $p_0$ and $q_0$ are* bisimilar*, denoted by $p_0 \leftrightarroweq q_0$, if there exists a symmetric binary* bisimulation relation $\mathcal{B}$ *on time-closed process terms such that*

- *$p_0 \mathcal{B} q_0$,*
- *if $p \xrightarrow{a[r]} p'$ and $p\mathcal{B}q$, then $q \xrightarrow{a[r]} q'$ for some process $q'$ with $p'\mathcal{B}q'$,*
- *if $p \xrightarrow{a[r]} \surd$ and $p\mathcal{B}q$, then $q \xrightarrow{a[r]} \surd$,*
- *if $U_r(p)$ and $p\mathcal{B}q$, then $U_r(q)$.*

The rules in Table 6.1 can be fit into the congruence format for action rules with types, data and variable binding of Bloom and Vaandrager [5] (see Chapter 7). Strong bisimulation defined by transition rules within this format is always a congruence on the algebra of closed terms, which means that if $p \leftrightarroweq p'$ and $q \leftrightarroweq q'$, then $p + q \leftrightarroweq p' + q'$ and $p\|q \leftrightarroweq p'\|q'$ and $\int_\phi \alpha[v] \cdot p \leftrightarroweq \int_\phi \alpha[v] \cdot p'$ and $(b)p \leftrightarroweq (b)p'$.

## 6.3 An Elimination Theorem

### 6.3.1 Regular processes

Usually, a process is called regular if it has a finite number of states. In the present setting this definition would backfire, since the integral construct causes even finite processes to have an infinite number of states. In untimed ACP one can prove, for suitable models, that a process is regular if and only if it is equal to a solution of a linear recursive specification [10]. Here, we use this property as the definition of regularity. A recursive specification is linear if its equations are of the form

$$X(v_1, ..., v_{ar(X)}) = \sum_i \int_{\phi_i} a_i[w] \cdot Y_i(b_{i1}, ..., b_{iar(Y_i)}) + \sum_j \int_{\psi_j} \alpha_j[w].$$

$$\frac{\phi[r/v] \quad r>0}{\int_\phi a[v] \xrightarrow{a[r]} \surd} \qquad \frac{\phi[r/v] \quad r>0}{\int_\phi a[v]\cdot x \xrightarrow{a[r]} x[r/v]}$$

$$\frac{x \xrightarrow{a[r]} \surd}{x+y \xrightarrow{a[r]} \surd \xleftarrow{a[r]} y+x} \qquad \frac{x \xrightarrow{a[r]} x'}{x+y \xrightarrow{a[r]} x' \xleftarrow{a[r]} y+x}$$

$$\frac{x \xrightarrow{a[r]} \surd \quad U_r(y)}{x\|y \xrightarrow{a[r]} (-r)y \xleftarrow{a[r]} y\|x} \qquad \frac{x \xrightarrow{a[r]} x' \quad U_r(y)}{x\|y \xrightarrow{a[r]} x'\|(-r)y \quad y\|x \xrightarrow{a[r]} (-r)y\|x'}$$

$$\frac{x \xrightarrow{a[r]} \surd \quad y \xrightarrow{a'[r]} \surd}{x\|y \xrightarrow{(a|a')[r]} \surd} \qquad \frac{x \xrightarrow{a[r]} x' \quad y \xrightarrow{a'[r]} y'}{x\|y \xrightarrow{(a|a')[r]} x'\|y'}$$

$$\frac{x \xrightarrow{a[r]} \surd \quad y \xrightarrow{a'[r]} y'}{x\|y \xrightarrow{(a|a')[r]} y' \xleftarrow{(a|a')[r]} y\|x}$$

$$\frac{x \xrightarrow{a[r]} \surd \quad r+b=s>0}{(b)x \xrightarrow{a[s]} \surd} \qquad \frac{x \xrightarrow{a[r]} x' \quad r+b=s>0}{(b)x \xrightarrow{a[s]} x'}$$

$$\frac{\langle t_X[r_1/v_1,...,r_{ar(X)}/v_{ar(X)}] \mid E\rangle \xrightarrow{a[r]} \surd}{\langle X(r_1,...,r_{ar(X)}) \mid E\rangle \xrightarrow{a[r]} \surd} \qquad \frac{\langle t_X[r_1/v_1,...,r_{ar(X)}/v_{ar(X)}] \mid E\rangle \xrightarrow{a[r]} y}{\langle X(r_1,...,r_{ar(X)}) \mid E\rangle \xrightarrow{a[r]} y}$$

$$\frac{\phi[s/v] \quad 0<r<s}{U_r(\int_\phi \alpha[v]) \quad U_r(\int_\phi \alpha[v]\cdot x)} \qquad \frac{U_r(x)}{U_r(x+y) \quad U_r(y+x)}$$

$$\frac{U_r(x) \quad r+b=s>0}{U_s((b)x)} \qquad \frac{0<r<b}{U_r((b)x)}$$

$$\frac{U_r(x) \quad U_r(y)}{U_r(x\|y)} \qquad \frac{U_r(t_X[b_1/v_1,...,b_{ar(X)}/v_{ar(X)}])}{U_r(X(b_1,...,b_{ar(X)}))}$$

Table 6.1: Action rules for ACP$rI$

A time-closed process is *regular* if it is bisimilar to a process $\langle X|E\rangle$, where $E$ is linear.

### 6.3.2 A counter-example

We want to prove an elimination theorem for a class of regular behaviours. An easy example will learn that such a theorem does not hold for the full class. This example uses the following lemma.

**Lemma 6.5** *Let $Q(p)$ denote the collection of rationals $r$ for which there exists a transition $p \xrightarrow{a[r]} p'$ where $p'$ can deadlock. If $p$ is regular, then $Q(p)$ consists of a finite number of intervals.*

**Proof.** Omitted. (It follows from the refinement lemma in Chapter 7.)

The following example shows that the merge of two regular processes is not always a regular process.

**Example 6.6** *Define*

$$\begin{aligned} X &= \textstyle\int_{0<w<1} a[w] \cdot Y(w) \\ Y(v) &= \textstyle\int_{w=v} a[w] \cdot Y(w). \end{aligned}$$

*The process $X \| a'[1]$ (with $a|a' = \delta$) is not regular.*

Each trace of the process $X$ is of the form $a[r] \cdot a[r] \cdot a[r] \cdot ...$ with $r \in \langle 0, 1\rangle$. If $r \in \langle 1/(n+1), 1/n\rangle$ for a natural $n$, then $X \| a'[1]$ will execute $a[r]$ $n$ times, followed by $a'[1 - nr]$, then $a[(n+1)r - 1]$, and after that only $a[r]$'s. And if $r = 1/n$, then $X \| a'[1]$ will get into a deadlock after $n - 1$ times executing $a$. So according to Lemma 6.5, $X \| a'[1]$ is not regular. *(End example)*

### 6.3.3 Strongly regular processes

According to Example 6.6, the class of regular processes is too large for an elimination theorem. On the other hand, if no occurrences of of time dependencies are allowed, then the collection is too small. Because in this class equivalences such as

$$\int_{0<v<1} a[v] \parallel \int_{1<w<2} a'[w] \;\leftrightarrows\; \int_{0<v<1} a[v] \cdot \int_{1-v<w<2-v} a'[w]$$

cannot be expressed anymore. Godskesen and Larsen [7] provided a rigorous proof that time dependencies are essential in order to obtain an expansion theorem in a timed setting. (Aceto and Murphy [1] proposed the notion of 'ill-timed' traces, in order to obtain an expansion theorem in the absence of time dependencies.)

So, is there an algebra in between, for which an elimination theorem can be deduced? The answer is yes.

**Definition 6.7** *A time-closed process is* strongly regular *if it is bisimilar to a process $\langle X_0|E\rangle$, where $E$ consists of equations*

$$X_I(v_1,...,v_{ar(X)}) = \sum_k \int_{\phi_k} a_k[w] \cdot X_{I_k}(b_{k1},...,b_{kar(Y_k)}) + \sum_l \int_{\psi_l} \alpha_l[w],$$

*that satisfy the following requirements.*

1. *The $\phi_k$ and the $\psi_l$ are in the class of conditions that is defined by*

$$\phi \ ::= \ w \leq b \ \mid \ b \leq w \ \mid \ \phi \wedge \phi \ \mid \ \neg\phi,$$

   *where $b$ is of the form $r$ or $r - v_i$.*

2. *The bounds $b_{kj}$ are of the form $r$ or $w + r$ or $v_i + w + r$.*

**Lemma 6.8** *Each strongly regular process can be described by a linear specification with the following more restrictive constraints.*

1. *The bounds $b$ in the $\phi_k$ and the $\psi_l$ are of the form $r$ or $r - v_i$ with $r \geq 0$.*

2. *The bounds $b_{kj}$ are of the form or $w$ or $v_i + w$.*

**Proof sketch.**

1. If a bound $b$ in the $\phi_k$ and the $\psi_l$ is of the form $r$ or $r - v_i$ with $r < 0$, then $r$ can be replaced by 0, because actions only occur after time zero.

2. If a bound $b_{kj}$ is of the form $r$ or $w + r$ or $v_i + w + r$, then $r$ can be removed by introducing a new recursion variable $\tilde{Y}_k(v_1,...,v_{ar(Y_k)})$, of which the linear equation is obtained by replacing expressions $s - v_j$ in the conditions of the equation of $Y_k(v_1,...,v_{ar(Y_v)})$ by $(s - r) - v_j$. □

We prove an elimination theorem for the algebra of strongly regular processes.

### 6.3.4 Two counter-examples

First, we present two more examples to show that this elimination result would get lost if the definition of strong regularity were less restrictive. Example 6.6 already indicated that this definition cannot be loosened by allowing not only expressions $r$ and $r - v_i$, but also variables $v_i$ as bounds $b$. The following example indicates that neither one can allow variables $v_i$ as bounds $b_{kj}$.

**Example 6.9** *Define*

$$X = \int_{0<w<1} a[w] \cdot Y(w)$$

$$Y(v) = \int_{w=1-v} a[w] \cdot Y(v).$$

*The process $X \| a'[2]$ (with $a|a' = \delta$) is not regular.*

Each trace of the process $X$ is of the form $a[r] \cdot a[1-r] \cdot a[1-r] \cdot ...$ with $r \in \langle 0,1 \rangle$. If $r \in \langle (n-2)/(n-1), (n-1)/n \rangle$ for some $n \geq 2$, then $X \| a'[2]$ will first execute $n+1$ $a$'s, then an $a'$ and then only $a$'s. And if $r = (n-1)/n$, then $X \| a'[2]$ will get into a deadlock after $n+1$ times executing $a$. So according to Lemma 6.5, $X \| a'[2]$ is not regular. *(End example)*

Finally, the following example indicates that one cannot allow expressions $r - sv_i$ as bounds $b$, where $s \in \mathbb{Q}$. This example is more involved than the previous ones.

**Example 6.10** *Define*

$$X_1 = \int_{0<v<1} a[v] \cdot X_2(v) \qquad Y_1 = a'[\tfrac{1}{2}] \cdot Y_2$$

$$X_2(v) = \int_{w=\frac{3}{2}-\frac{1}{2}v} a[w] \cdot X_2(w) \qquad Y_2 = a'[1] \cdot Y_2.$$

*The process $X_1 \| Y_1$ (with $a|a' = \delta$) is not regular.*

Clearly, the $n$th $a'$-action of $Y_1$ is executed at absolute time $n - 1/2$. An easy calculation learns that if $X_1$ executes its first action at time $r$, then its $n$th action will be executed at absolute time $n - (1-r)t_n$, where

$$t_n = \sum_{i=1}^{n} (-\frac{1}{2})^{i-1}.$$

So if $(1-r)t_n = 1/2$ for some $n$, then $X_1 \| Y_1$ will get into a deadlock after $n-1$ times executing $a$. The equalities $r = 1 - 1/(2t_n)$ for $n = 1, 2, ...$ yield an infinite partition of the interval $\langle 0,1 \rangle$, so according to Lemma 6.5 $X_1 \| Y_1$ is not regular. *(End example)*

### 6.3.5 Orderings on bounds

Before giving the proof of the elimination theorem for strongly regular processes, first we present some definitions and results on orderings on bounds that shall be crucial ingredients in this proof.

Assume a finite collection $B$ of bounds. An *ordering* $\mathcal{O}$ on $B$ is determined by a partition $B_1, ..., B_n$ of non-empty subsets of $B$, where $\cup_{i=1}^{n} B_i = B$, and $B_i \cap B_j = \emptyset$ if $i \neq j$. The condition $\mathcal{O}$ consists of the conjunct of expressions $b = b'$ for $b, b' \in B_i$ and $b < b'$ for $b \in B_i$ and $b' \in B_j$ with $i < j$. We say that two conditions $\phi$ and $\phi'$ are *equivalent under* $\mathcal{O}$ if for each valuation $\sigma$, $\sigma(\phi \wedge \mathcal{O})$ is true if and only if $\sigma(\phi' \wedge \mathcal{O})$ is true.

Fix a bound $b_i \in B_i$ for $i = 1, ..., n$, and fix a variable $w$ which does not occur in bounds in $B$. We define conditions $\psi_i$ which express all possible positions of $w$ with respect to the bounds $b_i$ under the ordering $\mathcal{O}$.

| | | | |
|---|---|---|---|
| $\psi_1$ | denotes | $w < b_1$ | |
| $\psi_{2i}$ | denotes | $w = b_i$ | for $i = 1, ..., n$ |
| $\psi_{2i+1}$ | denotes | $b_i < w < b_{i+1}$ | for $i = 1, ..., n-1$ |
| $\psi_{2n+1}$ | denotes | $b_n < w$ | |

Note that $\mathcal{O} \wedge \psi_i$ determines an ordering on $B \cup \{w\}$.

Consider the class $C$ of conditions defined by

$$\phi \ ::= \ b \leq w \mid w \leq b \ \mid \ \phi \wedge \phi \ \mid \ \neg\phi,$$

where $b \in B$.

**Lemma 6.11** *Under $\mathcal{O}$, each condition in $C$ is equivalent to a condition $\vee_{i\in I}\psi_i$.*

**Proof sketch.** Let $\phi$ be in $C$. We apply induction on the size of $\phi$. If $\phi$ is of the form $b \leq w$ with $b \in B_i$, then $\phi$ is equivalent to $\psi_{2i} \vee \cdots \vee \psi_{2n+1}$ under $\mathcal{O}$. If $\phi$ is of the form $w \leq b$ with $b \in B_i$, then $\phi$ is equivalent to $\psi_1 \vee \cdots \vee \psi_{2i}$ under $\mathcal{O}$.

Next, let $\phi$ be of the form $\phi_0 \wedge \phi_1$, where $\phi_0$ is equivalent to $\vee_{i\in I_0}\psi_i$ and $\phi_1$ is equivalent to $\vee_{i\in I_1}\psi_i$ under $\mathcal{O}$. Then $\phi$ is equivalent to $\vee_{i\in I_0\cap I_1}\psi_i$ under $\mathcal{O}$.

Finally, let $\phi$ be of the form $\neg\phi_0$, where $\phi_0$ is equivalent to $\vee_{i\in I}\psi_i$ under $\mathcal{O}$. Then $\phi$ is equivalent to $\vee_{i\in\{1,\ldots,2n+1\}\setminus I}\psi_i$ under $\mathcal{O}$. □

Hence, for each $\phi \in C$ we can calculate a condition $u(\phi \wedge \mathcal{O})$ which describes the ultimate delay of $\int_{\phi\wedge\mathcal{O}} \alpha[w]$ as follows.

**Definition 6.12** *Let $\phi$ be equivalent to $\vee_{i\in I}\psi_i$ under $\mathcal{O}$.*

- *If $I = \emptyset$, then put $u(\phi \wedge \mathcal{O})$ is ff.*
- *If $I$ has maximum $2i-1$ or $2i$ for some $i = 1, \ldots, n$, then put $u(\phi \wedge \mathcal{O})$ is $w < b_i$.*
- *If $I$ has maximum $2n+1$, then put $u(\phi \wedge \mathcal{O})$ is tt.*

For $B$ a finite set of bounds, and $r \in \mathbb{Q}$ and $N \in \mathbb{N}$, let $\mathcal{B}_{r,N}(B)$ denote the finite set of bounds

$$\{kr, \ kr - b \mid k = 0, \ldots, N, \ b \in B\}.$$

**Lemma 6.13** *For each ordering $\mathcal{O}$ of $\mathcal{B} = \mathcal{B}_{r,N}(v_1, \ldots, v_n) \cup \{w\}$, there is an ordering $\tilde{\mathcal{O}}$ of $\tilde{\mathcal{B}} = \mathcal{B}_{r,N}(w, v_1 + w, \ldots, v_n + w)$ such that if $\sigma(\mathcal{O})$ is true for a valuation $\sigma$, then $\sigma(\tilde{\mathcal{O}})$ is true.*

**Proof sketch.** Assume an ordering $\mathcal{O}$ on $\mathcal{B}$. We construct the desired ordering $\tilde{\mathcal{O}}$ on $\tilde{\mathcal{B}}$ by rewriting each possible relation in $\tilde{\mathcal{B}}$ to a relation in $\mathcal{B}$.

- The $\tilde{\mathcal{O}}$-order of $kr - (v_i + w)$ and $lr - (v_j + w)$ is the $\mathcal{O}$-order of $kr - v_i$ and $lr - v_j$.
- The $\tilde{\mathcal{O}}$-order of $kr - (v_i + w)$ and $lr - w$ is the $\mathcal{O}$-order of $kr - v_i$ and $lr$.
- If $k \geq l$, then the $\tilde{\mathcal{O}}$-order of $kr - (v_i + w)$ and $lr$ is the $\mathcal{O}$-order of $(k-l)r - v_i$ and $w$. Otherwise, if $k < l$, then $kr - (v_i + w) <_{\tilde{\mathcal{O}}} lr$.

- The $\tilde{\mathcal{O}}$-order of $kr - w$ and $lr - w$ is the $\mathcal{O}$-order of $kr$ and $lr$.
- If $k \geq l$, then the $\tilde{\mathcal{O}}$-order of $kr - w$ and $lr$ is the $\mathcal{O}$-order of $(k-l)r$ and $w$. Otherwise, if $k < l$, then $kr - w <_{\tilde{\mathcal{O}}} lr$.
- The $\tilde{\mathcal{O}}$-order of $kr$ and $lr$ is the $\mathcal{O}$-order of $kr$ and $lr$.

It follows immediately from the construction of $\tilde{\mathcal{O}}$ that if $\sigma(\mathcal{O})$ is true, then $\sigma(\tilde{\mathcal{O}})$ is true. □

### 6.3.6 The main theorem

**Theorem 6.14** *For each pair of strongly regular processes $p_0$ and $p_1$, there exists a strongly regular process $q$ such that $p_0 \| p_1 \leftrightarrows q$.*

**Proof sketch.** According to Lemma 6.8, $p_0$ and $p_1$ are bisimilar to processes $\langle X_0 | E_0 \rangle$ and $\langle X_1 | E_1 \rangle$, where the equations in $E_0$ and $E_1$ are of the form

$$X_I(v_{I1}, ..., v_{In(I)}) = \sum_{k \in K_I} \int_{\phi_k} a_k[w] \cdot X_{I_k}(b_{k1}, ..., b_{kn(I_k)}) + \sum_{l \in L_I} \int_{\phi'_l} \alpha_l[w],$$

such that

- the $\phi_k$ and the $\phi'_l$ are constructed from $w \leq b$ and $b \leq w$ and $\wedge$ and $\neg$, where $b$ is of the form $r$ or $r - v_{Ii}$ with $r \geq 0$,
- the bounds $b_{kj}$ are of the form $w$ or $v_{Ii} + w$.

We construct a suitable linear specification which describes the behaviour of $p_0 \| p_1$.

First, we introduce for each $I$ a fresh recursion variable $Y_I(v, v_{I1}, ..., v_{In(I)})$. Its linear equation is obtained from the equation of $X_I(v_{I1}, ..., v_{In(I)})$ by replacing conditions in $\phi_k$ and $\phi'_l$ of the form $w \leq r$ or $r \leq w$ by $w \leq r - v$ or $r - v \leq w$ respectively. Let $\tilde{\phi}_k$ and $\tilde{\phi}'_l$ denote the resulting conditions, and put

$$Y_I(v, v_{I1}, ..., v_{In(I)}) = \sum_{k \in K_I} \int_{\tilde{\phi}_k} a_k[w] \cdot X_{I_k}(b_{k1}, ..., b_{kn(I_k)}) + \sum_{l \in L_I} \int_{\tilde{\phi}'_l} \alpha_l[w].$$

For $v \geq 0$, $Y_I(v, v_{I1}, ..., v_{In(I)})$ yields the behaviour of $(-v)X_I(v_{I1} - v, ..., v_{In(I)} - v)$.

Next, we introduce a fresh recursion variable $Z_{IJ}(v, v_{I1}, ..., v_{In(I)}, v_{J1}, ..., v_{Jn(J)})$ for each $I$ and $J$, and construct its linear equation such that it describes the behaviour of $Y_I(v, v_{I1}, ..., v_{In(I)}) \| X_J(v_{J1}, ..., v_{Jn(J)})$. Only, it turns out that this behaviour cannot be described by a single linear equation, so we introduce a collection of variables $Z_{IJ}^{\mathcal{O}}$, where $\mathcal{O}$ ranges over the orderings on a collection of bounds $\mathcal{B}$.

This collection $\mathcal{B}$ is defined as follows. Ensure, by means of $\alpha$-conversion, that $v$ and the $v_{Ii}$ and the $v_{Ji}$ are distinct variables. Let $R$ denote the finite collection of rationals that occur in $E_0$ or in $E_1$. Define $N = \max\{r/r_0 \mid r \in R\}$, where $r_0$ is the greatest rational such that $r/r_0$ is a natural number for each $r \in R$. Put

$$\mathcal{B} = \mathcal{B}_{r_0,N}(v, v_{I1}, ..., v_{In(I)}, v_{J1}, ..., v_{Jn(J)}).$$

Fix an ordering $\mathcal{O}$ on $\mathcal{B}$. The bounds $b$ in the $\phi_k,\ \phi'_l,\ \tilde{\phi}_k,\ \tilde{\phi}'_l$ are of the form $kr_0$ or $kr_0 - v$ or $kr_0 - v_{Ii}$ or $kr_0 - v_{Ji}$ with $k = 0, ..., N$, so they are all in $\mathcal{B}$. Hence, according to Definition 6.12, we can define conditions $u_I$ and $\tilde{u}_I$ which yield the ultimate delays of $X_I(v_{I1}, ..., v_{In(I)})$ and $Y_I(v, v_{I1}, ..., v_{In(I)})$ under $\mathcal{O}$ respectively, namely

$$u_I = u((\vee_{k\in K_I}\phi_k \vee \vee_{l\in L_I}\phi'_l) \wedge \mathcal{O}),$$

$$\tilde{u}_I = u((\vee_{k\in K_I}\tilde{\phi}_k \vee \vee_{l\in L_I}\tilde{\phi}'_l) \wedge \mathcal{O}).$$

We construct the linear equation of $Z^{\mathcal{O}}_{IJ}$. The behaviour of $Y_I \| X_J$ originates from

**(a)** executing an initial action from $Y_I$,

**(b)** executing an initial action from $X_J$,

**(c)** executing a communication of initial actions from these two processes.

We describe these behaviours separately, under the ordering $\mathcal{O}$, which is determined by a partition $B_1, ..., B_n$ of $\mathcal{B}$. Fix $b_i \in B_i$ for $i = 1, ..., n$, and let $\psi_1, ..., \psi_{2n+1}$ denote the conditions $w < b_1,\ w = b_1,\ b_1 < w < b_n, ...,\ b_n < w$ respectively.
**(a)** Recall that a process $p\|q$ can only execute an initial action from $p$ before the ultimate delay of $q$. So initial actions from $Y_I(v, v_{I1}, ..., v_{In(I)})$ under $\mathcal{O}$ can only be executed under the condition $u_J$. After executing such an initial action $a_k[w]$ for some $k \in K_I$ under the condition $\phi_k \wedge u_J$, the resulting state is

$$X_{I_k}(b_{k1}, ..., b_{kn(I_k)}) \| (-w)X_J(v_{J1}, ..., v_{Jn(J)}).$$

This process is described by $Z^{\tilde{\mathcal{O}}}_{JI_k}(w, v_{J1} + w, ..., v_{Jn(J)} + w, b_{k1}, ..., b_{kn(I_k)})$ for some ordering $\tilde{\mathcal{O}}$ on $\tilde{\mathcal{B}} = \mathcal{B}_{r_0,N}(w, v_{J1} + w, ..., v_{Jn(J)} + w, b_{k1}, ..., b_{kn(I_k)})$. This ordering $\tilde{\mathcal{O}}$ can be determined as follows. The ordering $\mathcal{O}$ on $\mathcal{B}$ together with a condition $\psi_i$ for $i = 1, ..., 2n + 1$ determine an ordering $\mathcal{O}_i$ on $\mathcal{B} \cup \{w\}$. Since $b_{k1}, ..., b_{kn(I_k)} \in \{w, v_{I1} + w, ..., v_{In(I)} + w\}$, Lemma 6.13 says that there is an ordering $\tilde{\mathcal{O}}_i$ on $\tilde{\mathcal{B}}$ such that if $\sigma(\mathcal{O}_i)$ is true for a valuation $\sigma$, then $\sigma(\tilde{\mathcal{O}}_i)$ is true. Hence, we add the following subterms to the equation of $Z^{\mathcal{O}}_{IJ}$.

$$\sum_{i=1}^{2n+1} \int_{\tilde{\phi}_k \wedge u_J \wedge \psi_i} a_k[w] \cdot Z^{\tilde{\mathcal{O}}_i}_{JI_k}(w, v_{J1} + w, ..., v_{Jn(J)} + w, b_{k1}, ..., b_{kn(I_k)}).$$

After executing an initial $\alpha_l[w]$ for some $l \in L_I$ under the condition $\tilde{\phi}'_l \wedge u_J$, the resulting state is $(-w)X_J(v_{J1}, ..., v_{Jn(J)})$, which is described by $Y_J(w, v_{J1} + w, ..., v_{Jn(J)} + w)$. So we add the following subterm to the equation of $Z^{\mathcal{O}}_{IJ}$.

$$\int_{\tilde{\phi}'_l \wedge u_J} \alpha_l[w] \cdot Y_J(w, v_{J1} + w, ..., v_{Jn(J)} + w).$$

**(b)** Next, we construct the subterm that originates from executing an initial action of $X_J(v_{J1}, ..., v_{Jn(J)})$. Such initial actions can only be executed under the condition

$\tilde{u}_I$. After executing an initial action $a_k[w]$ for some $k \in K_J$ under the condition $\phi_k \wedge \tilde{u}_I$, the resulting state is

$$(-w)Y_I(v, v_{I1}, ..., v_{In(I)}) \| X_{J_k}(b_{k1}, ..., b_{kn(J_k)}),$$

which is described by $Z^{\tilde{\mathcal{O}}}_{IJ_k}(v + w, v_{I1} + w, ..., v_{In(I)} + w, b_{k1}, ..., b_{kn(J_k)})$. Again, the ordering $\mathcal{O}$ on $\mathcal{B}$ together with a condition $\psi_i$ for $i = 1, ..., 2n + 1$ determine an ordering $\tilde{\mathcal{O}}_i$ on $\mathcal{B}_{r_0,N}(v + w, v_{I1} + w, ..., v_{In(I)} + w, b_{k1}, ..., b_{kn(J_k)})$. So we add the following subterms to the equation of $Z^{\mathcal{O}}_{IJ}$.

$$\sum_{i=1}^{2n+1} \int_{\phi_k \wedge \tilde{u}_I \wedge \psi_i} a_k[w] \cdot Z^{\tilde{\mathcal{O}}_i}_{IJ_k}(v + w, v_{I1} + w, ..., v_{In(I)} + w, b_{k1}, ..., b_{kn(J_k)}).$$

After executing an initial $\alpha_l[w]$ for some $l \in L_J$ under the condition $\phi'_l \wedge \tilde{u}_I$, the resulting state is $(-w)Y_I(v, v_{J1}, ..., v_{Jn(J)})$, which is described by $Y_I(v + w, v_{J1} + w, ..., v_{Jn(J)} + w)$. So we add the following subterm to the equation of $Z^{\mathcal{O}}_{IJ}$.

$$\int_{\phi'_l \wedge \tilde{u}_I} \alpha_l[w] \cdot Y_I(v + w, v_{J1} + w, ..., v_{Jn(J)} + w).$$

**(c)** Finally, we construct the subterms that originate from executing the communication of initial actions of $Y_I$ and $X_J$.

Let $k \in K_I$ and $k' \in K_J$. After executing the action $(a_k|a_{k'})[w]$ under the condition $\tilde{\phi}_k \wedge \phi_{k'}$, the resulting state is $X_{I_k}(b_{k1}, ..., b_{kn(I_k)}) \| X_{J_{k'}}(b_{k'1}, ...b_{k'n(J_{k'})})$. Again, the ordering $\mathcal{O}$ on $\mathcal{B}$ together with a condition $\psi_i$ for $i = 1, ..., 2n + 1$ determine an ordering $\tilde{\mathcal{O}}_i$. So we add the following subterms to the equation of $Z^{\mathcal{O}}_{IJ}$.

$$\sum_{i=1}^{2n+1} \int_{\tilde{\phi}_k \wedge \phi_{k'} \wedge \psi_i} (a_k|a_{k'})[w] \cdot Z^{\tilde{\mathcal{O}}_i}_{I_k J_{k'}}(0, b_{k1}, ..., b_{kn(I_k)}, b_{k'1}, ..., b_{k'n(J_{k'})}).$$

For $k \in K_I$ and $l \in L_J$ we add the subterm

$$\int_{\tilde{\phi}_k \wedge \phi'_l} (a_k|\alpha_l)[w] \cdot Y_{I_k}(0, b_{k1}, ..., b_{kn(I_k)}).$$

For $l \in L_I$ and $k \in K_J$ we add the subterm

$$\int_{\tilde{\phi}'_l \wedge \phi_k} (\alpha_l|a_k)[w] \cdot Y_{J_k}(0, b_{k1}, ..., b_{kn(J_k)}).$$

Finally, for $l \in L_I$ and $l' \in L_J$, we add the subterm

$$\int_{\tilde{\phi}'_l \wedge \phi'_{l'}} (\alpha_l|\alpha_{l'})[w].$$

This finishes the construction of the linear equation of $Z^{\mathcal{O}}_{IJ}$.

Let $E$ denote the recursive specification that consists of the linear equations of the $Y_I$ and the $Z^{\mathcal{O}}_{IJ}$. Note that $E$ satisfies the constraints in Definition 6.7 for strongly regular processes. Consider the strongly regular process $\langle Z^{\mathcal{O}}_{01}(0)|E\rangle$, where the ordering $\mathcal{O}$ on $\mathcal{B}_{r_0,N}(v)$ is true under the valuation that maps $v$ to 0. The construction ensures that this process is bisimilar to $\langle X_0|E_0\rangle \| \langle X_1|E_1\rangle$, which finishes the proof of the elimination theorem. □

### 6.3.7 An example

We present an example of a merge of two simple strongly regular processes, which itself can only be described by a much more complicated linear specification.

**Example 6.15** *Define*

$$X = \int_{0<v<1} a[v] \cdot X.$$

*Let* $p = X \| a'[k]$ *(with* $a|a' = \delta$*), where* $k \in \mathbb{N}$*.*

The behaviour of $p$ can be described by the following linear specification.

$$\begin{aligned}
X_0 &= \textstyle\int_{0<v<1} a[v] \cdot X_1(v) \\
X_i(v) &= \textstyle\int_{0<w\leq i-v} a[w] \cdot X_i(v+w) + \int_{i-v<w<1} a[w] \cdot X_{i+1}(v+w) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, ..., k-1 \\
X_k(v) &= \textstyle\int_{0<w<k-v} a[w] \cdot X_k(v+w) + \int_{w=k-v} a'[w] \cdot Y(w) \\
Y(v) &= \textstyle\int_{0<w<1-v} a[w] \cdot X \\
X &= \textstyle\int_{0<v<1} a[v] \cdot X.
\end{aligned}$$

The idea behind this specification is quite easy. Process $p$ will execute $X$ until it reaches (absolute) time $k$, when it executes $a'$, after which it continues with $X$. The process $p$ has the possibility of executing $a'$ or at time $k$ if it has executed an $a$ after time $k-1$. So if this is the case, then the linear specification must take into account the execution of $a'$ at $k$. Similarly, $p$ can execute an $a$ after $k-1$ if it has executed an $a$ after $k-2$. So if this is the case, the linear specification must take into account the execution of $a$ after $k-1$, etc.

The equations of the $X_i$ for $i = 1, ..., k-1$ register whether $a$ is executed after time $i$ or not. If so, then $X_{i+1}$ is triggered, and otherwise $X_i$ is repeated. Finally, $X_k$ takes into account the execution of $a'$. *(End example)*

## 6.4 Timed Automata

An automaton consists of a set of states $S$, a set of start states $S_0 \subseteq S$, a set of labels $A$ and a set of transitions $E \subseteq S \times A \times S$. The language accepted by the automaton consists of all traces $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} ...$ such that $(s_i, a_i, s_{i+1}) \in E$ for $i = 0, 1, 2, ...$ Furthermore, the trace may have to satisfy certain fairness requirements, e.g. that it reaches a specific state an infinite number of times.

In the extension of automata with time from Alur and Dill [2], the elements of $E$ are supplied with time constraints on 'clock variables'. Such constraints are

disjunctions of expressions of the form $x \Box r$, with $x$ a clock variable and $r \in \mathbb{Q}_{>0}$ and $\Box \in \{<, >, \leq, \geq\}$. Moreover, there is a construct $x := 0$, which denotes that while executing a transition, clock $x$ is set back to zero. A trace is only accepted by a timed automaton if its transitions are performed at times that all clocks satisfy their constraints. Furthermore, accepted traces must satisfy the required fairness constraints. Finally, Zeno behaviour is excluded from timed automata, i.e. traces are only accepted if they progress beyond any moment in time.

The algebra of strongly regular processes can be linked to the class of timed automata. The fairness restrictions and the non-Zeno requirement are obstacles for the translation between timed automata and strongly regular processes, since ACP$rI$ does not take into account such semantic restrictions. However, if these restrictions are discarded, then the classes of strongly regular processes and of timed automata turn out to be equivalent.

A strongly regular process can be translated to the setting of timed automata as follows. A strongly regular processes executes an action $a[w]$ under restrictions of the form $w \Box r$ or $w \Box (r - v_i)$, with $r > 0$ and $\Box \in \{<, >, \leq, \geq\}$. These last inequalities can be rewritten to the form $(v_i + w) \Box r$. The $v_i$ and $w$ can be regarded as clocks, where $w$ has been set back to zero by $w := 0$ in the previous transition. The state that results after the execution of $a[w]$ is a recursive expression of the form $X(w, v_1 + w, ..., v_k + w)$. The $v_i + w$ and $w$ store the actual times of the clocks at the moment of the transition $a[w]$.

By means of the converse translation, it is easy to see that the language accepted by a timed automaton (without semantic restrictions) can always be described by a strongly regular process. We give a simple example.

**Example 6.16** *Consider a timed automaton with states $s_0, s_1$, with start state $s_0$, and clock variables $x, y$, which is defined by the following two transitions:*

- *$(s_0, a, s_1)$ with time constraints $x < 2,\ y := 0$,*
- *$(s_1, a', s_0)$ with time constraints $x < 3,\ y < 2,\ x := 0$.*

This timed automaton executes alternately $a$ and $a'$. Define

$$\begin{aligned} X &= \int_{0<w\leq 1} a[w] \cdot Y_1 + \int_{1\leq w<2} a[w] \cdot Y_2(w) \\ Y_1 &= \int_{0<w<2} a'[w] \cdot X \\ Y_2(v) &= \int_{0<w<3-v} a'[w] \cdot X. \end{aligned}$$

The process $X$ describes the behaviour of the automaton. *(End example)*

# References

[1] L. Aceto and D. Murphy. On the ill-timed but well-caused. In E. Best, editor, *Proceedings 4th Conference on Concurrency Theory (CONCUR'93)*, Hildesheim, *LNCS 715*, pages 97–111. Springer-Verlag, 1993.

[2] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[4] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[5] B. Bloom and F.W. Vaandrager. SOS rule formats for parametrized and state-bearing processes. Unpublished manuscript, 1994.

[6] W.J. Fokkink. Regular processes with rational time and silent steps. Report CS-R9231, CWI, Amsterdam, 1992.

[7] J.C. Godskesen and K.G. Larsen. Real time calculi and expansion theorems. In R. Shyamasundar, editor, *Proceedings 12th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 12),* New Delhi, *LNCS 652*, pages 302–315. Springer-Verlag, 1992.

[8] J.A. Hillebrand. The ABP and the CABP – a comparison of performances in real time process algebra. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Proceedings 1st Workshop on Algebra of Communicating Processes (ACP'94),* Utrecht, Workshops in Computing, pages 124–147. Springer-Verlag, 1994.

[9] A.S. Klusener. Completeness in real time process algebra. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings 2nd Conference on Concurrency Theory (CONCUR'91),* Amsterdam, *LNCS 527*, pages 376–392. Springer-Verlag, 1991.

[10] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[11] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.

# 7

# An Effective Axiomatization for Real Time ACP

Wan Fokkink & Steven Klusener

Baeten and Bergstra added real time to ACP, and introduced the notion of integration, which expresses the possibility of an action happening within a time interval. In order to axiomatize this feature, they needed an 'uncountable' axiom. This chapter deals with prefix integration, and integration is parametrized by conditions, which are inequalities between linear expressions of variables. We present an axiomatization for process terms, and propose a strategy to decide bisimulation equivalence between process terms, by means of this axiomatization.

## 7.1 Introduction

In recent years, many process algebras have been extended with some notion of time. This chapter is based on the approach of Baeten and Bergstra [3], who extended ACP with real time. Their algebra concerns (closed) process terms, constructed from timed actions $a(t)$, which denote the process that executes action $a$ at time $t$. This results in identities that do not hold in untimed ACP, such as

$$a(2) \cdot (b(1) + c(3)) = a(2) \cdot c(3).$$

After the execution of $a$, time has passed 2, so in the remaining subterm $b(1) + c(3)$ the first alternative is lost.

In [3], the notion of integration was introduced, which expresses the possibility that an action occurs somewhere within a time interval. The construct $\int_{v \in V} p$ executes the process $p$, where the behaviour of $p$ may depend on the value of $v$ in the time interval $V$. In [3], an axiomatization was proposed for processes that are time-closed, which means that if a process depends on a variable $v$, then it is guarded by some integral sign $\int_{v \in V}$. One of their axioms considerably hampers reasoning within the algebra, since in order to apply it one needs to check infinitely many equalities.

Namely, this axiom says that two processes $\int_{v\in V} p$ and $\int_{v\in V} q$ are equal if $p$ and $q$ are equal for all possible values for $v$ in the interval $V$.

In this chapter, we show how to get rid of this axiom. We restrict to prefix integration, and integration is parametrized by conditions, which consist of inequalities between linear expressions of variables. Furthermore, the notion of a conditional term is introduced, which is of the form $\phi :\rightarrow p$, where $\phi$ is a condition. The process $\phi :\rightarrow p$ executes $p$ under the condition that $\phi$ is true. We present an axiomatization for time-closed conditional terms, which allows to mix conditions through terms by the following two equations:

$$\int_\phi a(v) \cdot p = \int_\phi a(v) \cdot (\phi :\rightarrow p),$$

$$\phi :\rightarrow \int_\psi a(v) \cdot p = \int_{\phi\wedge\psi} a(v) \cdot p \qquad \text{if } v \text{ does not occur in } \phi.$$

Moreover, we present axiomatizations for bounds and for conditions.

Conditional terms can be reduced to a normal form, using the axioms, such that if two time-closed processes are bisimilar, then they have the same normal form. Hence, the axiom system decides bisimulation equivalence between time-closed processes.

Sections 7.2 and 7.4 originate from [13], which introduces, among other things, the notion of a conditional term together with its axiomatization. Section 7.3 originates from [9], which contains the decidability result. See [14, 15] how to deal with abstraction in our setting. A thorough treatment of real time ACP is provided in [16]. Section 7.5.2 provides a comparison of our results with related timed process algebras, which incorporate some integral construct which allows to express time dependencies.

**Acknowledgements.** A great number of people provided useful comments. Special thanks go to Jos Baeten, Frits Vaandrager and an anonymous referee, who suggested many substantial improvements.

## 7.2 The Syntax and Semantics

This section contains a description of the syntax and semantics for BPA$\delta$ with real time and prefix integration and conditions.

### 7.2.1 Bounds and conditions

In the sequel, we assume a countably infinite set of time variables *TVar*. Furthermore, we assume a set of time numbers *Time*, which is a field. So *Time* is closed under the binary operations addition and multiplication, which are associative and commutative and satisfy the distributivity laws. Moreover, it contains distinct units 0 and 1 for addition and multiplication respectively, and for each time number $t \neq 0$,

there are time numbers $-t$ and $t^{-1}$ such that $t + (-t) = 0$ and $t \cdot t^{-1} = 1$. We assume a (reflexive and transitive) total ordering $\leq$ on $Time$, which is preserved under addition and multiplication with time numbers greater than 0.

The full domain of time numbers will only be used in the operational semantics of process terms. In order to build the syntax of process terms from a finite signature, this syntax uses a countable sub-field $Time_0$ of $Time$, which is defined as follows. Fix functions $f_i : Time^{m_i} \to Time$ for $i = 1, ..., k$, and let $Time_0$ be the field that is generated by $f_1, ..., f_k$, i.e. $Time_0$ is the smallest sub-field of $Time$ such that if $r_1, ..., r_{m_i} \in Time_0$, then $f_i(r_1, ..., r_{m_i}) \in Time_0$.[1] We assume that for each pair of time numbers $r_0$ and $r_1$ in $Time_0$ it can be decided whether or not $r_0 \leq r_1$ holds.

Let $r \in Time_0$ and $v \in TVar$. The set of *bounds*, with typical element $b$, is defined by

$$b \ ::= \ r \ | \ v \ | \ b + b \ | \ r \cdot b.$$

The set of time variables that occur in a bound $b$ is denoted by $var(b)$. The expression $b_0 - b_1$ abbreviates $b_0 + (-1) \cdot b_1$.

A *condition* is constructed from conjunctions and negations of inequalities between bounds. So the set of conditions, with typical element $\phi$, is defined by

$$\phi \ ::= \ b_0 \leq b_1 \ | \ \phi \wedge \phi \ | \ \neg\phi.$$

The set of time variables that occur in a condition $\phi$ is denoted by $var(\phi)$. In the sequel, we use the abbreviations $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$, and $tt$ for $\neg(1 \leq 0)$, and $ff$ for $1 \leq 0$, and $b_0 < b_1$ for $\neg(b_1 \leq b_0)$, and $b_0 = b_1$ for $b_0 \leq b_1 \wedge b_1 \leq b_0$, and $b_0 < v < b_1$ for $b_0 < v \wedge v < b_1$.

### 7.2.2 Process terms

Assume a countable alphabet $A$ of atomic actions, together with a special constant $\delta$, representing deadlock. In the sequel, $a$ and $\alpha$ denote elements of $A$ and $A \cup \{\delta\}$ respectively.

*Integration* enables to express that the behaviour of a process may depend on the value of a time variable. If a process $p$ depends on the value of $v$ between 1 and 2, then we write

$$\int_{1<v<2} p.$$

Here, integration is parametrized by conditions, and we deal with *prefix* integration $\int_\phi \alpha(v) \cdot p$. If $\alpha \neq \delta$, then this process can execute the action $\alpha(t)$ under the condition that $\phi[t/v]$ is true, after which the process results to $p[t/v]$.

In BPA$\delta\rho I$, process terms are constructed from prefix integration, the alternative composition $p + q$, and the time shift $b \gg p$, where $b$ is a bound.[2] The time shift

[1] In the examples, $Time_0$ will consist of the rational numbers.

[2] We do not incorporate sequential composition $p \cdot q$ from untimed BPA in BPA$\delta\rho I$, because this construct can be eliminated from the syntax by a straightforward set of axioms, see [13].

is an auxiliary operator that is needed in the operational semantics of integration; the process $b \gg p$ displays the behaviour of $p$ after time $b$. Finally, we introduce the conditional construct $\phi :\rightarrow p$, where $\phi$ is a condition. This process displays the behaviour of $p$ under the condition that $\phi$ is true. Thus, the set of process terms, with typical element $p$, is defined by

$$p \ ::= \ \int_{\phi} \alpha(v) \ \mid \ \int_{\phi} \alpha(v) \cdot p \ \mid \ p + p \ \mid b \gg p \ \mid \ \phi :\rightarrow p.$$

As binding convention, integration and time shift bind stronger than alternative composition. Often $\alpha(b)$ will abbreviate $\int_{v=b} \alpha(v)$, where $v \notin var(b)$.

### 7.2.3 Free variables and substitutions

In general, one cannot attach a transition system to a process term that contains time variables which are not bound by an integral sign. Therefore, we need the notion of a time-closed process. In the term $\int_{\phi} \alpha(v)$, occurrences of $v$ in $\phi$ are bound, and in $\int_{\phi} \alpha(v) \cdot p$, occurrences of $v$ in $\phi$ and in $p$ are bound. A time variable in a process term is called free if it is not bound by any integral signs. The collection of free variables in a term $p$ is denoted by $fvar(p)$. A term $p$ is called *time-closed* if $fvar(p) = \emptyset$. As a model we will take the collection $\mathcal{T}^{cl}$ of time-closed process terms, modulo bisimulation (see Section 7.2.5).

A substitution is a mapping from time variables to bounds. For $b$ a bound and $\sigma$ a substitution, $\sigma(b)$ denotes the bound that results from substituting $\sigma(v)$ for each time variable $v$ in $b$. Substitutions extend to conditions as expected. A substitution $\sigma : TVar \rightarrow Time_0$ is called a *valuation*. For a condition $\phi$ and a valuation $\sigma$, $\sigma(\phi)$ results to either true or false. In the sequel, $[\phi]$ denotes the collection of valuations for which $\sigma(\phi)$ is true.

For a process term $p$, $\sigma(p)$ is obtained by replacing free occurrences of time variables $v$ in $p$ by $\sigma(v)$. In this definition however, there is one serious complication. Namely, if a free occurrence of $v$ in $p$ has been replaced by $\sigma(v)$, then variables $w$ that occur in $\sigma(v)$ may suddenly be bound in subterms $\int_{\phi} \alpha(w) \cdot q$ of $p$. A solution for this problem, which originates from the $\lambda$-calculus, is to allow unrestricted substitution by renaming bound variables. In the sequel, process terms are considered modulo $\alpha$-conversion, and when a substitution is applied, bound variables are renamed. Stoughton [20] presented a simple treatment of this technique.

If for a substitution $\sigma$ there is only one variable $v$ such that $\sigma(v) \neq v$, then often we will use the standard notation $p[\sigma(v)/v]$ for $\sigma(p)$.

### 7.2.4 Operational semantics

Table 7.1 contains an operational semantics for the collection $\mathcal{T}^{cl}$ of time-closed processes, in the style of Plotkin. In this operational semantics, we need a more general notion of bounds (and thus of conditions and process terms), which may

$$\frac{\phi[t/v]}{\int_\phi a(v) \xrightarrow{a(t)} \surd \qquad \int_\phi a(v) \cdot x \xrightarrow{a(t)} t \gg x[t/v]}$$

$$\frac{x \xrightarrow{a(t)} \surd}{x + y \xrightarrow{a(t)} \surd \quad y + x \xrightarrow{a(t)} \surd} \qquad \frac{x \xrightarrow{a(t)} x'}{x + y \xrightarrow{a(t)} x' \quad y + x \xrightarrow{a(t)} x'}$$

$$\frac{b < t \quad x \xrightarrow{a(t)} \surd}{b \gg x \xrightarrow{a(t)} \surd} \qquad \frac{b < t \quad x \xrightarrow{a(t)} x'}{b \gg x \xrightarrow{a(t)} x'}$$

$$\frac{\phi \quad x \xrightarrow{a(t)} \surd}{\phi :\rightarrow x \xrightarrow{a(t)} \surd} \qquad \frac{\phi \quad x \xrightarrow{a(t)} x'}{\phi :\rightarrow x \xrightarrow{a(t)} x'}$$

$$\frac{\phi[s/v] \quad t < s}{U_t(\int_\phi \alpha(v)) \quad U_t(\int_\phi \alpha(v) \cdot x)} \qquad \frac{U_t(x)}{U_t(x + y) \quad U_t(y + x)}$$

$$\frac{U_t(x)}{U_t(s \gg x)} \qquad \frac{t < s}{U_t(s \gg x)} \qquad \frac{\phi \quad U_t(x)}{U_t(\phi :\rightarrow x)}$$

Table 7.1: Action rules for time-closed terms

contain time numbers from *Time* instead of $Time_0$. That is, the BNF grammar of a bound in this section is $b ::= t \mid v \mid b + b \mid r \cdot b$, where $t$ is allowed to be in *Time*.

In most timed process algebras, the passing of time is expressed by idle transitions. For example, the process $a(1)$ can do an idle transition to time $t$ for $t < 1$, meaning that the process has reached time $t$. Finally, at time 1, it executes action $a$. Such an operational semantics can be found for timed CCS [22], timed CSP [19] and timed ACP [3]. We abstract from idle transitions, so here $a(1)$ only executes the $a$ at time 1. A similar operational semantics can be found in [12].

The timed deadlock $\delta(t)$ idles until time $t$. For example, the process $\int_{v<1} a(v) + \delta(3)$ either executes $a$ before time 1, or idles until time 3. On the other hand, the process $\int_{v<1} a(v) + \delta(1)$ will always execute $a$ before time 1. In order to distinguish processes that only differ in their deadlock behaviour, we introduce the delay predicate $U_t(p)$, which holds if $p$ can idle beyond time $t$. (Moller and Tofts [18] introduced a similar construct.) Processes that only differ in their deadlock behaviour have distinct delays. For example, $U_2(\int_{v<1} a(v) + \delta(3))$, but not $U_2(\int_{v<1} a(v) + \delta(1))$.

### 7.2.5 Bisimulation

Time-closed process terms are considered modulo *(strong) bisimulation*, which takes into account delays.

**Definition 7.1** *Two time-closed process terms $p_0, q_0$ are* strongly bisimilar*, notation $p_0 \leftrightarrows q_0$, if there exists a symmetric binary* bisimulation *relation $\mathcal{B}$ on time-closed process terms such that*

- *$p_0 \mathcal{B} q_0$,*
- *if $p \xrightarrow{a(t)} p'$ and $p \mathcal{B} q$, then $q \xrightarrow{a(t)} q'$ for some process $q'$ with $p' \mathcal{B} q'$,*
- *if $p \xrightarrow{a(t)} \surd$ and $p \mathcal{B} q$, then $q \xrightarrow{a(t)} \surd$,*
- *if $p \mathcal{B} q$ and $U_t(p)$, then $U_t(q)$.*

The rules in Table 7.1 can be fitted into the congruence format for action rules with types, data and variable binding of Bloom and Vaandrager [4].[3] Strong bisimulation defined by transition rules within this format is always a congruence on the algebra of closed terms, which means that if $p \leftrightarrows p'$ and $q \leftrightarrows q'$, then $p + q \leftrightarrows p' + q'$ and $\int_\phi \alpha(v) \cdot p \leftrightarrows \int_\phi \alpha(v) \cdot p'$ and $b \gg p \leftrightarrows b \gg p'$.

### 7.2.6 Axioms for bounds and conditions

Table 7.2 contains an axiomatization BA for bounds. It is assumed that BA incorporates the equalities $r_0 + r_1 = r_2$ and $r_0 \cdot r_1 = r_2$ between time numbers in $Time_0$. We consider bounds modulo AC, that is, modulo associativity and commutativity of the $+$. Using the axioms of BA, bounds can be reduced to a normal form $r_1 \cdot v_1 + ... + r_n \cdot v_n + r$, where $v_1, ..., v_n$ are distinct variables and $r_1, ..., r_n$ are unequal to zero. Using these normal forms, it is easy to deduce the following proposition.

**Proposition 7.2** *$b_0 = b_1 \Leftrightarrow \sigma(b_0)$ and $\sigma(b_1)$ result to the same time number for each valuation $\sigma$.*

Table 7.3 contains an axiom system CA for conditions. The construct $\phi \Rightarrow \psi$, which is used in two of the axioms, abbreviates $\phi \wedge \psi = \phi$. The six Boolean axioms are complete for the algebra generated by $tt$ and $\wedge$ and $\neg$ (see e.g. [17]), and the four ordering axioms for bounds characterize a linear ordered field (see e.g. [6]). It is assumed that CA incorporates the axiom system BA for bounds, that is, bounds in a condition may be manipulated by axioms in BA. In the sequel, $\phi = \psi$ denotes that this equality between conditions can be derived from CA.

The following lemma will be crucial in several constructions. Note that it would not hold if we had allowed bounds of the form $v^2$.

[3] The action rule for integration has to be adapted in order to fit it in the congruence format. Let *Bool* denote the type of Booleans and *Proc* the type of process terms. Integration is a function $\int(\lambda v.\phi, a, \lambda v.p)$, where $\int : (Time \to Bool) \times A \times (Time \to Proc) \longrightarrow Proc$. The action rule for integration takes the form

$$\frac{\texttt{pick}\ t : Time \quad \phi\, t}{\int(\phi, a, x) \xrightarrow{\langle a,t \rangle} t \gg (x\, t)}.$$

$$
\begin{array}{l}
b_0 + b_1 = b_1 + b_0 \\
(b_0 + b_1) + b_2 = b_0 + (b_1 + b_2) \\
b + 0 = b \\
1 \cdot b = b \\
0 \cdot b = 0 \\
r \cdot (b_0 + b_1) = r \cdot b_0 + r \cdot b_1 \\
r_0 \cdot b + r_1 \cdot b = (r_0 + r_1) \cdot b \\
r_0 \cdot (r_1 \cdot b) = (r_0 \cdot r_1) \cdot b
\end{array}
$$

Table 7.2: Axioms BA for bounds

$$
\begin{array}{ll}
\phi \wedge \psi \;=\; \psi \wedge \phi & b_0 + b \leq b_1 + b \;=\; b_0 \leq b_1 \\
(\phi_0 \wedge \phi_1) \wedge \phi_2 \;=\; \phi_0 \wedge (\phi_1 \wedge \phi_2) & r \cdot b_0 \leq r \cdot b_1 \;=\; b_0 \leq b_1 \quad \text{if } r > 0 \\
\phi \wedge (\psi_0 \vee \psi_1) \;=\; (\phi \wedge \psi_0) \vee (\phi \wedge \psi_1) & b_0 \leq b \wedge b \leq b_1 \;\Rightarrow\; b_0 \leq b_1 \\
\phi \;\Rightarrow\; \phi \vee \psi & b_0 \leq b_1 \vee b_1 \leq b_0 \;=\; tt \\
\phi \wedge \neg\phi = ff & \\
\neg\neg\phi \;=\; \phi &
\end{array}
$$

Table 7.3: Axioms CA for conditions

**Lemma 7.3** (refinement lemma) *Fix a time variable $v$. Each condition $\phi$ is provably equal to a condition of the form $\vee_i(\phi_i \wedge \phi'_i)$, where*

- *$var(\phi_i) \subseteq var(\phi) \backslash \{v\}$,*
- *$\phi'_i$ is of the form $v = b$ or $v < b$ or $b < v$ or $b < v < b'$, with $var(b + b') \subseteq var(\phi) \backslash \{v\}$.*

The axiom system CA is sound and complete in the following sense.

**Proposition 7.4** $\phi = \phi' \;\Leftrightarrow\; [\phi] = [\phi']$.

Soundness of CA can be verified by checking it for each axiom separately.

The proofs of the refinement lemma and of the completeness for CA are technical and do not contain any surprises. Hence, outlines of these proofs are provided in the appendix.

### 7.2.7 Axioms for process terms

From now on, occurrences of time numbers in bounds are restricted to $Time_0$ again. Table 7.4 contains an axiomatization for BPA$\delta\rho I$. The construct $P(v)$ represents

| | | | | |
|---|---|---|---|---|
| A1 | | $x+y$ | $=$ | $y+x$ |
| A2 | | $(x+y)+z$ | $=$ | $x+(y+z)$ |
| TA3 | | $\int_\phi P(v)+\int_\psi P(v)$ | $=$ | $\int_{\phi\vee\psi} P(v)$ |
| TA4 | | $\int_\phi P(v)+\int_\phi \delta(v)$ | $=$ | $\int_\phi P(v)$ |
| TA5 | | $\int_\phi \delta(v)\cdot x$ | $=$ | $\int_\phi \delta(v)$ |
| TA6 | | $\int_{ff} P(v)$ | $=$ | $\int_{ff} \delta(v)$ |
| TA7 | $v\notin var(b)$ | $\int_{v<b} \delta(v)$ | $=$ | $\delta(b)$ |
| TS1 | | $\int_\phi \alpha(v)\cdot x$ | $=$ | $\int_\phi \alpha(v)\cdot(v\gg x)$ |
| TS2 | $v\notin var(b)$ | $b\gg\int_\phi P(v)$ | $=$ | $\int_{\phi\wedge b<v} P(v)+\delta(b)$ |
| TS3 | | $b\gg(x+y)$ | $=$ | $b\gg x+b\gg y$ |
| TC1 | | $\int_\phi \alpha(v)\cdot x$ | $=$ | $\int_\phi \alpha(v)\cdot(\phi:\rightarrow x)$ |
| TC2 | $v\notin var(\phi)$ | $\phi:\rightarrow\int_\psi P(v)$ | $=$ | $\int_{\phi\wedge\psi} P(v)$ |
| TC3 | | $\phi:\rightarrow(x+y)$ | $=$ | $(\phi:\rightarrow x)+(\phi:\rightarrow y)$ |

Table 7.4: Axioms for BPA$\delta\rho I$

expressions of the form $\alpha(v)$ and $\alpha(v)\cdot x$. The equational theory for BPA$\delta\rho I$ incorporates the axiomatizations CA of conditions and BA of bounds. That is, conditions in terms can be manipulated by means of axioms in CA and BA.

For each axiom $p=q$, and for each valuation $\sigma$, we have $\sigma(p)\leftrightarrow\sigma(q)$. Hence, it is easy to see that the following proposition holds, which says that the axioms respect bisimulation equivalence between time-closed process terms. In the sequel, $p=q$ will mean that this equality between the terms $p$ and $q$ can be derived from the equational theory of BPA$\delta\rho I$.

**Proposition 7.5** $\forall p,q\in\mathcal{T}^{cl}$, $\qquad p=q \implies p\leftrightarrow q$.

The rest of this chapter is devoted to proving that the equational theory for BPA$\delta\rho I$ is complete for time-closed process terms, i.e. if for $p,q\in\mathcal{T}^{cl}$ we have $p\leftrightarrow q$, then $p=q$. We present an algorithm, based on the axioms, which decides bisimulation equivalence between time-closed process terms.

### 7.2.8 Basic terms

**Definition 7.6** *A term is* basic *if it is in the class defined by*

$$p ::= \int_\phi \alpha(v) \mid \int_\phi \alpha(v)\cdot p \mid p+p,$$

*and for each subterm $\int_\phi \alpha(v)\cdot p$ we have $\sigma(v\gg p)\leftrightarrow\sigma(p)$ for all valuations $\sigma$.*

**Proposition 7.7** *For each term $p$ there is a basic term $p'$ such that $p = p'$.*

**Proof.** First, we show how the axioms can be applied in order to get rid of time shifts and conditions. Let $p_0$ denote a term that does not contain time shifts nor conditions. Suppose that $p$ is of the form $b \gg p_0$. We delete the time shift from $p$ as follows, by induction on the *size* of $p_0$ (i.e. the number of function symbols in $p_0$). Ensure by means of $\alpha$-conversion that $v \notin var(b)$.

$$\begin{aligned} b \gg \textstyle\int_\phi P(v) &\overset{\text{TS2}}{=} \textstyle\int_{\phi \wedge b<v} P(v) + \delta(b), \\ b \gg (q + q') &\overset{\text{TS3}}{=} b \gg q + b \gg q'. \end{aligned}$$

Next, suppose that $p$ is of the form $\phi :\rightarrow p_0$. We delete the condition from $p$ as follows, by induction on the size of $p_0$. Ensure by means of $\alpha$-conversion that $v \notin var(\phi)$.

$$\begin{aligned} \phi :\rightarrow \textstyle\int_\psi P(v) &\overset{\text{TC2}}{=} \textstyle\int_{\phi \wedge \psi} P(v), \\ \phi :\rightarrow (q + q') &\overset{\text{TC3}}{=} (\phi :\rightarrow q) + (\phi :\rightarrow q'). \end{aligned}$$

If $p$ contains several time shifts and conditions, then these operators can be deleted one by one, by considering subterms of $p$ which contain only one time shift or one condition.

Finally, we show that $p$ equals a basic term, by induction on size. A term $\int_\phi \alpha(v)$ is already basic, and if $p$ and $p'$ are basic, then $p + p'$ is basic too. Since we have deleted time shifts and conditions, this only leaves the case $\int_\phi \alpha(v) \cdot p$, where $p$ is basic.

$$\begin{aligned} & \textstyle\int_\phi \alpha(v) \cdot \sum_i \int_{\psi_i} P_i(w) \\ \overset{\text{TS1}}{=} & \textstyle\int_\phi \alpha(v) \cdot v \gg \sum_i \int_{\psi_i} P_i(w) \\ \overset{\text{TS2,3}}{=} & \textstyle\int_\phi \alpha(v) \cdot \sum_i (\int_{\psi_i \wedge v<w} P_i(w) + \delta(v)). \end{aligned}$$

This last term is basic. □

## 7.3 Unique Normal Forms

We shall describe a strategy which reduces each basic term $p$ to a term which is called the *normal form* of $p$. All steps in the algorithm can be deduced from the axioms, so $p$ is equal to its normal form. Next, we will show that if two time-closed normal forms are bisimilar, then they are equal modulo AC, i.e. modulo associativity and commutativity of the $+$. This will imply completeness of the axiom system. From now on, terms are considered modulo AC, and this equivalence is denoted by $=_{\text{AC}}$.

In the following sections, we will present several equations between closed terms, that will be used in the construction of normal forms. These equations can all be deduced from the axioms. Finally, Section 7.3.6 provides a description of the construction of normal forms.

### 7.3.1 Some basic equations

The following equations for closed conditional terms can be deduced from the axioms.

**Equation 7.8** $\phi :\rightarrow (\psi :\rightarrow p) \;=\; \phi \wedge \psi :\rightarrow p.$

**Equation 7.9** $(\phi :\rightarrow p) + (\psi :\rightarrow p) \;=\; \phi \vee \psi :\rightarrow p.$

**Equation 7.10** $p + \int_{ff} \delta(v) \;=\; p.$

**Equation 7.11** $tt :\rightarrow p \;=\; p.$

**Equation 7.12** $ff :\rightarrow p \;=\; \int_{ff} \delta(v).$

In order to prove these equalities, it is sufficient to prove them for terms $p$ and $q$ of the form $\sum_{i=1}^{k} \int_{\phi_i} P_i(v)$ (with $k \geq 1$), because in the previous section we got rid of time shifts and conditions in closed terms. As an example, we prove Equations 7.9 and 7.10.

**Proof of Equation 7.9.** Ensure by means of $\alpha$-conversion that $v \notin var(\phi \vee \psi)$.

$$
\begin{array}{ll}
 & (\phi :\rightarrow \sum_{i=1}^{k} \int_{\phi_i} P_i(v)) + (\psi :\rightarrow \sum_{i=1}^{k} \int_{\phi_i} P_i(v)) \\
\overset{\text{TC2,3}}{=} & \sum_{i=1}^{k} \int_{\phi \wedge \phi_i} P_i(v) + \sum_{i=1}^{k} \int_{\psi \wedge \phi_i} P_i(v) \\
\overset{\text{TA3}}{=} & \sum_{i=1}^{k} \int_{(\phi \vee \psi) \wedge \phi_i} P_i(v) \quad \overset{\text{TC2,3}}{=} \quad \phi \vee \psi :\rightarrow \sum_{i=1}^{k} \int_{\phi_i} P_i(v). \; \Box
\end{array}
$$

**Proof of Equation 7.10.**

$$\sum_{i=1}^{k} \int_{\phi_i} P_i(v) + \int_{ff} \delta(v) \overset{\text{TA6}}{=} \sum_{i=1}^{k} \int_{\phi_i} P_i(v) + \int_{ff} P_1(v) \overset{\text{TA3}}{=} \sum_{i=1}^{k} \int_{\phi_i} P_i(v). \; \Box$$

### 7.3.2 Reducing conditions to intervals

A finite collection of conditions is called a *partition* if for each valuation $\sigma$ there is exactly one condition $\phi$ in this collection such that $\sigma \in [\phi]$.

**Equation 7.13** (lifting equation) *If* $\{\phi_1, ..., \phi_n\}$ *is a partition, then*

$$\int_{\phi} \alpha(v) \cdot \sum_{i=1}^{n} (\phi_i :\rightarrow p_i) \;=\; \sum_{i=1}^{n} \int_{\phi \wedge \phi_i} \alpha(v) \cdot p_i.$$

**Proof.** In this deduction we implicitly apply Equation 7.8.

$$
\begin{array}{lll}
 & \int_{\phi} \alpha(v) \cdot \sum_{i=1}^{n} (\phi_i :\rightarrow p_i) & \\
\overset{\text{TA3}}{=} & \sum_{j=1}^{n} \int_{\phi \wedge \phi_j} \alpha(v) \cdot \sum_{i=1}^{n} (\phi_i :\rightarrow p_i) & \text{because } \vee_j \phi_j = tt \\
\overset{\text{TC1,3}}{=} & \sum_{j=1}^{n} \int_{\phi \wedge \phi_j} \alpha(v) \cdot \sum_{i=1}^{n} (\phi_i \wedge \phi_j :\rightarrow p_i) & \\
\overset{\text{Eq 7.10}}{=} & \sum_{j=1}^{n} \int_{\phi \wedge \phi_j} \alpha(v) \cdot (\phi_j :\rightarrow p_j) & \text{because } \phi_i \wedge \phi_j = ff \text{ if } i \neq j \\
\overset{\text{TC1}}{=} & \sum_{j=1}^{n} \int_{\phi \wedge \phi_j} \alpha(v) \cdot p_j. \; \Box &
\end{array}
$$

The constraint in the lifting equation that $\{\phi_1, ..., \phi_n\}$ is a partition is an essential ingredient, because without it we would get equalities like

$$\begin{aligned} \textstyle\int_\phi \alpha(v) \cdot \int_{ff} \delta(v) &= \textstyle\int_{ff} \delta(v), \\ \textstyle\int_\phi \alpha(v) \cdot (p+q) &= \textstyle\int_\phi \alpha(v) \cdot p + \int_\phi \alpha(v) \cdot q. \end{aligned}$$

The following equation enables to reduce the collection of conditions $\{\phi_i\}$ in a term $\sum_i(\phi_i :\rightarrow p_i)$ to a partition.

**Equation 7.14** $(\phi :\rightarrow p) + (\psi :\rightarrow q) =$
$(\phi \wedge \psi :\rightarrow p+q) + (\phi \wedge \neg\psi :\rightarrow p) + (\neg\phi \wedge \psi :\rightarrow q) + (\neg\phi \wedge \neg\psi :\rightarrow \int_{ff} \delta(v))$.

**Proof**.
$$\begin{aligned} & (\phi :\rightarrow p) + (\psi :\rightarrow q) \\ \overset{\text{Eq 7.9}}{=}\ & (\phi \wedge \psi :\rightarrow p) + (\phi \wedge \neg\psi :\rightarrow p) + (\phi \wedge \psi :\rightarrow q) + (\neg\phi \wedge \psi :\rightarrow q) \\ \overset{\text{TC3}}{=}\ & (\phi \wedge \psi :\rightarrow p+q) + (\phi \wedge \neg\psi :\rightarrow p) + (\neg\phi \wedge \psi :\rightarrow q). \end{aligned}$$
According to Equation 7.10, we can add $\int_{ff} \delta(v) \overset{\text{TC2}}{=} \neg\phi \wedge \neg\psi :\rightarrow \int_{ff} \delta(v)$. □

We will use the lifting equation and the refinement lemma (see Section 7.2.1) to reduce conditions that parametrize integrals to the form $tt$ or $ff$ or $v = b$ or $v < b$ or $b < v$ or $b < v < b'$, with $v \notin var(b + b')$. In the sequel, such conditions will often be denoted by $v \in V$, where $V$ represents an interval. For example, $tt$ is denoted by $v \in \langle -\infty, \infty \rangle$, and $ff$ by $v \in \emptyset$, and $b \leq v < b'$ by $v \in [b, b'\rangle$, etc.

### 7.3.3 Adapting deadlocks

Using the refinement lemma, conditions in deadlocks can be reduced to either $tt$ or $ff$ or $v = b$ by means of TA7 and the following two equations. Let $v \notin var(b + b')$.

**Equation 7.15** $\int_{b<v} \delta(v) = \int_{tt} \delta(v)$.

**Proof**.
$$\begin{aligned} & \textstyle\int_{b<v} \delta(v) \overset{\text{TA3}}{=} \int_{b<v} \delta(v) + \int_{v=b+1} \delta(v) \\ \overset{\text{TA7}}{=}\ & \textstyle\int_{b<v} \delta(v) + \int_{v<b+1} \delta(v) \overset{\text{TA3}}{=} \int_{tt} \delta(v). \ \square \end{aligned}$$

**Equation 7.16** $\int_{b<v<b'} \delta(v) = b < b' :\rightarrow \delta(b')$.

This last equality can be deduced from TA3,7 and TC2.

Redundant deadlocks can be removed by means of TA4 and Equation 7.10 and the following equation, which can be deduced from TA3,4 and TC2.

**Equation 7.17** $b \leq \sup(V) :\rightarrow (\int_{v\in V} P(v) + \delta(b)) = b \leq \sup(V) :\rightarrow \int_{v\in V} P(v)$.

### 7.3.4 Removing redundant variables

In a process term $\int_{v=b} a(v) \cdot p$ (with $v \notin var(b)$), the time variable $v$ is 'redundant' in $p$, in the sense that it can only attain the value $b$. Occurrences of such redundant variables in $p$ can be removed by the following equation.

**Equation 7.18** $\int_{v=b} \alpha(v) \cdot p \;=\; \int_{v=b} \alpha(v) \cdot p[b/v]$.

This equation can be deduced from TC1-3 by induction on the *depth* of $p$ (i.e. the length of the longest execution trace of $p$).

### 7.3.5 Removing double terms

In the reduction to normal form, we will delete double terms. Let $V \sim W$ denote the condition that $V \cup W$ is an interval. Axiom TA3 induces the following equation.

**Equation 7.19** $V \sim W :\rightarrow (\int_{v \in V} P(v) + \int_{v \in W} P(v)) \;=\; V \sim W :\rightarrow \int_{v \in V \cup W} P(v)$.

Equation 7.19 is not always sufficient to reduce double terms. Namely, the equality

$$\int_{v<b} a(v) \cdot p + \int_{\phi} a(v) \cdot q \;=\; \int_{v \leq b} a(v) \cdot p + \int_{\phi} a(v) \cdot q$$

with $v \notin var(b)$, is sound if $\phi[b/v] = tt$ and $p[b/v] \leftrightarrow q[b/v]$. However, this equation cannot be deduced from Equation 7.19, because $p$ and $q$ need not be of the same form. We introduce an extra equation to deal with this example.

In the reduction to normal form, this equation is only needed in case $p[b/v]$ and $q[b/v]$ are of the same form, which can be expressed by a condition as follows. Consider a term $\sigma(p)$. Reduce its bounds to normal form, using the axioms from BA, and replace subterms of the form $\int_{v=b} a(v) \cdot p'$ by $\int_{v=b} a(v) \cdot p'[b/v]$. The resulting process term will be denoted by $\sigma(p)*$. For $p, q$ terms, let $\psi(p, q)$ denote a condition such that $\sigma \in [\psi(p, q)]$ if and only if $\sigma(p)* =_{\mathrm{AC}} \sigma(q)*$. Note that such a condition exists.

For $p, q$ terms and $b$ a bound with $v \notin var(b)$, the first example can be reduced by the following equation.

**Equation 7.20**
$$\phi[b/v] \wedge \psi(p[b/v], q[b/v]) :\rightarrow (\int_{v<b} a(v) \cdot p + \int_{\phi} a(v) \cdot q)$$
$$= \quad \phi[b/v] \wedge \psi(p[b/v], q[b/v]) :\rightarrow (\int_{v \leq b} a(v) \cdot p + \int_{\phi} a(v) \cdot q).$$

We also have a symmetric version of this equation, in order to reduce the process term $\int_{b<v} a(v) \cdot p + \int_{\phi} a(v) \cdot q$. We have two more symmetric equations to deal with the term $\int_{b_0<v<b_1} a(v) \cdot p + \int_{\phi} a(v) \cdot q$. These equations can be deduced from TA3 and TC2,3 and Equation 7.18.

There is one more example which cannot be reduced neither by Equation 7.19 nor by Equation 7.20.

$$\int_{v=b} a(v) \cdot p \;+\; \int_{\phi} a(v) \cdot q \quad \leftrightarrow \quad \int_{\phi} a(v) \cdot q$$

if $\phi[b/v] = tt$ and $p \leftrightarrow q[b/v]$. This second example can be reduced by the following equation.

**Equation 7.21** $\phi[b/v] \wedge \psi(p, q[b/v]) :\rightarrow (\int_{v=b} a(v) \cdot p + \int_{\phi} a(v) \cdot q)$

$$= \phi[b/v] \wedge \psi(p, q[b/v]) :\rightarrow \int_{\phi} a(v) \cdot q.$$

### 7.3.6 Construction of normal forms

We define an algorithm which reduces a basic term to a term which is called its *normal form*. This normal form is of the form $\sum_i(\phi_i :\rightarrow p_i)$, where $\{\phi_i\}$ is a partition and the $p_i$ are basic terms. The algorithm uses the equations that have been deduced in the previous sections. Some of these equations are of the form $\phi :\rightarrow p = \phi :\rightarrow q$. In the algorithm, such equations are applied in the form $p = (\phi :\rightarrow q) + (\neg\phi :\rightarrow p)$.

We apply induction on depth. So suppose that we have already constructed normal forms for basic terms of depth $\leq n$, and let $p$ be a basic term of depth $n+1$, of the form

$$\sum_{i\in I} \int_{\phi_i} \alpha_i(v) \cdot p_i \;+\; \sum_{j\in J} \int_{\phi_j} \alpha_j(v).$$

(In the basic induction step the sum over $I$ is empty.) The $p_i$ have depth $\leq n$, so by induction we already have constructed their normal forms $\sum_{k\in K_i}(\phi'_k :\rightarrow q_k)$. Replace the $p_i$ by their normal forms and apply the lifting equation to obtain

$$\sum_{(i,k)\in I\times K_i} \int_{\phi_i\wedge\phi'_k} \alpha_i(v) \cdot q_k \;+\; \sum_{j\in J} \int_{\phi_j} \alpha_j(v).$$

According to the refinement lemma, $\phi_i \wedge \phi'_k$ is equivalent to a condition $\vee_{l\in L_k}\psi_l \wedge v \in V_l$, and $\phi_j$ is equivalent to a condition $\vee_{l\in L_j}\psi_l \wedge v \in V_l$, where $var(\psi_l) \cup var(V_l) \subseteq var(p)\backslash\{v\}$, and the $V_l$ are either open or of the form $[b, b]$. Using TC2, we can reduce $p$ to the form

$$\sum_{(i,k,l)\in I\times K_i\times L_k} (\psi_l :\rightarrow \int_{v\in V_l} \alpha_i(v) \cdot q_k) \;+\; \sum_{j\in J\ l\in L_j} (\psi_l :\rightarrow \int_{v\in V_l} \alpha_j(v)).$$

Reduce the conditions $\psi_l$ to a partition by means of Equation 7.14. Reduce the bounds in the $V_l$ to normal form, using the axioms in BA.

Next, we remove redundant variables. If a $V_l$ is of the form $[b, b]$ and $v \in fvar(q_k)$, then apply Equation 7.18 to $\int_{v\in V_l} a_i(v) \cdot q_k$. Reduce the bounds in $q_k[b/v]$ to normal form again.

Next, we reduce deadlocks. We use the fact that the $V_l$ are either open or of the form $[b, b]$.

- Reduce deadlocks $\int_{v\in V} \delta(v) \cdot q$ to the form $\int_{v\in V} \delta(v)$.
- Apply TA7 and Equations 7.15 and 7.16 in order to reduce conditions in deadlocks to either $tt$ or $ff$ or $v \in [b, b]$.
- Reduce terms $\int_{ff} P(v)$ to $\int_{ff} \delta(v)$.

- Remove redundant deadlocks using TA4 and Equations 7.10 and 7.17.

Finally, we remove double terms. First apply Equation 7.20, and then Equations 7.19 and 7.21 to each pair $\int_{v\in V} a(v)\cdot q \;+\; \int_{v\in W} a(v)\cdot q'$ and $\int_{v\in V} a(v) \;+\; \int_{v\in W} a(v)$. The result is the normal form of $p$.

### 7.3.7 The main theorem

Since each term is equal to a basic term, it follows that each term is equal to a normal form. Let $p \in \mathcal{T}^{cl}$ have normal form $\Sigma_i(\phi_i :\to p_i)$. The construction of normal forms ensures that $var(\phi_i) \cup fvar(p_i) \subseteq fvar(p) = \emptyset$. In particular, each $\phi_i$ is equal to either $tt$ or $ff$, so we can reduce the normal form of $p$ to a time-closed term by applying Equations 7.11 and 7.12.

We prove that bisimilar time-closed normal froms are equal modulo AC. First, we formulate two lemmas which are needed in the proof of the main theorem.

**Lemma 7.22** *Let $p$ and $q$ be subterms of normal forms. If $p[r/v]* =_{\mathrm{AC}} q[r/v]*$ for infinitely many $r \in Time_0$, then $p =_{\mathrm{AC}} q$.*

The proof of this lemma, which is technical and straightforward, is presented in the appendix.

**Lemma 7.23** *Let $\int_{v\in V} a(v)\cdot p$ be a normal form. Then $p[r/v]*$ is a normal form for all $r \in V \cap Time_0$.*

This lemma can be proved by showing that the construction to normal form reduces $p[t/v]$ to $p[t/v]*$. It is left to the reader to check that this is indeed the case.

**Theorem 7.24** *If time-closed normal forms $p$ and $q$ are bisimilar, then $p =_{\mathrm{AC}} q$.*

**Proof.** We apply induction on the depth of $p$ and $q$. First, let

$$p \;=_{\mathrm{AC}} \sum_{i\in I} \int_{v\in V_i} \alpha_i(v), \qquad\qquad q \;=_{\mathrm{AC}} \sum_{j\in J} \int_{v\in W_j} \alpha'_j(v).$$

Fix an $i \in I$. First, let $\alpha_i \in A$. Then for $t \in V_i$ we have $p \xrightarrow{\alpha_i(t)} \surd$. Since $p \leftrightarroweq q$, for each $t \in V_i$ there is a $j(t) \in J$ with $t \in W_{j(t)}$ and $\alpha_i = \alpha'_{j(t)}$. In the reduction to normal form Equation 7.19 has been applied, so the intervals $W_{j(t)}$ for $t \in V_i$ have been united to one interval. Hence, there is a unique $j \in J$ with $V_i \subseteq W_j$ and $\alpha_i = \alpha'_j$. Similarly, for this $j$ there is a unique $i(j) \in I$ with $W_j \subseteq V_{i(j)}$ and $\alpha'_j = \alpha_{i(j)}$. Since $V_i \subseteq W_j \subseteq V_{i(j)}$ and $\alpha_i = \alpha'_j = \alpha_{i(j)}$, Equation 7.19 yields $i(j) = i$, so $V_i$ and $W_j$ coincide.

Next, let $\alpha_i = \delta$. The adaptation of deadlocks in the reduction of normal forms ensures that $v \in V_i$ is of the form $v < r$ or $tt$. Since Equation 7.17 has been applied in the reduction to normal form, the ultimate delay of $p$ is determined by the summand $\int_{v<r} \delta(v)$ or $\int_{tt} \delta(v)$. Since $p$ and $q$ are bisimilar, it follows that $q$ must contain this same summand.

Suppose that the theorem has been proved for depth $\leq n$. Let

$$p =_{\mathrm{AC}} \sum_{i\in I}\int_{v\in V_i} a_i(v)\cdot p_i + p', \qquad q =_{\mathrm{AC}} \sum_{j\in J}\int_{v\in W_j} a'_j(v)\cdot q_j + q',$$

where the $p_i$ and $q_j$ have depth $n$ and $p'$ and $q'$ have depth $\leq n$. Since $p \leftrightarrows q$, it follows that $p' \leftrightarrows q'$ and thus by the induction hypothesis $p' =_{\mathrm{AC}} q'$.

Fix an $i \in I$. The terms $p$ and $q$ are bisimilar and are basic terms, so for each $t \in V_i$ there is a $j(t) \in J$ with $t \in W_{j(t)}$ and $a_i = a'_{j(t)}$ and $t \gg p_i[t/v] \leftrightarrows t \gg q_{j(t)}[t/v]$. Since normal forms are basic terms, which means that they have ascending time stamps, it follows that $p_i[t/v] \leftrightarrows t \gg p_i[t/v] \leftrightarrows t \gg q_{j(t)}[t/v] \leftrightarrows q_{j(t)}[t/v]$

First, assume that $V_i$ contains more than one point. Let $J' \subseteq J$ be the collection of $j$ for which $a'_j = a_i$ and $q_j =_{\mathrm{AC}} p_i$, and define $W_{J'} := \cup_{j\in J'} W_j$. We show that $V_i \backslash W_{J'}$ is empty.

1. $V_i\backslash W_{J'}$ contains only a finite number of time numbers in $Time_0$.

   Suppose not. For each $t \in V_i\backslash W_{J'}$ we have $j(t) \notin J'$, so then there is an infinite subset $R$ of $(V_i\backslash W_{J'}) \cap Time_0$ and a $j_0 \in J\backslash J'$ such that $j(r) = j_0$ for all $r \in R$. Since $p_i[r/v] \leftrightarrows q_{j_0}[r/v]$, the induction hypothesis together with Lemma 7.23 yield $p_i[r/v]* =_{\mathrm{AC}} q_{j_0}[r/v]*$ for $r \in R$. Since $R$ is infinite, Lemma 7.22 implies $p_i =_{\mathrm{AC}} q_{j_0}$. Then $j_0 \in J'$, which is a contradiction.

2. $V_i\backslash W_{J'}$ does not contain time numbers in $Time\backslash Time_0$.

   Suppose that $V_i\backslash W_{J'}$ does contain a time number $t \notin Time_0$. Since $V_i\backslash W_{J'}$ consists of intervals that have bounds in $Time_0$, apparently it contains the time numbers in an interval $\langle r, s\rangle$ with $r, s \in Time_0$ and $r < t < s$. Then the time numbers $(r + (n-1)s)/n$ are distinct in $\langle r, s\rangle \cap Time_0$ for $n = 1, 2, ...$, so $V_i\backslash W_{J'}$ contains infinitely many time numbers in $Time_0$; contradiction.

3. $V_i\backslash W_{J'}$ is empty.

   Suppose not, so there exists an $r \in (V_i\backslash W_{J'}) \cap Time_0$. Since this set is finite, and since $V_i$ contains more than one point, there is a $j \in J'$ such that $W_j$ has lower bound or upper bound $r$. Since $q_j =_{\mathrm{AC}} p_i$ and $p_i[r/v] \leftrightarrows q_{j(r)}[r/v]$, it follows that $q_j[r/v] \leftrightarrows q_{j(r)}[r/v]$. Then the induction hypothesis together with Lemma 7.23 yield $q_j[r/v]* =_{\mathrm{AC}} q_{j(r)}[r/v]*$. Since Equation 7.20 has been applied to the pair

   $$\int_{v\in W_{j(r)}} a_i(v)\cdot q_{j(r)} \;+\; \int_{v\in W_j} a_i(v)\cdot q_j,$$

   we have $r \in W_j$, which is a contradiction.

Hence, $V_i\backslash W_{J'} = \emptyset$, or in other words, $V_i \subseteq W_{J'}$. Then Equation 7.19 implies that there is a unique $j \in J'$ with $V_i \subseteq W_j$.

Similarly for this $j$ there is an $i(j) \in I$ with $a'_j = a_{i(j)}$ and $q_j =_{\mathrm{AC}} p_{i(j)}$ and $W_j \subseteq V_{i(j)}$. Since Equation 7.19 has been applied, it follows that $i(j) = i$. Hence, $V_i$ coincides with $W_j$.

Next, assume that $V_i$ equals $[t, t]$. If $W_{j(t)}$ contains more than one point, then we have just proved that there is an $i(t) \in I$ with $a_{i(t)} = a'_{j(t)}$, and $V_{i(t)}$ and $W_{j(t)}$ coincide, and $p_{i(t)} =_{\mathrm{AC}} q_{j(t)}$. Since $V_{i(t)}$ and $W_{j(t)}$ coincide, it follows that $t \in V_{i(t)}$. And $p_{i(t)} =_{\mathrm{AC}} q_{j(t)}$ together with $p_i[t/v] \leftrightarrows q_{j(t)}[t/v]$ implies that $p_{i(t)}[t/v] \leftrightarrows p_i[t/v]$. Then the induction hypothesis together with Lemma 7.23 give $p_{i(t)}[t/v]* =_{\mathrm{AC}} p_i[t/v]*$. Since Equation 7.21 has been applied to the pair

$$\int_{v \in V_{i(t)}} a_i(v) \cdot p_{i(t)} \;+\; \int_{v \in V_i} a_i(v) \cdot p_i,$$

the term $\int_{v \in V_i} a_i(v) \cdot p_i$ should not be there at all. This is a contradiction, so it follows that $W_{j(t)}$ equals $[t, t]$.

Since Equation 7.18 has been applied, $p_i =_{\mathrm{AC}} p_i[t/v]*$ and $q_{j(t)} =_{\mathrm{AC}} q_{j(t)}[t/v]*$. Induction and Lemma 7.23 yield $p_i[t/v]* =_{\mathrm{AC}} q_{j(t)}[t/v]*$, so $p_i =_{\mathrm{AC}} q_{j(t)}$. □

**Corollary 7.25** *The axiomatization of BPAδρI is complete for time-closed processes modulo bisimulation equivalence.*

**Proof.** Let $p, q \in \mathcal{T}^{cl}$ be bisimilar. Each step in the reduction to normal form can be deduced from the axioms, so $p = p\!\downarrow$ and $q = q\!\downarrow$. Then $p\!\downarrow \leftrightarrows p \leftrightarrows q \leftrightarrows q\!\downarrow$, so Theorem 7.24 yields $p\!\downarrow \;=_{\mathrm{AC}} q\!\downarrow$. Hence $p = q$. □

**Corollary 7.26** *Bisimulation equivalence between time-closed processes in BPAδρI is decidable.*

### 7.3.8 An example

The normal form of a process term can be much larger than the term itself. For example, consider the term

$$\int_{0<v<5} a(v) \cdot (\int_{7-4v<w<5-v} b(w) \;+\; \int_{3<w<\frac{17}{6}+\frac{1}{3}v} b(w)).$$

Its normal form can be deduced from Figure 7.1. The lines that are drawn there intersect for $v \in \{\frac{1}{2}, \frac{2}{3}, \frac{25}{26}, 1, \frac{7}{5}, \frac{13}{8}, 2, \frac{5}{2}, 3, \frac{17}{4}\}$. Thus we get the following normal form:

$$\begin{aligned}
& \textstyle\int_{0<v\leq\frac{1}{2}} a(v)\cdot\delta(v) && + \textstyle\int_{\frac{1}{2}<v\leq\frac{2}{3}} a(v)\cdot\int_{3<w<\frac{17}{6}+\frac{1}{3}v} b(w)\\
+ & \textstyle\int_{\frac{2}{3}<v\leq\frac{25}{26}} a(v)\cdot(\int_{3<w<\frac{17}{6}+\frac{1}{3}v} b(w)+\int_{7-4v<w<5-v} b(w))\\
+ & \textstyle\int_{\frac{25}{26}<v\leq 1} a(v)\cdot\int_{3<w<5-v} b(w) && + \textstyle\int_{1\leq v\leq\frac{7}{5}} a(v)\cdot\int_{7-4v<v<5-v} b(w)\\
+ & \textstyle\int_{\frac{7}{5}\leq v\leq\frac{13}{8}} a(v)\cdot\int_{v<w<5-v} b(w) && + \textstyle\int_{\frac{13}{8}\leq v<2} a(v)\cdot\int_{v<w<\frac{17}{6}+\frac{1}{3}v} b(w)\\
+ & \textstyle\int_{2\leq v<\frac{5}{2}} a(v)\cdot(\int_{v<w<5-v} b(w)+\int_{3<w<\frac{17}{6}+\frac{1}{3}v} b(w))\\
+ & \textstyle\int_{\frac{5}{2}\leq v\leq 3} a(v)\cdot\int_{3<w<\frac{17}{6}+\frac{1}{3}v} b(w) && + \textstyle\int_{3\leq v<\frac{17}{4}} a(v)\cdot\int_{v<w<\frac{17}{6}+\frac{1}{3}v} b(w)\\
+ & \textstyle\int_{\frac{17}{4}\leq v<5} a(v)\cdot\delta(v).
\end{aligned}$$

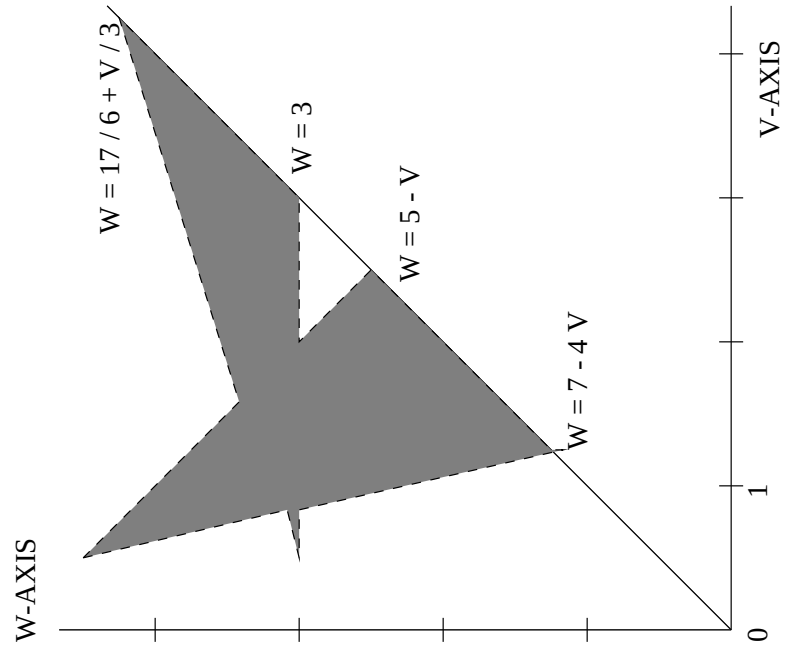


Figure 7.1: Graphical description of a process term

## 7.4 Parallelism and Synchronization

We introduce parallelism, synchronization and encapsulation, resulting in the theory ACP$\rho I$. We extend the syntax with the parallel merge $\|$, and we add the auxiliary

operators left merge $\between$ and communication merge $|$, which allow the definition of $\|$ in finitely many axioms. Moreover, we add the encapsulation operator $\partial_H$.

Assume a commutative communication function $| : A\cup\{\delta\}\times A\cup\{\delta\} \longrightarrow A\cup\{\delta\}$ which is commutative and associative and has $\delta$ as zero element. Communication between timed actions can only take place if they happen simultaneously. So if $a|a' = c$, then

$$\begin{array}{lcl} a(1)|a'(1) & = & c(1), \\ a(1)|a'(2) & = & \delta(1). \end{array}$$

In untimed ACP, $a\between p = a \cdot p$. However, in the real-time setting this definition would cause the process $a(1)\|a'(2)$ to have a deadlock:

$$a(1)\|a'(2) = a(1)\between a'(2) + a'(2)\between a(1) = a(1)\cdot a'(2) + a'(2)\cdot\int_{ff}\delta(v).$$

In order to avoid such deadlocks, the definition of the left merge is adapted in such a way that a process $p\between q$ can only idle as long as both $p$ and $q$ can idle. So for example:

$$\begin{array}{lcl} a(1)\between a'(2) & = & a(1)\cdot a'(2), \\ a'(2)\between a(1) & = & \delta(1). \end{array}$$

Note that this definition induces $a(1)\|a'(2) = a(1)\cdot a'(2)$.

For each subset $H$ of $A$, the encapsulation operator $\partial_H$ is defined on $A\cup\{\delta\}$ by $\partial_H(\alpha) = \delta$ if $\alpha \in H\cup\{\delta\}$ and $\partial_H(a) = a$ if $a \in A\backslash H$.

The constructs $\|$ and $\between$ and $|$ and $\partial_H$ are added to the notion of a process term, that is, the BNF grammar is extended with

$$p\|p \mid p\between p \mid p|p \mid \partial_H(p).$$

The notions free variables and substitutions extend to these operators as expected.

### 7.4.1 Operational semantics for ACP$\rho I$

Table 7.5 contains the action rules for the operators $\|, |, \between$ and $\partial_H$, taken from [13]. Once more, bounds may contain time numbers $t$ from *Time*. The rules are within the format of Bloom and Vaandrager, so bisimulation equivalence is a congruence for the added operators as well.

Furthermore, ACP$\rho I$ is a conservative extension of BPA$\rho\delta I$, which means that the operational semantics of BPA$\rho\delta I$ terms is not influenced by the extra action rules for communication and encapsulation. This follows from the fact that the action rules of BPA$\rho\delta I$ are all *source-dependent*, and that the extra action rules of ACP$\rho I$ all have a new function symbol in the left-hand side of their conclusion. See [10] for the definitions and a proof of this result.

$$\frac{x \xrightarrow{a(t)} x' \quad U_t(y)}{x\|y \xrightarrow{a(t)} x'\|(t \gg y) \quad y\|x \xrightarrow{a(t)} (t \gg y)\|x' \quad x \between y \xrightarrow{a(t)} x'\|(t \gg y)}$$

$$\frac{x \xrightarrow{a(t)} \surd \quad U_t(y)}{x\|y \xrightarrow{a(t)} t \gg y \quad y\|x \xrightarrow{a(t)} t \gg y \quad x \between y \xrightarrow{a(t)} t \gg y}$$

If $a|a' = c \neq \delta$, then

$$\frac{x \xrightarrow{a(t)} x' \quad y \xrightarrow{a'(t)} y'}{x\|y \xrightarrow{c(t)} x'\|y' \quad x|y \xrightarrow{c(t)} x'\|y'} \qquad \frac{x \xrightarrow{a(t)} \surd \quad y \xrightarrow{a'(t)} \surd}{x\|y \xrightarrow{c(t)} \surd \quad x|y \xrightarrow{c(t)} \surd}$$

$$\frac{x \xrightarrow{a(t)} \surd \quad y \xrightarrow{a'(t)} y'}{x\|y \xrightarrow{c(t)} y' \quad y\|x \xrightarrow{c(t)} y' \quad x|y \xrightarrow{c(t)} y' \quad y|x \xrightarrow{c(t)} y'}$$

$$\frac{x \xrightarrow{a(t)} \surd \quad a \notin H}{\partial_H(x) \xrightarrow{a(t)} \surd} \qquad \frac{x \xrightarrow{a(t)} x' \quad a \notin H}{\partial_H(x) \xrightarrow{a(t)} \partial_H(x')}$$

$$\frac{U_t(x) \quad U_t(y)}{U_t(x\|y) \quad U_t(x \between y) \quad U_t(x|y)} \qquad \frac{U_t(x)}{U_t(\partial_H(x))}$$

Table 7.5: Additional action rules for ACP$\rho I$

### 7.4.2 Axioms for ACP$\rho I$

The axiomatization for ACP$\rho I$ consists of the (old) axioms in Table 7.4, together with the (new) axioms in Table 7.6. Some of these new axioms contain the construct $U_b(p)$, which represents a condition which results to true under a valuation $\sigma$ if and only if $U_{\sigma(b)}(\sigma(p))$. At the end of Table 7.6, three axioms U1-3 are added which enable to reduce the construct $U_b(p)$ to a condition of the form as defined in Section 7.2.1. Table 7.6 uses the abbreviation $U_v(\phi)$ for $U_v(\int_\phi \delta(v))$.

In the sequel, $p = q$ means that equality between these terms can be derived from the equational theory of ACP$\rho I$. We extend our decidability result to ACP$\rho I$ by showing how to eliminate the new operators from the syntax, using these axioms.

Similarly as in Proposition 7.5 we can deduce that if two time-closed terms are provably equal in the conditional axiom system, then they are bisimilar.

The following Proposition 7.27 says that the axioms of ACP$\rho I$ are sufficient to eliminate the communication and encapsulation operators from the syntax. Godskesen and Larsen [11] provided a rigorous proof that time dependencies are essential in order to obtain such a theorem in a timed setting. (Aceto and Murphy [1] pro-

| | | |
|---|---|---|
| CM1 | | $x\|\|y = x\between y + y\between x + x\|y$ |
| TCM2 | $v \notin fvar(y)$ | $\int_\phi \alpha(v)\between y = \int_{\phi\wedge U_v(y)} \alpha(v)\cdot y + \int_{U_v(\phi)\wedge U_v(y)} \delta(v)$ |
| TCM3 | $v \notin fvar(y)$ | $(\int_\phi \alpha(v)\cdot x)\between y = \int_{\phi\wedge U_v(y)} \alpha(v)\cdot(x\|\|y) + \int_{U_v(\phi)\wedge U_v(y)} \delta(v)$ |
| CM4 | | $(x+y)\between z = x\between z + y\between z$ |
| TCF | | $\int_\phi \alpha(v)\|\int_\psi \alpha'(v) = \int_{\phi\wedge\psi}(\alpha\|\alpha')(v) + \int_{U_v(\phi)\wedge U_v(\psi)} \delta(v)$ |
| TCM5 | | $(\int_\phi \alpha(v)\cdot x)\|\int_\psi \alpha'(v) = \int_{\phi\wedge\psi}(\alpha\|\alpha')(v)\cdot x + \int_{U_v(\phi)\wedge U_v(\psi)} \delta(v)$ |
| TCM6 | | $\int_\phi \alpha(v)\|\int_\psi \alpha'(v)\cdot y = \int_{\phi\wedge\psi}(\alpha\|\alpha')(v)\cdot y + \int_{U_v(\phi)\wedge U_v(\psi)} \delta(v)$ |
| TCM7 | | $(\int_\phi \alpha(v)\cdot x)\|\int_\psi \alpha'(v)\cdot y = \int_{\phi\wedge\psi}(\alpha\|\alpha')(v)\cdot(x\|\|y) + \int_{U_v(\phi)\wedge U_v(\psi)} \delta(v)$ |
| CM8 | | $(x+y)\|z = x\|z + y\|z$ |
| CM9 | | $x\|(y+z) = x\|y + x\|z$ |
| TD1 | | $\partial_H(\int_\phi \alpha(v)) = \int_\phi \partial_H(\alpha)(v)$ |
| TD2 | | $\partial_H(\int_\phi \alpha(v)\cdot x) = \int_\phi \partial_H(\alpha)(v)\cdot\partial_H(x)$ |
| D3 | | $\partial_H(x+y) = \partial_H(x) + \partial_H(y)$ |
| U1 | | $U_b(\int_{v\in V} P(v)) = b < \sup(V)$ |
| U2 | | $U_b(x+y) = U_b(x)\vee U_b(y)$ |
| U3 | | $U_b(\phi :\rightarrow x) = \phi\wedge U_b(x)$ |

Table 7.6: Axioms for ACP$\rho I$

posed the notion of 'ill-timed' traces, in order to obtain an expansion theorem for the merge in the absence of time dependencies.)

**Proposition 7.27** *For each ACP$\rho I$ term $p$, there is a BPA$\delta\rho I$ term $p'$ such that $p = p'$.*

*Proof.* Let $p_0$ and $p_1$ denote two BPA$\delta\rho I$ terms. First, suppose that $p$ is of the form $p_0\between p_1$. We show how to eliminate $\between$ from this term, by induction on the sizes of $p_0$ and $p_1$. Previously, we have shown for terms that do not contain communication nor encapsulation operators, that time shifts can be deleted. Hence, only the following cases need to be considered.

$$\begin{aligned}
\int_\phi \alpha(v)\between p_1 &= \int_{\phi\wedge U_v(p_1)} \alpha(v)\cdot p_1 + \int_{U_v(\phi)\wedge U_v(p_1)} \delta(v) \qquad v\notin fvar(p_1)\\
(\int_\phi \alpha(v)\cdot q)\between p_1 &= \int_{\phi\wedge U_v(p_1)} \alpha(v)\cdot(q\|p_1) + \int_{U_v(\phi)\wedge U_v(p_1)} \delta(v) \qquad v\notin fvar(p_1)\\
(q+q')\between p_1 &= q\between p_1 + q'\between p_1.
\end{aligned}$$

Similarly, we can delete $|$ from $p_0|p_1$, using the TCM axioms from Table 7.6. Since $p_0\|p_1 = p_0\between p_1 + p_1\between p_0 + p_0|p_1$, the case $p_0\|p_1$ reduces to the previous two cases. Finally, the $\partial_H$ can be deleted from $\partial_H(p_0)$ by means of the TD axioms.

Next, we deal with the general case, where may $p$ contain several communication and encapsulation operators. These operators can be deleted one by one, by considering subterms of $p$ of the form $p_0 \| p_1$ or $p_0 \between p_1$ or $p_0 | p_1$ or $\partial_H(p_0)$. $\Box$

In the previous section we have seen that by means of the axioms in Table 7.4 we can decide bisimulation equivalence between time-closed terms that do not contain communication nor encapsulation operators. Together with Proposition 7.27, it follows that the equational theory of ACP$\rho I$ decides bisimulation equivalence between time-closed terms.

**Corollary 7.28** *The equational theory of ACP$\rho I$ is complete for time-closed processes modulo bisimulation equivalence.*

**Corollary 7.29** *Bisimulation equivalence between time-closed processes in ACP$\rho I$ is decidable.*

## 7.5 Related Work

Recently, many process algebras have been extended with some notion of time. Most timed process algebras are based on relative time, in which the time stamp of an action refers to the point in time when the previous action was executed. This chapter focused on absolute time, which means that the time stamp of an action $a(t)$ refers to the start time of the entire process (time zero). A real time version of ACP with relative time was proposed in [3], in which square brackets are used to denote relative time; so $a[1] \cdot b[2]$ corresponds with the (absolute time) expression $a(1) \cdot b(3)$. Without any complications our results can be translated to ACP with relative time

We consider timed process algebras with a notion of integration and time dependencies. For example, we do not consider the work of Holmer, Larsen and Wang [12] on decidability in real time CCS, because their algebra does not incorporate time dependencies.

### 7.5.1 Timed CCS

Wang [23] introduced the construct $a@v.p$, which executes $a$ at some time $t$, after which it evolves into $p[t/v]$. His calculus is based on relative time, and unlike timed ACP, consecutive actions can be executed at the same point in time. The construct $a@v.p$ is equivalent to the expression $\int_{0 \leq v} a[v] \cdot p$ in BPA with relative time.

Chen [7] presented a generalization of Wang's construct $a@v.p$, namely $a(v)|_b^{b'}.p$, which executes $a$ at some point in time $t \in [b, b']$, after which it evolves into $p[t/v]$. This construct is similar to the expression $\int_{b \leq v \leq b'} a[v] \cdot p$ in BPA with relative time. The bounds $b$ and $b'$ allow to express time dependencies.

Chen obtained a decidability result by introducing for every pair of processes $p, q$ a first order formula $WC(p, q)$ which is the least condition such that $p$ and $q$ are

bisimilar. Decidability follows from the decidability of the first order theory of the underlying time domain, according to Tarski [21].

In [8], Chen introduced an axiom system with conditions. Derivations are relative to some condition; if two process terms $p, q$, possibly containing free time variables, are equal under the condition $\phi$, then $\phi \vdash p = q$. He shows that $WC(p, q)$ can be expressed by a condition, and that $WC(p, q) \vdash p = q$ is derivable, which induces that his axiom system is effective. In Chen's setting it is not possible to mix conditions through the terms. In order to explain the difference with our axiomatization, we rephrase axiom TC1 as a conditional proof rule:

$$\frac{\phi \vdash x = y}{tt \vdash \int_\phi \alpha(v) \cdot x = \int_\phi \alpha(v) \cdot y}$$

A derivation starting from a term $\int_\phi \alpha(v) \cdot x$ where $\phi$ is used 'deep down' in $x$, gives rise to a proof tree, while in our setting derivations are always equational.

### 7.5.2 Timed automata

Alur and Dill [2] proposed an extension of Büchi and Muller automata with time. Transitions are supplied with time constraints on 'clock variables', and while executing a transition, a clock can be set back to zero. A trace is accepted by a timed automaton if its transitions are performed at times that all clocks satisfy their constraints. Furthermore, accepted traces have to satisfy required fairness constraints, and Zeno behaviour is excluded from timed automata, i.e. traces are only accepted if they progress beyond any moment in time. The fairness restrictions, the non-Zeno requirement and the fact that only infinite traces are considered, are obstacles for the translation between timed automata and real time ACP with recursion. However, if these restrictions are discarded, then the classes of timed automata corresponds with a subalgebra of real time ACP with recursion which allows an elimination theorem for the merge, see Chapter 6.

Čerāns [5] introduced a more general notion of timed automata, which he calls timed graphs. For example, in his setting edges of automata are painted a color, red or black, which determines whether or not idling is allowed when the edge becomes enabled. Čerāns proved that bisimulation equivalence is decidable for timed graphs. This result is incomparable with ours, because his timed algebra is so different.

## 7.6 Appendix: Three Proofs

**Refinement Lemma.** Fix a time variable $v$. Each condition $\phi$ is equal to a condition of the form $\vee_i(\phi_i \wedge \phi'_i)$, where

- $var(\phi_i) \subseteq var(\phi) \backslash \{v\}$,
- $\phi'_i$ is of the form $v = b$ or $v < b$ or $b < v$ or $b < v < b'$, with $var(b + b') \subseteq var(\phi) \backslash \{v\}$.

**Proof sketch.** First, rewrite $\phi$ to a condition of the form $\vee_i \psi_i$, with each $\psi_i$ of the form $\wedge_j (b_j < b'_j) \wedge \wedge_k (c_k = c'_k)$. Reduce the bounds in $\psi_i$ to normal form, i.e. to the form $r_1 \cdot v_1 + ... + r_l \cdot v_l + s$. In each (in)equality, collect factors $r \cdot v$ at one side, and collect the remaining summands of the bounds on the other side, such that either $v$ is deleted from the (in)equality, or it takes the form $r \cdot v = b$ or $r \cdot v < b$, with $r \neq 0$ and $v \notin var(b)$. In the latter case, replace the (in)equality by $v = (1/r) \cdot b$ or by $v < (1/r) \cdot b$ if $r > 0$ or by $(1/r) \cdot b < v$ if $r < 0$. Thus we can reduce each $\psi_i$ to an equivalent condition $\psi'_i$ of the form

$$\psi \ \wedge \ \bigwedge_{j \in J} b_j < v \ \wedge \ \bigwedge_{k \in K} v < c_k \ \wedge \ \bigwedge_{l \in L} v = d_l$$

where $v$ does not occur in $\psi, b_j, c_k, d_l$. We show that such a $\psi'_i$ is equivalent to a condition of the form $\vee_j (\phi'_j \wedge v \in V_j)$, with $v \notin var(\phi'_j) \cup var(V_j)$.

First, suppose $L \neq \emptyset$. Fix an $l_0 \in L$ and put $d = d_{l_0}$. The following condition is equivalent to $\psi'_i$.

$$(\psi \wedge \bigwedge_{j \in J} b_j < d \wedge \bigwedge_{k \in K} d < c_k \wedge \bigwedge_{l \in L} d = d_l) \ \wedge \ v = d.$$

So we may assume $L = \emptyset$. If $J \neq \emptyset$ and $K \neq \emptyset$, then the following condition is equivalent to $\psi'_i$.

$$\bigvee_{(j,k) \in J \times K} (\psi \wedge \bigwedge_{j' \in J} b_{j'} \leq b_j \wedge \bigwedge_{k' \in K} c_k \leq c_{k'} \wedge b_j < v < c_k).$$

Similarly, we can find suitable conditions equivalent to $\psi'_i$ if $J$ or $K$ is empty. □

**Completeness of CA.** If $[\phi] = [\phi']$, then $\phi = \phi'$.

**Proof sketch.** We apply induction on the number of variables that occur in $\phi$ and $\phi'$. If this number is zero, then the proposition is trivial, since then both $\phi$ and $\phi'$ reduce to either *tt* or *ff*. So assume that we have proved the case for $n$ variables, and let $\phi$ and $\phi'$ contain $n+1$ variables. Fix a variable $v$ that occurs in $\phi$ or in $\phi'$. According to the refinement lemma, we have

$$\phi = \bigvee_i \psi_i \wedge (v \in V_{i1} \vee ... \vee v \in V_{im_i}) \qquad \phi' = \bigvee_i \psi_i \wedge (v \in W_{i1} \vee ... \vee v \in W_{in_i})$$

where $v$ does not occur in the $\psi_i$, $V_{ij}$, $W_{ij}$, and $\{\psi_i\}$ is a partition. Furthermore, ensure that under condition $\psi_i$ both the $V_{ij}$ and the $W_{ij}$ are pairwise disjoint and non-empty. Moreover, ensure that if $\sigma \in [\psi_i]$, then the elements in $\sigma(V_{ij})$ and in $\sigma(W_{ij})$ are smaller than the elements in $\sigma(V_{ij+1})$ and in $\sigma(W_{ij+1})$ respectively.

Consider a $\psi_i$, and suppose that $\sigma \in [\psi_i]$. Since $\sigma(\phi) = \sigma(\phi')$, and since $\{\psi_i\}$ is a partition, it follows that $\sigma(v \in V_{i1} \vee ... \vee v \in V_{im_i}) = \sigma(v \in W_{i1} \vee ... \vee v \in W_{in_i})$. The $\sigma(V_{ij})$ and the $\sigma(W_{ij})$ are pairwise disjoint and of increasing order, so $m_i = n_i$ and $\sigma(V_{ij}) = \sigma(W_{ij})$ for all $j$. So if $V_{ij}$ has lower bound $b_j$ and upper bound $b'_j$, and let $W_{ij}$ have lower bound $c_j$ and upper bound $c'_j$, then $\sigma(\psi_i \wedge b_j = c_j \wedge b'_j = c'_j) = \sigma(\psi_i)$. This equality holds as well if $\sigma(\psi_i)$ results to false, so the induction hypothesis yields

$(\psi_i \wedge b_j = c_j \wedge b'_j = c'_j) = \psi_i$. Hence, $\psi_i \wedge v \in V_{ij} \;=\; \psi_i \wedge v \in W_{ij}$. This holds for all $i$ and $j$, so

$$\bigvee_i \psi_i \wedge (v \in V_{i1} \vee ... \vee v \in V_{im_i}) \;=\; \bigvee_i \psi_i \wedge (v \in W_{i1} \vee ... \vee v \in W_{in_i}). \;\Box$$

**Lemma 7.22.** Let $p$ and $q$ be subterms of normal forms. If $p[r/v]* =_{\mathrm{AC}} q[r/v]*$ for infinitely many $r \in \mathit{Time}_0$, then $p =_{\mathrm{AC}} q$.

**Proof.** For a bound $b$, let $b\!\downarrow$ be its normal form. Note that if $b_0\!\downarrow \neq_{\mathrm{AC}} b_1\!\downarrow$, then there is at most one $r \in \mathit{Time}_0$ such that $b_0[r/v]\!\downarrow \;=_{\mathrm{AC}} b_1[r/v]\!\downarrow$. Let $\lhd b_0, b_1 \rhd\!\downarrow$ denote $\lhd b_0\!\downarrow, b_1\!\downarrow \rhd$.

We use induction on the depth of $p$ and $q$. Let

$$p \;=_{\mathrm{AC}}\; \sum_i \int_{w \in V_i} a_i(w) \cdot p_i \;+\; \sum_j \int_{w \in W_j} \alpha_j(w),$$

$$q \;=_{\mathrm{AC}}\; \sum_k \int_{w \in V'_k} a'_k(w) \cdot q_k \;+\; \sum_l \int_{w \in W'_l} \alpha'_l(w).$$

Assume that $p \neq_{\mathrm{AC}} q$; we show that $p[r/v]* =_{\mathrm{AC}} q[r/v]*$ for only finitely many $r \in \mathit{Time}_0$. We distinguish two cases.

1. There is a $j$ such that for all $l$ we have $\int_{w \in W_j} \alpha_j(w) \neq_{\mathrm{AC}} \int_{w \in W'_l} \alpha'_l(w)$.

   Fix an $l$. If $\alpha_j \neq \alpha'_l$, then clearly $\int_{w \in W_j[r/v]\!\downarrow} \alpha_j(w) \neq_{\mathrm{AC}} \int_{w \in W'_l[r/v]\!\downarrow} \alpha'_l(w)$ for all $r$.

   So assume that $\alpha_j = \alpha'_l$. Then $W_j \neq_{\mathrm{AC}} W'_l$, so there is no more than one $r \in \mathit{Time}_0$ such that $W_j[r/v]\!\downarrow = W'_l[r/v]\!\downarrow$.

   It follows that the set $\{r \in \mathit{Time} \mid p[r/v]* =_{\mathrm{AC}} q[r/v]*\}$ is smaller or equal to the number of $l$'s (and thus finite).

2. There is an $i$ such that for all $k$ we have $\int_{w \in V_i} a_i(w) \cdot p_i \neq_{\mathrm{AC}} \int_{w \in V'_k} a'_k(w) \cdot q_k$.

   Fix a $k$ with $a_i = a'_k$. If $V_i \neq_{\mathrm{AC}} V'_k$, then it follows as in 1 that there is no more than one $r$ such that

   $$(\int_{w \in V_i} a_i(w) \cdot p_i)[r/v]* =_{\mathrm{AC}} (\int_{w \in V'_k} a'_k(w) \cdot q_k)[r/v] * .$$

   So assume that $p_i \neq_{\mathrm{AC}} q_k$. Then by the induction hypothesis there is only a finite number of $r$ such that $p_i[r/v]* =_{\mathrm{AC}} q_k[r/v]*$. Furthermore, if $V_i$ or $V'_k$ is not of the form $[b, b]$, then there is no more than one $r$ such that $V_i[r/v]\!\downarrow$ or $V'_k[r/v]\!\downarrow$ does have this form respectively.

   It follows that $\{r \in V \cap \mathit{Time}_0 \mid p[r/v]* =_{\mathrm{AC}} q[r/v]*\}$ is finite. $\Box$

# References

[1] L. Aceto and D. Murphy. On the ill-timed but well-caused. In E. Best, editor, *Proceedings 4th Conference on Concurrency Theory (CONCUR'93)*, Hildesheim, *LNCS 715*, pages 97–111. Springer-Verlag, 1993.

[2] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[4] B. Bloom and F.W. Vaandrager. SOS rule formats for parametrized and state-bearing processes. Unpublished manuscript, 1994.

[5] K. Čerāns. Decidability of bisimulation equivalence for processes with parallel timers. Technical report, University of Latvia, Riga, 1992.

[6] C.C. Chang and H.J. Keisler. *Model Theory*. North-Holland, 1990.

[7] L. Chen. An interleaving model for real-time systems. In A. Nerode and M. Taitslin, editors, *Proceedings 2nd Symposium on Logical Foundations of Computer Science,* Tver, Russia, *LNCS 620*, pages 81–92. Springer-Verlag, 1992.

[8] L. Chen. Axiomatising real-timed processes. In S. Brooks, M. Main, A. Melton, M. Mislove and D. Schmidt, editors, *Proceedings 9th Conference on Mathematical Foundations of Programming Semantics (MFPS'93),* New Orleans, *LNCS 802*, pages 215–229. Springer-Verlag, 1993.

[9] W.J. Fokkink. Normal forms in real time process algebra. Report CS-R9149, CWI, Amsterdam, 1991.

[10] W.J. Fokkink and C. Verhoef. Unpublished manuscript on conservative extensions in operational semantics with types and variable binding.

[11] J.C. Godskesen and K.G. Larsen. Real time calculi and expansion theorems. In R. Shyamasundar, editor, *Proceedings 12th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 12),* New Delhi, *LNCS 652*, pages 302–315. Springer-Verlag, 1992.

[12] U. Holmer, K.G. Larsen, and Y. Wang. Deciding properties of regular real timed processes. In K.G. Larsen and A. Skou, editors, *Proceedings 3rd Workshop on Computer Aided Verification (CAV'91),* Aalborg, *LNCS 575*, pages 432–442. Springer-Verlag, 1991.

[13] A.S. Klusener. Completeness in real time process algebra. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings 2nd Conference on Concurrency Theory (CONCUR'91),* Amsterdam, *LNCS 527*, pages 376–392. Springer-Verlag, 1991.

[14] A.S. Klusener. Abstraction in real time process algebra. In J.W. de Bakker, C. Huizing, W.P. de Roever and G. Rozenberg, editors, *Proceedings REX Workshop "Real Time: Theory in Practice",* Mook, *LNCS 600*, pages 325–352. Springer-Verlag, 1991.

[15] A.S. Klusener. The silent step in time. In W.R. Cleaveland, editor, *Proceedings 3rd Conference on Concurrency Theory (CONCUR'92),* Stony Brook, *LNCS 630*, pages 421–435. Springer-Verlag, 1992.

[16] A.S. Klusener. *Models and Axioms for a Fragment of Real Time Process Algebra*. PhD thesis, Eindhoven University of Technology, 1993.

[17] S. Koppelberg. General theory of Boolean algebras. In J.D. Monk, editor, *Handbook of Boolean Algebras, Volume 1.* North-Holland, 1989.

[18] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings 1st Conference on Concurrency Theory (CONCUR'90),* Amsterdam, *LNCS 458*, pages 401–415. Springer-Verlag, 1990.

[19] S.A. Schneider. An operational semantics for timed CSP. Report TR1-91, Oxford University, 1991. To appear in *Information and Computation.*

[20] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.

[21] A. Tarski. *A New Decision Method for Elementary Algebra*. University of California Press, 1951.

[22] Y. Wang. Real time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings 1st Conference on Concurrency Theory (CONCUR'90),* Amsterdam, *LNCS 458*, pages 502–520. Springer-Verlag, 1990.

[23] Y. Wang. CCS + time = an interleaving model for real time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings 18th International Colloquium on Automata, Languages and Programming (ICALP'91),* Madrid, *LNCS 510*, pages 217–228. Springer-Verlag, 1991.

# 8

# Complete Axioms for Timed Regular Processes with Silent Steps

Wan Fokkink

First, we consider BPA$\delta$ with recursion, extended with relative time. It is proved that the axioms for BPA$\delta$ with time, together with two standard axioms for (untimed) recursion, are complete for regular processes modulo bisimulation.
Next, the syntax is extended with the silent step $\tau$, and a timed version of branching bisimulation is defined. We add one axiom for the silent step to the axiom system, and prove that this axiomatization is complete for regular processes modulo timed branching bisimulation.

## 8.1 Introduction

Over the years, process algebras such as CCS, CSP and ACP have been extended with a notion of time. This chapter is based on the approach of Baeten and Bergstra [2], which extends ACP with real time.

The first part of this chapter considers BPA$\delta r$, which denotes Basic Process Algebra with deadlock and relative time. Terms are constructed from timed atoms $\alpha[r]$, the alternative composition $x+y$ and the sequential composition $x\cdot y$. Processes are considered in relative time, i.e. a timed action $a[r]$ executes action $a$ exactly $r$ time units after the previous action has been executed. In this chapter, the algebra BPA$\delta r$ is extended with a notion of recursion.

We consider the two axioms R1,2 for regular processes from Bergstra and Klop [5]. In that paper it is proved that R1,2 together with the standard axioms for BPA$\delta$ are complete for the algebra of (untimed) regular processes modulo strong bisimulation. In this chapter, we prove that R1,2 together with the standard axioms for the algebra BPA$\delta r$ are complete for the algebra of timed regular processes. This completeness result is obtained by means of an algorithm which reduces bisimilar terms to the same normal form.

Milner [15] derived completeness of an axiomatization for regular behaviours in the untimed case, with respect to strong bisimulation. Chen [7] deduced a similar result for his extension of CCS with real time. Aceto and Jeffrey [1], deduced a similar result for the regular subcalculus of Wang Yi's timed CCS [20, 11]. Their setting incorporates abstraction in the setting of strong bisimulation.

Next, we extend the syntax with the silent step $\tau$. The semantics for abstraction is based on the untimed branching bisimulation of van Glabbeek and Weijland [9]. This equivalence preserves the branching structure of processes; a $\tau$-transition is silent if and only if it does not lose possible behaviours. Klusener [12] introduced a timed version of branching bisimulation. Again, $\tau[r]$-transitions may be omitted under the condition that they do not lose possible behaviours. However, this same intuition in the timed setting gives rise to quite a different mathematical interpretation than in the untimed case. As always, the definition of bisimulation with abstraction in the setting of time becomes deplorably complicated, cf weak bisimulation in [17, 6, 18, 10]. However, its axiomatization is crystal clear. In our setting, one simple axiom from Klusener [14] is sufficient to describe equivalences that involve abstraction. Hence, performing calculations in the algebra is quite feasible, in spite of the complicated definition of its semantics.

We deduce a completeness result for our algebra of timed regular processes with abstraction. Milner [16] and Bergstra and Klop [5] have deduced similar results for the untimed case. In that last paper, no less than three extra axioms were needed to deal with complications regarding recursion in the presence of abstraction. This is mainly due to the fact that $\tau$ is not a 'guard' for recursion; the recursive equation $X = \tau \cdot X$ has infinitely many solutions $\tau \cdot p$ for $X$. However, this complication disappears in the presence of time; the recursive equation $X = \tau[r] \cdot X$ has only one solution $\tau[r] \cdot \delta[\infty]$ for $X$. A similar observation was already made by Reed and Roscoe [19], in a setting of timed CSP with topological models. As a consequence of this phenomenon, no extra axioms are needed for recursion in the timed setting. To obtain a complete axiomatization, we only have to add the characterizing axiom for abstraction to the axiom system.

To cut down notational overhead as much as possible, we leave out the communication operators and the encapsulation operator from ACP. Although these operators are important for the expressive power of the formalism, they are not essential for presenting the main ideas of this paper. A straightforward collection of axioms from [2], together with axioms R1,2, suffice to eliminate these operators from the syntax, see [8].[1]

**Acknowledgements.** Steven Klusener and Frits Vaandrager provided helpful comments.

[1]For this elimination result it is essential that our time domain consists of the rationals, instead of the reals. For example, if $X = a[1] \cdot X$ and $Y = b[\sqrt{2}] \cdot Y$, then the merge cannot be eliminated from $X \| Y$.

## 8.2 Timed Regular Processes

We study the formalism BPA$\delta r$, which stands for Basic Process Algebra with deadlock, extended with relative time. We assume an alphabet $A$ of atomic actions, together with the special constant $\delta$ to represent deadlock. In the sequel, $a$ and $\alpha$ denote elements of $A$ and $A \cup \{\delta\}$ respectively. A *timed action* is of the form $\alpha[r]$ with $r \in \mathbb{Q} \cup \{\infty\}$. Here, $\infty$ denotes a special time element 'infinity' that is greater than any rational. Moreover, we assume the binary operators alternative composition $+$ and sequential composition $\cdot$.

The timed deadlock $\delta[r]$ can only idle until time $r$. For example, the process $a[1]+\delta[2]$ can either execute the $a$ at time 1 or idle until time 2. On the other hand, the process $a[1] + \delta[1]$ will always execute the $a$ at time 1.

### 8.2.1 Recursion

A *recursive specification* $E$ is a finite set of equations $\{X_i = t_i |\ i = 1, ..., n\}$, where the $X_i$ are recursion variables, and the $t_i$ are process terms constructed from timed actions, the alternative composition, the sequential composition and the variables $X_j$ for $j = 1, ..., n$.

A *solution* of the recursive specification $E$, in a certain model of BPA$\delta r$, is a collection of processes $\{p_i |\ i = 1, ..., n\}$ such that the equations $X_i = t_i$ become true (in the model) if the $p_i$ are substituted for the $X_i$.

The syntactic construct $\langle X|E\rangle$ denotes a solution of $X$ with respect to $E$. It can be regarded as some kind of variable, ranging over the collection of solutions of $X$. By abuse of notation $\langle X|E\rangle$ is often abbreviated by $X$.

In the sequel, we only consider *linear* recursive specifications, which consist of equations of the form

$$X = \sum_i \alpha_i[r_i] \cdot Y_i + \sum_j \beta_j[s_j].$$

### 8.2.2 Operational semantics

Table 8.1 contains the action rules that define the operational semantics for BPA$\delta r$ with recursion. This operational semantics, taken from [13], does not yield any idle transitions; a process $a[r]$ only executes the $a$ at time $r$. A similar operational semantics can be found in [11].

Processes are considered in relative time. This means that an action $a[r]$ executes $a$ exactly $r$ time units after the previous action has been executed. For example, the process $a[1] \cdot b[2]$ first executes $a$ at time 1, and then $b$ at (absolute) time 3. Note that time starts at zero and never reaches infinity, so actions $a[r]$ with $r \leq 0$ and $r = \infty$ do not display any behaviour.

The expression $E$ in the two action rules for recursion represents a linear recursive specification. Moreover, $\langle t_1|E\rangle$ denotes $t_1$ with occurrences of variables $X_i$ replaced by $\langle X_i|E\rangle$.

$$a[r] \xrightarrow{a[r]} \surd \qquad \text{if } 0 < r < \infty$$

$$\frac{x \xrightarrow{a[r]} \surd}{x + y \xrightarrow{a[r]} \surd \xleftarrow{a[r]} y + x} \qquad \frac{x \xrightarrow{a[r]} x'}{x + y \xrightarrow{a[r]} x' \xleftarrow{a[r]} y + x}$$

$$\frac{x \xrightarrow{a[r]} \surd}{x \cdot y \xrightarrow{a[r]} y} \qquad \frac{x \xrightarrow{a[r]} x'}{x \cdot y \xrightarrow{a[r]} x' \cdot y}$$

$$\frac{\langle t_1 | E\rangle \xrightarrow{a[r]} \surd}{\langle X_1 | E\rangle \xrightarrow{a[r]} \surd} \qquad \frac{\langle t_1 | E\rangle \xrightarrow{a[r]} y}{\langle X_1 | E\rangle \xrightarrow{a[r]} y}$$

Table 8.1: Action rules for BPA$\delta r$

### 8.2.3 Strong bisimulation

The *ultimate delay* $U(p)$ is the latest moment in time up to which process $p$ can idle without executing an initial action. It is defined inductively as follows:

$$\begin{array}{lcl} U(\alpha[r]) & = & \max\{r, 0\} \\ U(p + q) & = & \max\{U(p), U(q)\} \\ U(p \cdot q) & = & U(p) \\ U(\langle X_i | E\rangle) & = & U(\langle t_i | E\rangle). \end{array}$$

$U(p)$ enables to distinguish processes that only differ in their deadlock behaviour, such as $a[1] + \delta[1]$ and $a[1] + \delta[2]$.

**Definition 8.1** *Two process expressions $p_0, q_0$ are* (strongly) bisimilar, *notation $p_0 \leftrightarrows q_0$, if there exists a symmetric, binary* bisimulation relation $\mathcal{B}$ *on processes such that*

1. $p_0 \mathcal{B} q_0$.
2. *If $p \xrightarrow{a[r]} p'$ and $p \mathcal{B} q$, then $q \xrightarrow{a[r]} q'$ for some process $q'$ with $p' \mathcal{B} q'$.*
3. *If $p \xrightarrow{a[r]} \surd$ and $p \mathcal{B} q$, then $q \xrightarrow{a[r]} \surd$.*
4. *If $p \mathcal{B} q$, then $U(p) = U(q)$.*

### 8.2.4 Regular processes

As a model for BPA$\delta r$ with recursion we take the collection $\mathcal{R}$ of *regular* process terms, modulo bisimulation. Regular terms are constructed from the timed actions,

expressions $\langle X|E\rangle$ with $E$ a linear recursive specification, and the alternative and the sequential composition. Process terms in $\mathcal{R}$ are considered modulo bisimulation equivalence.

The terminology 'regular' is justified by the following observation. For each process $p$, let $Trans(p)$ be the smallest collections of transitions such that the following statements are true, where $r$ ranges over $\mathbb{Q}_{>0}$.

$$p \underline{\leftrightarrow} p + a[r] \cdot q \implies (p, a[r], q) \in Trans(p),$$

$$p \underline{\leftrightarrow} p + a[r] \implies (p, a[r]) \in Trans(p),$$

$$(p, a[r], q) \in Trans(p) \implies Trans(q) \subseteq Trans(p).$$

For each $p \in \mathcal{R}$ the collection $Trans(p)$ is finite, if bisimilar terms are identified (cf [4] for the untimed case).

### 8.2.5 An axiom system

Strong bisimulation is a congruence on $\mathcal{R}$, which means that if $p \underline{\leftrightarrow} p'$ and $q \underline{\leftrightarrow} q'$, then $p+q \underline{\leftrightarrow} p'+q'$ and $p \cdot q \underline{\leftrightarrow} p' \cdot q'$. This property follows from the *path* format of Baeten and Verhoef [3]. They proved that if a collection of transition rules is within this format, then the strong bisimulation equivalence it induces on the algebra of closed terms is always a congruence. We can extend our operational semantics with rules that define predicates $U(p) = r$ for $r \in \mathbb{Q}_{>0} \cup \{\infty\}$. Our extended operational semantics fits the path format, and Definition 8.1 yields exactly the definition for strong bisimulation in the setting with the ultimate delay predicates.

Table 8.2 contains an axiom system for BPA$\delta r$ with recursion. Axioms A1-5 are the standard axioms from BPA, and the axioms TA6-9 are taken from [13]. In that paper it is proved that the axioms constitute a complete proof system for the model of closed terms (i.e. process terms not containing recursion variables) modulo bisimulation. Finally, the axioms R1,2 for recursion stem from [5]. R1 induces equalities like $\langle X|X = a[r] \cdot X\rangle = a[r] \cdot \langle X|X = a[r] \cdot X\rangle$, and R2 (or the *Recursive Specification Principle*) says that each linear recursive specification has only one solution. In both axioms, $E$ denotes a linear recursive specification of the form $\{X_i = t_i |\ i = 1, ..., n\}$.

### 8.2.6 Completeness

Consider the model $\mathcal{R}$. It is easy to see that the axioms A1-5 and TA6-9 and R1 are sound with respect to bisimulation. A detailed proof of the soundness of R2 can be found in [8]. We now prove that these axioms together constitute a complete axiomatization for $\mathcal{R}$. Namely, we shall present a strategy to reduce each solution of a linear specification to normal form, by means of the axioms. Next, we shall show that if two normal forms are bisimilar, then they are syntactically equal modulo $\alpha$-conversion (i.e. modulo renaming of variables). This proves completeness.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A1 | | | | | $x+y$ | $=$ | $y+x$ |
| A2 | | | | | $(x+y)+z$ | $=$ | $x+(y+z)$ |
| A3 | | | | | $x+x$ | $=$ | $x$ |
| A4 | | | | | $(x+y)\cdot z$ | $=$ | $x\cdot z+y\cdot z$ |
| A5 | | | | | $(x\cdot y)\cdot z$ | $=$ | $x\cdot(y\cdot z)$ |
| TA6 | | | $s\leq r$ | $\Longrightarrow$ | $\alpha[r]+\delta[s]$ | $=$ | $\alpha[r]$ |
| TA7 | | | | | $\delta[r]\cdot x$ | $=$ | $\delta[r]$ |
| TA8 | | | $r\leq 0$ | $\Longrightarrow$ | $\alpha[r]$ | $=$ | $\delta[0]$ |
| TA9 | | | | | $a[\infty]$ | $=$ | $\delta[\infty]$ |
| R1 | $p_i = \langle X_i\|E\rangle$ | $i=1,...,n$ | $\Longrightarrow$ | $p_1$ | $=$ | $t_1[p_1/X_1,...,p_n/X_n]$ | |
| R2 | $p_i = t_i[p_1/X_1,...,p_n/X_n]$ | $i=1,...,n$ | $\Longrightarrow$ | $p_1$ | $=$ | $\langle X_1\|E\rangle$ | |

Table 8.2: Axioms for timed regular processes

Let $E=\{X_i=t_i|\ i=1,...,n\}$ be a linear specification. In order to reduce the process $\langle X_1|E\rangle$ to normal form, we reduce $E$ in several steps.

## Step 1: Removal of redundant deadlocks

- First, replace each expression in $t_i$ of the form $\alpha[r]$ with $r\leq 0$ by $\delta[0]$ and each expression of the form $a[\infty]$ by $\delta[\infty]$.

- Next, replace each expression in $t_i$ of the form $\delta[r]\cdot X$ by $\delta[r]$.

- Finally, remove each expression $\delta[r]$ from $t_i$ for which there is an expression $a[s]\cdot X$ or $\alpha[s]$ in $t_i$ with $r\leq s$.

## Step 2: Identification of bisimilar variables

If $\langle X_j|E\rangle \leftrightarrow \langle X_k|E\rangle$ with $j<k$, then rename $X_k$ in the $t_i$ into $X_j$.

## Step 3: Removal of double edges

If an expression $a[r]$ or $a[r]\cdot X_j$ occurs in $t_i$ more than once, then remove all but one of the occurrences of this expression in $t_i$.

## Step 4: Removal of redundant variables

Let the collection $dep(X_1)$ of variables in $E$ that occur in the 'dependency graph' of $X_1$ be defined as follows:

$$X_1 \in dep(X_1),$$
$$X_i \in dep(X_1) \text{ and } X_j \text{ occurs in } t_i \implies X_j \in dep(X_1).$$

If $X_j \notin dep(X_1)$, then remove the equation $X_j = t_j$ from $E$.

Thus we have constructed the normal form of $\langle X_1|E\rangle$. Step 1 is provable from R1,2 and TA6-9, Step 3 from R1,2 plus A3, and Step 4 from R1,2. We show that Step 2 can be proved from R1,2+A3. Let $\tilde{E}$ be the specification that results after identifying bisimilar variables in $E$. Let $X_{i(j)}$ denote the bisimilar variable that has been substituted for $X_j$ in $\tilde{E}$, for $j = 1, ..., n$.

**Proposition 8.2** $\quad \mathrm{R1,2} + \mathrm{A3} \vdash \langle X_1|E\rangle = \langle X_1|\tilde{E}\rangle$.

**Proof**. Let $T_j$ denote the process $t_j[\langle X_{i(1)}|\tilde{E}\rangle/X_1, ..., \langle X_{i(n)}|\tilde{E}\rangle/X_n]$. It is easy to see that $\langle X_j|E\rangle \leftrightarrows T_j$ for $j = 1, ..., n$. Since $\langle X_j|E\rangle \leftrightarrows \langle X_{i(j)}|E\rangle$, this implies $T_j \leftrightarrows T_{i(j)}$ for $j = 1, ..., n$.

So if $T_j$ has a subterm $a[r] \cdot \langle X_k|\tilde{E}\rangle$, then $T_{i(j)}$ has a subterm $a[r] \cdot \langle X_l|\tilde{E}\rangle$ with $\langle X_k|\tilde{E}\rangle \leftrightarrows \langle X_l|\tilde{E}\rangle$. Bisimilar variables have been identified in $\tilde{E}$, so $k = l$. By the same argument, each subterm $a[r] \cdot \langle X_k|\tilde{E}\rangle$ of $T_{i(j)}$ is also a subterm of $T_j$. Similarly, $T_j$ has a subterm $a[r]$ if and only if $T_{i(j)}$ has a subterm $a[r]$. Finally, since $U(T_j) = U(T_{i(j)})$, Step 1 in the reduction to normal form ensures that $T_j$ has a subterm $\delta[r]$ if and only if $T_{i(j)}$ has a subterm $\delta[r]$. Thus $\mathrm{A3} \vdash T_j = T_{i(j)}$.

Then $\langle X_{i(j)}|\tilde{E}\rangle \overset{\mathrm{R1}}{=} T_{i(j)} \overset{\mathrm{A3}}{=} T_j$. This holds for all $j$, so $\langle X_{i(1)}|\tilde{E}\rangle, ..., \langle X_{i(n)}|\tilde{E}\rangle$ is a solution for $E$. Then R2 implies $\langle X_j|E\rangle = \langle X_{i(j)}|\tilde{E}\rangle$ for $j = 1, ..., n$. □

The next theorem implies that A1-5+TA6-9+R1,2 constitutes a complete axiomatization for regular processes modulo strong bisimulation.

**Theorem 8.3** *If two normal forms $\langle X_1|E\rangle$ and $\langle Y_1|E'\rangle$ are bisimilar, then they are syntactically equivalent modulo $\alpha$-conversion.*

**Proof**. Let

$$\begin{aligned} E &= \{X_i = t_i |\ i = 1, ..., m\}, \\ E' &= \{Y_j = s_j |\ j = 1, ..., n\}. \end{aligned}$$

We construct inductively a mapping $\phi$ from the variables of $E$ to the variables of $E'$, such that $\langle X_i|E\rangle \leftrightarrows \langle \phi(X_i)|E'\rangle$ for each $i$, and if $\phi(X_i) = Y_j$, then $\phi \circ t_i$ yields $s_j$.

Put $\phi(X_1) = Y_1$. Now suppose that we have defined $\phi(X_i) = Y_j$ for some $i$. Let

$$t_i = \sum_k a_k[r_k] \cdot X_{i_k} + \sum_l \alpha_l[s_l].$$

Since $\langle X_i|E\rangle \leftrightarrow \langle Y_j|E'\rangle$, $s_j$ has a subterm $\alpha_l[s_l]$ for each $l$, and a subterm $a_k[r_k]\cdot Y_{j_k}$ with $\langle X_{i_k}|E\rangle \leftrightarrow \langle Y_{j_k}|E'\rangle$ for each $k$.

If $\phi(X_{i_k})$ has already been defined, then $\langle \phi(X_{i_k})|E'\rangle \leftrightarrow \langle X_{i_k}|E\rangle \leftrightarrow \langle Y_{j_k}|E'\rangle$ by induction. Since bisimilar variables have been identified in Step 2, it follows that $\phi(X_{i_k}) = Y_{j_k}$. If $\phi(X_{i_k})$ has not yet been defined, then put $\phi(X_{i_k}) = Y_{j_k}$.

In Step 3 double edges have been removed, so subterms $\alpha_l[s_l]$ and $a_k[r_k]\cdot X_{i_k}$ and $a_k[r_k]\cdot Y_{j_k}$ occur in $t_i$ and $s_j$ only once. Hence, $\phi \circ t_i$ yields $s_j$.

By Step 4 each variable is in the dependency graph of $X_1$, so in the end $\phi$ is defined for all variables in $E$. It is easy to see that $\phi$ is bijective, since by a symmetric construction one can define its inverse. □

**Corollary 8.4** A1-5+TA6-9+R1,2 *form a complete axiomatization for* $\mathcal{R}$ *modulo strong bisimulation.*

## 8.3 Abstraction

The previous section treated the model $\mathcal{R}$ modulo strong bisimulation. In this section we extend the syntax with the special constant $\tau$, and we consider the model $\mathcal{R}_\tau$, where processes are considered modulo *rooted branching* bisimulation.

### 8.3.1 The time shift

In order to define branching bisimulation, we need the *time shift* operator $(r)p$, which takes a rational $r$ and a process term $p$. The process $(r)p$ denotes the behaviour of $p$ that is shifted forward $r$ units in time. Its operational semantics and axioms are given in Table 8.3. We extend the syntax with this construct. It is easy to see that, using axioms TS1-4, the time shift can be eliminated from all process terms.

$$\frac{x \xrightarrow{a[r]} \surd \quad r+s>0}{(s)x \xrightarrow{a[r+s]} \surd} \qquad \frac{x \xrightarrow{a[r]} x' \quad r+s>0}{(s)x \xrightarrow{a[r+s]} x'}$$

| | | | | | |
|---|---|---|---|---|---|
| TS1 | $s>0$ | $\Longrightarrow$ | $(r)\alpha[s]$ | $=$ | $\alpha[r+s]$ |
| TS2 | | | $(r)\delta[0]$ | $=$ | $\delta[r]$ |
| TS3 | | | $(r)(x+y)$ | $=$ | $(r)x+(r)y$ |
| TS4 | | | $(r)(x\cdot y)$ | $=$ | $(r)x\cdot y$ |

Table 8.3: Action rules and axioms for the time shift

### 8.3.2 Branching bisimulation

We add the silent step $\tau$ to the alphabet. In the sequel, $a$ and $\alpha$ will represent elements from $A \cup \{\tau\}$ and $A \cup \{\delta, \tau\}$ respectively. The operational semantics still consists of the action rules from Table 8.1. Only, the definition of strong bisimulation is adapted to that of *branching* bisimulation from [14].

In untimed branching bisimulation, a $\tau$-transition is invisible if it does not lose possible behaviours, or in other words, $\tau p + q$ is equivalent to $p$ if $q$ is semantically included in $p$. The same intuition is used to define timed branching bisimulation. In timed branching bisimulation a transition $p \xrightarrow{\tau[r]} p'$ may be matched with the passing of time in $q$ if $U(q) > r$ and $p' \leftrightarrow_b (-r)q$. Because this implies that executing the $\tau[r]$-transition in $p$ and idling beyond $r$ in $q$ result in equivalent behaviours. However, this same intuition gives rise to a mathematical interpretation that is quite different from the untimed case. This is shown by the following examples.

**Example 8.5** *In the untimed setting,* $\tau(a + b) + a \leftrightarrow_b a + b$. *However,*

$$\tau[1] \cdot (a[1] + b[1]) + a[2] \not\leftrightarrow_b a[2] + b[2].$$

*Not executing the $\tau$ at 1 in the process on the left means a decision that the $a$, and not the $b$, will be executed at 2.*

**Example 8.6** *In the untimed setting,* $\tau a + b \not\leftrightarrow_b a + \tau b$. *However,*

$$\tau[1] \cdot a[1] + b[2] \leftrightarrow_b a[2] + \tau[1] \cdot b[1].$$

*In both processes it is decided at time 1 whether the $a$ or the $b$ will be executed at 2.*

In the definition of timed branching bisimulation, we need an auxiliary definition. Suppose that two processes $p$ and $q$ are branching bisimilar, and that $p$ can execute an $a$-action. Unlike strong bisimulation, it may not be the case that $q$ can execute the same initial $a$-action. Possibly, $q$ will first execute a number of $\tau$-actions, which can all be matched with idling in $p$. Finally, the resulting state $q'$ can execute the $a$-action. We say that $q \Rightarrow q'$ is *equivalent with $p$.*

**Definition 8.7** *Let $\mathcal{B}$ be a binary relation on processes. For $p$ a process and $r \geq 0$, we inductively define the relation '$q \Rightarrow_r q'$ $\mathcal{B}$-*equivalent with *$p$'.*

1. *If $U(q) \geq r$, and $(-t)p\mathcal{B}(-t)q$ for $0 \leq t \leq r$, then $q \Rightarrow_r (-r)q$ $\mathcal{B}$-equivalent with $p$.*

2. *If $q \xrightarrow{\tau[s]} q'$, and $(-t)p\mathcal{B}(-t)q$ for $0 \leq t < s$, and $q' \Rightarrow_{r-s} q''$ $\mathcal{B}$-equivalent with $(-s)p$, then $q \Rightarrow_r q''$ $\mathcal{B}$-equivalent with $p$.*

**Definition 8.8** *Two process terms $p_0$ and $q_0$ are* branching bisimilar*, denoted by $p_0 \leftrightarrow_b q_0$, if there exists a symmetric binary relation $\mathcal{B}$ on processes such that*

1. *$p_0 \mathcal{B} q_0$.*

2. *If* $p\mathcal{B}q$ *and* $p \xrightarrow{a[r]} p'$, *then* $q \Rightarrow_s q'$ $\mathcal{B}$*-equivalent with* $p$ *for some* $s < r$, *such that*
   - *either* $q' \xrightarrow{a[r-s]} q''$ *with* $p'\mathcal{B}q''$,
   - *or* $a = \tau$ *and* $U(p') > 0$ *and* $p'\mathcal{B}(s-r)q'$.
3. *If* $p\mathcal{B}q$ *and* $p \xrightarrow{a[r]} \surd$, *then* $q \Rightarrow_s q'$ $\mathcal{B}$*-equivalent with* $p$ *such that* $q' \xrightarrow{a[r-s]} \surd$.
4. *If* $p\mathcal{B}q$ *and* $U(p) > r$, *then* $q \Rightarrow_r q'$ $\mathcal{B}$*-equivalent with* $p$.

Branching bisimulation is not a congruence. For example, $a[2] \leftrightarrows_b \tau[1]\cdot a[1]$, but $a[2] + b[2] \not\leftrightarrows_b \tau[1]\cdot a[1] + b[2]$. We need a rootedness condition.

**Definition 8.9** *Two process terms* $p$ *and* $q$ *are* rooted branching bisimilar*, denoted by* $p \leftrightarrows_{rb} q$, *if*

1. $p \xrightarrow{a[r]} p'$ *if and only if* $q \xrightarrow{a[r]} q'$ *with* $p' \leftrightarrows_b q'$.
2. $p \xrightarrow{a[r]} \surd$ *if and only if* $q \xrightarrow{a[r]} \surd$.

Rooted branching bisimulation is a congruence on $\mathcal{R}_\tau$.

### 8.3.3 One axiom for abstraction

Using the intuition for branching bisimulation, we can express rooted branching bisimulation equivalence in one axiom TT, from [14]. Surprisingly, we obtain a complete axiomatization for $\mathcal{R}_\tau$ by adding only this axiom to the axiom system.

| TT $\quad U(x) \leq r \ \wedge\ U(y) > 0 \implies a[s]\cdot(x + \tau[r]\cdot y) = a[s]\cdot(x + (r)y)$ |
|---|

### 8.3.4 Completeness

Consider the model $\mathcal{R}_\tau$. As before, we can deduce that the axioms A1-5 and TA6-9 and R1,2 and TT and TS1-4 are sound with respect to rooted branching bisimulation. This section is devoted to proving that these axioms are complete.

We reduce each solution of a linear specification to a normal form and show that if two normal forms are bisimilar, then they are syntactically equivalent. Let $E = \{X_i = t_i |\ i = 1, ..., n\}$ be a linear specification. We reduce $\langle X_1|E\rangle$ to normal form in several steps.

### Step 1: Removal of redundant deadlocks

Replace expressions of the form $\alpha[r]$ with $r \leq 0$ in $t_i$ by $\delta[0]$ and expressions of the form $a[\infty]$ by $\delta[\infty]$. Next, replace expressions of the form $\delta[r]\cdot X$ by $\delta[r]$. Finally, remove expressions $\delta[r]$ from $t_i$ for which there is an expression $a[s]\cdot X$ or $\alpha[s]$ in $t_i$ with $r \leq s$.

### Step 2: Root unwinding

Add an equation $X_{\rm root} = t_1$ to $E$, where $X_{\rm root}$ does not yet occur in $E$.

### Step 3: Adding $\tau$-steps

Consider the equation for a variable $X \neq X_{\rm root}$ in $E$:

$$X = \sum_j a_j[r_j] \cdot X_j + \sum_k \alpha_k[s_k].$$

Let $t_0$ be the smallest time number that occurs in this equation. If there is an $r_j$ or $s_k$ greater than $t_0$, then replace this equation in $E$ by the following two equations:

$$X = \sum_{\{j|r_j=t_0\}} a_j[t_0] \cdot X_j + \sum_{\{k|s_k=t_0\}} \alpha_k[t_0] + \tau[t_0] \cdot Y,$$

$$Y = \sum_{\{j|r_j>t_0\}} a_j[r_j - t_0] \cdot X_j + \sum_{\{k|s_k>t_0\}} \alpha_k[s_k - t_0],$$

where $Y$ is a variable that does not yet occur in $E$. Repeat this procedure until the equations in $E$ for variables unequal to $X_{\rm root}$ have all become of the form

$$X = \sum_j a_j[r] \cdot X_j + \sum_k \alpha_k[r].$$

### Step 4: Identification of bisimilar variables

If $\langle X|E\rangle \leftrightarrows_b \langle X'|E\rangle$ with $X \neq X'$ and $X, X' \neq X_{\rm root}$, then rename all occurrences of $X'$ at the right-hand side of equations from $E$ into $X$.

### Step 5: Removal of double edges

If an expression $\alpha[r]$ or $a[r] \cdot X'$ occurs more than once at the right-hand side of an equation in $E$, then remove all but one of these occurrences.

### Step 6: Removal of $\tau$-loops

If for a variable $X \neq X_{\rm root}$ its equation in $E$ is of the form $X = \tau[r] \cdot X$, then replace this equation in $E$ by $X = \delta[\infty]$.

### Step 7: Removal of redundant $\tau$-steps

Suppose that there is an equation of the form $X = \tau[r] \cdot X'$ in $E$ with $X \neq X', X_{\rm root}$. Let the equation for $X'$ in $E$ be of the form

$$X' = \sum_j a_k[s] \cdot X_j + \sum_k \alpha_k[s].$$

Then replace the equation for $X$ in $E$ by

$$X = \sum_j a_j[s + r] \cdot X_j + \sum_k \alpha_k[s + r].$$

## Step 8: Removal of redundant variables

If a variable $X$ in $E$ is not in $dep(X_{\rm root})$, then remove its equation from $E$.

Thus we have constructed the normal form $\langle X_{\rm root}|\tilde{E}\rangle$ of $\langle X_1|E\rangle$. Step 1 can be proved by R1,2+TA6-9, Steps 2 and 8 by R1,2, Steps 3, 6 and 7 by R1,2+TT+TS1-4, and Step 5 by R1,2+A3. We show that Step 4 is provable. Let $\tilde{E}$ be the specification that results after identifying bisimilar variables in $E$.

**Proposition 8.10** $\quad {\rm R1,2+A3+TT+TS1-4} \vdash \langle X_{\rm root}|E\rangle = \langle X_{\rm root}|\tilde{E}\rangle$.

**Proof.** Let $Q$ be the collection of positive rationals that occur as a time stamp in $E$. Since $Q$ is finite, and since it contains only rational numbers, there is a greatest rational $t_0$ such that $t/t_0$ is a natural number for all $t \in Q$.

Let $\tilde{E}$ be the specification that results after identifying bisimilar variables in $E$. We reduce the specifications $E$ and $\tilde{E}$ as follows. Consider an equation

$$X = \sum_j a_j[r] \cdot X_j + \sum_k \alpha_k[r],$$

with $X \neq X_{\rm root}$. If $r = \infty$, then replace this equation by

$$X = \tau[t_0] \cdot X.$$

If $t_0 < r < \infty$, then replace it by the following two equations:

$$X = \tau[t_0] \cdot Y,$$

$$Y = \sum_j a_j[r - t_0] \cdot X_j + \sum_k \alpha_k[r - t_0].$$

where the variable $Y$ does not yet occur in $E$ nor in $\tilde{E}$. Repeat this procedure until all equations in $E$ and $\tilde{E}$ for variables unequal to $X_{\rm root}$ have become of the form

$$X = \sum_j a_j[t_0] \cdot X_j + \sum_k \alpha_k[t_0] \qquad \text{or} \qquad X = \sum_j \delta[0].$$

The resulting specifications are denoted by $E^*$ and $\tilde{E}^*$. It is easy to see that the axioms TT+TS1-4 induce $\langle X_{\rm root}|E\rangle = \langle X_{\rm root}|E^*\rangle$ and $\langle X_{\rm root}|\tilde{E}\rangle = \langle X_{\rm root}|\tilde{E}^*\rangle$.

Since $\langle X_{\rm root}|E\rangle \leftrightarrows_{rb} \langle X_{\rm root}|\tilde{E}\rangle$, it follows that $\langle X_{\rm root}|E^*\rangle \leftrightarrows_{rb} \langle X_{\rm root}|\tilde{E}^*\rangle$. The rooted branching bisimulation relation between $\langle X_{\rm root}|E^*\rangle$ and $\langle X_{\rm root}|\tilde{E}^*\rangle$ is a strong bisimulation relation. Namely, the rootedness condition compels that initial transitions $\langle X_{\rm root}|E^*\rangle \xrightarrow{a[r]} p$ are matched with initial transitions $\langle X_{\rm root}|\tilde{E}^*\rangle \xrightarrow{a[r]} q$, and vice versa. Moreover, the construction of $E^*$ and $\tilde{E}^*$ ensures that non-initial transitions in the transitions systems of $\langle X_{\rm root}|E^*\rangle$ and $\langle X_{\rm root}|\tilde{E}^*$ have labels of the form $a[t_0]$, so such transitions $p \xrightarrow{a[t_0]} p'$ in the one transition system are matched with transitions $q \xrightarrow{a[t_0]} q'$ in the other. Hence, $\langle X_{\rm root}|E^*\rangle \leftrightarrows \langle X_{\rm root}|\tilde{E}^*\rangle$. Then the completeness result from the previous section yields $\langle X_{\rm root}|E^*\rangle = \langle X_{\rm root}|\tilde{E}^*\rangle$, so finally

$$\langle X_{\rm root}|E\rangle = \langle X_{\rm root}|E^*\rangle = \langle X_{\rm root}|\tilde{E}^*\rangle = \langle X_{\rm root}|\tilde{E}\rangle. \quad \Box$$

**Theorem 8.11** *If two normal forms* $\langle X_{\mathrm{root}}|E\rangle$ *and* $\langle Y_{\mathrm{root}}|E'\rangle$ *are rooted branching bisimilar, then they are syntactically equivalent modulo* $\alpha$*-conversion.*

**Proof.** Similar to the proof of Theorem 8.3.

**Corollary 8.12** A1-5+TA6-9+TT+TS1-4+R1,2 *is a complete axiomatization for* $\mathcal{R}_\tau$ *modulo rooted branching bisimulation.*

# References

[1] L. Aceto and A.S.A. Jeffrey. A complete axiomatization of timed bisimulation for a class of timed regular behaviours. Report 4/94, University of Sussex, 1994.

[2] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[3] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, *Proceedings 4th Conference on Concurrency Theory (CONCUR'93),* Hildesheim, *LNCS 715*, pages 477–492. Springer-Verlag, 1993.

[4] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings 11th International Colloquium on Automata, Languages and Programming (ICALP'84),* Antwerp, *LNCS 172*, pages 82–95. Springer-Verlag, 1984.

[5] J.A. Bergstra and J.W. Klop. A complete inference system for regular processes with silent moves. In F.R. Drake and J.K. Truss, editors, *Proceedings Logic Colloquium 1986,* Hull, pages 21–81. North-Holland, 1988.

[6] L. Chen. A model for real-time process algebras. In A.M. Borzyszkowski and S. Sokołowski, editors, *Proceedings 18th Symposium on Mathematical Foundations of Computer Science (MFCS'93),* Gdansk, *LNCS 711*, pages 372–381. Springer-Verlag, 1993.

[7] L. Chen. Axiomatising real-timed processes. In S. Brooks, M. Main, A. Melton, M. Mislove and D. Schmidt, editors, *Proceedings 9th Conference on Mathematical Foundations of Programming Semantics (MFPS'93),* New Orleans, *LNCS 802*, pages 215–229. Springer-Verlag, 1993.

[8] W.J. Fokkink. Regular processes with rational time and silent steps. Report CS-R9231, CWI, Amsterdam, 1992.

[9] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. In G.X. Ritter, editor, *Information Processing 89, Proceedings of the 11th IFIP World Computer Congress,* San Francisco, pages 613–618. North-Holland, 1989. Under revision for *Journal of the ACM.*

[10] C. Ho-Stuart, H.S.M. Zedan, and M. Fang. Congruent weak bisimulation with dense real-time. *Information Processing Letters*, 46:55–61, 1993.

[11] U. Holmer, K.G. Larsen, and Y. Wang. Deciding properties of regular real timed processes. In K.G. Larsen and A. Skou, editors, *Proceedings 3rd Workshop on Computer Aided Verification (CAV'91),* Aalborg, *LNCS 575*, pages 432–442. Springer-Verlag, 1991.

[12] A.S. Klusener. Abstraction in real time process algebra. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX workshop "Real Time: Theory in Practice",* Mook, *LNCS 600*, pages 325–352. Springer-Verlag, 1991.

[13] A.S. Klusener. Completeness in real time process algebra. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings 2nd Conference on Concurrency Theory (CONCUR'91),* Amsterdam, *LNCS 527*, pages 376–392. Springer-Verlag, 1991.

[14] A.S. Klusener. The silent step in time. In W.R. Cleaveland, editor, *Proceedings 3rd Conference on Concurrency Theory (CONCUR'92),* Stony Brook, *LNCS 630*, pages 421–435. Springer-Verlag, 1992.

[15] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[16] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. Report ECS-LFCS-86-8, University of Edinburgh, 1986.

[17] F. Moller and C. Tofts. Behavioural abstraction in TCCS. In W. Kuich, editor, *Proceedings 19th International Colloquium on Automata, Languages and Programming (ICALP'92),* Vienna, *LNCS 623*, pages 559–570. Springer-Verlag, 1992.

[18] J. Quemada, D. de Frutos, and A. Azcorra. TIC: A TImed Calculus. *Formal Aspects of Computing*, 5(3):224–252, 1993.

[19] M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[20] Y. Wang. *A Calculus of Real Time Systems.* PhD thesis, Chalmers University of Technology, Göteborg, 1991.

# Samenvatting

Dit proefschrift omvat zeven artikelen op het gebied van de procesalgebra.

In de eerste twee hoofdstukken wordt de compleetheid bewezen van axiomatizeringen voor twee verschijningsvormen van iteratie. In Hoofdstuk 2 wordt aangetoond dat basic CCS uitgebreid met prefix iteratie $a^*x$ compleet geaxiomatizeerd is door de vier standaardaxioma's voor basic CCS tezamen met twee extra axioma's:

$$\begin{aligned} a \cdot a^*x + x &= a^*x \\ a^*(a^*x) &= a^*x \end{aligned}$$

Ruim een jaar geleden stelden Bergstra, Bethke en Ponse de vraag of BPA uitgebreid met binaire iteratie $x^*y$ compleet geaxiomatizeerd is door de vijf standaardaxioma's voor BPA tezamen met drie extra axioma's:

$$\begin{aligned} x \cdot x^*y + y &= x^*y \\ x^*y \cdot z &= x^*(yz) \\ x^*(y \cdot (x+y)^*z + z) &= (x+y)^*z \end{aligned}$$

In Hoofdstuk 3 wordt deze vraag bevestigend beantwoord.

Groote and Vaandrager definieerden het tyft/tyxt formaat voor transitieregels, en zij toonden aan dat transitiesystemen voortgebracht door 'well-founded' tyft/tyxt regels altijd een congruentie opleveren voor sterke bisimulatie. In Hoofdstuk 4 wordt aangetoond dat de restrictie van well-foundedness overbodig is voor dit congruentieresultaat. Namelijk, het blijkt dat er voor iedere collectie transitieregels in tyft/tyxt formaat een equivalente collectie transitieregels in het restrictievere tree-formaat is. Tree-regels zijn well-founded, dus de congruentiestelling van Groote en Vaandrager is van toepassing op dit formaat.

Een bekende stelling uit de unificatietheorie zegt dat iedere eindige, unificeerbare collectie vergelijkingen een idempotente, meest algemene unificator heeft. In Hoofdstuk 5 wordt aangetoond dat deze stelling ook opgaat voor oneindige collecties vergelijkingen.

Baeten and Bergstra hebben een uitbreiding van ACP gedefinieerd met reële tijd en integratie, waardoor het mogelijk is het gedrag van een proces af te laten hangen van het tijdstip waarop een eerdere actie is uitgevoerd. De laatste drie hoofdstukken nemen deze algebra onder de loep.

In Hoofdstuk 6 wordt aangetoond dat er een deelalgebra van de reguliere processen in ACP met reële tijd en recursie bestaat waarvoor de merge geëlimineerd

kan worden. Deze deelalgebra is gelijk aan de klasse van automaten met tijd van Alur and Dill.

In Hoofdstuk 7 wordt bewezen dat sterke bisimulatie beslisbaar is voor ACP met reële tijd en prefix integratie. Dit resultaat is gebaseerd op de axiomatizering voor conditionele termen van Klusener.

Hoofdstuk 8 presenteert een complete axiomatizering voor $\mathrm{BPA}_{\delta\tau}$ met recursie en tijd, maar zonder integratie, modulo branching bisimulatie.