doeko
jan
barend
bosscher

grammars modulo bisimulation

Grammars Modulo Bisimulation

Grammars Modulo Bisimulation

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. J.J.M. Franse
ten overstaan van een door het college van dekanen
ingestelde commissie in het openbaar te verdedigen
in de Aula der Universiteit
op donderdag 23 oktober 1997 te 13.00 uur

door

Doeko Jan Barend Bosscher
geboren te Leiden

Promotor:	prof. dr. J.A. Bergstra
Co-Promotor:	dr. A. Ponse
Faculteit:	Wiskunde, Informatica, Natuur- en Sterrenkunde

The work in this thesis has been carried out at CWI, Amsterdam in the context of the ESPRIT BRA project no. 7166 "Concur2".

für Marilise

Dankwoord

Het proefschrift dat voor u ligt heb ik gemaakt, maar had echter niet tot stand kunnen komen zonder een aantal bijzondere mensen:

In de eerste plaats mijn grootmoeder en grootvader Dickie Conijn en Doeko Bosscher en moeder en vader Cathrien Boon en Flip Bosscher. Ik ben hen dankbaar voor de liefde en de liberale levenshouding die zij hebben doorgegeven.

Ik dank Steffen van Bakel ervoor dat hij mij in contact heeft gebracht met het Centrum voor Wiskunde en Informatica (CWI) in de periode van mijn afstuderen. Het contact met Jan Rutten en zijn uitstekende afstudeerbegeleiding heeft de kiem gelegd voor mijn promotie aan het CWI.

Ik dank Frits Vaandrager, mijn eerste begeleider op het CWI, voor de absolute vrijheid die hij mij gaf om onderzoek te doen. Ik ken weinig mensen met zo'n precisie en zo'n fijne neus voor interessante oplosbare problemen.

Ik dank Jan Bergstra voor zijn invulling van het promotorschap. Zijn persoonlijke aanpak en de combinatie van humane en paradoxale logica hebben de voltooiing van dit proefschrift mogelijk gemaakt en mij als mens verrijkt.

Ik dank Alban Ponse voor de latere dagelijkse begeleiding. Albans waardering voor eenvoud in procestheorie ("alleen eenvoudige noties overleven") heeft mij succesvol geïnfecteerd. Van de discussies over expressiviteit van de ster van Kleene die hebben geresulteerd in hoofdstuk 3 heb ik genoten. Ik dank hem voor de buitengewoon plezierige manier waarop hij co-promotor bij mijn proefschrift is geweest.

Ik dank Krzysztof Apt, Peter van Emde Boas en Frits Vaandrager uit de leescommissie voor hun inspanningen voor mijn promotie.

Je veux remercier Didier Caucal pour ses observations très utiles sur le texte de cette thèse et pour sa bonne volonté de participer aux déliberation de la "commission de lecture".

Ik dank Jaco de Bakker die als hoofd van de oude afdeling Programmatuur (AP) voor anderen en mij een vrijzinnig onderzoeksklimaat heeft geschapen. Ik dank mijn collega's van AP en de afdeling van Programmatuur van de Universiteit van Amsterdam voor de vriendschap die ik heb ondervonden tijdens mijn promotie: Met mijn collega's van het oude AP2 kun je stuk voor stuk paarden stelen.

Doeko Bosscher

Amsterdam, September 1997

* Op de voorkant staat het schilderij de Toren van Babel van Breughel dat hangt in het museum Boymans van Beuningen in Rotterdam. Het grafisch ontwerp is van Plano/Stijn van Diemen. De boekenlegger toont een andere afbeeling van Gustave Doré.

Beide afbeeldingen refereren naar Genesis 11:1-9 in de Bijbel waarin wordt verklaard dat mensen *verschillende* talen spreken als straf voor de hoogmoed om een stad te bouwen die tot in de hemel reikt. Dit proefschrift handelt over een vergelijking en interpretatie van talen als processen.

Contents

1. Preface 7

Chapter 1. Decidability of Bisimulation Equivalence of Context-free Processes definable with δ 15
1. Introduction 15
2. An Operational Semantics of BPA_δ 16
3. Restricted Greibach Normal Form extended with δ 18
4. Using Mimicking for Deciding Bisimilarity 20
5. Decidability of Bisimulation Equivalence of Context-Free Processes definable with ϵ ? 24

Chapter 2. Regularity of Context-Free Processes definable with δ 31
1. Introduction 31
2. Regularity and NRD specifications 34
3. Regularity Implies an Upper Bound Modulo Bisimulation 39
4. Using Mimicking for Deciding Regularity 47

Chapter 3. Regular Process Graphs and Regular Expressions 49
1. Introduction 49
2. Regular Expressions, Process Graphs and Operations on Process Graphs 53
3. A Criterion for Process Graphs expressible with Regular Expressions definable *without* ϵ: the $\star$ property 64
4. A Criterion for Process Graphs expressible with Regular Expressions definable *with* ϵ: the $\star_\epsilon$ property 89
5. Decidability of Expressibility and the $\star$ properties 107
6. Identifying Expressible Process Graphs 119

Chapter 4. Term Rewriting Properties of SOS Axiomatizations 125
1. Introduction 125
2. The Axiomatization of a GSOS system as a TRS 126
3. Confluency and Weak Normalization 139

4. A Strongly Normalizing Subclass 145

Bibliography 153

1. Preface

In this thesis I present a collection is of four theoretical papers, written at CWI in Amsterdam in the last four and a half years. Each chapter corresponds to a separate research project and can be read on its own.

I am the author of all chapters, but there is a more grandiloquent binding factor. In the title I have tried to put this concisely. I am mainly concerned with bisimulation equivalence of grammars of formal languages expressing process behavior. This last sentence I will explain in detail.

Grammars

Traditionally expressivity of grammars is the subject of a formal language theory. It is of great importance to computer science, since there is an intimate relationship between different kinds of grammars and the abstract machines that can accept or produce the languages defined by these grammars. In turn these abstract machines are models for different forms of computation.

The word "language" in formal language theory must be understood in a mathematical way. A language is a collection of sentences or a lexicon built up of a given alphabet. This definition means that Dutch sentences are a language, but also the strings over the alphabet consisting only of the symbol "1". Languages of interest are mostly infinite collections that can be represented by a finite specification or grammar. Languages can be ordered with respect to the specific format of the grammar allowed. Thus in the well-known hierarchy of formal languages by Chomsky regular, context-free, context-sensitive and recursively enumerable languages are discerned [Cho59]. In this thesis I am concerned with the "simplest" languages in the hierarchy, i.e. regular and context-free languages.

Text books on formal languages usually start with the study of the regular expressions as formulated by Kleene [Kle56]. Regular expressions are very simple grammars which define the forming of a language in a very restricted way: by a union of two languages, concatenation of words belonging to two languages and the Kleene star, which forms

a new language by a finite concatenation of words from the old language. This last construct gives the power to describe an infinite language in a finite way. The regular expression a^*b defines the language of words ending with a b and starting with an arbitrary number of a's, i.e. $\{b, ab, aab, aaab, ...\}$. Regular languages are called *regular*, since they exhibit a very regular structure.

Regular languages look very simple, but are more interesting if one realizes that the regular languages are exactly the languages which can be accepted by finite state automata. Finite state automata are an elementary form of computing device, which use no auxiliary memory. A program which controls a vending machine can typically be modeled by a finite state machine. For every regular expression a finite state machine exists which accepts the language generated by it and vice versa [Kle56, MY60, Wat95]. With finite state machines it is possible to model non-deterministic computation, although surprisingly it was proved that for every non-deterministic automaton, a deterministic automaton exists which accepts the same language [RS59]. This implies that a deterministic parser can be constructed for a regular language.

The second class of languages traditionally studied are the context-free languages. These are the languages produced by context-free grammars. The context-free grammar $\{S \rightarrow ab, S \rightarrow aSb\}$ defines the language of words of a's followed by an equal number of b's. The context-free languages are precisely the languages which can be accepted by a much more powerful computing device than a finite state automaton, called a push down automaton [Sch63]. Push down automata have a special form of memory, called a stack, with which they can perform a limited form of counting.

It is a standard result that the class of regular languages is smaller than the class of context-free languages. Every language defined by a regular expression is equal to a context-free language which has a so-called right or left-linear context-free grammar [HU79]. However the converse does not hold. E.g. the language of a's followed by an equal number of b's is not regular, i.e. cannot be defined by a regular expression. This result is best understood if one thinks about regular languages as being accepted by a machine which cannot count, in particular cannot keep track of the number of a's and b's.

Context-free languages are sufficiently rich to express languages of "practical" interest, such as programming languages. They are also

inherently more difficult than regular languages. Parsers for context-free languages are much more complex programs than those for regular languages. Therefore a useful question is whether a given context-free language is regular. Regularity of context-free languages is however undecidable, which means that there is no algorithm to verify whether a context-free language can be defined by a regular expression [BH64]. The so-called pumping theorems can be used to establish that the above context-free language of a's followed by an equal number of b's is not regular, but this requires human intuition [BHPS61]. It is also undecidable whether two context-free grammars define the same context-free language [BHPS61].

... Modulo Bisimulation

An alternative interpretation of regular expressions and context-free grammars is provided by process theory. The regular expression a^*b can be said to express a process behavior where at any time a choice can be made between an a action and a b action, but ends when the b is performed. Similarly the context-free grammar $\{S \to ab, S \to aSb\}$ can be said to define a regular process behavior where a number of a actions can be performed, but at any moment can be chosen to perform b actions and is stopped when as many a's as b's have been performed.

In this thesis I study regular expressions and context-free grammars defining process behavior. The only difference between process expressions as defining process behaviors and grammars defining languages lies in the interpretation of the syntax. Process expressions define processes, whereas grammars define languages. The difference between grammars and their interpretations as process expressions vanishes if the set of traces of a process expression is examined. The strings formed by a concatenation of actions of some action set, then correspond immediately to the strings which are formed in accordance to a grammar over an alphabet. The equivalence which identifies process expressions which have the same set of traces is suitably called *language* or *trace equivalence.*

In process algebra a familiar way to represent a process is by means of a process graph, in which the nodes denote the states the process

expressions have evolved to and transitions labeled with an action between states denote that a state can evolve to another state while performing an action. An operational semantics describes how the graphs are constructed from process expressions. For the regular and context-free grammars (and others) this is done mostly by means of transition rules in the so-called SOS format [Plo81].

The equivalence of process expressions is generally also given in terms of their process graphs. In contrast to the situation in language theory, in process algebra language equivalence is one of many equivalences studied intensively to identify process expressions. There are many applications for which trace equivalence is considered to be too coarse a notion. This becomes manifest when the context-free processes are extended with the extra constant δ from ACP [BK84b] which has no equivalent in formal language theory[1], and defines a deadlocking process, i.e. a process that never terminates, but also cannot do anything. Significantly, trace equivalence does not distinguish between a terminating and a deadlocking process.

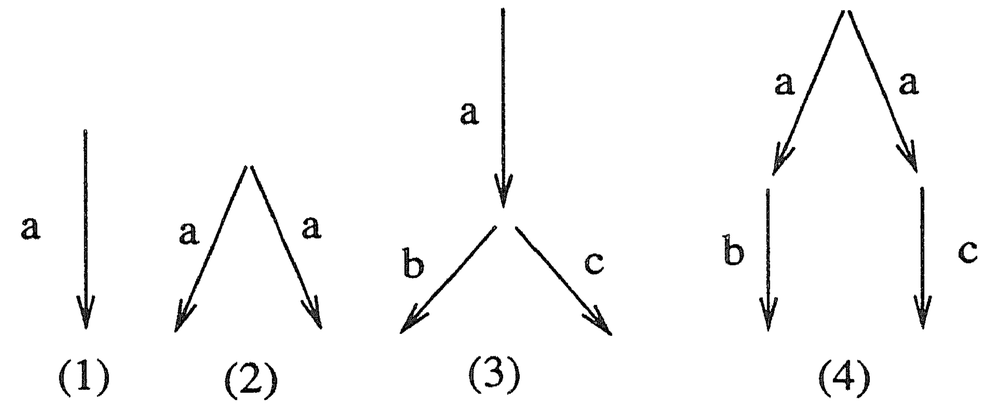
A comprehensive list of equivalences known as the linear time–branching time spectrum is given in [Gla90, Gla93]. With linear time only the traces are taken into account, whereas branching time takes the branching structure of a process graph into account.

A frequently used equivalence on processes is strong bisimulation [Par81]. This equivalence essentially distinguishes processes with a different branching structure. The process expressions a and $a + a$ are language equivalent and bisimilar, since their set of traces ($\{a\}$) and branching structure is the same. The process expressions $a \cdot (b + c)$ and $a \cdot b + a \cdot c$ are language equivalent but not bisimilar. They are language equivalent, since the sets of traces $\{a, ab, ac\}$ are the same. However they do have a different branching structure: the first process can still choose to do a b or c after the a is performed, whereas in the second process the choice for a consequent b or c is fixed the moment a is performed.

EXAMPLE 1.1. In the picture below the process depicted by (1) and (2) (defined by a and $a + a$) are *bisimilar*, whereas the processes (3)

[1]The constant ϵ is usually regarded as the natural interpretation of the empty word e or λ of formal language theory.

and (4) (defined by $a \cdot (b + c)$ and $a \cdot b + a \cdot c$) are not.



There is however a strong relation between bisimulation and trace equivalence. In a deterministic setting, where at every point of choice only distinctive actions are possible, the linear time–branching time spectrum collapses, i.e. all forms of equivalences between strong bisimulation equivalence and language equivalence are the same [Par81, Eng85].

In this thesis the focus is on grammars for process expressions in the context of strong bisimulation. This choice is somewhat arbitrary, but can be motivated by the fact that bisimulation is a popular equivalence with some nice mathematical properties. It is a congruence for the most frequently used operators in process algebra, notably the operators concatenation and union of languages, which are referred to in a process algebraic setting as sequential composition and non-deterministic choice. Furthermore bisimulation equivalence is a decidable equivalence for regular expressions and context-free grammars, whereas that is not the case for the other equivalences in the linear–branching time spectrum [GH94]. This motivates my interest for grammars modulo bisimulation.

Overview of this thesis

It appears that in the context of bisimulation not only is more distinguished, but also more becomes decidable. One of the remarkable achievements in process theory in recent years is the proof that bisimulation equivalence is decidable for context-free processes [BBK87b, CHS92]. It basically shows that there is a finite characterization of a possible infinite bisimulation. This is not at all obvious, since language equivalence of context-free languages is undecidable.

The question of decidability of bisimulation equivalence can also be asked for context-free processes that can be defined with the extra

constants δ and ϵ. The constant ϵ denotes the process which terminates immediately. In Chapter 1 I prove that a recursive map exists which maps context-free processes definable with δ to "ordinary" context-free processes. This is done in such a way that the mapped processes are bisimilar if and only if the original processes are bisimilar. Basically I show that there is no difference between dead lock and live lock in the world of the context-free processes. I give a tentative explanation why the same technique does not suffice to prove the same result for context-free processes definable with ϵ.

Another important result is that regularity of context-free processes is decidable [BCS96]. The decidability of regularity of context-free processes modulo bisimulation is closely related to the decidability of bisimulation equivalence for context-free processes. This is usually understood as the question as to which of the processes defined by a context-free grammar have a regular process graph, i.e. are bisimilar to a process graph which has finitely many states or which allow only finite traces of non-bisimilar states. This is essentially the same question as to which context-free processes can be defined with a finite state machine [2]. In turn finite state machines are associated in a natural way with process graphs with finitely many states. In Chapter 2 I show that for an interesting class of context-free processes definable with δ not only can regularity be decided, but also an upper bound can be given on the size of the state space modulo bisimulation. Furthermore I show that the results of Chapter 1 and [BCS96] can be combined to prove the decidability of regularity of all context-free processes definable with δ.

Chapter 3 is the largest chapter of this thesis. Here I deal at length with the relation between regular expressions and their machine model in language theory, the finite state machines. In contrast to the case for context-free processes it is common knowledge that it is decidable whether two regular expressions or two finite automata define the same process modulo bisimulation [Mil84]. However it is not the case that the processes which are defined by regular expressions, coincide with the processes defined by finite state automata. It was shown in [Mil84] that there are very simple finite automata for which no regular expression exists with the same process interpretation. I consider the question as to whether an effective criterion exists that describes finite

[2] and not the question which context-free processes can be expressed modulo bisimulation with a regular expression !

automata which can be expressed as a process by a regular expression. I will show that indeed such a criterion exists and present algorithms to compute the regular expression which has the same process interpretation modulo bisimulation as a finite automaton, notably for the regular expressions which can be defined with and without the constant ϵ.

In Chapter 4 I treat a subject about the relation between the algebraic and operational approach to define equality on process expressions. As stated earlier, many more equivalences have been studied in process theory than in language theory. Process algebras such as CCS [Mil86] start with an operational semantics and an equivalence, and then prove soundness with respect to a set of axioms. Another style is to specify the equality of processes by means of axioms and an equational logic, and then search for sound models. This approach is in some sense more algebraic and is chosen in ACP [BBK87a]. The relation between the operational and axiomatic approach is a well-researched subject. A question I have elaborated on in Chapter 4 is whether the axiomatizations for bisimulation equivalence in [ABV94] for a given operational semantics have nice term rewriting properties. I show that a well-defined class of oriented axiomatizations are (strongly) normalizing. It is nice if axiomatizations are normalizing, because this means that in principle equality proofs can be carried out mechanically. In this particular case it means that there is another way to check bisimulation equivalence, besides computing the bisimulation.

1.1. Origins of the Chapters. Chapter 1 has not yet been published elsewhere. The subject arose from the idea that the results in [CHS92] could be "re-used" rather than that a whole new proof should be given. The proofs in [CHS92] are very elegant, but also rather difficult.

Chapter 2 is the result of a cooperation with David Griffioen at CWI. The results were presented earlier at the ICALP '96 conference in Paderborn, Germany [BG96]. The paper evolved from a careful analysis of the example in [MM94] of a regular process whose BPA system is not regular[3].

Chapter 3 is as yet unpublished elsewhere. The subject arose from a discussion with Alban Ponse about the expressiveness of the Kleene

[3]I.e. the context-free process defined by X over the specification (3) in Example 1.1 of Chapter 2.

star. The chapter considers the second open question in [Mil84] *"What structural property of finite charts is necessary and sufficient for star behavior ?"*

Chapter 4 is the first paper I wrote at CWI. The idea came from my supervisor at that time at the CWI, Frits Vaandrager, who had just finished [ABV92] (subsequently published as [ABV94]) in which the problem was posed. The chapter was presented at the conference TACS '94 in Sendai, Japan [Bos94].

Conventions used in this thesis.

I have made a deliberate sparse use of special fonts to give a quiet picture for reading. I use *emphasis* only when I introduce a new concept or want to stress the importance of (a part of) a sentence. Sometimes I make use of a [...] construct in conditions, which means that the part in between brackets belongs together, e.g. ... [whenever $s \downarrow$ and $s \xrightarrow{a} s$, $r \xrightarrow{a} s'$]... . For the rest everything is pretty standard, I think.

CHAPTER 1

Decidability of Bisimulation Equivalence of Context-free Processes definable with δ

In [CHS92] it is proved that (strong) bisimulation equivalence is decidable for all context-free processes. We show that there exists a recursive map from context-free processes definable with δ of ACP [BK84b] to context-free processes which preserves bisimulation equivalence. We conjecture that no "similar" map exists with which we can reduce context-free processes definable with the constant ϵ to context-free processes.

1. Introduction

A classical result from formal language theory is that language equivalence is undecidable for context-free languages. This is proved to be a consequence of the well-known Correspondence Lemma of Post [Pos46] in [BH64]. The picture changes if these languages are studied as processes. In [BBK87b, BBK93] it is proved that bisimulation equivalence is decidable for *normed* context-free processes, also known as normed processes over the signature of Basic Process Algebra (BPA). In that remarkable paper it is shown for the first time that context-free processes express some finite periodicities which allow an effective comparison of processes. Simpler proofs have been given in [Cau90, HS91, Gro92]. A polynomial time algorithm for deciding bisimulation for normed context-free processes is presented in [HJM94].

The previous result is extended in [CHS92] by showing decidability of bisimulation equivalence for *all* context-free processes, also known as processes over the signature of Basic Process Algebra (BPA). The proof is very elegant, exploiting a deep insight in the structure of such processes. An exponential time algorithm for deciding bisimulation equivalence has been given in [BCS95].

As far as we know the question of decidability of bisimulation equivalence for BPA_δ, i.e. those context-free processes which can be defined

with the extra constant δ from ACP [BK84a], has not been studied. In this chapter we extend the decidability of bisimulation equivalence of context-free processes to that of context-free processes definable with δ, also known as the processes definable over the signature of BPA_δ and recursive process names. We give the name δ *mimicking* for the interpretation of context-free processes definable with δ as context-free processes while preserving bisimulation equivalence. We do this by interpreting δ as a special context-free process and by proving that bisimulation equivalence of transformed BPA_δ processes is (logically) equivalent to that of the original processes. Furthermore we investigate whether the same technique can be used to prove the decidability of bisimulation equivalence over BPA_ϵ, i.e. those context-free processes which can be defined with the constant ϵ, denoting the empty or *terminated* process.

2. An Operational Semantics of BPA_δ

As usual we refer to context-free processes as the processes which are defined by process expressions over specifications over the signature of Basic Process Algebra (BPA), actions and process variables. In this section we define some notions of BPA and give syntax and semantics. In the Introduction of this thesis we motivated our interest for context-free processes as an alternative interpretation of context-free grammars. For a detailed description of the relation between language and process theory we refer the interested reader to [HM96].

Let Act be a countable set of *actions* with typical elements a, b, c with the usual annotations. Let Var be a countable set of process variables disjoint from van Act with typical elements X, Y, Z. The process expressions $\mathbf{P}^{rec}_{\delta,\mathrm{Var}}$ over the signature of $\mathrm{BPA}(Act)$ and Var are given by the abstract syntax

$$p ::= a \mid \delta \mid p + p \mid p \cdot p \mid X$$

where $a \in Act$ and $X \in \mathrm{Var}$. We use that $\cdot$ binds stronger than $+$, e.g. $a{\cdot}b{+}c$ is the same as $(a{\cdot}b){+}c$. In this chapter and Chapter 2 we will also use $\mathbf{P}^{rec}_{\mathrm{Var}}$ which is a subset of $\mathbf{P}^{rec}_{\delta,\mathrm{Var}}$ of process expressions *without* δ. $p_1 + p_2$ is the usual process algebraic notation for the *choice* between p_1 and p_2 and $p_1 \cdot p_2$ for *sequential composition*. A *specification* Δ over the

signature of BPA$_\delta$ and Var is a set of tuples $\{(X, t_X(V_\Delta))|X \in V_\Delta\}$[1], where V_Δ is a *finite* subset of Var of *variables of* Δ and $t_X(V_\Delta) \in \mathbf{P}^{rec}_{\delta,\mathrm{Var}}$ a process expression in which only process variables of V_Δ occur. For convenience we write $\{X = p, ...\}$ [2] instead of $\{(X, p), ...\}$. We call two specifications Δ and Δ' *disjoint* if $V_\Delta \cap V_{\Delta'} = \emptyset$. Furthermore we use $A_\Delta \subseteq Act$ for the set of actions occurring in Δ. The set of process expressions over the signature of BPA(Act) and the variables of Δ is denoted by $\mathbf{P}^{rec}_{\delta,\Delta}$.

Let Δ be a specification over $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$. The operational semantics of a process expression $p \in \mathbf{P}^{rec}_{\delta,\Delta}$ is given by a binary termination relation $\cdot \dot{\rightarrow}_\Delta \surd \subseteq \mathbf{P}^{rec}_{\delta,\Delta} \times Act$ and a ternary transition relation $\cdot \dot{\rightarrow}_\Delta \cdot \subset \mathbf{P}^{rec}_{\delta,\Delta} \times Act \times \mathbf{P}^{rec}_{\delta,\Delta}$ with the transitions provable by the following rules

$$\frac{}{a \rightarrow \surd}$$

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad \frac{p \xrightarrow{a} p'}{q + p \xrightarrow{a} p'} \qquad \frac{p \xrightarrow{a} \surd}{p + q \xrightarrow{a} \surd} \qquad \frac{p \xrightarrow{a} \surd}{q + p \xrightarrow{a} \surd}$$

$$\frac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q} \qquad \frac{p \xrightarrow{a} \surd}{p \cdot q \xrightarrow{a} q}$$

$$\frac{p \xrightarrow{a} p' \; X = p \in \Delta}{X \xrightarrow{a} p'} \qquad \frac{p \xrightarrow{a} \surd \; X = p \in \Delta}{X \xrightarrow{a} \surd}$$

where $a \in Act$. A *derivative* of a process expression over Δ is an process expression which can be reached with a sequence of contigious transitions. We feel free to write *state* instead for derivative. For the *set of reachable process expressions* from a process expression we also use the term *state space.*

DEFINITION 2.1. Let Δ be a specification over $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$.

1. Let $R \subseteq \mathbf{P}^{rec}_{\delta,\Delta} \times \mathbf{P}^{rec}_{\delta,\Delta}$ be a relation satisfying for every $(p, q) \in R$,
   (a) $p \xrightarrow{a}_\Delta \surd$ iff $q \xrightarrow{a}_\Delta \surd$, and
   (b) if $p \xrightarrow{a}_\Delta p'$ for some $a \in Act$, $p' \in \mathbf{P}^{rec}_{\delta,\Delta}$, then there is a $q' \in \mathbf{P}^{rec}_{\delta,\Delta}$ so that $q \xrightarrow{a}_\Delta q'$ and $(p', q') \in R$, and

[1] Notice that for every X, there is *precisely one* t_X.

[2] Notice that the equality sign is not interpreted as such.

(c) if $q \xrightarrow{a}_{\Delta} q'$ for some $a \in Act$, $q' \in \mathbf{P}^{rec}_{\delta,\Delta}$, then there is a $p' \in \mathbf{P}^{rec}_{\delta,\Delta}$ so that $p \xrightarrow{a}_{\Delta} p'$ and $(p', q') \in R$,

then R *is a bisimulation over* Δ,

2. If there exists a bisimulation R over Δ, such that $(p, q) \in R$, then p and q *are bisimilar over* Δ, shortly $p \leftrightarrows_{\Delta} q$.

DEFINITION 2.2. Let Δ be a $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification.

1. If $p \in \mathbf{P}^{rec}_{\delta,\Delta}$ and every occurrence of variable $X \in V_\Delta$ in p is in a subexpression of the form $a \cdot p'$, where $a \in Act$, then p is *syntactically guarded.*
2. If [whenever $X = p \in \Delta$, p is syntactically guarded], then Δ is *syntactically guarded.*

EXAMPLE 2.3. The process expression $a \cdot X \cdot X + b \cdot X \cdot X$ is syntactically guarded, whereas the process expressions $(a+b) \cdot Y$ and $Y \cdot a + Y \cdot b$ are not, and so $\{X = a \cdot X \cdot X + b \cdot X \cdot X\}$ is a syntactically guarded specification and $\{Y = (a + b) \cdot Y \cdot Y\}$, $\{Y = Y \cdot a + Y \cdot b\}$ are not syntactically guarded.

DEFINITION 2.4. Let Δ be a syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var})$ specification. The set of tuples $\{(X, \Delta) | X \in V_\Delta)\}$ is a *solution* of Δ.

It is folklore that syntactically guarded specifications satisfy the so-called Recursive Definition Principle (RDP), i.e. every specification has *a* solution, and the Recursive Specification Principle (RSP), i.e. every syntactically guarded specification has *at most one* solution [BBK87c, BV95].

We feel free now to speak of the set $\mathbf{P}^{rec}_{\delta,\Delta}$, i.e. of process expressions over the signature of $\mathrm{BPA}_\delta(Act)$ and the (unique) solution of Δ. Abusing notation we write X instead of (X, Δ) and we call X a *process name.* E.g. for the process expression $a \cdot (X, \Delta)$ we write $a \cdot X$, when Δ is known from the context.

3. Restricted Greibach Normal Form extended with δ

In the literature often specifications in an even more restricted format are studied. The restricted *Greibach Normal Form* is a format which allows specifications, which are as expressive as syntactically guarded specifications, as we will prove in this section. We present here an extension of this format for the c tant δ.

DEFINITION 3.1. Let Δ be a $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification.

1. δ, a, $a \cdot X$ and $a \cdot X \cdot Y$ for $a \in Act$, $X, Y \in V_\Delta$ are *in* $rGNF_\delta$.
2. If $p \in \mathbf{P}^{rec}_{\delta,\Delta}$ and $p = \delta$ or [p consists only of non δ summands in rGNF_δ and every summand appears only once], then p is *in* $rGNF_\delta$.
3. If [whenever $X = p \in \Delta$, p is in rGNF_δ], then Δ is *in* $rGNF_\delta$.

EXAMPLE 3.2. The process expression $a \cdot X \cdot X$ is in rGNF_δ, and $a \cdot X \cdot X \cdot X$ and $\delta \cdot X$ are not in rGNF_δ. The specification $\{X = a \cdot X \cdot X\}$ is in rGNF_δ and $\{X = a \cdot X \cdot X \cdot X + \delta \cdot X\}$ is not in rGNF_δ.

REMARK 3.3. Notice that a rGNF_δ specification is always syntactically guarded.

It is sufficient to consider specifications in rGNF_δ, since any syntactically guarded specification can be reduced to one.

LEMMA 3.4. Let Δ be a syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification and $V_\Delta = \{X_1, ..., X_k\}$. Then there is a computable $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification Δ' in rGNF_δ disjoint from Δ with $V_{\Delta'} = \{Y_1, ..., Y_l\}$ and $X_i \leftrightarrows_{\Delta \cup \Delta'} Y_i$ for $i \leq k \leq l$.

Proof. The proof is well-known for context-free processes [BBK87b, BBK93, HM96] and we sketch an extension with δ.[3] First we rewrite each p for $X = p \in \Delta$ to normal form modulo the associativity of $\cdot$ and modulo the commutativity and associativity of the $+$ with the axioms (A3) $x + x = x$, (A4) $(x + y) \cdot z = x \cdot z + y \cdot z$, (A6) $x + \delta = x$ and (A7) $\delta \cdot x = \delta$ oriented from left-to-right (Axiom names from [BW90]). Second we substitute all summands of the form $\alpha \cdot p$, $\alpha \in Act \cup \mathrm{Var}$, $p \notin \mathrm{Var}$ by $\alpha \cdot X'$ and introduce a new equation $X' = p$, where X' is "fresh" process variable. We repeat this procedure until there are only rGNF_δ summands and summands of the form $X \cdot Y$. Notice that the rGNF_δ summands are linear, i.e. of the form δ, a or $a \cdot X$.

Last we substitute all summands of the form $X \cdot Y$ by $q \cdot Y$, where $X = q$ is an equation and rewrite the specification to normal form with (A3,A4,A6,A7).

The whole procedure terminates since rewriting with (A3,A4,A6,A7) is strongly normalizing and every time an equation is added which is

[3] The extension with ϵ used in Section 5 is similar, where the axioms (A8) $x \cdot \epsilon = x$ and (A9) $\epsilon \cdot x = x$ are used, instead of (A6) and (A7).

not in rGNF_δ the (single) summand has a smaller size than the summand it stems from. The final specification is in rGNF_δ, since at the last step a process expression is substituted of the form a or $a \cdot X$. We omit the tedious proof that a subset of the solution of the new specification is bisimilar to the solution of the original specification □

EXAMPLE 3.5. For the specification $\Delta_1 = \{X = a \cdot X \cdot X \cdot \delta + b\}$ according to the proof in Lemma 3.4 can safely be used the rGNF_δ specification $\Delta_2 = \{Y = a \cdot Y' + b, Y' = a \cdot Y' \cdot Y'' + b \cdot Y'', Y'' = a \cdot Y' \cdot Y''' + b \cdot Y''', Y''' = \delta\}$ and $X \leftrightarrows_{\Delta_1 \cup \Delta_2} Y$.

A pleasant property of specifications in rGNF_δ is that the state space of a process expression which is a sequence of process names is a *set* of sequences of process names.

LEMMA 3.6. Let Δ be a $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification in rGNF_δ. If $\sigma \in {V_\Delta}^+$, then whenever $\sigma \xrightarrow{a}_\Delta p$ and $p \in \mathbf{P}^{rec}_{\delta,\Delta}$, $p \in {V_\Delta}^+$.

Proof. let Δ and σ be as in the premise. Let X be the first process name of σ, i.e. (0) $\sigma = X$ or (1) $\sigma = X \cdot \sigma'$. Since $\sigma \xrightarrow{a} p$, by the operational semantics there is a p_i a summand of p' and $X = p' \in \Delta$ which was used in the transition. Since Δ in rGNF_δ p_i is of the form (0) a, (1) $a \cdot X_1$, (2) $a \cdot X_1 \cdot X_2$ and p is of the form (0.1) X_1, (0.2) $X_1 \cdot X_2$, (1.0) σ', (1.1) $X_1 \cdot \sigma'$ or (1.1) $X_1 \cdot X_2 \cdot \sigma'$. In all five cases we find $p \in {V_\Delta}^+$ □

PROPOSITION 3.7. Let Δ be a specification in rGNF_δ and $p \in \mathbf{P}^{rec}_{\delta,\Delta}$. There exists a specification Δ' in rGNF_δ disjoint and computable from Δ and $Y \in V_{\Delta'}$ such that $p \leftrightarrows_{\Delta \cup \Delta'} Y$.

4. Using Mimicking for Deciding Bisimilarity

A crucial ingredient of our proof is the result in [CHS92] that bisimulation equivalence of process expressions over syntactically guarded $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specifications (thus without δ !) is decidable.

THEOREM 4.1. Let E be a syntactically guarded $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specification. Let $p, q \in \mathbf{P}^{rec}_E$. Then $p \leftrightarrows_E q$ is decidable.

We prove the same result for process expressions over guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications, by showing that bisimulation equivalence of sequences of process names over $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications in rGNF_δ is decidable.

In this chapter and Chapter 2 we assume that all specifications $\Delta, \Delta' ...$ are $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications in rGNF_δ, unless specified explicitly otherwise. We prove that bisimulation equivalence of process expressions over such specifications can be reduced to bisimulation equivalence of process expressions over $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specifications, by replacing δ.

Here we give a formal definition of the interpretation of context-free processes, which we refer to as δ mimicking.

DEFINITION 4.2. Let $P_\delta \in \mathrm{Var} - V_\Delta$ and $a_\delta \in Act - A_\Delta$. We define the maps $\mu_{\Delta,a_\delta,P_\delta} : \mathbf{P}^{rec}_{\delta,\Delta} \to \mathbf{P}^{rec}_{\mathrm{Var}}$, $\mu_{\Delta,a_\delta,P_\delta} : \wp_{fin}(\mathrm{Var} \times \mathbf{P}^{rec}_{\delta,\Delta}) \to \wp_{fin}(\mathrm{Var} \times \mathbf{P}^{rec}_{\mathrm{Var}})$ as follows.

$$
\begin{array}{lcl}
\mu_{\Delta,a_\delta,P_\delta}(a) & = & a \\
\mu_{\Delta,a_\delta,P_\delta}(\delta) & = & a_\delta \cdot P_\delta \\
\mu_{\Delta,a_\delta,P_\delta}(X) & = & \text{if } X \leftrightarrows_\Delta \delta, \text{ then } P_\delta, \text{ else } X \\
\mu_{\Delta,a_\delta,P_\delta}(p \oplus q) & = & \mu_{\Delta,a_\delta,P_\delta}(p) \oplus \mu_{\Delta,a_\delta,P_\delta}(q),\ \oplus \in \{+, \cdot\} \\
\mu_{\Delta,a_\delta,P_\delta}(\emptyset) & = & \{P_\delta = a_\delta \cdot P_\delta\} \\
\mu_{\Delta,a_\delta,P_\delta}(\{X = p\} \cup \Delta') & = & \text{if } X \leftrightarrows_\Delta \delta, \text{ then } \mu_{\Delta,a_\delta,P_\delta}(\Delta'), \\
 & & \text{else } \{X = \mu_{\Delta,a_\delta,P_\delta}(p)\} \cup \mu_{\Delta,a_\delta,P_\delta}(\Delta')
\end{array}
$$

where $a \in A_\Delta$, $X \in V_\Delta$ and $p, q \in \mathbf{P}^{rec}_{\delta,\Delta}$.

REMARK 4.3. If X is a process name over some specification Δ, then $\mu_{\Delta,a_\delta,P_\delta}(X)$ is the *δ mimicker of X* over the *δ mimicking specification* $\mu_{\Delta,a_\delta,P_\delta}(\Delta)$.

LEMMA 4.4. If $X \in V_\Delta$, then $X \leftrightarrows_\Delta \delta$ is decidable.

Proof. Since Δ is in rGNF_δ, it is sufficient to verify whether $X = \delta \in \Delta$ □

LEMMA 4.5. The maps $\mu_{\Delta,a_\delta,P_\delta} : \mathbf{P}^{rec}_{\delta,\Delta} \to \mathbf{P}^{rec}_{\mathrm{Var}}$ and $\mu_{\Delta,a_\delta,P_\delta} : \wp_{fin}(\mathrm{Var} \times \mathbf{P}^{rec}_{\delta,\Delta}) \to \wp_{fin}(\mathrm{Var} \times \mathbf{P}^{rec}_{\mathrm{Var}})$ as defined in Definition 4.2 are computable. Moreover the size[4] of the mimicked specification

[4]e.g. the number of symbols or the symbol complexity extended to specifications of Definition 3.48 in Chapter 3.

$\mu_{\Delta,a_\delta,P_\delta}(\Delta)$ is linear in the size of Δ.

Proof. Immediate by definition of μ and Lemma 4.4 □

In this section we further assume that $a_\delta \in Act - A_\Delta$ and $P_\delta \in \mathrm{Var} - V_\Delta$. For readability we drop the subscripts of the mimicking maps μ if they are clear from the context.

PROPOSITION 4.6. $\mu(\Delta)$ is a $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specification in rGNF.

LEMMA 4.7. If $\sigma, \sigma' \in {V_\Delta}^+$, $X \in V_\Delta$ and $a \in A_\Delta$, then

1. $\sigma \xrightarrow{a}_\Delta \sigma'$ iff $\mu(\sigma) \xrightarrow{a}_{\mu(\Delta)} \mu(\sigma')$, and
2. $X \xrightarrow{a}_\Delta \surd$ iff $\mu(X) \xrightarrow{a}_{\mu(\Delta)} \surd$, and
3. If $\mu(\sigma) \xrightarrow{a}_{\mu(\Delta)} \rho$ for some $\rho \in (V_\Delta \cup P_\delta)^+$, then there is a $\sigma'' \in {V_\Delta}^+$ such that $\sigma \xrightarrow{a}_\Delta \sigma''$ and $\mu(\sigma'') = \rho$.

Proof. Notice that μ is an identity on $\{X \in V_\Delta | X \not\leftrightarrows_\Delta \delta\}$ □

We first prove "soundness" of the mimicking translation, i.e. if two process expressions are bisimilar, their mimicked versions are bisimilar as well.

LEMMA 4.8. If $\sigma, \nu \in {V_\Delta}^+$ and $\sigma \leftrightarrows_\Delta \nu$, then $\mu(\sigma) \leftrightarrows_{\mu(\Delta)} \mu(\nu)$.

Proof. Let $R \subseteq (V_\Delta \cup \{P_\delta\})^+ \times (V_\Delta \cup \{P_\delta\})^+$ be the set $\{(\mu(\sigma'), \mu(\nu')) \mid \sigma' \leftrightarrows_\Delta \nu'\}$. We prove that R is a bisimulation. Suppose $(\mu(\sigma'), \mu(\nu')) \in R$.

1. ($\Rightarrow$) Suppose $\mu(\sigma') \xrightarrow{a}_{\mu(\Delta)} \surd$. By definition of μ $a \neq a_\delta$. By the operational semantics $\sigma' = X$ for some $X \in V_\Delta$ and by Lemma 4.7 $X \xrightarrow{a}_\Delta \surd$. By definition of bisimulation $\nu' \xrightarrow{a}_\Delta \surd$ and by the operational semantics $\nu' \in V_\Delta$. By Lemma 4.7 $\mu(\nu') \xrightarrow{a}_{\mu(\Delta)} \surd$.
2. ($\Rightarrow$) Suppose $\mu(\sigma') \xrightarrow{a}_{\mu(\Delta)} \rho_\sigma$. Distinguish $(\frac{T}{F})$ whether or not $a = a_\delta$. (T) If $a = a_\delta$, then $\sigma' \leftrightarrows_\Delta \delta$. Now (0) $\sigma' = X$ or (1) $\sigma' = X \cdot \sigma_2$ for some $\sigma_2 \in {V_\Delta}^+$ and $X \leftrightarrows_\Delta \delta$. By definition of μ (0) $\rho_\sigma = \mu(X)$ or (1) $\rho_\sigma = \mu(X \cdot \sigma_2)$ and $X \cdot \sigma_2 \leftrightarrows_\Delta X \leftrightarrows_\Delta \delta$. By definition of R $\nu' \leftrightarrows_\Delta \delta$, (0) $\nu' = Y$ or (1) $\nu' = Y \cdot \nu_2$ for some $\nu_2 \in {V_\Delta}^+$ and $Y \leftrightarrows_\Delta \delta$. By definition of μ (0) $\mu(\nu') \xrightarrow{a_\delta}_\Delta P_\delta$ or (1) $\mu(\nu') \xrightarrow{a_\delta}_\Delta P_\delta \cdot \mu(\nu_2)$. By definition of R (0) $(\mu(\sigma'), \mu(Y)) \in R$ and $\mu(Y) = P_\delta$ or (1) $(\mu(\sigma'), \mu(Y \cdot \nu_2)) \in R$ and $\mu(Y \cdot \nu_2) = P_\delta \cdot \mu(\nu_2)$.

   (F) If $a \neq a_\delta$, then by Lemma 4.7 $\sigma' \xrightarrow{a}_\Delta \sigma''$ for some $\sigma'' \in {V_\Delta}^+$ and $\mu(\sigma'') = \rho_\sigma$. By definition of R there is a ν'' such that

$\nu' \xrightarrow{a} \nu''$ and $\nu'' \leftrightarrows_{\Delta} \sigma''$. By Lemma 4.7 $\mu(\nu') \xrightarrow{a}_{\mu(\Delta)} \mu(\nu'')$ and by definition of R, $(\rho_\sigma, \mu(\nu'')) \in R$.

Obviously $(\mu(\sigma), \mu(\nu)) \in R$ and hence $\mu(\sigma) \leftrightarrows_{\mu(\Delta)} \mu(\nu)$ □

Next we prove "completeness" of the mimicking translation, i.e. if two mimicked process expressions are bisimilar, the original process expressions are bisimilar.

LEMMA 4.9. If $\sigma, \nu \in {V_\Delta}^+$ and $\mu(\sigma) \leftrightarrows_{\mu(\Delta)} \mu(\nu)$, then $\sigma \leftrightarrows_{\Delta} \nu$.

Proof. Let $R \subseteq {V_\Delta}^+ \times {V_\Delta}^+$ be the set $\{(\sigma', \nu') \mid \mu(\sigma') \leftrightarrows_{\mu(\Delta)} \mu(\nu')\}$. We prove that R is a bisimulation. Suppose $(\sigma', \nu') \in R$.

1. ($\Rightarrow$) Suppose $\sigma' \xrightarrow{a}_{\Delta} \surd$. Since δ is in rGNF_δ by the operational semantics $\sigma' = X$ for some $X \in V_\Delta$ and by Lemma 4.7 $\mu(\sigma') \xrightarrow{a}_{\mu(\Delta)} \surd$. By definition of R $\mu(\nu') \xrightarrow{a}_{\mu(\Delta)} \surd$. Similarly $\nu' = Y$ for some $Y \in V_\Delta$ and and by Lemma 4.7 $\nu' \xrightarrow{a}_{\Delta} \surd$.
2. ($\Rightarrow$) Suppose $\sigma' \xrightarrow{a}_{\Delta} \sigma''$. Since $a \neq a_\delta$, by Lemma 4.7 $\mu(\sigma') \xrightarrow{a}_{\mu(\Delta)} \mu(\sigma'')$. By definition of R $\mu(\nu') \xrightarrow{a}_{\mu(\Delta)} \rho$, for some $\rho \leftrightarrows_{\mu(\Delta)} \mu(\sigma'')$. By Lemma 4.7 $\nu' \xrightarrow{a}_{\Delta} \nu''$ and $\mu(\nu'') = \rho$.

Obviously $(\sigma, \nu) \in R$ and hence $\sigma \leftrightarrows_{\Delta} \nu$ □

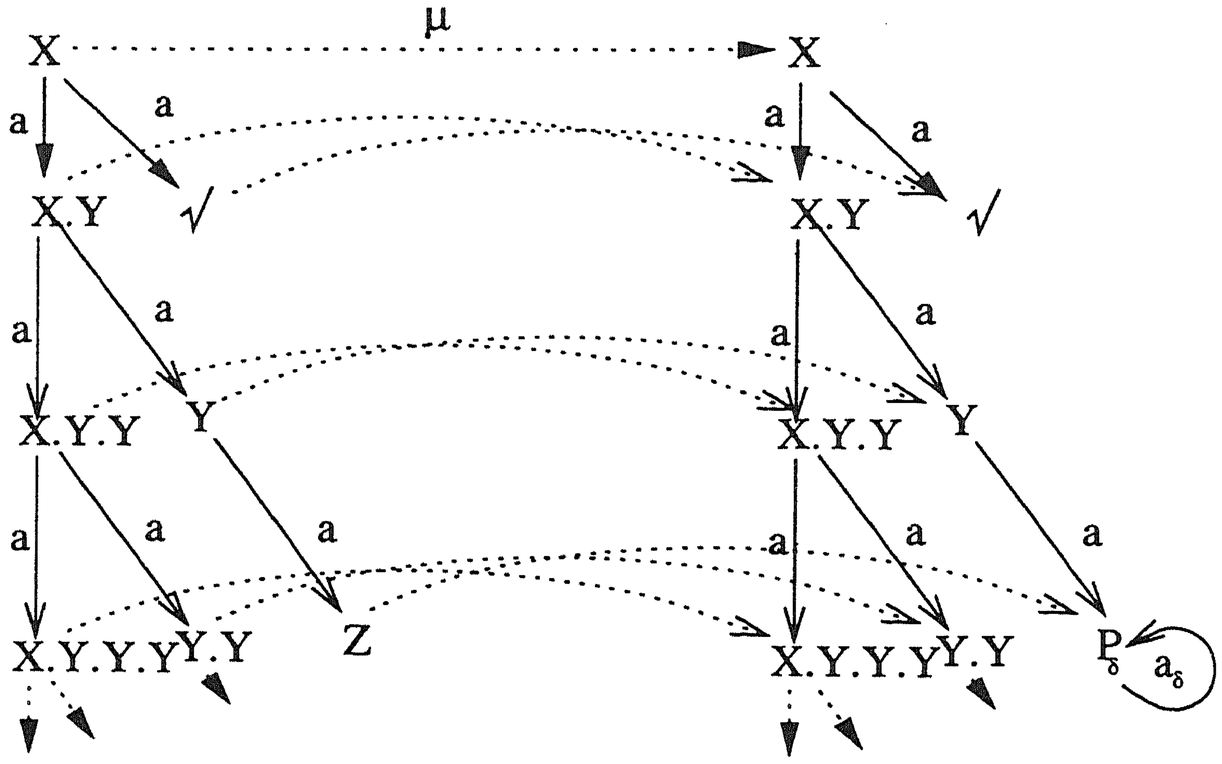
THEOREM 4.10. Let $\sigma, \nu \in {V_\Delta}^+$. Then $\sigma \leftrightarrows_{\Delta} \nu$ is decidable.

Proof. By Theorem 4.1 and Proposition 4.5 we are finished if we can effectively reduce the question whether or not $\sigma \leftrightarrows_{\Delta} \nu$ to that of $\mu(\sigma) \leftrightarrows_{\mu(\Delta)} \mu(\nu)$. Hence the result follows by Lemma 4.8 and 4.9 □

COROLLARY 4.11. Bisimilarity of process expressions over syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications is decidable.

EXAMPLE 4.12. Let Δ be the specification $\{X = a \cdot X \cdot Y + a, Y = a \cdot Z, Z = \delta\}$. In the picture below left the process graphs of X over Δ

is depicted and right X $(= \mu(X))$ over $\mu(\Delta)$.



In [BCS95] it is proved that decidability of bisimulation equivalence has exponential time complexity [5] thus improving the result of [CHS92] which proved decidability of the full class but permits no analysis of the complexity. Since computing $\mu(\Delta)$ is relatively cheap, we expect that the complexity of deciding bisimulation of process expressions over syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications is exponential too.

5. Decidability of Bisimulation Equivalence of Context-Free Processes definable with ϵ ?

The results in the previous section show that decidability of bisimulation equivalence over context-free processes with δ is easily reduced to that of context-free processes *without* δ. An interesting question is whether the same method can be used to show interconnections of other process algebras.

---

[5] Unfortunately we cannot deduce the measure of the complexity used in [BCS95], i.e. is the complexity formulated in terms of the number of process names defined, the number of summands of the specification .. ? We expect the authors to have meant the *size* of the specification, in which the complexity is usually measured e.g. the (polynomial) time complexity of deciding bisimulation of normed context-free processes [HJM94].

A question still open to our knowledge is whether or not bisimulation equivalence of context-free processes defined with the extra constant ϵ, otherwise known as the process expressions over $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications is decidable. In this section we explain why we think the mimicking technique *does not* apply to the setting with ϵ.

Syntax and Semantics of BPA_ϵ

We present the main differences of syntax and semantics with respect to BPA_δ. The syntax of the set of process expressions $\mathbf{P}^{rec}_{\epsilon,\mathrm{Var}}$ over the signature of BPA_ϵ and Var is given by the abstract syntax

$$p ::= a \mid \epsilon \mid p + p \mid p \cdot p \mid X$$

where $a \in Act$ and $X \in \mathrm{Var}$.

The operational semantics of process expressions over $\Sigma(\mathrm{BPA}_\epsilon(Act, E_\epsilon))$, where E_ϵ is a $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specification is given by the unary termination relation $\cdot\downarrow \subseteq \mathbf{P}^{rec}_{\epsilon,\mathrm{Var}}$ and the ternary transition relation $\cdot\xrightarrow{\cdot}\cdot \subseteq \mathbf{P}^{rec}_{\epsilon,E_\epsilon} \times Act \times \mathbf{P}^{rec}_{\epsilon,E_\epsilon}$ satisfying the following rules.

$$\frac{p\downarrow \;\; X = p \in E_\epsilon}{X\downarrow} \qquad \frac{p\downarrow}{(p+q)\downarrow} \qquad \frac{p\downarrow}{(q+p)\downarrow} \qquad \frac{p\downarrow, q\downarrow}{(p\cdot q)\downarrow} \qquad \frac{}{\epsilon\downarrow}$$

$$\frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{p \xrightarrow{a} p'}{q+p \xrightarrow{a} p'}$$

$$\frac{p \xrightarrow{a} p'}{p\cdot q \xrightarrow{a} p'\cdot q} \qquad \frac{p\downarrow \;\; q \xrightarrow{a} q'}{p\cdot q \xrightarrow{a} q'} \qquad \frac{}{a \xrightarrow{a} \epsilon}$$

$$\frac{p \xrightarrow{a} p' \;\; X = p \in E_\epsilon}{X \xrightarrow{a} p'}$$

where $a \in Act$.

DEFINITION 5.1. Let E_ϵ be a $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specification.

1. Let $R \subseteq \mathbf{P}^{rec}_{\epsilon,E_\epsilon} \times \mathbf{P}^{rec}_{\epsilon,E_\epsilon}$ be a relation satisfying for every $(p, q) \in R$,
   (a) $p\downarrow$ iff $q\downarrow$, and

(b) if $p \xrightarrow{a} p'$ for some $a \in Act$, there is a $q' \in \mathbf{P}^{rec}_{\epsilon,\mathrm{Var}}$ such that $q \xrightarrow{a} q'$ and $(p', q') \in R$, and
(c) if $q \xrightarrow{a} q'$ for some $a \in Act$, there is a $p' \in \mathbf{P}^{rec}_{\epsilon,\mathrm{Var}}$ such that $p \xrightarrow{a} p'$ and $(p', q') \in R$,

then R is *a bisimulation over* E_ϵ,

2. If there exists a bisimulation over E_ϵ such that $(p, q) \in R$, then p and q are *bisimilar over* E_ϵ, shortly $p \leftrightarroweq_{E_\epsilon} q$.

DEFINITION 5.2. Let Δ be a $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specification.

1. If $p \in \mathbf{P}^{rec}_{\delta,\Delta}$ and every occurrence of variable $X \in \mathrm{Var}$ in p is in a subexpression of the form $a \cdot p'$, where $a \in Act$, then p is *syntactically guarded.*
2. If [whenever $X = p \in \Delta$, p is syntactically guarded], then Δ is *syntactically guarded.*

DEFINITION 5.3. Let E_ϵ be a $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specification.

1. ϵ, a, $a \cdot X$ and $a \cdot X \cdot Y$ for $a \in Act$, $X, Y \in V_{E_\epsilon}$ are *in* $rGNF_\epsilon$.
2. If $p \in \mathbf{P}^{rec}_{\epsilon,E_\epsilon}$, every summand of p is in rGNF_ϵ and every summand appears only once, then p is *in* $rGNF_\epsilon$.
3. If [whenever $X = p \in E_\epsilon$, p is in rGNF_ϵ], then E_ϵ is *in* $rGNF_\epsilon$.

It is folklore that a $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specification E_ϵ in rGNF_ϵ has an unique solution $\{(X, E_\epsilon), ...\}$ as defined in Section 2 [BV95] of this chapter. In the same way as in the previous section we take the liberty to speak of the *process name* X, instead of (X, E_ϵ) and we consider process expressions over the signature of BPA_ϵ and the solution of E_ϵ.

In the remainder of this chapter we assume specifications to be over $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ and in rGNF_ϵ.

Using Mimicking for Deciding Bisimulation over BPA_ϵ ?

If we use the mimicking technique of the previous section to mimic ϵ, we replace occurrences of ϵ by a fresh process name X_ϵ. Also we extend specifications with a new equation $X_\epsilon = p_\epsilon$, where p_ϵ is a process expression which performs only fresh actions with respect to A_Δ. We leave here the precise form of p_ϵ open, e.g. $a_\epsilon \cdot X_\epsilon$, a direct translation of δ mimicking or $p_\epsilon = a_\epsilon$, where a_ϵ is of course a fresh action.

A first observation is that the replacement forces to distinguish process expressions which are obviously bisimilar. This is the case for

process expressions specifications where the ϵ is not essential. This is if the specification can be substituted by an ϵ-*free* specification so that a process expression over the original specification and a process expression over a new (disjoint) specification are bisimilar, but the new specification has no ϵ occurrences. We describe such a specification E_ϵ in the next example.

EXAMPLE 5.4. Let E_ϵ be the specification defined as

$$\begin{aligned} E_\epsilon \quad = \quad & \{X = b \cdot Y \cdot Z \\ & Y = b + \epsilon \\ & Z = b \cdot Z\} \end{aligned}$$

and its ϵ mimicked version $\mu_\epsilon(E)$ defined as

$$\begin{aligned} \mu_\epsilon \quad = \quad & \{X = b \cdot Y \cdot Z \\ & Y = b + X_\epsilon \\ & Z = b \cdot Z \\ & X_\epsilon = p_\epsilon\}. \end{aligned}$$

The process names X and Z are obviously bisimilar over E_ϵ, whereas their ϵ mimicked versions are not: The ϵ mimicker for X has a derivative $p_\epsilon \cdot Z$ which can perform a fresh action due to the translation of the summand p_ϵ in Y, which the ϵ mimicker for Z cannot.

This immediately poses the question
Question 1: *Is it decidable whether an equivalent ϵ-free specification for a process expression over a syntactically guarded* $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ *specification exists ?*

In the above example it is easily seen whether or not there exists an ϵ-free specification for X, but it is already much more difficult to prove for T and the specification in Example 5.8 (Of which we prove the ϵ to be essential in the specification). We have no idea whether this is decidable. A tangible criterion would be that the process expression does not terminate intermediately, i.e. has no derivatives which can terminate *and* perform an action. The next supposition is that if a process expression is not intermediately terminating, then the defining specification can be effectively replaced by an equivalent ϵ-free specification. This would then immediately imply decidability of bisimulation

equivalence for process expressions over $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications.

LEMMA 5.5. Let $\sigma \in {V_{E_\epsilon}}^+$. It is decidable whether σ is intermediately terminating.

Proof. Notice that since E_ϵ is in rGNF_ϵ, a process name X is intermediately terminating iff its defining equation in E_ϵ has a summand ϵ *and* a summand $\neq \epsilon$. Furthermore if $\sigma = X_1 \cdot ... \cdot X_k$ and σ is intermediately terminating, then $X_i \downarrow$ for all i and there is a X_j such that X_j is intermediately terminating □

Notice that this results extends to syntactically guarded specifications with straight forward extensions of Lemma 3.4 and Proposition 3.7.

We next present an example of a process expression which is not intermediately terminating, but is nevertheless not bisimilar to a process expression over some $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specification. This specification is even more interesting, since it shows that process expressions over $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications need not have a bound on the branching degree. This makes it even more doubtful that process expressions over $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications can be encoded faithfully with respect to bisimulation as context-free processes.

DEFINITION 5.6. The *branching degree* $bf : \mathbf{P}^{rec}_{\epsilon,E_\epsilon} \to \mathbf{N} \cup \{\infty\}$ of a process expression is the size of the set $\{p \xrightarrow{a}_\Delta p' | a \in Act, p' \in \mathbf{P}^{rec}_{\epsilon,E_\epsilon}\}$.

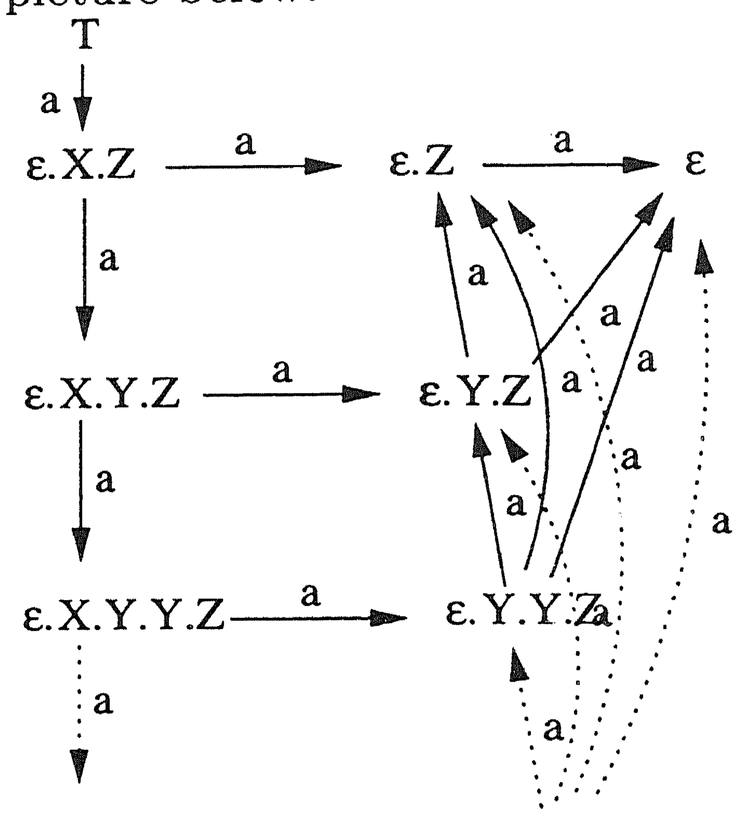
LEMMA 5.7. Let E be a syntactically guarded $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specification. If $p \in \mathbf{P}^{rec}_E$, then $bf(p)$ is bounded by the maximal number of summands of the equations of E.

Proof. By the operational semantics the maximum number of different derivatives of p is bounded by the number of summands of the equations of E □

EXAMPLE 5.8. Let E_ϵ be a specification defined as

$$\begin{aligned} E_\epsilon \;=\; \{&T = a \cdot X \cdot Z \\ &X = a \cdot X \cdot Y + a \\ &Y = a + \epsilon \\ &Z = a\} \end{aligned}$$

We can verify that that the process expressions $\epsilon \cdot Y^m \cdot Z$ have a branching degree of $m+1$ for every $m > 0$, i.e. $\epsilon \cdot Y^m \cdot Z$ has derivatives $\epsilon \cdot Z, ..., \epsilon \cdot Y^{m-1} \cdot Z$. This holds also modulo bisimulation since $\epsilon \cdot Y^k \cdot Z \not\leftrightarrows_{E_\epsilon} \epsilon \cdot Y^l \cdot Z$ for $k \neq l$, where $k, l > 0$ as can verified with the picture below.



COROLLARY 5.9. The following statement is *false*.

If $p \in V_{E_\epsilon}{}^+$ is not intermediately terminating, then there is a process expression p' over a $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specification E disjoint from E_ϵ so that $p \leftrightarrows_{E_\epsilon \cup E} p'$.

We expect that the proof of decidability of bisimulation equivalence of process expressions over $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications needs a completely new proof. Most likely we cannot scrounge the result of [CHS92] as we did in this chapter with the decidability of bisimulation equivalence of BPA_δ by interpreting δ as a live lock.

We end with a last open question.

Question 2: *Is bisimulation equivalence of process expressions over syntactically guarded $\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications decidable ?*

We conjecture that in case one would establish decidability of bisimulation equivalence of process expressions over syntactically guarded

$\Sigma(\mathrm{BPA}_\epsilon(Act, \mathrm{Var}))$ specifications, we can reduce decidability of bisimulation equivalence over syntactically guarded $\Sigma(\mathrm{BPA}_{\delta,\epsilon}(Act, \mathrm{Var}))$ specifications to the former question using an extension of δ mimicking.

REMARK 5.10. In this chapter (and the rest of this thesis) we are primarily concerned with *strong* bisimulation equivalence. In recent years also the decidability of *branching* bisimulation equivalence [GW89] over context-free processes with a silent step (τ) has been solved for the normed case [Hüt91].

It is an open question whether branching bisimulation equivalence over all context-free processes with a silent step is decidable. This is relevant to our problem since the behavior of τ in the context of branching bisimulation has some similarities with ϵ, as is noticed by several authors [BV95, Vra97].

Our key argument against the possibility of reduction of the decidability question in this section is that context-free processes definable *with* ϵ can have an unbounded branching degree, whereas context-free processes definable *without* ϵ have a bounded branching degree.

Context-free processes definable with the extra τ do not have this restriction, if we assume the standard operational semantics in the setting of branching bisimulation[6].

We see no way to translate the decision of arbitrary (possibly *un*normed) context-free processes definable with ϵ to a decision of branching bisimilarity of normed context processes. We conjecture that an interpretation as a decision problem of branching bisimilarity of *arbitrary* context-free processes fails because there is no method to get around the ϵ axiom $\epsilon \cdot x = x$: whereas $a + \epsilon \cdot b$ and $a + b$ are strongly bisimilar, $a + \tau \cdot b$ and $a + b$ are not branching bisimilar.

Acknowledgements. I thank Jan Friso Groote, Bas Luttik and Alban Ponse for simplifications in the proofs and fruitful discussions.

[6] If we replace ϵ by τ in Example 5.8 T has no upper bound on the branching degree modulo *branching* bisimulation.

CHAPTER 2

Regularity of Context-Free Processes definable with δ

In [BG96] an upper bound is given of the size of the state space modulo bisimulation equivalence of regular process expressions over so-called NRD specifications, a non-trivial subclass of the context-free processes. This result implies the decidability of regularity of process expressions over these context-free processes. The decidability of regularity of all context-free processes is proved in [BCS96] . We show that the upper bound result directly extends to context-free processes definable with δ of ACP [BK84b]. Furthermore, the problem of decidability of regularity of context-free processes definable with δ can be effectively reduced with the mimicking technique of Chapter 1 to one deciding the regularity of context-free processes definable *without* δ.

1. Introduction

In Chapter 1 we proved that bisimulation equivalence of context-free processes is decidable. Here context-free processes are defined by the process expressions over syntactically guarded specifications over the signature of Basic Process Algebra (BPA) with the constant δ from ACP [BK84a]. This leads one to believe that bisimulation is a sufficiently strong equivalence to imply decidability of *regularity* of context-free processes, where regularity means that the state space of a process is finite modulo bisimulation equivalence. As with bisimulation equivalence, decidability of regularity has been mostly studied for context-free processes definable *without* δ.

In the past five years the decidability of regularity has been extended from process expressions over *linear* specifications to process expressions over *arbitrary* syntactically guarded specifications (without δ), which we can illustrate with the example below.

EXAMPLE 1.1.

$$(1)\quad \begin{aligned} X &= a\cdot Y + c\cdot Z\\ Y &= d\cdot X + e\cdot Y\\ Z &= c \end{aligned} \qquad (2)\quad \begin{aligned} X &= a\cdot X\cdot Y + c\\ Y &= b\cdot Y \end{aligned}$$

$$(3)\quad \begin{aligned} X &= a\cdot Y\cdot Z\\ Y &= a\cdot Y\cdot B + c\\ Z &= b\cdot Z\\ B &= b \end{aligned} \qquad (4)\quad \begin{aligned} X &= a\cdot T\cdot Z\\ T &= b + c\cdot T\cdot Y\\ Y &= d + d\cdot Y + d\cdot Z\\ Z &= e\cdot Z \end{aligned}$$

All four specifications define X as a regular process name. X is regular with respect to specification (1) because the specification is *linear*. It is well-known that the class of regular process expressions is the same as the class of process expressions which can be denoted by a linear specification [Mil84, BV95]. In fact one of the reasons for using regular process expressions is that these can be described precisely by the class of linear specifications, which allows an easy implementation and checking of modal and temporal properties [Hol89].

The specification (2) also defines X as a regular process name, however to see this is already more difficult since the specification is not linear (X has the summand $a\cdot X\cdot Y$ in the defining equation). In [MM94] the decidability of regularity of BPA *systems* [1] is proved. It is proved that specifications which employ no "normed stackings" define only regular process expressions, a result also implied by [Kru95]. Both papers describe a method to generate a linear specification for a process expression, provided that it is regular.

In [BG96] we proved that we can extend the class of specifications further by allowing specifications like (3). The specification is not linear and not a regular BPA system (i.e. not all process names are regular), since Y obviously is not regular. However X is regular: the idea is that the "context-free behavior" of Y is somehow neutralized by Z. New in this approach is that we do not consider the specification as a whole, but investigate the regularity of the individual process names by explicitly distinguishing a "root" process name.

[1] Which has the same meaning as that *all* process names defined by a specification are regular. This implies that the elements of $\mathbf{P}^{rec}_{\delta,(2)}$ are regular, i.e. the set of all process expressions definable over the signature of BPA with δ and the process names defined by the specification (2).

The result in [BG96] establishes decidability of regularity of process expressions over the class of specifications such that the weakly Normed Repeats are weakly Deterministic (NRD, see Section 2.1). For this class of specifications we defined an effective criterion for the regularity of process expressions. The effectiveness proof follows an "upper bound argument": We prove that a process expression over a NRD specification is regular iff the state space modulo bisimulation has less than an upper bounded number of states [2]. This gives a decision procedure which lists new states until no more non bisimilar states can be generated, but stops when the approximation of the state space has more states than the upper bound. The test stops before an upper bounded number of non-bisimilar states is generated *if and only if* the process expression is regular. Since generation of new states and testing of bisimilarity is effective [CHS92, BCS96], this describes an effective procedure.

Almost at the same time that [BG96] was published, decidability of regularity for *all* context-free processes was proved independently in [BCS96]. Besides establishing the regularity of the process name X over the specifications (1)–(3) in Example 1.1, this approach can establish regularity of X over (4), which the approach in [BG96] cannot (as was discovered by the authors of [BCS96]). The approach of [BCS96] is radically different from ours and uses a sophisticated argument to show that the question of regularity of context-free processes can be reduced to the (decidable) question of whether a regular graph has a finite grammar. The method of [BCS96] does not say anything about the size of the state space modulo bisimulation equivalence of a regular context-free process.

In this chapter we present again the main results of [BG96] and extend these to specifications definable with δ. We first prove that the upper bound result holds also for regular process expressions over NRD specifications. As indicated above, this proves that regularity of process expressions over NRD specifications is decidable. Second we prove that the technique of δ mimicking as presented in Chapter 1 extends the decidability result of regularity of all context-free processes of [BCS96] to a setting with δ.

[2] Which is an exponential bound in the number of different process names of a specification, see Lemma 3.16.

The plan of this chapter is as follows. In Section 2 we define regularity formally and give preliminary definitions for the upper bound result, which we present in Section 3. In Section 4 we prove that the δ mimicking technique of Chapter 1 and the decidability result of [BCS96] imply decidability of regularity for all context-free processes definable with δ.

2. Regularity and NRD specifications

For convenience we recall in this chapter all results of Section 2 of Chapter 1 about the process expressions, consisting of sequences of process names over $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications in rGNF_δ. Unless explicitly stated otherwise we assume that a specification Δ is over $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ and in rGNF_δ, V_Δ is the set of process names defined by Δ with typical elements X, Y, Z and $\nu, \rho, \sigma \in {V_\Delta}^+$ are sequences of process names.

For convenience we introduce some extra terminology, which is mostly straightforward: If $\rho = X$ or $\rho = X \cdot \sigma$, then X is the *head process name* of ρ. We feel free to write a transition as $p \to_\Delta q$ and not $p \xrightarrow{a}_\Delta q$, if we have no interest in the action a. For the process expressions p and q we use also the words *derivatives* and *states*. We use the Greek letters φ, χ, ψ to range over transition sequences, possibly ending with a termination predicate. If φ ends with a $\to_\Delta \surd$ step, shortly a *tick step* then φ is *terminated*.

We mean by $|\varphi|$ the length of a transition sequence φ given by the number of transitions and tick steps, e.g. $|X \to_\Delta Y \to_\Delta \surd| = 2$ and $|X \to_\Delta Y| = |Y \to_\Delta \surd| = 1$. We say that φ is *finite*, if φ has a finite length. If $i < |\varphi|$, then $\varphi(i)$ denotes the $i+1$-th state of φ. If φ is finite and not terminated, then $\varphi(|\varphi|)$ denotes the last state of φ, which we also denote by $last(\varphi)$. Hence $X \to_\Delta Y \to_\Delta \surd(0) = X$ and $X \to_\Delta Y \to_\Delta \surd(1) = Y$. The last state of a terminating transition sequence is undefined, e.g. $X \to_\Delta Y \to_\Delta \surd(2)$ is not defined.

We say that σ is *weakly normed over* Δ, shortly $\sigma \Downarrow_\Delta$, if there exists a terminated transition sequence starting with σ [3] and *perpetual* if $\sigma \not\Downarrow_\Delta$. By $|V_\Delta|$ we mean the *size* of the set V_Δ, and $|\nu|$ the *number* of process variables of ν, e.g. $|X| = 1$ and $|X \cdot \sigma| = 1 + |\sigma|$ for $X \in V_\Delta$ and $\sigma \in {V_\Delta}^+$. By ν^k for $k > 0$ we mean the sequential composition

[3] Notice that a weakly normed process can come after one or more transitions in a state that is perpetual, i.e. it is not *normed* in the usual meaning.

of k $\nu's$. If no confusion can arise, we use $|\varphi|, |V_\Delta|, |\nu|$ without stating whether respectively the number, length or size is meant.

The most important notion that we use in this chapter is regularity that we define on process expressions.

DEFINITION 2.1. Let $p \in \mathbf{P}^{rec}_{\delta,\Delta}$. If there exist only transition sequences $p = p_0 \to p_1 \to p_2 \ldots$ such that there are only finitely many pair-wise non bisimilar p_i, then p is *regular over* Δ.

For convenience we define regularity as the *absence* of the possibility of infinite sequences of non bisimilar states. If process expressions are finitely branching this implies that the state-space modulo bisimulation is finite, which is another frequently seen formulation in the literature [MM94, Kru95].

LEMMA 2.2. The state-space modulo bisimulation equivalence of a process expression that is regular over Δ is finite.

Proof. It is folklore that process expressions over finite syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications are finitely branching. By König's Lemma a regular process expression hence gives rise to a finite state space modulo bisimulation □

2.1. NRD specifications. In Section 3 we prove decidability of regularity of process expressions over a restricted class of specifications, called NRD specifications. In this section we give the definition and some auxiliary results.

DEFINITION 2.3. If $X = p \in \Delta$ and there is only one summand[4] in p starting with the action $a \in Act$, then a is *unique in* X *over* Δ.

EXAMPLE 2.4. Let Δ be the specification $X = \{X = a \cdot X + b \cdot Y + b \cdot X \cdot Y + c \cdot Y + d, Y = \delta\}$. Then the actions a, c and d are unique in X over Δ and b is not.

PROPOSITION 2.5. It is decidable whether a is unique in X over Δ.

DEFINITION 2.6. Let $W_{\Delta,0}, W_{\Delta,i}, \cdots \subseteq V_\Delta$ be sets inductively defined as follows

[4] Recall that specifications in rGNF_δ have only equations with unequal summands.

1. $W_{\Delta,0} = \emptyset$,
2. $X \in W_{\Delta,i+1}$ iff either $X \in W_{\Delta,i}$ or $X = p \in \Delta$ and there is an action a such that a is unique in X over Δ, and the summand starting with a is either a, or has all subsequent process names in $W_{\Delta,i}$.

Let $W_{\Delta,i} = W_{\Delta,i+1}$ for some $i \in \mathbf{N}$. Then $W_{\Delta,i}$ is the set of *weakly normed deterministic* process names for Δ, denoted as W_Δ.

EXAMPLE 2.7. In the specifications in Example 1.1,

1. $W_{(1),0} = \emptyset$, $W_{(1),1} = \{Z\}$, $W_{(1),2} = \{Z, X\}$ and $W_{(1),i} = \{Z, X, Y\}$ for $i > 2$, and
2. $W_{(2),0} = \emptyset$ and $W_{(2),i} = \{X\}$ for $i > 0$, and
3. $W_{(3),0} = \emptyset$ and $W_{(3),i} = \{Y, B\}$ for $i > 0$, and
4. $W_{(4),0} = \emptyset$ and $W_{(4),i} = \{T\}$ for $i > 0$.

PROPOSITION 2.8. For $i < j$ $W_{\Delta,i} \subseteq W_{\Delta,j}$, $|W_{\Delta,j}| - |W_{\Delta,i}| \leq |V_\Delta|$ and if $W_{\Delta,i} = W_{\Delta,i+1}$, then $W_{\Delta,i} = W_{\Delta,j}$.

LEMMA 2.9. W_Δ is effectively computable.

Proof. Immediate by Propositions 2.5 and 2.8 □

DEFINITION 2.10. Let $X \in V_\Delta$ and $\varphi = X \rightarrow \ldots$ be a non terminated, finite transition sequence. If $last(\varphi) = X \cdot \sigma$ for some $\sigma \in {V_\Delta}^+$, $last(\varphi) \Downarrow_\Delta$ and [whenever $i < j < |\varphi|$, $\varphi(i)$ and $\varphi(j)$ do not have the same head process name], then φ is a *weakly normed repeat* and σ is the *stacking of* φ.

EXAMPLE 2.11. In the specifications in Example 1.1,

1. there are no weakly normed repeats defined by (1) and (2), and
2. $Y \xrightarrow{a}_{(3)} Y \cdot B$ is the only weakly normed repeat defined by (3), and
3. $T \xrightarrow{c}_{(4)} T \cdot Y$ is the only weakly normed repeat defined by (4).

LEMMA 2.12. If X starts a weakly normed repeat defined by Δ, then X is not regular over Δ.

Proof. Let X start a weakly normed repeat $X \rightarrow_\Delta \ldots X \cdot \sigma$ for some $\sigma \in {V_\Delta}^+$. Observe that there is an infinite transition sequence $X \rightarrow_\Delta \ldots X \cdot \sigma \rightarrow_\Delta \ldots X \cdot \sigma^n \rightarrow_\Delta \ldots X \cdot \sigma^{n+1} \rightarrow_\Delta \ldots$ and for $k > l \geq 1$

$X \cdot \sigma^k \not\trianglelefteq {}_\Delta X \cdot \sigma^l$. This implies that X starts a transition sequence of infinitely many pair-wise non-bisimilar states □

LEMMA 2.13. Let $\sigma \in {V_\Delta}^+$. It is decidable whether $\sigma \Downarrow_\Delta$.

Proof. Let $N_{\Delta,i}$ for $i \geq 0$ be the sets inductively defined as follows:

1. $N_{\Delta,0} = \emptyset$, and
2. $N_{\Delta,i+1} = V_{\Delta,i} \cup \{X | X = p \in \Delta, p \text{ has summands } a,\ a \cdot \vec{Y},\ \vec{Y} \in N_{\Delta,i}\}$

It is straight forward that $\sigma \Downarrow_\Delta$ iff $\sigma \in {N_{\Delta,|\Delta|}}^+$ and $N_{\Delta,|\Delta|}$ is effectively computable from Δ□

LEMMA 2.14. The set of weakly normed repeats defined by a specification is effectively computable.

Proof. Immediate by Definition 2.10 and Lemma 2.13 □

DEFINITION 2.15. A specification Δ is *NRD* (weakly Normed Repeat variables are weakly Deterministic) iff the set of process names of V_Δ appearing in stackings of weakly normed repeats is a subset of W_Δ.

EXAMPLE 2.16. In the specifications in Example 1.1,

1. (1) and (2) are NRD, since they define no weakly normed repeats,
2. (3) is NRD, since it defines only one weakly normed repeat $Y \xrightarrow{a}_{(3)} Y \cdot B$ and the process name B appearing in its stacking is an element of $W_{(3)} = \{Y, B\}$,
3. (4) is not NRD, since it defines one weakly normed repeat $T \xrightarrow{c}_{(4)} T \cdot Y$ such that $Y \notin W_{(4)} = \{T\}$.

LEMMA 2.17. It is is decidable whether a specification is NRD.

Proof. Immediate by Lemmas 2.9 and 2.14 □

In the proofs in the next section we need two other forms of repeats.

DEFINITION 2.18. Let $X \in V_\Delta$ and $\varphi = X \rightarrow \ldots$ be a non terminated, finite transition sequence. If $last(\varphi)$ has X as head process name and [for $i < j < |\varphi|$, $\varphi(i)$ and $\varphi(j)$ do not have the same head process name], then

1. if $last(\varphi) = X$, then φ is an *empty stacking* repeat, shortly a *cyclic* repeat, and

2. if $last(\varphi) = X \cdot \sigma$ for some $\sigma \in {V_\Delta}^+$ and $last(\varphi) \not\Downarrow_\Delta$, then φ is a *perpetually stacking* repeat, shortly a *perpetual* repeat.

EXAMPLE 2.19. In the specifications in Example 1.1,

1. (1) defines only cyclic repeats, i.e. $X \xrightarrow{a}_{(1)} Y \xrightarrow{d}_{(1)} X$, $Y \xrightarrow{e}_{(1)} Y$ and $Y \xrightarrow{d}_{(1)} X \xrightarrow{a}_{(1)} Y$, and
2. (2) defines one cyclic repeat $Y \xrightarrow{b}_{(2)} Y$ and one perpetual repeat $X \xrightarrow{a}_{(2)} X \cdot Y$, and
3. (3) defines one cyclic repeat $Z \xrightarrow{b}_{(3)} Z$ and no perpetual repeats, and
4. (4) defines two cyclic repeats $Y \xrightarrow{d}_{(4)} Y$ and $Z \xrightarrow{e}_{(4)} Z$ and no perpetual repeats.

2.2. Properties of NRD specifications.

DEFINITION 2.20. A transition sequence χ is *weakly deterministic* iff for every transition $\chi(i) \xrightarrow{a}_\Delta \chi(i+1)$ or $\chi(i) \xrightarrow{a}_\Delta \surd$ in χ, a is unique in the head process name of $\chi(i)$ over Δ.

LEMMA 2.21. If $\sigma \in {W_\Delta}^+$, then there is a weakly deterministic transition sequence $\sigma \xrightarrow{a_0}_\Delta \ldots \xrightarrow{a_n}_\Delta \surd$.

Proof. We prove the result by a simultaneous well-founded induction on $|\sigma|$, and i, the "lowest" set $W_{\Delta,i}$ of Definition 2.6 the head process name of σ belongs to. Let the Induction Hypothesis (I.H.) be that all σ' with head process name in $W_{\Delta,j}$, such that $(|\sigma'|, j)$ is lexicographically smaller than $(|\sigma|, i)$ start a weakly deterministic transition sequence. By definition of W_Δ there is a transition possible of the form (0) $\sigma \xrightarrow{a}_\Delta \surd$ or (1) $\sigma \xrightarrow{a}_\Delta \rho$, where a is unique in the head process name of σ over Δ and $\rho \in {V_\Delta}^+$. In case (0) we are obviously finished. In case (1) we can distinguish the following cases according to the form of ρ:

1. $\rho \equiv X$ for some $X \in W_{\Delta,j}$ and $j < i$. We are finished by the I.H., and
2. $\rho \equiv X \cdot Y$ for some $X, Y \in W_{\Delta,j}$ and $j < i$. By the I.H. there are weakly deterministic transition sequences possible from X and Y, which can be "glued" together in the obvious way, and
3. $|\rho| < |\sigma|$. We are finished by the I.H., and

4. $\rho \equiv \rho_1 \cdot \rho_2$, $|\rho_1| < |\sigma|$, $|\rho_2| < |\sigma|$ and $\rho_1, \rho_2 \in {W_\Delta}^+$. By the I.H. there are weakly deterministic transition sequences possible from ρ_1 and ρ_2, which can be "glued" together in the obvious way □

EXAMPLE 2.22. The transition sequence $Y^n \xrightarrow{c}_{(3)} Y^{n-1} \ldots Y \xrightarrow{c}_{(3)} \surd$ for $n > 2$ is weakly deterministic over the specification (3) of Example 1.1.

The main reason for using NRD specifications is that the next "cancelation property" holds for "prefixes" from ${W_\Delta}^+$.

LEMMA 2.23. Let $\nu \in {W_\Delta}^+$ and $\rho, \sigma \in {V_\Delta}^+$. If $\nu \cdot \rho \leftrightarrows_\Delta \nu \cdot \sigma$, then $\rho \leftrightarrows_\Delta \sigma$.

Proof. Because ν is weakly normed deterministic, from Lemma 2.21 follows that a weakly normed deterministic transition sequence from $\nu \cdot \rho$ to ρ exists. A corresponding transition subsequence starts in $\nu \cdot \sigma$ and ends in σ, because each transition is labeled with an unique action (unique with respect to the head process name in the state). By definition of bisimulation we may conclude that $\rho \leftrightarrows_\Delta \sigma$ □

The cancelation property does not hold for arbitrary elements of V_Δ. Take the specification $\Delta = \{X = a \cdot X, Y = a, Z = b\}$. Then $X \cdot Y \leftrightarrows_\Delta X \cdot Z$, but obviously $Y \not\leftrightarrows_\Delta Z$. The next lemma shows that it is not sufficient to strengthen it to normed elements of ${V_\Delta}^+$.

LEMMA 2.24. Let $\nu, \rho, \sigma \in {V_\Delta}^+$ and $\nu \Downarrow_\Delta$. Then it is not necessarily so that $\nu \cdot \rho \leftrightarrows_\Delta \nu \cdot \sigma$ implies that $\rho \leftrightarrows_\Delta \sigma$.

Proof. The counter example is due to the authors of [BCS96]. Let (4) be the specification as defined in Example 1.1. Notice that $Y \cdot Z \leftrightarrows_{(4)} Y \cdot Y \cdot Z$ although $Z \not\leftrightarrows_{(4)} Y \cdot Z$ and $Y \Downarrow_\Delta$ [5] □

3. Regularity Implies an Upper Bound Modulo Bisimulation

In this section we prove that for NRD specifications we can establish an upper bound on the number of non-bisimilar derivatives of a regular process expression. As a corollary this gives that regularity of process expressions over such specifications is decidable.

---

[5] See the picture in Lemma 4.1.

DEFINITION 3.1. Let φ, χ be finite, non terminated transition sequences and Ξ the collection of weakly normed repeats defined by Δ.

1. If $|\varphi| = |\chi|$ and whenever $0 \leq i \leq |\chi|$ $\varphi(i) = \chi(i) \cdot \rho$ for some $\rho \in {V_\Delta}^+$, then $\varphi = \chi \cdot \rho$.
2. If $\varphi(0) \in V_\Delta$, the states of φ are pair-wise non-bisimilar, φ has no transition subsequence in Ξ and no transition subsequence $\chi \cdot \rho$ such that $\chi \in \Xi$ for some $\rho \in {V_\Delta}^+$, then φ is a *possible entry to a weakly normed repeat*, shortly an *entry*.
3. If $X \in V_\Delta$, X starts no weakly normed repeats and for every entry φ starting in X and $\chi \in \Xi$ holds that $last(\varphi) \neq \chi(0)$ and [whenever $last(\varphi) = \chi(0) \cdot \rho$ for some $\rho \in {V_\Delta}^+$, $\chi(0) \cdot \rho \leftrightarrows {}_\Delta last(\chi) \cdot \rho$], then X is *weakly normed repeat invariant over* Δ.

EXAMPLE 3.2. In the specifications in Example 1.1,

1. All process names are weakly normed repeat invariant over (1) and (2), since (1) and (2) define no weakly normed repeats, and
2. X is weakly normed repeat invariant over (3), since X starts no weakly normed repeat and the entry $X \xrightarrow{a}_{(3)} Y \cdot Z$ is invariant for $Y \cdot Z \xrightarrow{a}_{(3)} Y \cdot B \cdot Z$, since $Y \cdot Z \leftrightarrows {}_\Delta Y \cdot B \cdot Z$. For the other entry $X \xrightarrow{a}_{(3)} Y \cdot Z \xrightarrow{c}_{(3)} Z$, Z starts no weakly normed repeat. Y is not weakly normed repeat invariant over (3) since it starts a weakly normed repeat. Z and B are weakly normed repeat invariant over (3) since no sequence corresponding to a weakly normed repeat is reachable, and
3. X is not weakly normed repeat invariant over (4), since the entry $X \xrightarrow{a}_{(4)} T \cdot Z$ is not invariant for $T \cdot Z \xrightarrow{c}_{(4)} T \cdot Y \cdot Z$ since $T \cdot Y \cdot Z \not\leftrightarrows {}_{(4)} T \cdot Z$ and $T \cdot Z \xrightarrow{c}_{(4)} T \cdot Y \cdot Z$ corresponds to the weakly normed repeat $T \xrightarrow{c}_{(4)} T \cdot Y$. The process name T is not weakly normed repeat invariant over (4), since it starts a weakly normed repeat. Y and Z are weakly normed repeat invariant over (4) since no sequence corresponding to a weakly normed repeat is reachable.

We first prove that the following equivalence holds, which gives a relation between regularity and sequences of non-bisimilar states from a process name to a weakly normed repeat.

THEOREM 3.3. Let Δ be a NRD specification and $X \in V_\Delta$. Then X is regular over Δ iff X is weakly normed repeat invariant over Δ.

3.1. X is regular over Δ implies X is weakly normed repeat invariant over Δ. In the first part of the proof we need the Approximation Induction Principle (AIP), which states that if the unfolded graphs of processes are bisimilar down to an arbitrary depth, then the processes are bisimilar. The principle and its proof are described in [BBK87c, BV95][6].

PROPOSITION 3.4. Let $\pi_i : \mathbf{P}^{rec}_{\delta,\Delta} \to \mathbf{P}^{rec}_{\delta,\Delta}$, $i > 0$ be the projection operators defined as follows.

$$\begin{array}{lcll} \pi_1(\alpha \cdot p) & = & \alpha & \\ \pi_i(\alpha) & = & \alpha & \\ \pi_i(X) & = & \pi_i(p), & X = p \in \Delta \\ \pi_{i+1}(\alpha \cdot p) & = & \alpha \cdot \pi_i(p) & \\ \pi_i(p_1 + p_2) & = & \pi_i(p_1) + \pi_i(p_2). & \end{array}$$

where $\alpha \in Act \cup \{\delta\}$ and $p \in \mathbf{P}^{rec}_{\delta,\Delta}$. Then the following statement holds for $p_1, p_2 \in \mathbf{P}^{rec}_{\delta,\Delta}$,

1. (AIP) If $\pi_m(p_1) \leftrightarrows_\Delta \pi_m(p_2)$ for all $m > 0$, then $p_1 \leftrightarrows_\Delta p_2$, and
2. If $p_2 \leftrightarrows_\Delta p_2$, then $\pi_m(p_1) \leftrightarrows_\Delta \pi_m(p_2)$ for all $m > 0$.

LEMMA 3.5. Let Δ be a NRD specification and $Y \in V_\Delta$. If Y is regular over Δ, then Y is weakly normed repeat invariant over Δ.

Proof. Let $\varphi \equiv Y \to_\Delta \ldots$ be an entry. By Lemma 2.12 there is no weakly normed repeat χ such that $\chi(0) = last(\varphi)$.

Suppose $last(\varphi) = \psi(0)$, $\psi = \chi \cdot \rho$ for some $\rho \in {V_\Delta}^+$ and $\chi = X \to_\Delta \ldots X \cdot \nu$ is a weakly normed repeat for $X \in V_\Delta$ and $\nu \in {W_\Delta}^+$. We set out to prove that for all $m > 0$, $\pi_m(X \cdot \rho) \leftrightarrows_\Delta \pi_m(X \cdot \nu \cdot \rho)$, that gives $X \cdot \rho \leftrightarrows X \cdot \nu \cdot \rho$ with AIP.

It is easily established that ψ can be extended with a transition sequence $X \cdot \nu \cdot \rho \to_\Delta \ldots \to_\Delta X \cdot \nu^2 \cdot \rho \ldots$ etc. Y can reach states $\nu^n \cdot \rho$ for every $n > 0$. Because Y is regular over Δ, the pigeon hole principle gives that there are smallest $k > l \geq 1$ such that $\nu^k \cdot \rho \leftrightarrows_\Delta \nu^l \cdot \rho$. Here the (repeated) use of Lemma 2.23 is crucial to our proof. It establishes

[6] In [BBK87c] the proof for BPA is described and in [BV95] the extension to BPA_δ (and other process algebras).

that $\nu^{k-l} \cdot \rho \leftrightarrows_\Delta \rho$ and hence for $i \geq 1$, $\pi_i(X \cdot \nu \cdot \rho) \leftrightarrows_\Delta \pi_i(X \cdot \nu^{i.(k-l)+1} . \rho) \leftrightarrows_\Delta \pi_i(X \cdot \nu^{i.(k-l)} \cdot \rho) \leftrightarrows_\Delta \pi_i(X \cdot \rho)$ □

3.2. X is weakly normed repeat invariant implies X is regular over Δ. We begin by proving that a transition sequence which uses *no* repeat has a finite, bounded length.

LEMMA 3.6. Let Ξ be the collection of all repeats defined by Δ. The length of a transition sequence φ which has no subsequence in Ξ, or subsequence ψ so that $\psi = \chi \cdot \rho$ for some $\rho \in {V_\Delta}^+$ and $\chi \in \Xi$ is maximally $|\varphi(0)|.(2^{|V_\Delta|} - 1)$.

Proof. We prove a slightly stronger result, which immediately implies the desired result. We prove by a well-founded simultaneous induction on $|\chi(0)|$ and $V_\Delta(\chi)$, i.e. the set of process names that occur left-most in states of χ that $|\chi| \leq |\chi(0)|.(2^{|V_\Delta(\chi)|} - 1)$. Now assume the Induction Hypothesis (I.H.) holds for all tuples which are lexicographically smaller than $(|\chi(0)|, V_\Delta(\chi))$. Distinguish the following cases in the Induction Step.

1. If $|\chi(0)| = 1$, then $\chi \equiv X \rightarrow_\Delta \rho \rightarrow_\Delta \ldots$ or $\chi \equiv X \rightarrow_\Delta \surd$ for some $X \in V_\Delta$. We only prove it for the first case, the case that $\chi \equiv X \rightarrow_\Delta \surd$ is clear. Now $X \notin V_{\Delta(\psi)}$, where $\psi \equiv \rho \rightarrow_\Delta \ldots$ because X cannot start a repeat. By definition Δ is in rGNF_δ and so $|\rho| \leq 2$. This implies $|\chi| = 1 + |\psi| \leq 1 + 2.(2^{|V_\Delta(\chi) - \{X\}|} - 1) = |\chi(0)|.(2^{|V_\Delta(\chi)|} - 1)$,
2. If $|\chi(0)| > 1$, then $\chi \equiv X \cdot \rho \rightarrow_\Delta \ldots$, for some $X \in V_\Delta, \rho \in {V_\Delta}^+$. By the I.H. any transition sequence without using repeats $X \rightarrow_\Delta \ldots \rightarrow_\Delta \surd$ has a length not exceeding $2^{|V_\Delta(\chi)|} - 1$ and so does the transition sequence $X \cdot \rho \rightarrow_\Delta \ldots \rightarrow_\Delta \rho$, which does not use the presence of the tail ρ. Also by the I.H. the transition sequence $\rho \rightarrow_\Delta \ldots$ has a maximal length of $|\rho|.(2^{|V_\Delta(\chi)|} - 1)$, which gives a total maximal length of $|\chi(0)|.(2^{|V_\Delta(\chi)|} - 1)$ □

REMARK 3.7. The estimate of the length of a transition sequence of pair-wise non-bisimilar states starting in a process name that uses *no* cycles *cannot* be improved. If we take the specification Δ defined by

$$\begin{array}{rcl} \Delta = \{X & = & a \cdot A \cdot A, \\ A & = & a\} \end{array}$$

then the transition sequence $X \xrightarrow{a}_\Delta A \cdot A \xrightarrow{a}_\Delta A \xrightarrow{a}_\Delta \surd$ uses no cycles. It has the maximal length according to Lemma 3.6.

LEMMA 3.8. If φ is a perpetual repeat, then there is a state ρ in φ with a perpetual process name at the first or second position.

Proof. Let φ be as in the premise. Let N_Δ and P_Δ be the subsets of weakly normed and perpetual process names of V_Δ respectively. By definition $last(\varphi)$ is of the form $X \cdot \rho$ with $X \cdot \rho \not\Downarrow_\Delta$ for some $\rho \in {V_\Delta}^+$. If $X \in P_\Delta$, $\rho \equiv P$ or $\rho \equiv P \cdot \sigma$ for some $P \in P_\Delta$ or $\sigma \in {V_\Delta}^+$ we are finished. Otherwise there is a $P \in P_\Delta$ so that $\rho \equiv \nu \cdot P$ or $\rho \equiv \nu \cdot P \cdot \sigma$ for some $\nu \in {N_\Delta}^+$ and $\sigma \in {V_\Delta}^+$. Let $\varphi(j)$ with $1 \leq j \leq |\varphi|$ be the first state so that P is present in the process name sequence. Suppose the preceding state $\varphi(j-1)$ has the process name $Y \in N_\Delta$ at the first position in the process name sequence. Using the definition of Y in Δ, P is introduced. Because Δ is in rGNF_δ, this implies that the perpetual process name P is introduced at the first or second position
□

LEMMA 3.9. The length of an entry φ is maximally $|V_\Delta|^2.2^{|V_\Delta|}$.

Proof. Let φ be as in the premise. With Lemma 3.6 we know that after $2^{|V_\Delta|}$ transitions φ has used at least one cyclic or perpetual repeat, i.e. φ has a subsequence, which is a cyclic or perpetual repeat or there is a cyclic or perpetual repeat χ and $\sigma \in {V_\Delta}^+$ such that $\chi \cdot \sigma$ is a subsequence of φ. This repeat cannot be cyclic, because φ has then at least two bisimilar states. So φ has used a perpetual repeat in the first $2^{|V_\Delta|}$ transitions. By Lemma 3.8 it has has either passed through a state ρ of the form (1.0) $\rho \equiv P$ or (1.1) $\rho \equiv P \cdot \sigma$ or (2.0) $\rho \equiv Y \cdot P$ or (2.1) $\rho \equiv Y \cdot P \cdot \sigma$, where $Y \in N_\Delta$ and $P \in P_\Delta$. We determine now the size of the "gap" between two states with a perpetual process name at the first or second position: In cases (1.x) in the state ρ' following ρ in φ, there is a perpetual process name at the first or second position. Because P is perpetual the tail σ cannot shift left-most in the transition from ρ to ρ'. Δ is in rGNF_δ and therefore ρ' is of the form (1.0) $\rho' \equiv \nu$ (1.1) $\rho' \equiv \nu \cdot \sigma$ and $1 \leq |\nu| \leq 2$. Every transition from P is to another perpetual state, so ν has *at least* one perpetual process name. But then ρ' has a perpetual process name at the first or second position.

In cases (2.x) φ passes through a state with a perpetual process name at the first or second position in $2^{|V_\Delta|}$ transitions. If φ does not use the presence of the tail of ρ, then this implies with the previous argument that in $2^{|V_\Delta|}$ steps from ρ, φ passes through a state with a perpetual process name at the first or second position. If φ does use the presence of the tail of ρ, then by Lemma 3.6 in $2^{|V_\Delta|}$ transitions X and its reducts have disappeared and is P the left-most process name. So in the gaps between two states with a perpetual process name at the first or second position in φ there are at most $2^{|V_\Delta|}$ connecting transitions.

There are maximally $|P_\Delta|+|N_\Delta|.|P_\Delta| \leq |V_\Delta|^2$ of such non-bisimilar perpetual states in φ, because $P \leftrightarrows {}_\Delta P \cdot \sigma$, $P \cdot \nu \leftrightarrows {}_\Delta P \cdot \sigma$ etc. and $X \cdot P \leftrightarrows {}_\Delta X \cdot P \cdot \sigma$, $X \cdot P \cdot \nu \leftrightarrows {}_\Delta X \cdot P \cdot \sigma$ etc. for $P \in P_\Delta$ and $\nu, \sigma \in {V_\Delta}^+$. Hence the maximal length is the number of different perpetual states with a perpetual process name at the first or second position *multiplied* with the maximal length of a gap, i.e. $|V_\Delta|^2.2^{|V_\Delta|}$ □

REMARK 3.10. In the above lemma, we actually conjecture the length of an entry to have an upper bound of $2^{|V_\Delta|}-1$. The reader can verify this in the two simple cases that either *no* process names are perpetual (Lemma 3.6) or all process names in V_Δ are perpetual. In the last case the maximum number modulo bisimulation is even $|V_\Delta|$ and hence the maximal length $|V_\Delta|-1$.

The previous lemma implies an extension to specifications definable with δ of the main result of [MM94], i.e. regularity of BPA systems[7] is decidable.

COROLLARY 3.11. It is decidable whether a syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification defines only regular process expressions.

Proof. As before it is sufficient to prove the result for rGNF_δ specifications. Notice that by Lemma 3.9, if Δ defines no weakly normed repeats, then the length of a sequence of transitions starting in a process name and passing through pair-wise non-bisimilar states is bounded. Furthermore, if Δ defines a weakly normed repeat, then by Lemma 2.12 there is an element in ${V_\Delta}^+$ which is not regular. Hence the question

[7] See footnote 1.

whether a specification defines only regular process expressions comes down to the question, whether a specification defines a weakly normed repeat. By Lemma 2.13 this is decidable □

EXAMPLE 3.12. In the Introduction we stated that X is regular over the specification (2) in Example 1.1. Notice that (2) does not define any weakly normed repeats and hence X (and Y) are regular over (2), and the same for the other elements of $\{X, Y\}^+$.

LEMMA 3.13. If X is weakly normed repeat invariant over Δ, then X is regular over Δ.

Proof. Assume that X is weakly normed repeat invariant over Δ, and suppose X is not regular over Δ. By Definition 3.1 X does not start a weakly normed repeat. The non regularity implies that there is a transition sequence $\varphi = X \rightarrow_\Delta ...$ passing through *infinitely* many pair-wise non-bisimilar states.

First suppose φ uses no weakly normed repeat, i.e. φ has no subsequence ψ such that ψ is a weakly normed repeat or $\psi = \chi \cdot \rho$ for some weakly normed repeat χ and $\rho \in {V_\Delta}^+$. This implies that φ is an entry according to Definition 3.1. By Lemma 3.9 we conclude that in this case φ passes through a finite number of pair-wise non-bisimilar states (namely at most ${V_\Delta}^2.2^{V_\Delta}$). Therefore φ uses a weakly normed repeat.

Let ψ be the first subsequence corresponding to a weakly normed repeat, i.e. ψ is a weakly normed repeat or $\psi = \chi \cdot \rho$ for some $\rho \in {V_\Delta}^+$ and weakly normed repeat χ. By the assumption that X is weakly normed repeat invariant over Δ $\psi(0)$ starts no weakly normed repeats. This implies that $\psi(0) \leftrightarrows last(\psi)$. Contradiction with the assumption that φ has infinitely many pair-wise non-bisimilar states □

EXAMPLE 3.14. The previous lemma establishes that X is indeed regular over the specification (3) in Example 1.1, since we showed in Example 3.2 that X is weakly repeat invariant over (3).

3.3. Regular Process Expressions over a NRD specification have a Bounded Number of Non-Bisimilar Derivatives. .

DEFINITION 3.15. The branching degree $bf(\Delta)$ of a specification Δ is the maximum number of summands[8] in the right-hand sides of the equations of Δ, i.e. $max(\{i | p \equiv p_1 + ... + p_i, X = p \in \Delta\})$.

[8]Same as footnote 4.

LEMMA 3.16. Let Δ be a NRD specification and $\sigma \in {V_\Delta}^+$ regular over Δ.

1. The maximal length of a transition sequence starting in σ and passing only through pair-wise non-bisimilar states is maximally $|\sigma|.|V_\Delta|^2.2^{|V_\Delta|}$,
2. The size of the state space of σ modulo bisimulation is bounded by $|\sigma|.max(bf(\Delta), 2)^{|V_\Delta|^2.2^{|V_\Delta|}+1}$.

Proof. (1) follows with a straight forward induction on $|\sigma|$ by Theorem 3.5 and Lemma 3.9.

Notice that for (2) the maximal depth of the graph modulo bisimulation of σ is given by (1). This gives rise to a graph with maximally $\sum_{i=1}^{|\sigma|.|V_\Delta|^2.2^{|V_\Delta|}} bf(\Delta)^i$ non-bisimilar states, which is less or equal $max(bf(\Delta), 2)^{(|\sigma|.|V_\Delta|^2.2^{|V_\Delta|})+1}$ □

COROLLARY 3.17. The regularity of process expressions over NRD specifications is decidable.

Proof. By Proposition 3.7 in Chapter 1 we can safely reduce the decidability of regularity to that of a single process name X over a specification Δ in rGNF_δ. It is sufficient to list $max(2, bf(\Delta))^{|V_\Delta|^2.2^{|V_\Delta|}+1}$ non bisimilar derivatives of X, which is possible since bisimulation equivalence of these derivatives is decidable [CHS92, BCS95]: By Lemma 3.16 X is regular over Δ iff the listing fails to establish more than the upper bound non-bisimilar derivatives □

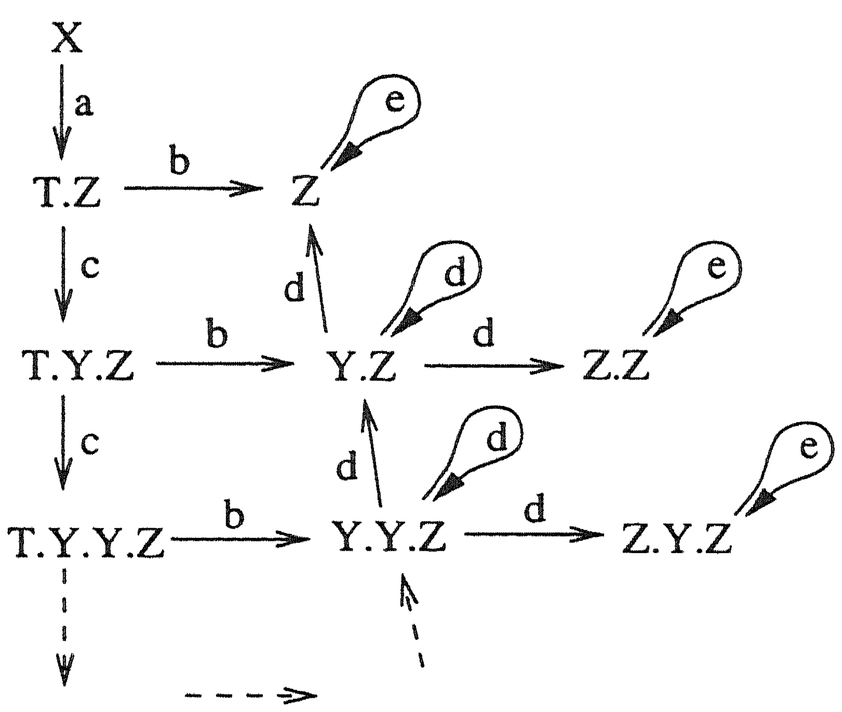
The proof of decidability of regularity over NRD specifications hints at a more than double exponential complexity, since we made it plausible in Section 4 of Chapter 1 that the complexity of deciding bisimulation equivalence of context-free processes definable with δ was exponential.

4. Using Mimicking for Deciding Regularity

In the previous section we proved the decidability of regularity of process expressions over NRD specifications. This fails to prove however the decidability of regularity of all process expressions over arbitrary syntactically guarded specifications. We present a counter example first and then proceed with the decidability proof for the full class.

LEMMA 4.1. There exists a regular process expression over a syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specification in rGNF_δ, which is not a NRD specification.

Proof. The process name X over the specification (4) of Example 1.1 is a subtle counter example found by the authors of [BCS96]. Example 2.16 shows that (4) is not NRD.



Notice that X is regular modulo bisimulation over (4) as the picture above shows □

For the decidability proof for the full class we use the notions of mimicking as developed in Section 4 of Chapter 1. Let $a_\delta \in Act - A_\Delta$ and $P_\delta \in \mathrm{Var} - V_\Delta$, and $\mu_{\Delta,a_\delta,P_\delta} : \mathbf{P}^{rec}_{\delta,\Delta} \to \mathbf{P}^{rec}_{\mathrm{Var}}$, $\mu_{\Delta,a_\delta,P_\delta} : \wp_{fin}(\mathrm{Var} \times \mathbf{P}^{rec}_{\delta,\Delta}) \to \wp_{fin}(\mathrm{Var} \times \mathbf{P}^{rec}_{\mathrm{Var}})$ be the mimicking maps for respectively process expressions and specifications. As before we abbreviate $\mu_{\Delta,a_\delta,P_\delta}$ to μ for readability.

A crucial ingredient is the result in [BCS96] that regularity of process expressions over syntactically guarded $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specifications (thus without δ !) is decidable.

THEOREM 4.2. Let E be a guarded $\Sigma(\mathrm{BPA}(Act, \mathrm{Var}))$ specification. Then regularity over E of process expressions in $\mathbf{P}_E^{rec}$ is decidable.

LEMMA 4.3. Let $\sigma \in {V_\Delta}^+$. Then σ is regular over Δ iff $\mu(\sigma)$ is regular over $\mu(\Delta)$.

Proof. ($\Rightarrow$) Assume σ is regular over Δ. Let $\varphi \equiv \varphi(0) \rightarrow_\Delta \varphi(1)...$ be a (possibly infinite) transition sequence. By assumption there is only a finite number of non-bisimilar $\varphi(i)$. Let $\mu(\varphi)$ be the transition sequence $\mu(\varphi(0)) \rightarrow_{\mu(\Delta)} \rho_1....$ By Lemma 4.7 of Chapter 1 $\mu(\varphi(0)) \rightarrow_{\mu(\Delta)} \mu(\varphi(1))...$. By Lemma 4.8 of Chapter 1 there is also a finite number of non-bisimilar $\mu(\varphi(i))$. Hence $\mu(\sigma)$ is regular over $\mu(\Delta)$.

($\Leftarrow$) Assume $\mu(\sigma)$ is regular over $\mu(\Delta)$. Let $\mu(\varphi) \equiv \mu(\varphi(0)) \rightarrow_{\mu(\Delta)} \rho_1...$ be a (possibly infinite) transition sequence. By assumption there is only a finite number of non-bisimilar ρ_i and by Lemma 4.7 of Chapter 1 $\varphi(0) \rightarrow_\Delta \varphi(1)....$ By Lemma 4.9 of Chapter 1 there is also a finite number of non-bisimilar $\varphi(i)$. Hence σ is regular over Δ □

COROLLARY 4.4. Regularity of process expressions over syntactically guarded $\Sigma(\mathrm{BPA}_\delta(Act, \mathrm{Var}))$ specifications is decidable.

Proof. Immediate by Lemma 4.5 of Chapter 1, Lemma 4.3 and and Theorem 4.2 □

In [BCS96] no estimate is given of the complexity of deciding regularity of context-free processes. Since the complexity of regularity for all context-free processes definable with δ depends crucially on this result, we cannot give an estimate as we did for deciding of regularity of process expressions over NRD specifications.

Acknowledgements. We thank Olaf Burkhart, Didier Caucal, Bas Luttik, Faron Moller, Bernhard Steffen, Colin Stirling and Frits Vaandrager for comments on earlier versions of this chapter.

CHAPTER 3

Regular Process Graphs and Regular Expressions

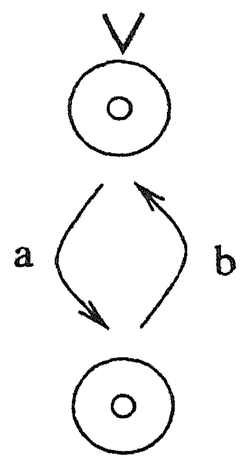
In this chapter we investigate the relation between finite automata and regular expressions in the setting of process theory. Kleene found the now well-known result that in formal language theory the languages generated by regular expressions coincide with languages accepted by finite automata [Kle56]. It was shown by Robin Milner [Mil84] that this result does not hold in the setting of process theory: there are very simple process graphs for which no regular expression exists with a bisimilar graph interpretation. This poses the question as to which criterion completely describes expressible process graphs. In this chapter we define an effective criterion which we prove identifies those regular process graphs which are expressible with regular expressions with and without the constant ϵ. Further we present two algorithms which compute the (smallest) regular expression expressing a process graph, *if it exists*.

1. Introduction

Recent years have seen a renewed interest in the Kleene star operation $\cdot^*\cdot$ [Mil84, BBP94, Fok96]. The famous American mathematician Kleene described regular expressions in the setting of formal language theory using this operator in 1956 [Kle56]. It was introduced in a *binary* form [1] to describe iteration of regular expressions, i.e. concatenation of an arbitrary number of words generated by regular expressions, one of the basic concepts for forming infinite languages. A wealth of results exist on regular expressions and the regular languages generated by them [HU79, LP79]. A classical result from formal language theory is that for every regular expression a finite automaton can be constructed which accepts the language generated by it and vice versa [Kle56, MY60].

[1] In fact in [CEW58] the more frequently seen *unary* form was introduced. We use the original notion as defined by Kleene.

EXAMPLE 1.1. The regular expression $a+(ab)^*\epsilon+(ab)^*a$ generates the language $\{\epsilon, a, ab, aba, ...\}$ which is accepted by the finite automaton depicted below.



(1)

Here is the only time that we take it as the picture of a finite automaton in the style of [LP79]: the small circles ∘ denote *states* of the finite automaton. A triple $\circ \xrightarrow{a} \circ$ denotes a *transition* from a state to another state with an element of the automaton's alphabet. A bigger circle ◯ around a small circle denotes that the state is a *terminal state*; a wedge ⋁ on the small or big circle of a state denotes that it is a *begin state* too.

In the following will use these pictures to denote *process graphs*. The difference with finite automata is very small: a wedge denotes the *root* of a process graph and a big circle around a small circle now denotes that a state has a *termination label*. An element of the alphabet is interpreted as an *action*. The rest stays the same. Of course only the reachable states are depicted, i.e. the states which can be reached with a sequence of contagious transitions from the root.

In process theory the Kleene star operation has also been studied [Mil84, BBP94, FZ94, Fok97, AFIb, AFIa]. Here the semantics is no longer a concatenation of words, but a (sequential) composition of processes. The key axiom is the same as the one already formulated by Kleene [Kle56] for formal languages

$$\text{(BKS1)} \quad x^*y = x \cdot x^*y + y$$

but now captures a different meaning: the process expression x^*y can perform x *or* y, and upon completion of x has this choice again and upon completion of y terminates.

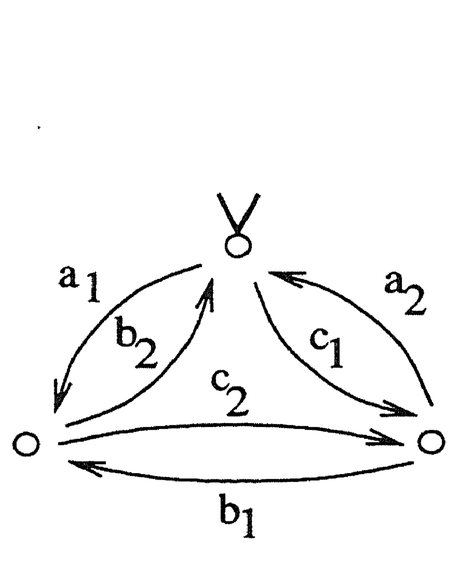
Regular process expressions have the same syntax as regular expressions, but the operators have a different meaning: the alphabet becomes the set of actions, concatenation $(\cdot)$ is sequential composition of processes, union $(+)$ is a non-deterministic choice between processes and the empty word e or λ is interpreted as the terminated process ϵ.

Several intriguing questions about regular expressions in process theory are still open. Only recently it has been proved by [FZ94] that the axiomatization proposed by [BBP94] consisting of (BKS1) and

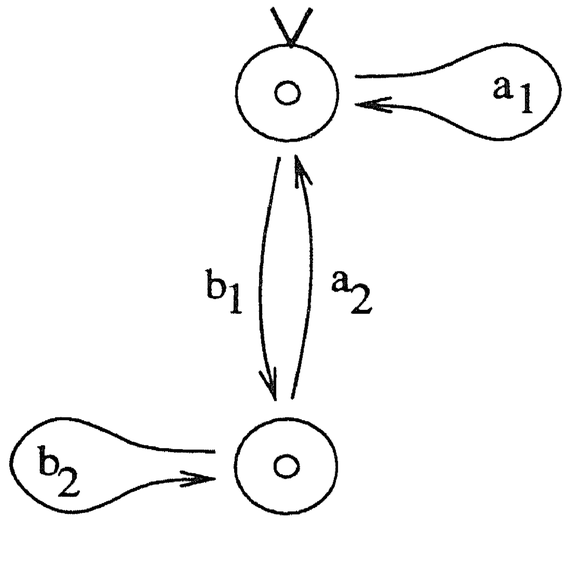
$$\begin{array}{ll} \text{(BKS2)} & x^*(y \cdot z) = (x^*y) \cdot z \\ \text{(BKS3)} & x^*(y \cdot ((x+y)^*z) + z) = (x+y)^*z \end{array}$$

is complete for bisimulation equivalence of process expressions over the signature of Basic Process Algebra (BPA) with iteration, i.e. the set of the process expressions which can be built up from actions by means of sequential composition, choice and Kleene star. For process expressions with the special constants ϵ and δ (the deadlocked process) in ACP [BK84c, BW90] the question is still open.

EXAMPLE 1.2. The process graphs (1) and (2) are derived from [Mil84], where



(1)



(2)

it is *proved* of (1) and *conjectured* of (2) that there is no process expression over the signature of BPA with iteration and ϵ which defines a bisimilar process graph.

In a slightly different setting [Mil84] [2], it was posed as an open problem which regular process graphs can be expressed as regular expressions in the context of bisimulation. As was noticed by several authors [Mil84, BBP94] the class of regular process graphs does not coincide with the class of regular process expressions. There are some very simple regular graphs which are known to be *in*expressible, e.g. the process graph (1) in Example 1.2, or are seriously suspected to be inexpressible, e.g. the one of Example 1.1[3] [Pon94] and (2) in Example 1.2.

In this chapter we present an effective criterion that shows that expressibility is decidable. We give a general method to prove expressibility, which shows in a clear cut way that all of the presented process graphs here so far are inexpressible.

The question as to which regular process graphs can be expressed is in the first place of theoretical interest. Expressing process graphs by means of regular expressions has some advantages over expressing them by means of process expressions over recursive specifications. For instance, the inductive structure of process expressions can be exploited and there is no need for complicated restrictions -such as guardedness- to obtain meaningful specifications. It is evident that the operational semantics of the Kleene star is much than that for a recursive mechanism.

Expressibility is also of practical interest if we see it at as the problem of which recursive programs can be stated in a WHILE language. It is claimed by some that for iteration much simpler implementations are sufficient. Recently a general program to structure process behavior has been developed, with a script language based on iteration [BK94] to obtain a simple implementation and transparent behavior.

The plan of this chapter is as follows. In Section 2 we describe regular expressions formally as process expressions over the signature of BPA, Kleene star and ϵ, process graphs and the graph interpretations of

---

[2] In [Mil84] *finite charts* are compared with process expressions over a CCS like language. Furthermore the *unary* Kleene star is used, instead of the binary form used in this thesis. However in the presence of ϵ this difference is non-essential, since the unary Kleene star can be expressed in the binary form and the other way round. See also Section 6.

[3] i.e. if we see the picture as a *process graph*.

regular expressions. Throughout this chapter we use the non standard notion of a *rum* as defined in Section 2.2, i.e. the root unwinding of a minimization modulo bisimulation equivalence of a process graph, which allows for an easy formulation of graph operations and proofs.

In Section 3 we define the $\star$ *property*, the criterion on process graphs which we prove in detail to correspond with the criterion that a process graph can be expressed with a regular expression. In Section 4 we repeat the same exercise for the $\star_\epsilon$ *property*, which we prove to correspond with the criterion that a process graph can be expressed with a regular expression definable with ϵ. The $\star$ and $\star_\epsilon$ property are formulated in an inductive way on the rum's of process graphs. They can be seen as recipes for constructing a regular expression expressing a process graph modulo bisimulation. For their definition we developed the (new) graph operations ϵ-unknotting and ϵ-exit and ϵ-iterative part, that are interesting in their own right.

In Section 5 we show that a process graph which has the $\star$ or $\star_\epsilon$ property can be expressed with a regular expression with a *lower bounded* and *upper bounded* size determined by the process graph. Since the $\star$ or $\star_\epsilon$ property corresponds to the property that process graphs can be expressed with a regular expression, the upper bound result proves that the $\star$ and $\star_\epsilon$ property *and* expressibility with a regular expression with and without ϵ are decidable.

In Section 6 we use the upper bound result to formulate an algorithm which computes the *smallest* regular expression with and without ϵ expressing a process graph. A second algorithm using the the inductive structure of the $\star$ and $\star_\epsilon$ is suspected to have a better complexity.

Second we relate the results of this chapter to Milner's open question and ponder on the question of expressibility of regular expressions with other constants and operators. We end this chapter with a (partial) criterion for *in*expressibility of process graphs, which is more descriptive than the exit property of [BBP94, Bos95] and *simpler* than the $\star$ and $\star_\epsilon$ property.

2. Regular Expressions, Process Graphs and Operations on Process Graphs

In this section we define the basic concepts that we need in this chapter. In Section 2.1 we define formally what we mean by regular expressions by presenting the operational semantics of Basic Process

Algebra with the special constant ϵ. In Section 2.2 we define regular process graphs and rum's formally. In Section 2.3 we give the interpretation of process expressions as process graphs.

2.1. An Operational Semantics for Regular Expressions.

In the sequel we will assume a finite set Act of actions with typical elements a, b, c, d, e with the usual annotations. Process expressions over the signature of BPA with δ, ϵ and $*$ [4], $\mathbf{P}^*_{\delta,\epsilon}$ for short, are given by the abstract syntax

$$p ::= a|\delta|\epsilon \mid p + p \mid p \cdot p \mid p^* p$$

where $a \in Act$. We use that * binds stronger than $\cdot$, which in turn binds stronger than $+$, e.g. $c^*a \cdot b + c$ is the same as $((c^*a) \cdot b) + c$. In this chapter we use the full set $\mathbf{P}^*_{\delta,\epsilon}$ only to prove some auxiliary results[5]; our main interest goes to its subsets $\mathbf{P}^*_\epsilon$ -process expressions over $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ (without using δ)- and $\mathbf{P}^*$ -without δ and ϵ. Typical elements of $\mathbf{P}^*_{\delta,\epsilon}$ and the other subsets are p, q.

The operational semantics of process expressions over $\Sigma(\mathrm{BPA}^*_{\delta,\epsilon}(Act))$ is given by the unary termination predicate $\cdot\downarrow \subseteq \mathbf{P}^*_{\delta,\epsilon}$ and the ternary transition relation $\cdot \xrightarrow{\cdot} \cdot \subseteq \mathbf{P}^*_{\delta,\epsilon} \times Act \times \mathbf{P}^*_{\delta,\epsilon}$ satisfying the following rules.

$$\frac{p\downarrow}{(p+q)\downarrow} \qquad \frac{p\downarrow}{(q+p)\downarrow} \qquad \frac{p\downarrow, q\downarrow}{(p\cdot q)\downarrow} \qquad \frac{q\downarrow}{(p^*q)\downarrow} \qquad \frac{}{\epsilon\downarrow}$$

$$\frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{p \xrightarrow{a} p'}{q+p \xrightarrow{a} p'}$$

$$\frac{p \xrightarrow{a} p'}{p\cdot q \xrightarrow{a} p'\cdot q} \qquad \frac{p\downarrow \;\; q \xrightarrow{a} q'}{p\cdot q \xrightarrow{a} q'} \qquad \frac{}{a \xrightarrow{a} \epsilon}$$

$$\frac{p \xrightarrow{a} p'}{p^*q \xrightarrow{a} p'\cdot p^*q} \qquad \frac{q \xrightarrow{a} q'}{p^*q \xrightarrow{a} q'}$$

where $a \in Act$.

Intuitively $p\downarrow$ states that p has the option to terminate immediately. $p \xrightarrow{a} q$ states that p can evolve into q by performing an a action.

[4] Also known as $\mathrm{BPA}^*_{\delta,\epsilon}(Act)$.

[5] We use δ to prove that we can replace an impure iterator safely wrt. bisimulation with a pure iterator, see Lemma 4.47 and Lemma 4.48.

We use shortly $p \rightarrow^* q$ and $p \rightarrow^+ q$ to denote respectively that a p can go in sequence of (zero or) more than zero transitions to q.

DEFINITION 2.1. Let $p, q \in \mathbf{P}^*_{\delta,\epsilon}$.

1. Let $R \subseteq \mathbf{P}^*_{\delta,\epsilon} \times \mathbf{P}^*_{\delta,\epsilon}$ be a relation. If $(p,q) \in R$ iff
   (a) $p \downarrow$ iff $q \downarrow$, and
   (b) if $p \xrightarrow{a} p'$ for some $a \in Act$, there is a $q' \in \mathbf{P}^*_{\delta,\epsilon}$ such that $q \xrightarrow{a} q'$ and $(p', q') \in R$, and
   (c) if $q \xrightarrow{a} q'$ for some $a \in Act$, there is a $p' \in \mathbf{P}^*_{\delta,\epsilon}$ such that $p \xrightarrow{a} p'$ and $(p', q') \in R$,

   then R is *a bisimulation.*
2. If there is a bisimulation R such that $(p,q) \in R$, then p and q are *bisimilar*, shortly $p \leftrightarrows q$.

LEMMA 2.2. Let $p, q \in \mathbf{P}^*_{\delta,\epsilon}$. Then $p \leftrightarrows q$ is decidable.
Proof. Folklore e.g. [Mil89, BW90] □

DEFINITION 2.3. Let $p, q \in \mathbf{P}^*_{\delta,\epsilon}$.

1. q is a *derivative of* p iff $p \rightarrow^* q$,
2. q is a *proper* derivative of p iff $p \rightarrow^+ q$,
3. q is a *child of* p iff there is an $a \in Act$ such that $p \xrightarrow{a} q$,
4. p is *terminating* iff p has a derivative q such that $q \downarrow$,
5. p is *strongly normed* iff every derivative of p is terminating.

DEFINITION 2.4. Let $p, q \in \mathbf{P}^*_{\delta,\epsilon}$.

1. p is *returning via* q iff q is a proper derivative of p and p is bisimilar with a derivative of q,
2. p is *returning* if p is returning via some of its children.

EXAMPLE 2.5. Let $p \equiv (a \cdot (b + \epsilon))^*(c + \epsilon)$. Using the operational semantics of $\mathrm{BPA}^*_{\delta,\epsilon}(Act)$ we establish that the derivatives [6] of p are

1. $(a \cdot (b + \epsilon))^*(c + \epsilon)$,
2. $\epsilon \cdot (b + \epsilon) \cdot ((a \cdot (b + \epsilon))^*(c + \epsilon))$,
3. $\epsilon \cdot ((a \cdot (b + \epsilon))^*(c + \epsilon))$, and
4. ϵ.

[6] we only print the significant brackets, i.e. we work modulo associativity of $\cdot$ and identify $a \cdot (b \cdot c)$, $(a \cdot b) \cdot c$, and $a \cdot b \cdot c$

All derivatives of p, except p itself are the proper derivatives. The children are $\epsilon \cdot (b + \epsilon) \cdot (a \cdot (b + \epsilon))^* c$ and ϵ. All derivatives of p are terminating, since ϵ is a derivative of each of them, hence p is strongly normed. Moreover p is returning via every proper derivative except ϵ.

PROPOSITION 2.6. Let $p \in \mathbf{P}^*$. If $p \rightarrow^+ p'$ and $p' \not\leftrightarrows \epsilon$, then there is a $p'' \in \mathbf{P}^*$ such that $p'' \leftrightarrows p'$.

PROPOSITION 2.7. Let $p \in \mathbf{P}^*$. Then $p \not\downarrow$.

PROPOSITION 2.8. If $p \in \mathbf{P}^*_\epsilon$, then p is strongly normed.

PROPOSITION 2.9. Let $p_1{}^* p_2 \in \mathbf{P}^*_{\delta,\epsilon}$. If $p_1{}^* p_2$ is strongly normed, then $p_1 \leftrightarrows \delta$ or p_1 is strongly normed.

2.2. Process Graphs. In this section we describe process graphs. We see every process graph as built up from *states* from a *fixed* countable, infinite state space S, which is a superset of $\mathbf{P}^*_{\delta,\epsilon}$. As usual, one special state is distinguished, the *root state* or *root* of the process graph.

The *transitions* of a process graph are a ternary transition set $\cdot \stackrel{\cdot}{\rightarrow} \cdot \subseteq S \times Act \times S$ of triples which denote that left-hand side states go to right-hand states with an action label taken from Act. Furthermore, the set $(\rightarrow)$ of transitions respects *reachability*: if $s \xrightarrow{a} s'$, then there is a sequence of contagious transitions from $\rightarrow$ such that $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s$, where s_0 is the *root*.

Moreover on the set of reachable states of a process graph a unary termination relation $\cdot \downarrow \subseteq S$ is defined which denotes that a state has a termination label or can terminate immediately. We allow states with a termination label to start transitions as well, thus we allow *intermediate termination*, the choice to terminate *or* to perform a transition.

For states we use typical elements s, t. Like for process expressions we use shortly $s \rightarrow^* t$ (, $s \rightarrow^+ t$) to denote that a state s can go with zero or more (or more than zero, respectively) transitions to t. With $|\rightarrow|$ and $|\downarrow|$ we mean respectively the number of transitions and termination labels.

DEFINITION 2.10. A *process graph* over Act is a quadruple $(r, S, \downarrow, \rightarrow)$, where $r \in S$ is the *root*, $S \supset \mathbf{P}^*_{\delta,\epsilon}$ is the *state space*, $\cdot \downarrow \subseteq S$ is the set of *termination labels*, and $\cdot \stackrel{\cdot}{\rightarrow} \cdot \subseteq S \times Act \times S$ the set of *transitions*, such that

1. if $s \xrightarrow{a} t$, then $r \rightarrow^* s$,

2. if $t\downarrow$, then $r \rightarrow^* t$.

We only consider *finite* process graphs, i.e. process graphs of which the transition set is finite.

REMARK 2.11. The definition of process graphs is chosen to simplify proofs, notably we shall use that the states of all process graphs are derived from the same fixed state space S. Notice that we thus allow states from $\mathbf{P}^*_{\delta,\epsilon}$, but that we have detached the meaning of the states from the operational semantics as defined earlier. E.g. the process graph with root state $a + b$, can very well be trivial, or have a transition with a c action.

However, unless explicitly stated we use elements from $S - \mathbf{P}^*_{\delta,\epsilon}$ as reachable states for process graphs. Furthermore if we do use reachable states from $\mathbf{P}^*_{\delta,\epsilon}$, the concerning transitions are faithful to the operational semantics as defined in Section 2.1.

We use letters g and h to denote process graphs.

DEFINITION 2.12. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite process graph. R_g is the *set of reachable states of* g, i.e. $s \in R_g$ iff $r \rightarrow^* s$.

DEFINITION 2.13. Let $g = (r, S, \downarrow, \rightarrow)$ and $g_1 = (r_1, S, \downarrow_1, \rightarrow_1)$ be finite process graphs. Then

1. g is *trivial* iff the transition relation $\rightarrow$ is empty,
2. g_1 is a *derivative of* g iff $\downarrow_1 \subseteq \downarrow$ and $\rightarrow_1 \subseteq \rightarrow$ are the smallest sets so that
   (a) if $r_1 \rightarrow^* t$ and $t\downarrow$, then $t\downarrow_1$,
   (b) if $r_1 \rightarrow^* s \xrightarrow{a} s'$, then $s \xrightarrow{a}_1 s'$.
3. g_1 is a *proper* derivative of g iff g_1 is a derivative of g and $r_1 \rightarrow^+ r_1$,
4. g_1 is a *child of* g iff g_1 is a proper derivative and there is an $a \in Act$ such that $r \xrightarrow{a} r_1$,
5. g is *terminating* iff g has a derivative g_1 with $r\downarrow_1$,
6. g is *strongly normed* iff every derivative of g is terminating,
7. g is a *terminated graph*, shortly *terminated* iff g is trivial and terminating.

DEFINITION 2.14. Let $g_1 = (r_1, S, \downarrow_1, \rightarrow_1)$, $g_2 = (r_2, S, \downarrow_2, \rightarrow_2)$ be process graphs.

1. Let $R \subseteq R_{g_1} \times R_{g_2}$ be a relation. If for all $(s, t) \in R$,
   (a) $s\downarrow_1$ iff $t\downarrow_2$, and

(b) if $s \xrightarrow{a}_1 s'$ for some $a \in Act$, $s' \in S$, there is a $t' \in S$ such that $t \xrightarrow{a}_2 t'$ and $(s', t') \in R$, and
(c) if $t \xrightarrow{a}_2 t'$ for some $a \in Act$, $t' \in S$, there is a $s' \in S$ such that $s \xrightarrow{a}_1 s'$ and $(s', t') \in R$,

then R is a *bisimulation*.

2. If there is a bisimulation R such that $(r_1, r_2) \in R$, then g_1 and g_2 are *bisimilar*, shortly $g_1 \leftrightarrows g_2$.

LEMMA 2.15. Bisimulation equivalence is decidable for finite process graphs.

Proof. Folklore e.g. [Mil89],[BW90] □

For finite process graphs it is meaningful to speak of the minimal number of states and transitions of the graph modulo bisimulation.

DEFINITION 2.16. Let g and h be finite process graphs such that g and h are bisimilar and there is no process graph h' with less states than h and bisimilar to it. Then h is a *minimization modulo bisimulation of* g, or shortly a *minimization of* g. Furthermore g is a *minimization*, if it is a minimization of itself.

To define the interpretations of the binary operators in the next section we need the standard notion of root unwinding as defined e.g. in [BW90]. Intuitively the root unwinding of a process graph is a graph bisimilar to the original process graph, possibly with an extra state which is a copy of the root, without incoming transitions.

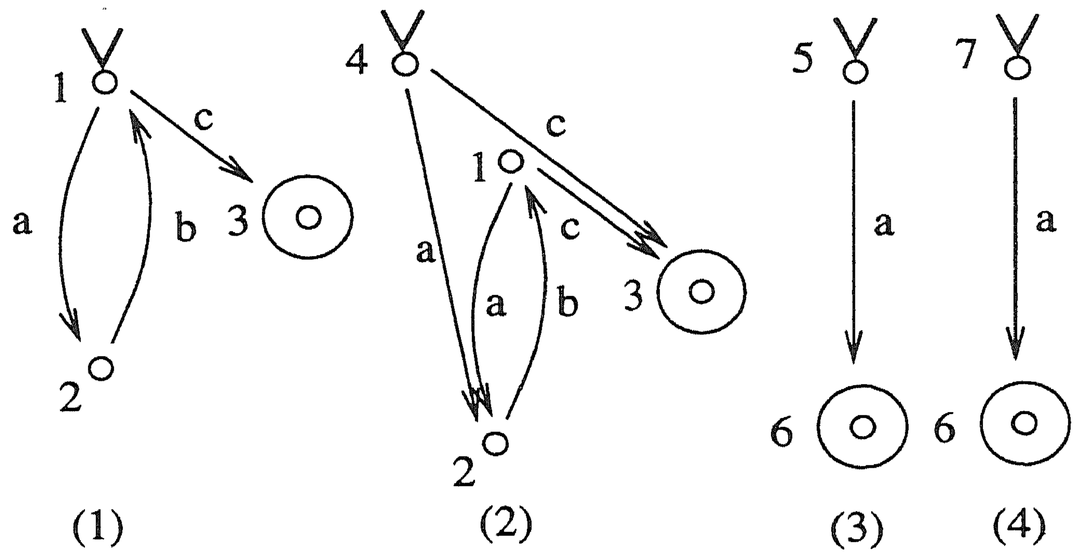
DEFINITION 2.17. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite process graph. If $g_{ru} = (r_{ru}, S, \downarrow_{ru}, \rightarrow_{ru})$ is a finite process graph, where $r_{ru} \in S - R_g$ is some unreachable state of g and $\downarrow_{ru}$ and $\rightarrow_{ru}$ are the smallest sets such that

1. If $r \downarrow$, then $r_{ru} \downarrow_{ru}$, and
2. If $r \xrightarrow{a} s$, then $r_{ru} \xrightarrow{a}_{ru} s$, and
3. If $r_{ru} \rightarrow_{ru}{}^{+} s$ and $s \xrightarrow{a} s'$, then $s \xrightarrow{a}_{ru} s'$, and
4. If $r_{ru} \rightarrow_{ru}{}^{+} s$ and $s \downarrow$, then $s \downarrow_{ru}$, and

then g_{ru} is *a root unwinding with* r_{ru} *of* g, shortly *a root unwinding of* g.

EXAMPLE 2.18. In the picture below the process graph depicted by (2) is a root unwinding with 4 of (1); (4) is a root unwinding with

7 of (3).



Notice that (1) and (2) are bisimilar, and so are (3) and (4).

PROPOSITION 2.19. A finite process graph is bisimilar with a root unwinding of itself.

DEFINITION 2.20. Let $g_1 = (r_1, S, \downarrow_1, \rightarrow_1)$ and $g_2 = (r_2, S, \downarrow_2, \rightarrow_2)$ be finite process graphs. If there exists a bijective map $\varphi : R_{g_1} \rightarrow R_{g_2}$ such that

1. $\varphi(r_1) = r_2$, and
2. $s \downarrow_1$ iff $\varphi(s) \downarrow_2$, and
3. $s \xrightarrow{a}_1 s'$ iff $\varphi(s) \xrightarrow{a}_2 \varphi(s')$,

then φ is *an isomorphism from* g_1 *to* g_2. If there is an isomorphism from g_1 to g_2, then g_1 and g_2 are *isomorphic*, shortly $g_1 \cong g_2$.

PROPOSITION 2.21. Let g be a finite process graph. If g_1 and g_2 are root unwindings of g, then $g_1 \cong g_2$.

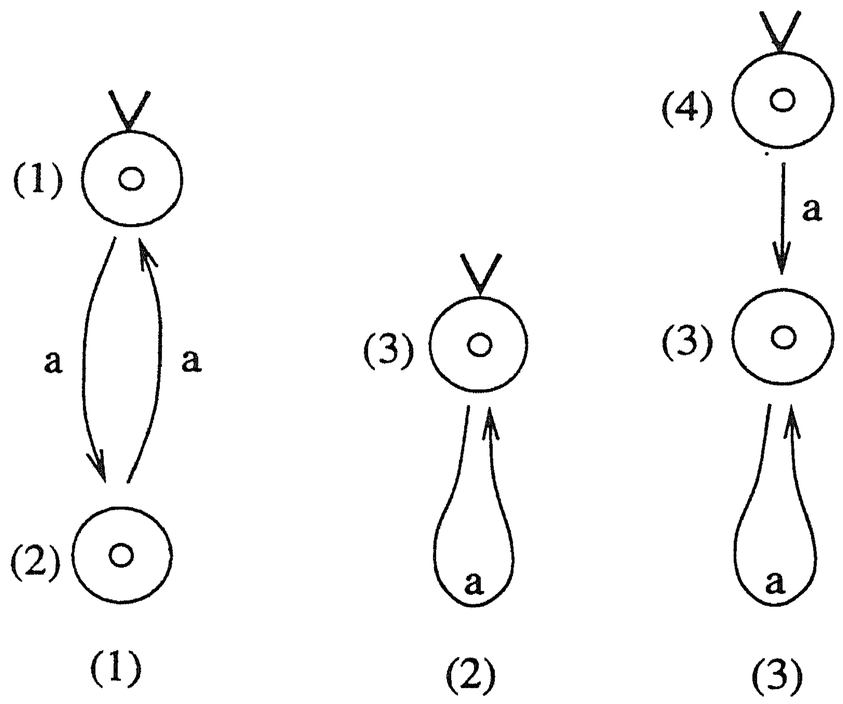
DEFINITION 2.22. A process graph is *root unwound invariant*, shortly *rui* iff it is isomorphic to a root unwinding of itself.

EXAMPLE 2.23. In Example 2.18 (1) is not a rui process graph, since it is bisimilar but not isomorphic with a root unwinding (2) with state 4. Conversely (2) is rui process graph, for it is isomorphic with a root unwinding of itself with state 7.

For technical reasons we consider mostly rui process graphs, i.e. we define all relations and operations primarily on rui process graphs. This allows a pleasant formulation of e.g. the interpretation of the binary operators from the syntax of $\mathrm{BPA}^*_{\delta,\epsilon}(Act)$ on on process graphs.

DEFINITION 2.24. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rui process graph and $g' = (r', S, \downarrow', \rightarrow')$ a minimization of g. If h is a root unwinding with some $u \in S - (R_{g'} \cup R_g)$ of g', then h is a root unwound minimization with u, or shortly a *root unwound minimization* or *rum*.

EXAMPLE 2.25. In the picture below, (2) is a minimization of (1) and (3) and is a root unwinding of (2) and a rum of (1).



PROPOSITION 2.26. Let g be a finite rui process graph. If h and h' are rum's of g, then h and h' are isomorphic.

LEMMA 2.27. Let g be a finite rum. If g_d is a root unwinding of a derivative of g, then g is a rum.

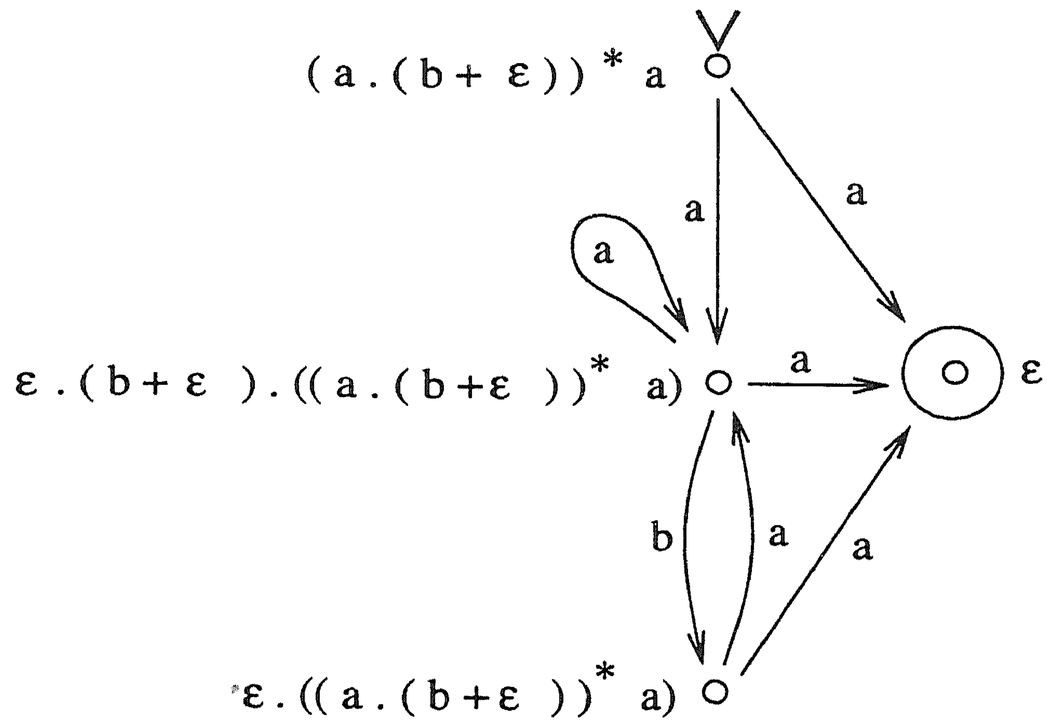
Proof. By definition of a rum, any derivative $g' \neq g$ is a minimization. Hence a root unwinding of g' is a rum □

2.3. Graph Interpretations of Regular Expressions. In the previous two sections we defined the notions of process expressions and graphs. In this section we will develop some relations between them.

The main relation between process expressions and process graphs is given by the following interpretation.

DEFINITION 2.28. The process graph interpretation $\gamma(p)$ of a process expression $p \in \mathbf{P}^*_{\delta,\epsilon}$ is defined as the process graph $(p, S, \downarrow_p, \rightarrow_p)$, where $\downarrow_p$ and $\rightarrow_p$ are the smallest set of terminations and transitions provable by the operational semantics of p as defined in Section 2.1, but restricted to the reachable states of p.

EXAMPLE 2.29. The graph interpretation of the process expression $(a \cdot (b + \epsilon))^* a$ is depicted below.



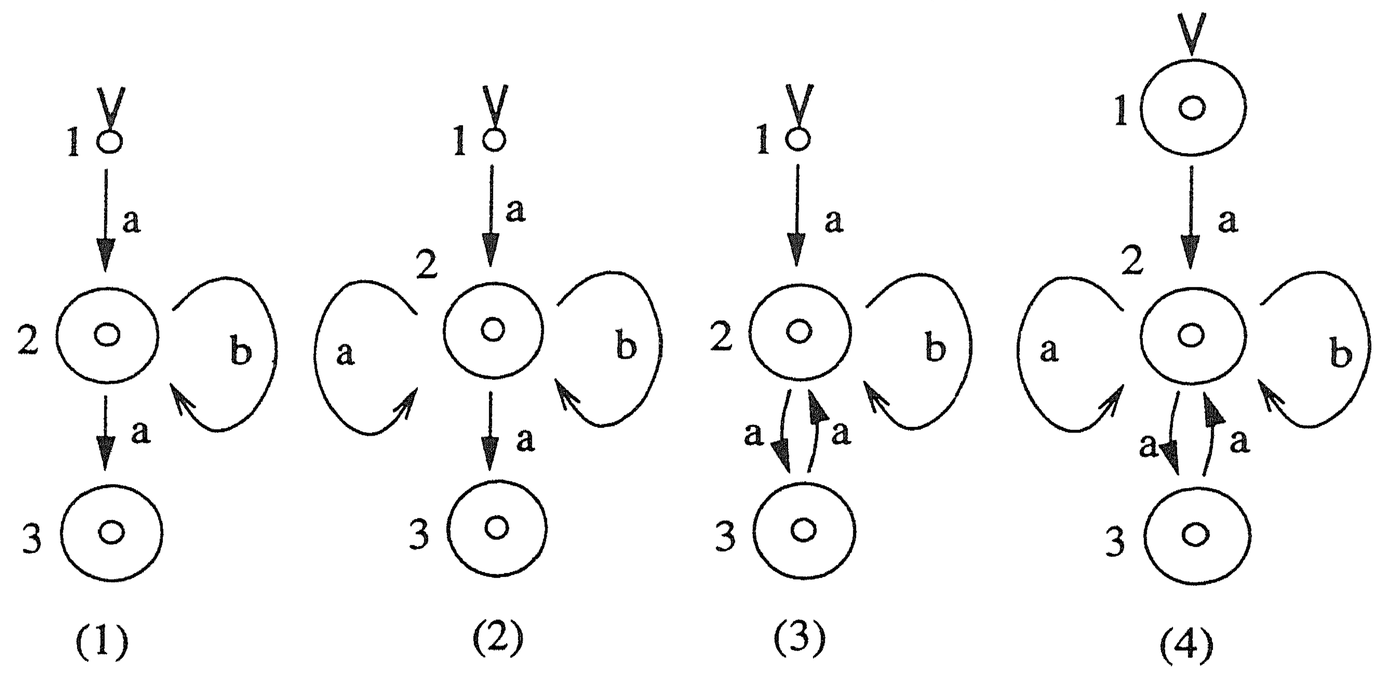
Notice that we do not depict any unreachable states of the state space S.

In the above definition, the graph interpretation of process expressions is derived directly from the operational semantics. Another way is to give an interpretation of process expressions inductively with an interpretation of the operators from the signature.

For the interpretation of the binary Kleene star $\cdot^*\cdot$ we use an operation, to which we subsequently refer as "knotting" and which can be thought of as the graph interpretation of the unary Kleene star $(\cdot^*\epsilon)$.

DEFINITION 2.30. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rui process graph. If $g_k = (r, S, \downarrow \cup \{r\}, \rightarrow_k)$ is a process graph where $\rightarrow_k \supseteq \rightarrow$ is the smallest set such that [if $s \downarrow$ and $r \xrightarrow{a} t$ for some $a \in Act$, $t \in S$, then $s \xrightarrow{a}_k t$], then g_k is *a knotting of* g.

EXAMPLE 2.31. In the picture below, the process graph depicted by (4) is a knotting of the process graphs (1)–(4).



Notice that (1),(2) and (4) are the rum's of graph interpretations of respectively $a \cdot (b^*(\epsilon + a))$, $a \cdot (a + b)^*(\epsilon + a)$ and $(a \cdot (b^*(\epsilon + a)))^*\epsilon$; (3) is *in*expressible with an element of $\mathbf{P}^*_\epsilon$.

REMARK 2.32. Notice that the knotting of a finite rui process graph is an idempotent operation, as (4) in Example 2.31 shows.

In the sequel we speak of *the* knotting.

PROPOSITION 2.33. The knotting of a finite rui process graph is unique.

LEMMA 2.34. Let g be a finite rui process graph. If g is rui, then the knotting of g is a finite rui process graph.

Proof. Knotting adds no root incoming transitions □

DEFINITION 2.35. Let $g_1 = (r_1, S, \downarrow_1, \rightarrow_1), g_2 = (r_2, S, \downarrow_2, \rightarrow_2)$ be finite rui process graphs such that $R_{g_1} \cap R_{g_2} = \emptyset$. The process graph interpretation of

1. $g_1 + g_2$ is defined as $(r_1, S, \downarrow_+, \rightarrow_+)$, where $\downarrow_+ \supseteq \downarrow_1 \cup \downarrow_2 - \{r_2\}$ and $\rightarrow_+ \supseteq \rightarrow_1$ are the smallest sets such that
   (a) if $r_2 \downarrow_2$, then $r_1 \downarrow_+$, and
   (b) if $r_2 \xrightarrow{a}_2 t$, then $r_1 \xrightarrow{a}_+ t$, and
   (c) if $s \xrightarrow{a}_2 t$ and $s \neq r_2$, then $s \xrightarrow{a}_+ t$.
2. $g_1 \cdot g_2$ is defined as $(r_1, S, \downarrow_\cdot, \rightarrow_\cdot)$, where $\downarrow_\cdot$ and $\rightarrow_\cdot \supseteq \rightarrow_1$ are the smallest sets such that
   (a) if $\downarrow_1 \neq \emptyset$, then $\downarrow_\cdot \supseteq \downarrow_2 - \{r_2\}$, and

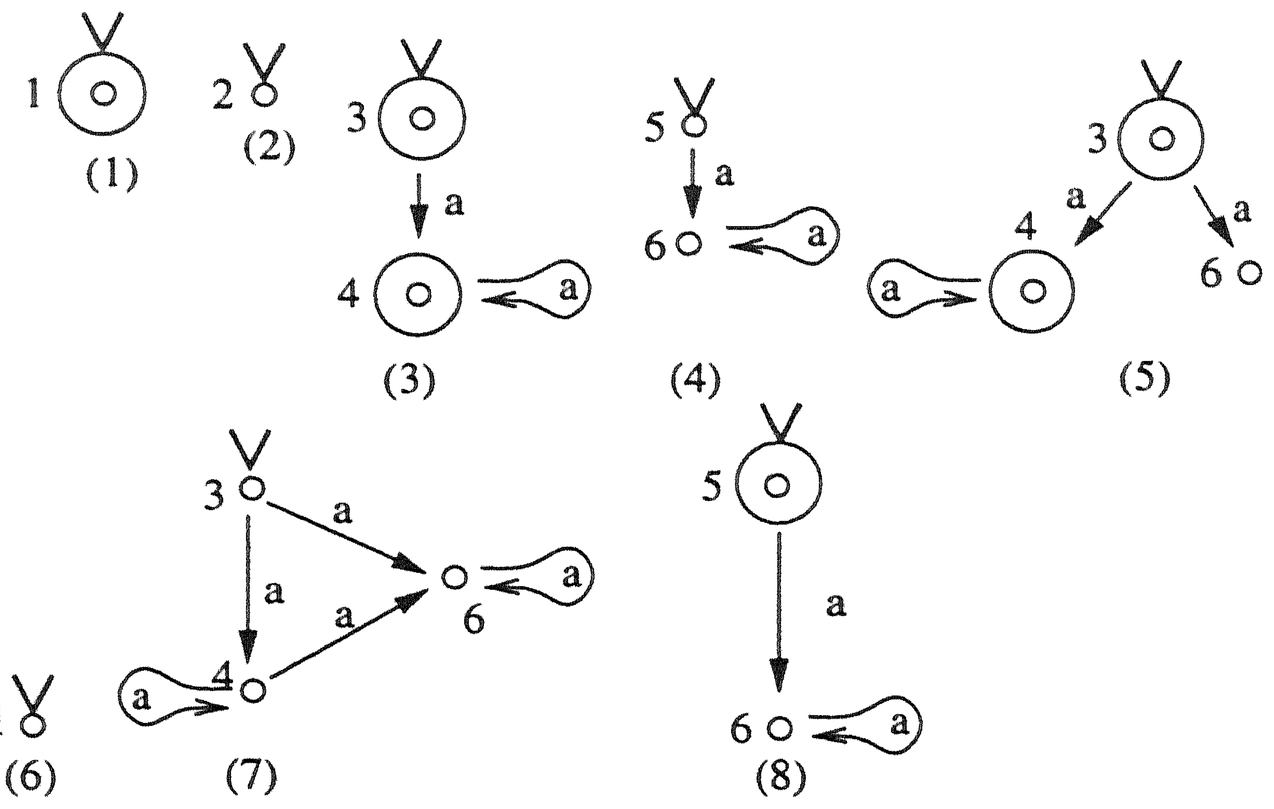
(b) if $r_2 \downarrow_2$, then $\downarrow_1 \subseteq \downarrow$., and
(c) if $r_2 \xrightarrow{a}_2 t$ and $s \downarrow_1$, then $s \xrightarrow{a} t$, and
(d) if $s \xrightarrow{a}_2 t$, $\downarrow_1 \neq \emptyset$ and $s \neq r_2$, then $s \xrightarrow{a} t$.

3. $g_1{}^* g_2$ is the process graph $g_1{}^k \cdot g_2$, where $g_1{}^k$ is the knotting of g_1 according to Definition 2.30.

REMARK 2.36. Notice that $g_1 \cdot g_2 = g_1$ if the set of termination labels of g_1 is empty. Furthermore, notice that $+$, $\cdot$ and * require disjoint reachable state sets of the process graphs for their arguments.

EXAMPLE 2.37. In the picture below the process graph

1. (1) equals (1)+(2), (5) equals (3)+(4) and (4) equals (4)+(2), and
2. (6) equals (1) · (2), (7) equals (3) · (4), and
3. (8) equals (4)*(1) and (5) equals (4)*(3)



Notice that (1)–(5) are rum's of graph interpretations of respectively ϵ, δ, $a^*\epsilon$, $a^*\delta$ and $a^*\epsilon + a^*\delta$ and (6)–(8) of respectively δ, $a^*(a^*\delta)$ and $(a^*\delta)^*\epsilon$.

We leave it to the reader to verify that the construction of the new graph from two process graphs with the interpretation of a syntactic operator is well-defined.

PROPOSITION 2.38. Let g_i for $i \in \mathbf{N}$ be finite rui process graphs such that $R_{g_i} \cap R_{g_j} = \emptyset$ for $i \neq j$. For $\oplus \in \{+, \cdot, *\}$ we have

1. $g_i \oplus g_{i+1}$ is a finite rui process graph, and
2. if $g_i \leftrightarrows g_{i+2}$ and $g_{i+1} \leftrightarrows g_{i+3}$, then $g_i \oplus g_{i+1} \leftrightarrows g_{i+2} \oplus g_{i+3}$.

PROPOSITION 2.39. Let $p_i \in \mathbf{P}^*_{\delta,\epsilon}$. Let g_i be finite rui process graphs which are respectively isomorphic to root unwindings with $u_i \in S - R_{\gamma(p_i)}$ of $\gamma(p_i)$, such that $R_{g_i} \cap R_{g_j} = \emptyset$ if $i \neq j$. For $\oplus \in \{+, \cdot, *\}$, $i \neq j$ we have

1. $\gamma(p_i \oplus p_j) \leftrightarrows g_i \oplus g_j$,
2. if $p_i \leftrightarrows p_{i+1}$ and $p_{i+2} \leftrightarrows p_{i+3}$, then $g_i \oplus g_{i+2} \leftrightarrows g_{i+1} \oplus g_{i+3}$.

DEFINITION 2.40. A process graph g is Ξ *expressible*, where $\Xi \in \{\Sigma(\mathrm{BPA}^*_\epsilon(Act)), \Sigma(\mathrm{BPA}^*(Act))\}$ iff there is a process expression $p \in P(\Xi)$ such that $g \leftrightarrows \gamma(p)$.

3. A Criterion for Process Graphs expressible with Regular Expressions definable *without* ϵ: the $\star$ property

In this section we introduce the first definitions to study expressivity by means of the system with only the binary Kleene star and the operators from the signature of BPA(Act). We proceed in the following way. First we present a criterion which is supposed to identify the expressible process graphs. Next we give a proof of the soundness of the criterion, i.e. every process graph that satisfies the criterion is expressible with a regular expression. We end with the completeness proof that the criterion exhaustively identifies the expressible process graphs.

Auxiliary graph operations for a criterion for $\Sigma(\mathrm{BPA}^*(Act))$ *expressible process graphs*

For the criterion we need a few auxiliary definitions. The first is on the exit and iterative part of a process graph. Intuitively, the exit part is the part of a process graph which consists of the root, the roots of non returning children and of derivatives of these children. Later we prove that if $p_1{}^*p_2$ satisfies a simple condition, then $\gamma(p_2)$ is bisimilar to the exit part and $\gamma(p_1{}^*\epsilon)$ bisimilar to an iterative part of a rum of $\gamma(p_1{}^*p_2)$ (Lemma 3.46).

We present the definition of an exit and iterative part first and next give some examples and general properties.

DEFINITION 3.1. Let g be a finite process graph. Rt_g is the set of *roots of derivatives of g bisimilar to g*.

DEFINITION 3.2. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum.

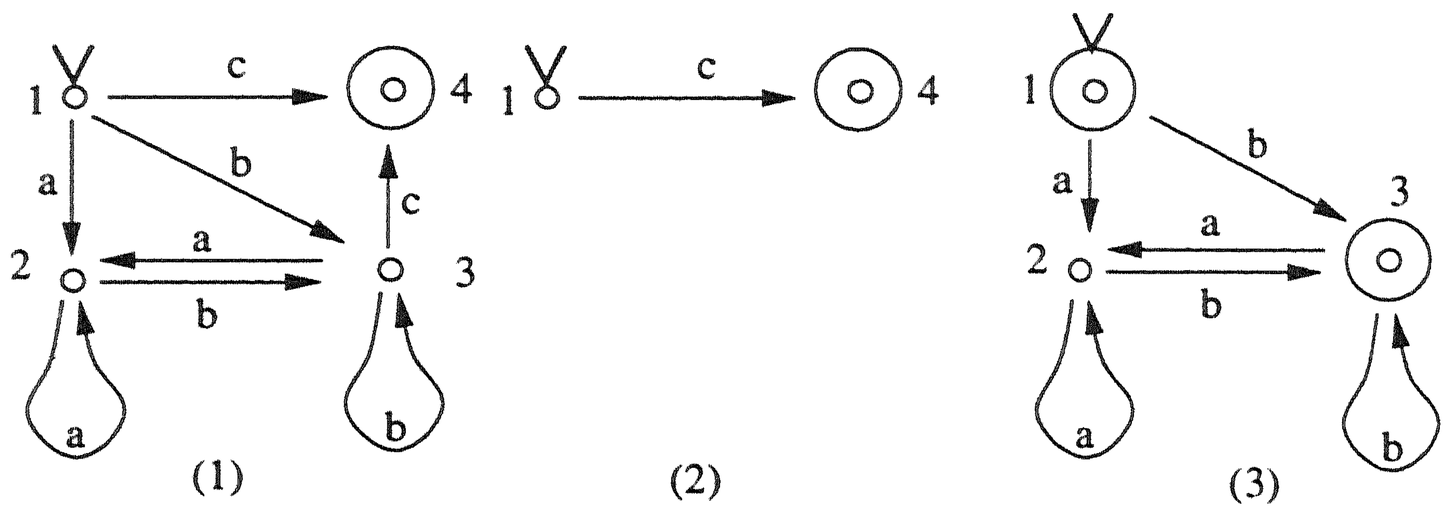
1. If $g_e = (r, S, \downarrow_e, \rightarrow_e)$ is a finite process graph where $\downarrow_e \subseteq \downarrow - Rt_g$ and $\rightarrow_e \subseteq \rightarrow$ are the smallest sets such that
   (a) if $r \xrightarrow{a} s$ and $s \not\rightarrow^* s'$, such that $s' \in Rt_g$, then $r \xrightarrow{a}_e s$, and
   (b) if $r \rightarrow_e{}^+ s$ and $s \xrightarrow{a} s'$, then $s \xrightarrow{a}_e s'$, and
   (c) if $r \rightarrow_e{}^+ s$ and $s \downarrow$, then $s \downarrow_e$,

   then g_e is *an exit part of* g.
2. If $g_i = (r, S, \downarrow_i, \rightarrow_i)$ is a finite process graph where $\downarrow_i = Rt_g$ and $\rightarrow_i \subseteq \rightarrow$ is the smallest set such that
   (a) if $r \rightarrow_i{}^* s$, $s \xrightarrow{a} s'$, $s \downarrow_i$ and $r \not\xrightarrow{a}_e s'$, then $s \xrightarrow{a}_i s'$, and
   (b) if $r \rightarrow_i{}^+ s$, $s \not\downarrow_i$ and $s \xrightarrow{a} s'$, then $s \xrightarrow{a}_i s'$,

   then g_i is *an iterative part of* g.

REMARK 3.3. Notice that we can safely omit that $\downarrow_e \subseteq \downarrow - Rt_g$.

EXAMPLE 3.4. In the picture below, (2) is an exit part of (1) and (3) is an iterative part of (1).



Notice that (1)–(3) are rum's of graph interpretations of respectively $(a^*b)^*c$, c and $(a^*b)^*\epsilon$.

LEMMA 3.5. An exit part and an iterative part of a finite rum are finite rui process graphs.

Proof. No root incoming transitions are added □

PROPOSITION 3.6. An exit and an iterative part of a finite rum are unique.

REMARK 3.7. Henceforth we speak of *the* exit and *the* iterative part.

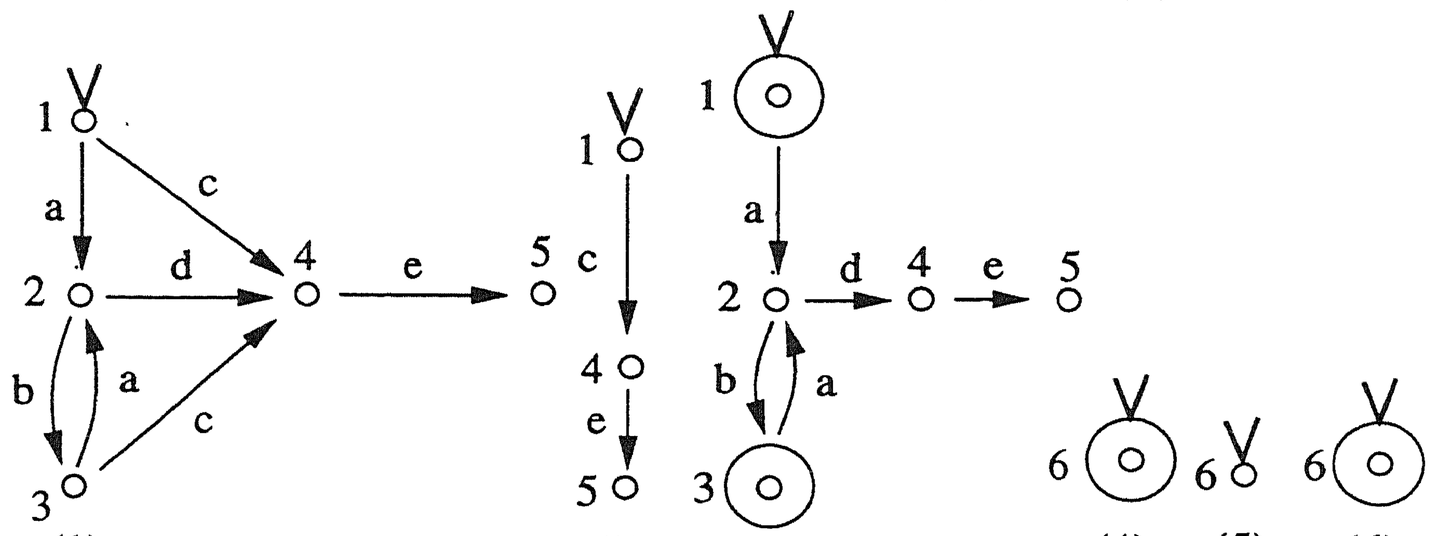
PROPOSITION 3.8. Let g be a finite rum and g_e and g_i respectively the exit and iterative part of g. Then $R_g = R_{g_e} \cup R_{g_i}$.

DEFINITION 3.9. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum, g_e the exit part of g and $g_i = (r, S, \downarrow_i, \rightarrow_i)$ the iterative part of g. If whenever $s \in R_{g_i}$,

1. $s \not\downarrow$, and
2. there is a $s' \in Rt_g$, such that $s \rightarrow_i{}^* s'$,

then g is *safely decomposable in the exit part g_e and the iterative part g_i*.

EXAMPLE 3.10. The process graph (1) in Example 3.4 is safely decomposable in the exit part (2) and the iterative part (3). The process graph (1) depicted below, is not safely decomposable in (2) and (3). (4) is not safely decomposable in (5) and (6).



(1) (2) (3) (4) (5) (6)

Notice that (1)–(6) are rum's of graph interpretations of respectively $(a \cdot (b + d \cdot e \cdot \delta))^* c \cdot e \cdot \delta$, $c \cdot e \cdot \delta$, $(a \cdot (b + d \cdot e \cdot \delta))^* \epsilon$, ϵ, δ and ϵ.

LEMMA 3.11. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If g is safely decomposable in the exit part g_e and the iterative part g_i, then $R_{g_i} \cap R_{g_e} = \{r\}$, $\downarrow_e \cap \downarrow_i = \emptyset$ and $\rightarrow_e \cap \rightarrow_i = \emptyset$.

Proof. Let g, g_e and g_i be as in the premise. Suppose towards a contradiction that there is a $s' \neq r \in R_g$ such that $[s' \in R_{g_i} \cap R_{g_e}$ or $s' \in \downarrow_e \cap \downarrow_i$ or $s \xrightarrow{a} s' \in \rightarrow_e \cap \rightarrow_i]$. By assumption that g is safely decomposable in g_e and g_i there is a transition sequence $s' \rightarrow_i{}^* s''$ for some $s'' \in Rt_g$. By definition of an iterative part $s' \rightarrow^* s''$. By definition of an exit part $r \rightarrow_e t \rightarrow_e{}^* s' \rightarrow_e{}^* s''$, where t is the root of a child of g_e. Contradiction □

EXAMPLE 3.12. The process graph (1) in Example 3.4 is safely decomposable and strongly normed. The process graph (1) in Example 3.10 is not strongly normed. The exit part (2) and iterative part (3)

have more overlapping states than the root of (1) and overlapping transitions, as the reader can verify.

PROPOSITION 3.13. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum and safely decomposable in the exit part $g_e = (r, S, \downarrow_e, \rightarrow_e)$ and the iterative part $g_i = (r, S, \downarrow_i, \rightarrow_i)$ and $s \xrightarrow{a} s'$ and $s \not\xrightarrow{a}_i s'$. If $s \downarrow_i$, then $r \xrightarrow{a}_e s'$, else $s \xrightarrow{a}_e s'$ and $s, s' \in R_{g_e} - R_{g_i}$.

LEMMA 3.14. Let g be a finite rum. If g is safely decomposable in the exit part g_e and the iterative part g_i, and $g_{e'}$ is a finite process graph such that $g_{e'} \cong g_e$ and $R_{g_{e'}} \cap R_g = \emptyset$, then $g_i \cdot g_{e'} \cong g$.

Proof. Let $g = (r, S, \downarrow, \rightarrow)$, $g_e = (r, S, \downarrow_e, \rightarrow_e)$, $g_i = (r, S, \downarrow_i, \rightarrow_i)$ and $g_{e'} = (r_{e'}, S, \downarrow_{e'}, \rightarrow_{e'})$ be as in the premise. Let h be $g_i \cdot g_{e'}$ and $\chi : R_{g_e} \rightarrow R_{g_{e'}}$ the isomorphism from g_e to $g_{e'}$. It is sufficient to prove that the map $\varphi : R_g \rightarrow R_h$ defined as

$$\begin{aligned} \varphi(s) &= s, && \text{if } s \in R_{g_i}, \\ \varphi(s) &= \chi(s), && \text{if } s \in R_{g_e} - \{r\}, \end{aligned}$$

is the isomorphism from g to h. Notice that φ is a bijective map because φ is injective since

1. By definition of φ, φ restricted to R_{g_i} is bijective, and
2. χ is a bijective map and hence χ restricted to $R_{g_e} - \{r\}$ is injective, and
3. By Proposition 3.11 $R_{g_e} \cap R_{g_i} = \{r\}$,

and surjective since

1. By definition of sequential composition $R_h = R_{g_i} \cup R_{g_{e'}} - \{r_{e'}\}$, and
2. $\varphi(R_{g_i}) = R_{g_i}$, and
3. $\chi(R_{g_e} - \{r\}) = R_{g_{e'}} - \{r_{e'}\}$.

Now for the other demands for an isomorphism:

1. By definition of sequential composition the root of h is the state r. By definition of an iterative part $r \in g_i$. By definition of φ, $\varphi(r) = r$,
2. ($\downarrow \Rightarrow \downarrow_h$) Suppose $s \downarrow$. Distinguish ($\frac{T}{F}$) whether or not $s \in R_{g_i}$. (T) If $s \in R_{g_i}$, then g is not safely decomposable. Contradiction with this assumption.

(F) If $s \notin R_{g_i}$, then by Proposition 3.8 $s \in R_{g_e}$ and $s \neq r$. By definition of isomorphism $\chi(s)\downarrow_{e'}$. By definition of an iterative part $\downarrow_i \neq \emptyset$. Hence by definition of sequential composition and h, $\varphi(s)\downarrow_h$, since $\varphi(s) = \chi(s)$.

($\downarrow_h \Rightarrow \downarrow$) Suppose $\varphi(s)\downarrow_h$. Distinguish $(\frac{T}{F})$ whether or not $s\downarrow_i$. (T) If $s\downarrow_i$, then by definition of sequential composition $\chi(r)\downarrow_{e'}$. By definition of isomorphism $r\downarrow_e$. Contradiction with the definition of an exit part.

(T) If $s \not\downarrow_i$, then by definition of sequential composition $\chi(s)\downarrow_{e'}$. By definition of isomorphism $s\downarrow_e$ and $s \neq r$. By definition of an exit part $s\downarrow$.

3. ($\rightarrow \Rightarrow \rightarrow_h$) Suppose $s \xrightarrow{a} s'$. By definition of an iterative part $r\downarrow_i$. Distinguish $(\frac{T}{F})$ whether or not $s \xrightarrow{a}_i s'$. (T) If $s \xrightarrow{a}_i s'$, then by definition of sequential composition $s \xrightarrow{a}_h s'$. Since $\varphi(s) = s$ and $\varphi(s') = s'$, because $s, s' \in R_{g_i}$ by definition of φ, $\varphi(s) \xrightarrow{a}_h \varphi(s')$.

(F) If $s \not\xrightarrow{a}_i s'$, then distinguish (F . $\frac{T}{F}$) whether or not $s\downarrow_i$. (F.T) If $s\downarrow_i$, then by definition of an iterative part $r \xrightarrow{a}_e s'$. By definition of isomorphism $\chi(r) \xrightarrow{a}_{e'} \chi(s')$. By definition of sequential composition $s \xrightarrow{a}_h \chi(s')$. Since $\varphi(s) = s$ and $\varphi(s') = \chi(s')$, by definition of φ, $\varphi(s) \xrightarrow{a}_h \varphi(s')$.

(F.F) If $s \not\downarrow_i$, then by Proposition 3.13 $s \xrightarrow{a}_e s'$ and $s, s' \in R_{g_e} - R_{g_i}$. By definition of isomorphism $\chi(s) \xrightarrow{a}_{e'} \chi(s')$ and $\chi(s) \neq \chi(r)$. By definition of an iterative part $\downarrow_i \neq \emptyset$. By definition of sequential composition $\chi(s) \xrightarrow{a}_h \chi(s')$. Since $\chi(s) = \varphi(s)$ and $\chi(s') = \varphi(s')$, by definition of φ, $\varphi(s) \xrightarrow{a}_h \varphi(s')$.

($\rightarrow \Leftarrow \rightarrow_h$) Suppose $\varphi(s) \xrightarrow{a}_h \varphi(s')$. Distinguish $(\frac{T}{F})$ whether or not $s \xrightarrow{a}_i s'$. (T) If $s \xrightarrow{a}_i s'$, then by definition of an iterative part $s \xrightarrow{a} s'$.

(F) If $s \not\xrightarrow{a}_i s'$, then distinguish (F . $\frac{T}{F}$) whether or not $s\downarrow_i$. (F.T) If $s\downarrow_i$, then by definition of an iterative part $s \in Rt_g$. By definition of sequential composition $\chi(r) \xrightarrow{a}_{e'} \chi(s')$. By definition of an isomorphism $r \xrightarrow{a}_e s'$. By definition of an exit part $r \xrightarrow{a} s'$ and hence $s \xrightarrow{a} s'$.

(F.F) If $s \not\downarrow_i$, then $\chi(s) \xrightarrow{a}_{e'} \chi(s')$ by definition of sequential composition. By definition of an isomorphism $s \xrightarrow{a}_e s'$. By definition of an exit part $s \xrightarrow{a} s'$ □

LEMMA 3.15. Let g be a finite rum. If g is safely decomposable in the exit part g_e and the iterative part g_i, then g_e and g_i are rum's.

Proof. Let g, g_e and g_i be as in the premise. Suppose towards a contradiction that g_e or g_i are not rum's. Let $g_{e'}$ be a rum of g_e with root $r_{e'}$ and $g_{i'}$ a rum of g_i such that $R_{g_{i'}} \cap R_{g_{e'}} = \emptyset$. By definition of sequential composition $g_{i'} \cdot g_{e'}$ is rui, because $g_{e'}$ and $g_{i'}$ are rui by assumption that they are rum's. Furthermore by Proposition 3.14 and Proposition 2.38 $g_{i'} \cdot g_{e'} \leftrightarrows g$. By definition of sequential composition follows that

$$\begin{array}{lcl} \mid R_{g_{i'} \cdot g_{e'}} \mid & = & \{\downarrow_i \neq \emptyset\} \\ \mid R_{g_{i'}} \mid + \mid R_{g_{e'}} - \{r_{e'}\} \mid & < & \{\mid R_{g_{i'}} \mid < \mid R_{g_i} \mid \text{ or } \mid R_{g_{e'}} - \{r_{e'}\} \mid < \mid R_{g_e} - \{r_e\} \mid\} \\ \mid R_{g_i} \mid + \mid R_{g_e} - \{r\} \mid & = & \{ \text{ Lemma 3.11, Proposition 3.8 } \} \\ \mid R_g \mid . & & \end{array}$$

Contradiction with the assumption that g is a rum. Hence g_e and g_i are rum's $\square$

The second definition for the criterion is needed for the graph interpretations of the (unary) Kleene star. Intuitively, an *un*knotting is a possible original for the knotting (Lemma 3.23). We will prove also that a rum of $\gamma(p^*\epsilon)$ has an unknotting bisimilar to $\gamma(p)$ (Lemma 3.47).

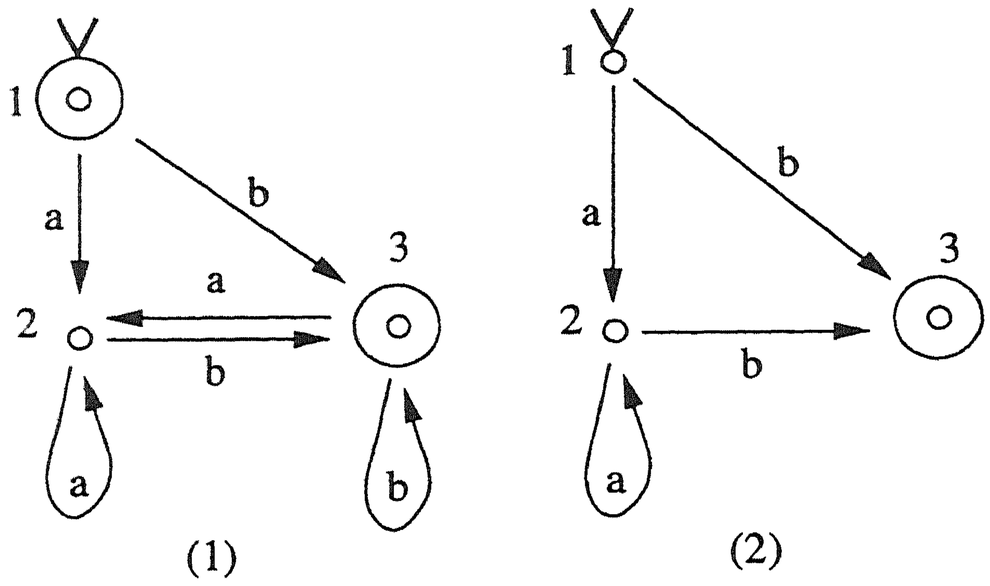
We present the definition of unknotting first and next give some examples and general properties.

DEFINITION 3.16. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If $g_{\bar{k}} = (r, S, \downarrow_{\bar{k}}, \rightarrow_{\bar{k}})$ is a process graph where $\downarrow_{\bar{k}} = \downarrow - \{r\}$ and $\rightarrow_{\bar{k}} \subseteq \rightarrow$ is the smallest set such that

1. if $r \xrightarrow{a} s$, then $r \xrightarrow{a}_{\bar{k}} s$, and
2. if $r \rightarrow_{\bar{k}}{}^* s$, $s \xrightarrow{a} s'$ and $s \not\downarrow$, then $s \xrightarrow{a}_{\bar{k}} s'$,

then $g_{\bar{k}}$ is *an unknotting of* g.

EXAMPLE 3.17. In the picture below, (2) is an unknotting of (1).



(1) (2)

Notice that (1)–(2) are rum's of graph interpretations of respectively $(a^*b)^*\epsilon$ and a^*b.

LEMMA 3.18. Let g be a finite rum. If $g_{\bar{k}}$ is an unknotting of g, then $g_{\bar{k}}$ is a finite rui process graph.

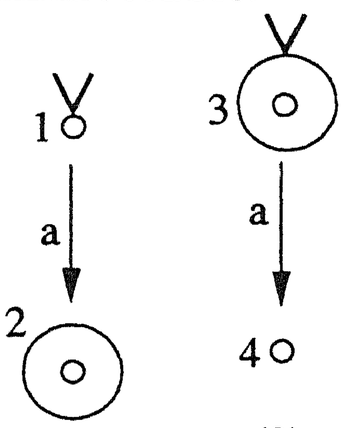
Proof. No root incoming transitions are added by an unknotting □

PROPOSITION 3.19. An unknotting of a finite rum is unique.

REMARK 3.20. Henceforth we speak of *the* unknotting.

DEFINITION 3.21. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If $g_{\bar{k}}$ is the unknotting of g such that $g_{\bar{k}}$ is strongly normed and $\downarrow = Rt_g$, then g is *safely unknottable with* $g_{\bar{k}}$.

EXAMPLE 3.22. The process graph (1) in Example 3.17 is safely unknottable with (2). In the picture below, (1) and (2) are not safely unknottable.



(1) (2)

Notice that (1)–(2) are rum's of graph interpretations of respectively a and $a \cdot \delta + \epsilon$.

LEMMA 3.23. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If g is safely unknottable with $g_{\bar{k}}$, then the knotting of $g_{\bar{k}}$ is g.

Proof. Let $g = (r, S, \downarrow, \rightarrow)$ and $g_{\bar{k}} = (r, S, \downarrow_{\bar{k}}, \rightarrow_{\bar{k}})$ be as in the premise. Let $g_{\bar{k},k}$ be the knotting of $g_{\bar{k}}$ according to Definition 2.14. It is sufficient to prove that $\downarrow = \downarrow_{\bar{k},k}$ and $\rightarrow = \rightarrow_{\bar{k},k}$.

1. ($\downarrow \subseteq \downarrow_{\bar{k},k}$) Suppose $s \downarrow$. Distinguish ($\frac{T}{F}$) whether or not $s = r$. (T) If $s = r$, then $s \not\downarrow_{\bar{k}}$ by definition of unknotting. By definition of knotting $s \downarrow_{\bar{k},k}$.

   (F) If $s \neq r$, then $s \downarrow_{\bar{k}}$ by definition of unknotting. By definition of knotting, $s \downarrow_{\bar{k},k}$.

   ($\downarrow \supseteq \downarrow_{\bar{k},k}$) Suppose $s \downarrow_{\bar{k},k}$. Distinguish ($\frac{T}{F}$) whether or not $s = r$. (T) If $s = r$, then by the assumption that g is safely unknottable with $g_{\bar{k}}$, $s \downarrow$.

   (F) If $s \neq r$, then $s \downarrow_{\bar{k}}$ by definition of knotting. By definition of unknotting, $s \downarrow$.
2. ($\rightarrow \subseteq \rightarrow_{\bar{k},k}$) Suppose $s \xrightarrow{a} s'$. Distinguish ($\frac{T}{F}$) whether or not $s \xrightarrow{a}_{\bar{k}} s'$. (T) If $s \xrightarrow{a}_{\bar{k}} s'$, then $s \xrightarrow{a}_{\bar{k},k} s'$, by definition of knotting.

   (F) If $s \not\xrightarrow{a}_{\bar{k}} s'$, then $s \downarrow$ and $s \neq r$ by definition of unknotting. By the assumption that g is safely unknottable with $g_{\bar{k}}$ follows $s \downarrow_{\bar{k}}$, $s \in Rt_g$ and $r \xrightarrow{a} s'$. By definition of unknotting $r \xrightarrow{a}_{\bar{k}} s'$. By definition of knotting we find $s \xrightarrow{a}_{\bar{k},k} s'$.

   ($\rightarrow \supseteq \rightarrow_{\bar{k},k}$) Suppose $s \xrightarrow{a}_{\bar{k},k} s'$. Distinguish ($\frac{T}{F}$) whether or not $s \xrightarrow{a}_{\bar{k}} s'$. (T) If $s \xrightarrow{a}_{\bar{k}} s'$, then by definition of unknotting $s \xrightarrow{a} s'$.

   (F) If $s \not\xrightarrow{a}_{\bar{k}} s'$, then by definition of knotting $r \xrightarrow{a}_{\bar{k}} s'$ and $s \downarrow_{\bar{k}}$. By definition of unknotting this implies that $r \xrightarrow{a} s'$ and $s \downarrow$. From the assumption that g is safely unknottable with $g_{\bar{k}}$ follows that $s \in Rt_g$ and so $s \xrightarrow{a} s'$ □

LEMMA 3.24. Let g be a finite rum. If g is safely unknottable with $g_{\bar{k}}$, then $g_{\bar{k}}$ is a rum.

Proof. Let g and $g_{\bar{k}}$ be as in the premise. By Lemma 3.18 $g_{\bar{k}}$ is rui. Suppose towards a contradiction that $g_{\bar{k}}$ has a rum h with less states than $g_{\bar{k}}$. By definition of knotting, the knotting h_k of h has also less states than $g_{\bar{k}}$, and by definition of unknotting $g_{\bar{k}}$ has as much states as g. Hence h_k has less states than g and by Lemma 2.34 h_k is a finite rui process graph. By Lemma 3.23 h_k is bisimilar to g. Contradiction with the assumption that g is a rum □

A criterion for $\Sigma(\mathrm{BPA}^*(Act))$ *expressible process graphs*

We now present a classification of finite rum's in order to define the criterion that characterizes the $\Sigma(\mathrm{BPA}^*(Act))$ expressible process graphs.

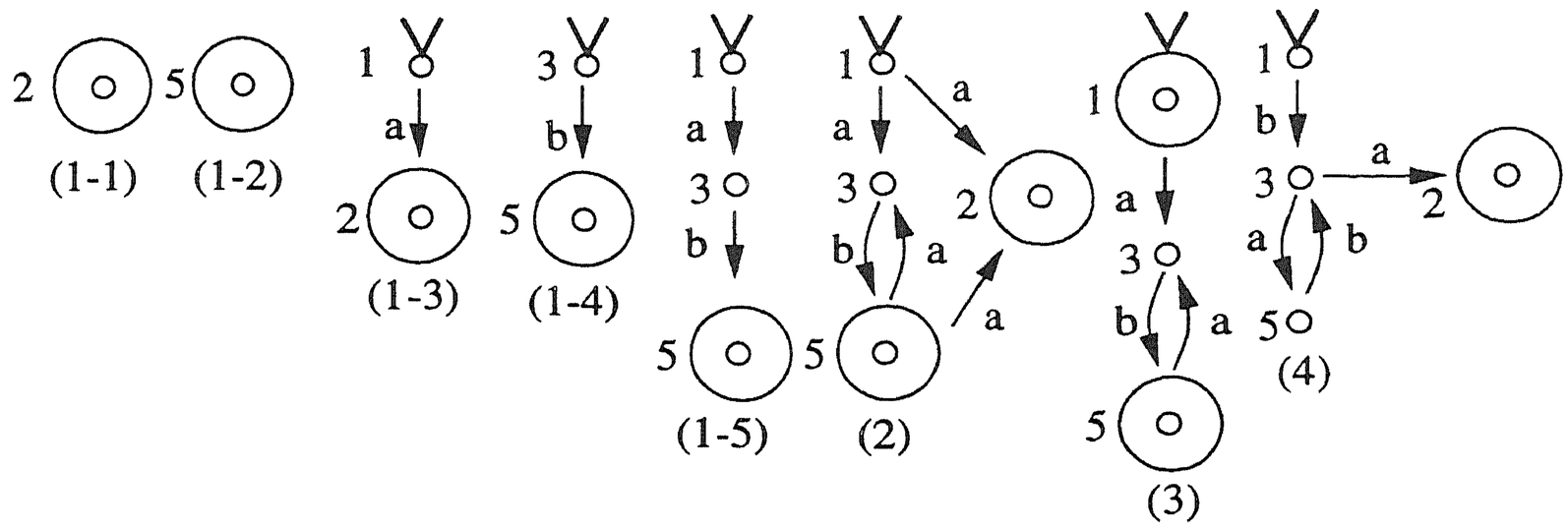
DEFINITION 3.25. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum.

1. g is of type 1 iff
   (a) g is terminating, and
   (b) if $r \downarrow$, then g is terminated, and
   (c) g is not returning, and
   (d) the root unwindings of children of g, if any, are of type 1,2 or 4.
2. g is of type 2 iff g is safely decomposable in the exit part g_e and the iterative part g_i and
   (a) g_e is not trivial and of type 1,2 or 4, and
   (b) g_i is not trivial and of type 3.
3. g is of type 3 iff g is safely unknottable with the unknotting $g_{\bar{k}}$ of type 1,2 or 4.
4. g is of type 4 iff g is returning via a derivative $g' \not\leftrightarrows g$, which has a root unwinding of type 2.

REMARK 3.26. The intuition for the definition of a type of a rum is the following. A rum of a process graph interpretation is of type 1, if the process expression is of the form $\Sigma_i a_i \cdot p_i + \Sigma_j b_j$, where the non trivial children are expressed by the p_i. It is of type 2, if the process expression is of the form ${p_1}^* p_2$, $\gamma(p_2) \not\leftrightarrows \gamma(\epsilon)$ is bisimilar to the *exit part* and $\gamma({p_1}^* \epsilon)$ is bisimilar to the the iterative part. It is of type 3, if the process expression is of the form $p^* \epsilon$ and the *unknotting* is bisimilar to $\gamma(p)$. It is of type 4, if it is the derivative of a process expression of another type, i.e. the process expression is of the form ${p_1}' \cdot {p_1}^* p_2$ and ${p_1} \rightarrow^+ {p_1}'$. (In Section 5 we develop a graph operation with which we can give the part of the rum of $\gamma({p_1}' \cdot {p_1}^* p_2)$ bisimilar to $\gamma({p_1}')$ called a *root search graph.*)

The children, exit and iterative part and unknotting are expressed in a similar inductive way.

EXAMPLE 3.27. The process graphs depicted below all have a type.



Notice that (1-1)–(1-5) are the rum's of graph interpretations of respectively ϵ, ϵ, a, b and $a \cdot b$, (2) of $(a \cdot b)^* a$ and (3)–(4) of $(a \cdot b)^* \epsilon$, $b \cdot (a \cdot b)^* a$.

1. The process graphs depicted by (1-1) and (1-2) are obviously of type 1. (1-3) and (1-4) are of type 1 since they have respectively children (1-1) and (1-2). (1-5) is of type 1, since its only child (1-4) is of type 1.
2. (2) is of type 2: its exit part is (1-3) is of type 1 and (3) is an iterative part of type 3.
3. (3) is of type 3, since it has a unknotting, namely (1-5) which is of type 1.
4. (4) is of type 4, since it is a root unwinding with state 6 of the child of (2) with root 3 of type 2.

PROPOSITION 3.28. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum and g_i be the iterative part of g and $s \in R_g$. If $r \rightarrow^* s$ and $s \rightarrow^+ s'$ for some $s' \in Rt_g$, then $s \in R_{g_i}$.

LEMMA 3.29. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. g is at most of one type 1–4.

Proof. We have to show that all

	1	2	3	4
1				
2	$\perp$			
3	$\perp$	$\perp$		
4	$\perp$	$\perp$	$\perp$	

cases marked with $\perp$ indeed give a contradiction.

Let $g = (r, S, \downarrow, \rightarrow)$ be as in the premise and g_e, g_i and $g_{\bar{k}}$ respectively the exit part, iterative part and the unknotting of g[7]. Distinguish the followings case.

1. Suppose g is of type 1.
   (a) Suppose g is of type 2. By definition of type 1 g is not returning. By definition of type 2 g has a non trivial iterative part, such that $s \neq r \in R_{g_i}$. By definition g is safely decomposable with g_e and g_i and so $r \rightarrow_i{}^+ s \rightarrow_i{}^* s'$ for some $s' \in Rt_g$. By definition of an iterative part $r \rightarrow^* s'$ and so g is returning. Contradiction.
   (b) Suppose g is of type 3. By definition of type 1 if $r \downarrow$, then g is terminated. By definition of type 3 g is safely unknottable with $g_{\bar{k}}$ and so $r \downarrow$. This implies that g is terminated. By definition of unknotting $g_{\bar{k}}$ is not terminated. Contradiction with the supposition that g is safely unknottable with $g_{\bar{k}}$.
   (c) Suppose g is of type 4. By definition of type 1, g is not returning. By definition of type 4 g is returning. Contradiction.
2. Suppose g is of type 2.
   (a) Suppose g is of type 3. By definition of type 2, g is safely decomposable in g_e and g_i and so $r \not\downarrow$. By definition of type 3 is g safely unknottable with $g_{\bar{k}}$ and so $r \downarrow$. Contradiction.
   (b) Suppose g is of type 4. By definition of type 4, g is returning via a derivative $g' \not\trianglelefteq g$ which has a root unwinding g_d of type 2. Notice that $R_{g_d} = (R_g \cup \{r_d\}) - \{r\}$ for some $r_d \notin R_g$, where r_d is the root of g_d. This implies that there are $s \neq r \in Rt_g$ and $t \neq r_d \in Rt_{g_d}$ such that $s \in R_g$ and $t \in R_{g_d}$ and $s \rightarrow^+ t \rightarrow^+ s$. By Proposition 3.28 $t \in R_{g_i}$. Since g_d is of type 2, it has a non trivial exit part $g_{de} = (r_d, S, \downarrow_{de}, \rightarrow_{de})$ and for some $t' \in \rightarrow \cap \rightarrow_{de} \cap \rightarrow_d$ there exists a transition $t \xrightarrow{a}_{de} t'$. Since g is safely decomposable in g_e and g_i, $t \not\downarrow_i$. By definition of an iterative part $t \xrightarrow{a}_i t'$ and since g is safely decomposable in g_e and g_i by supposition $t' \rightarrow_i{}^+ s$. By definition of an iterative part

[7] Notice that the exit part, iterative part and the unknotting of a rum are well-defined.

$t' \rightarrow^+ s \rightarrow^+ t$ and by definition of an exit part $t' \rightarrow_{de}{}^+ t$. Contradiction.

3. Suppose g is of type 3 and of type 4. By definition of type 3, g is safely unknottable with $g_{\bar{k}}$ and so $r \downarrow$. By definition of type 4, g is returning via a derivative $g' \not\leftrightarrows g$ which has a root unwinding $g_d = (r_d, S, \downarrow_d, \rightarrow_d)$ of type 2. Notice that $R_{g_d} = (R_g \cup \{r_d\}) - \{r\}$ for some $r_d \notin R_g$, where r_d is the root of g_d. This implies that there are $s \neq r \in Rt_g$ and $t \neq r_d \in Rt_{g_d}$ such that $t \rightarrow_d{}^+ s \rightarrow_d{}^+ t$ and $s \downarrow_d$. Since g_d is of type 2, it is safely decomposable in the exit part g_{de} and the iterative part g_{di} and $R_{g_{di}} \cap \downarrow_d = \emptyset$. By Proposition 3.28 $s \in R_{g_{di}}$ and hence $s \not\downarrow_d$. Contradiction.

Hence if g is of a type, then the type is unique □

We can now present the criterion, which we have baptized the $\star$ property:

DEFINITION 3.30. Let g be a finite process graph. Then g has the $\star$ property iff g has a non trivial rum of type 1,2 or 4.

EXAMPLE 3.31. All process graphs in Example 3.27 except (3) have the $\star$ property.

We postpone the proof that the $\star$ property is decidable to Section 5.

The $\star$ property identifies the process graphs expressible by $\Sigma(\mathrm{BPA}^*(Act))$

PROPOSITION 3.32. If a process graph is Ξ expressible all its derivatives are Ξ expressible, where $\Xi \in \Sigma(\mathrm{BPA}^*_\epsilon(Act)), \Sigma(\mathrm{BPA}^*(Act))\}$.

PROPOSITION 3.33. Let g be a finite rum and h a minimization of g. If g is not returning, a minimization of a child of h has less transitions than h.

LEMMA 3.34. Let g be a finite rum and $h = (r, S, \downarrow_h, \rightarrow_h)$ a minimization of g. If g is safely decomposable in the non trivial exit part g_e and the non trivial iterative part g_i and $h_e = (r, S, \downarrow_{h_e}, \rightarrow_{h_e})$ and $h_i = (r, S, \downarrow_{h_i}, \rightarrow_{h_i})$ are minimizations of respectively g_e and g_i, then

1. $1 \leq |\rightarrow_{h_e}| < |\rightarrow_h|$ and $1 \leq |\rightarrow_{h_i}| < |\rightarrow_h|$, and
2. $|\rightarrow_h| \geq 2$.

Proof. By the assumption that g is safely decomposable with g_e and g_i follows by Lemma 3.11 that the transitions of the exit and iterative part are disjoint □

PROPOSITION 3.35. Let g be a finite rum and $h = (r_h, S, \downarrow_h, \rightarrow_h)$ a minimization of g. If $g_{\bar{k}} \neq g$ is the unknotting of g and $h_{\bar{k}} = (r_{h_{\bar{k}}}, S, \downarrow_{h_{\bar{k}}}, \rightarrow_{h_{\bar{k}}})$ a minimization of $g_{\bar{k}}$, then

1. $|\rightarrow_h| \geq |\rightarrow_{h_{\bar{k}}}|$, $|\downarrow_h| \geq |\downarrow_{h_{\bar{k}}}|$ and $|\rightarrow_h| + |\downarrow_h| > |\rightarrow_{h_{\bar{k}}}| + |\downarrow_{h_{\bar{k}}}|$, and
2. whenever g is returning, $|\rightarrow_h| > |\rightarrow_{h_{\bar{k}}}|$.

PROPOSITION 3.36. Let g be a process graph such that $\epsilon \notin R_g$. g is bisimilar to $g \cdot \gamma(\epsilon)$.

LEMMA 3.37. Let g be a finite rum. If g is safely unknottable with the unknotting $g_{\bar{k}}$ and $\epsilon \notin R_{g_{\bar{k}}}$, then $g \leftrightarrows g_{\bar{k}}{}^* \gamma(\epsilon)$.

Proof. Immediate by Propositions 3.18, 3.23, Lemma 3.36 and Proposition 2.39 □

PROPOSITION 3.38. Let g be a finite rum and $h = (r, S, \downarrow_h, \rightarrow_h)$ a minimization of g. If g is returning via a derivative g_d, then minimizations of g and g_d have the same number of transitions and $|\rightarrow_h| \geq 2$.

THEOREM 3.39. If g is a finite process graph and has the $\star$ property, then g is $\Sigma(\mathrm{BPA}^*(Act))$ expressible.

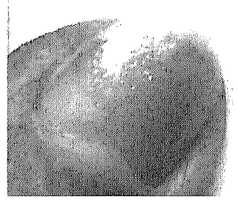
Proof. Assume without loss of generalization that $g = (r, S, \downarrow, \rightarrow)$ is a rum. We prove the result by a well-founded induction on the number of transitions m of a minimization h of g.

Let the Induction Hypothesis (I.H.) be that all rum's

1. whose minimizations have less than m transitions, and
2. are not trivial, and
3. are of type 1,2 or 4,

are $\Sigma(\mathrm{BPA}^*(Act))$ expressible. Distinguish the following cases in the Induction Step (I.S.) as to the type of g.

1. If g is of type 1, then minimizations of all (root unwindings of) children of g have less than m transitions by Proposition 3.33. The root unwindings of the children of g are of type 1,2 or 4. By the I.H. the non trivial children are $\Sigma(\mathrm{BPA}^*(Act))$ expressible.



g is expressible as the smallest expression $p \equiv p_1 + ... + p_n \in \mathbf{P}^*$, so that for every transition $r \xrightarrow{a} s$ in $\rightarrow$, where s is the root of a child g_c of g, such that

if g_c is trivial then a is a summand of p
else $a \cdot q$ is a summand of p,
where $\gamma(q)$ is bisimilar to a
root unwinding of g_c.

2. If g is of type 2, by assumption and Proposition 3.34 the exit part g_e of g is not trivial and has a minimization with less than m transitions. Since the exit part is of type 1,2 or 4, it is by assumption a rum and hence by the I.H. is $\Sigma(\mathrm{BPA}^*(Act))$ expressible as some $p_2 \in \mathbf{P}^*$.

   By Proposition 3.34 the iterative part g_i has a minimization h_i with less than m transitions. By Proposition 4.35 the unknotting $g_{i,\bar{k}} \neq g_i$ of g_i is not trivial and has a minimization $h_{i,\bar{k}}$ with less transitions than h_i. Together this implies that $h_{i,\bar{k}}$ has less transitions than h. By assumption $g_{i,\bar{k}}$ is of type 1,2 or 4 and is thus a rum. By the I.H. $g_{i,\bar{k}}$ is expressible as some $p_1 \in \mathbf{P}^*$. By Proposition 3.37 and Proposition 2.39 g_i is expressible as ${p_1}^*\epsilon$.

   By assumption that g is of type 2, g is safely decomposable with g_e and g_i. By Lemma 3.14 and Proposition 2.39 g is expressible as $({p_1}^*\epsilon) \cdot p_2 \leftrightarrows {p_1}^*p_2 \in \mathbf{P}^*$.
3. If g is of type 4, then by assumption and Proposition 3.38, g is returning via a derivative g_d whose minimizations have m transitions. The case for rum's of type 2 in the I.S. establishes that a root unwinding of g_d is $\Sigma(\mathrm{BPA}^*(Act))$ expressible. Now apply Proposition 3.32 to obtain $\Sigma(\mathrm{BPA}^*(Act))$ expressibility of g □

The graph interpretations of process expressions over $\Sigma(\mathrm{BPA}^*(Act))$ *have the* $\star$ *property*

DEFINITION 3.40. Let ${p_1}^*p_2 \in \mathbf{P}^*_\epsilon$. If there is a p_2' such that $p_2 \rightarrow^* p_2'$ and whenever ${p_1}^*p_2 \xrightarrow{a} p'$, there is a p'' so that $p_2' \xrightarrow{a} p''$ and $p' \leftrightarrows p''$, then ${p_1}^*p_2$ is an *impure iterator.* If ${p_1}^*p_2$ is not an impure iterator, then ${p_1}^*p_2$ is a *pure iterator.*

EXAMPLE 3.41. We present a pure and an *im*pure iterator.

1. The process expression a^*b is a pure iterator.

2. The process expression $a^*(a^*b)$ is an impure iterator.

PROPOSITION 3.42. Let $p_1{}^*p_2 \in \mathbf{P}^*_\epsilon$. If $p_1{}^*p_2$ is returning, then whenever $p_1 \rightarrow^+ p_1{}'$, $p_1{}^*p_2$ is returning via $p_1{}' \cdot p_1{}^*p_2$.

DEFINITION 3.43. Let $p^*\epsilon \in \mathbf{P}^*_\epsilon$. If there are p', p'', p''' such that $p \rightarrow^+ p'$, $p' \downarrow$, $p' \xrightarrow{a} p''$, $p \xrightarrow{a} p'''$ and $p'' \cdot p^*\epsilon \leftrightarrows p''' \cdot p^*\epsilon$, then $p^*\epsilon$ is an *impure knotting*. If $p^*\epsilon$ is not an impure knotting, then $p^*\epsilon$ is a *pure knotting*.

EXAMPLE 3.44. We present a pure knotting and two *im*pure knottings.

1. The process expression $(a \cdot b)^*\epsilon$ is a pure knotting.
2. The process expressions $(a \cdot (a + \epsilon))^*\epsilon$ and $(a \cdot (a + b)^*(b + \epsilon))^*\epsilon$ are impure knottings.

LEMMA 3.45. If $p \in \mathbf{P}^*$, then $p^*\epsilon$ is a pure knotting.

Proof. p has no derivatives p' such that $p' \downarrow$ and $p \leftrightarrows \epsilon$ □

LEMMA 3.46. Let $p_1{}^*p_2 \in \mathbf{P}^*$. If $p_1{}^*p_2$ is a pure iterator, $p_1{}^*\epsilon$ is a pure knotting and g is a rum of $\gamma(p_1{}^*p_2)$, then

1. the exit part g_e of g is bisimilar to $\gamma(p_2)$ and the iterative part g_i of g is bisimilar to $\gamma(p_1{}^*\epsilon)$,
2. g is safely decomposable in g_e and g_i.

Proof. Let $p_1{}^*p_2$ and $g = (r, S, \downarrow, \rightarrow)$ be as in the premise[8].

1. *Exit part.* We start with a construction of a process graph $g_{e'}$ by leaving out transitions and transition labels from g (A1). Next we prove that the thus obtained process graph is an exit part (B1) and hence *the* exit part by Proposition 3.6. Last we prove that the process graph $g_{e'}$ is bisimilar to $\gamma(p_2)$ (C1).

   (A1) Let $g_{e'} = (r, S, \downarrow_{e'}, \rightarrow_{e'})$ be the process graph where $\downarrow_{e'} \subseteq \downarrow - Rt_g$ and $\rightarrow_{e'} \subseteq \rightarrow$ are the smallest sets such that

   (a) if $p_2 \rightarrow^+ p_2{}'$ and $p_2 \downarrow$, then $s \downarrow_{e'}$, where s is the root of a derivative of g bisimilar to $\gamma(p_2{}')$, and

   (b) If $p_2 \xrightarrow{a} p_2{}'$, then $r \xrightarrow{a}_{e'} s'$, where s is the root of the derivative of g bisimilar to $\gamma(p_2{}')$, and

[8] Notice that it is in principle redundant to demand that $p_1{}^*\epsilon$ is a pure knotting, since by Lemma 3.45 all knottings $p^*\epsilon$ are pure if $p \in \mathbf{P}^*$. However we need it explicitly for a similar proof of Lemma 4.49.

(c) If $p_2 \rightarrow^+ p_2'$ and $p_2' \xrightarrow{a} p_2''$, then $s \xrightarrow{a}_{e'} s'$, where s, s' are the roots of the derivatives of g bisimilar to respectively $\gamma(p_2')$ and $\gamma(p_2'')$.

(B1) To prove that $g_{e'}$ is indeed an exit part of g we check the demands of Definition 3.2.

$(\downarrow_{e'} \subseteq \downarrow - Rt_g)$ Immediate by definition of $g_{e'}$.

$(\rightarrow_{e'} \subseteq \rightarrow)$ Immediate by definition of $g_{e'}$. Now for the rest of the demands:

(a) If $r \xrightarrow{a} s$ and $s \not\rightarrow^* s'$ for some $s' \in Rt_g$, then by definition of g $s \neq r$ is the root of a proper derivative of g. From Proposition 3.42 and the assumption that $p_1{}^* p_2$ is a pure iterator follows that s is the root of a derivative of g bisimilar to $\gamma(p_2')$ and $p_2 \xrightarrow{a} p_2'$. By definition of $g_{e'}$ $r \xrightarrow{a}_{e'} s$.

(b) If $r \rightarrow_{e'}{}^+ s$ and $s \xrightarrow{a} s'$, this implies by definition of $g_{e'}$ and g that $p_2 \rightarrow^+ p_2'$ and $p_2' \xrightarrow{a} p_2''$ such that s, s' are the roots of the derivatives of g bisimilar to $\gamma(p_2')$ and $\gamma(p_2'')$. By definition of $g_{e'}$ $s \xrightarrow{a}_{e'} s'$,

(c) If $r \rightarrow_{e'}{}^+ s$ and $s \downarrow$, then by definition of $g_{e'}$, s is the root of a derivative of g bisimilar to some p_2' such that $p_2 \rightarrow^+ p_2'$ and $p_2' \downarrow$. By definition of $g_{e'}$ $s \downarrow_{e'}$.

(C1) To prove that $g_{e'}$ and $\gamma(p_2)$ are bisimilar, we present the relation $R_1 \subseteq R_{g_{e'}} \times R_{\gamma(p_2)}$, which we prove to be a bisimulation. Let R_1 be the smallest superset of the tuple (r, p_2), of tuples of the form (s, p_2'), where s is the root of a derivative of g bisimilar to $\gamma(p_2')$ and $p_2 \rightarrow^+ p_2'$.

We check the demands of Definition 2.14. Suppose $(s, q) \in R_1$.

(a) $(\downarrow_{e'} \Rightarrow \downarrow)$ Suppose $s \downarrow_{e'}$. By definition of $g_{e'}$ this implies that s is the root of a derivative of g bisimilar to $\gamma(p_2')$, such that $p_2 \rightarrow^+ p_2'$ and $p_2' \downarrow$. By definition of R_1, $q \leftrightarrows p_2'$ and hence $q \downarrow$.

$(\downarrow_{e'} \Leftarrow \downarrow)$ Suppose $q \downarrow$. By definition of R_1 $p_2 \rightarrow^+ q$ and s is the root of the derivative of g bisimilar to $\gamma(q)$. By definition of $g_{e'}$ $s \downarrow_{e'}$.

(b) $(\rightarrow_{e'} \Rightarrow \rightarrow)$ Suppose $s \xrightarrow{a}_{e'} s'$. Distinguish $(\frac{T}{F})$ whether or not $s = r$. (T) If $s = r$, then by definition of $g_{e'}$ s' is the root of a derivative of g bisimilar to some p_2'' and $p_2 \xrightarrow{a} p_2''$.

By definition of R_2 $q \equiv p_2$. By definition of R_1 $(s', {p_2}') \in R_1$.
(F) If $s \neq r$, then by definition of $g_{e'}$ $p_2 \rightarrow^+ {p_2}'$ and s, s' are the roots of derivative of g bisimilar to respectively $\gamma({p_2}')$, $\gamma({p_2}'')$. By definition of R_1 $q \leftrightarrows {p_2}'$ and hence $q \xrightarrow{a} q'$ for some $q' \leftrightarrows {p_2}''$. By definition of R_1 $(s', q') \in R_1$.

(c) $(\rightarrow_{e'} \Leftarrow \rightarrow)$ Suppose $q \xrightarrow{a} q'$. Distinguish $(\frac{T}{F})$ whether or not $q \equiv p_2$. (F) If $q \equiv p_2$, then by definition of R_1, $s = r$. By definition of $g_{e'}$, this implies $s \xrightarrow{a}_{e'} s'$, where s' is the root of a derivative of g bisimilar to $\gamma(q')$. By definition of R_1 $(s', q') \in R_1$.
(T) If $q \not\equiv p_2$, then by definition of R_1, s is the root of the derivative of g bisimilar to $\gamma(q)$ and $p_2 \rightarrow^+ q$. By definition of $g_{e'}$ this implies $s \xrightarrow{a}_{e'} s'$, where s' is the root of a derivative of g bisimilar to $\gamma(q')$. By definition of R_1 $(s', q') \in R_1$.

By definition $(r, p_2) \in R_1$ and hence $g_{e'} \leftrightarrows \gamma(p_2)$.

Iterative part. We start again with a construction of a process graph $g_{i'}$ by leaving out transitions and transition labels from g (A2). Next we prove that the thus obtained process graph is an iterative part (B2) and hence *the* iterative part by Proposition 3.6. Last we prove that the process graph is bisimilar to $\gamma({p_1}^*\epsilon)$ (C2).

(A2) Let $g_{i'} = (r, S, \downarrow_{i'}, \rightarrow_{i'})$ be the process graph where $\downarrow_{i'} = Rt_g$ and $\rightarrow_{i'} \subseteq \rightarrow$ is the smallest set such that

(a) If ${p_1}^*\epsilon \xrightarrow{a} {p_1}' \cdot {p_1}^*\epsilon$, then $r \xrightarrow{a}_{i'} s$, where $s \neq r$ is the root of the derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^* p_2)$, and

(b) If ${p_1}^*\epsilon \rightarrow^+ {p_1}' \cdot {p_1}^*\epsilon$ and ${p_1}' \cdot {p_1}^*\epsilon \xrightarrow{a} {p_1}'' \cdot {p_1}^*\epsilon$, then $s \xrightarrow{a}_{i'} s'$, where $s \neq r$, $s' \neq r$ are the roots of the derivatives of g bisimilar to respectively $\gamma({p_1}' \cdot {p_1}^* p_2)$ and $\gamma({p_1}'' \cdot {p_1}^* p_2)$.

(B2) To prove that $g_{i'}$ is indeed an iterative part of g we check the demands of Definition 3.2.

$(\downarrow_{i'} = Rt_g)$ Immediate by definition of $g_{i'}$.

$(\rightarrow_{i'} \subseteq \rightarrow)$ Immediate by definition of $g_{i'}$.

We use that the previously defined process graph $g_{e'}$ is proved to be the exit part for the other demands of Definition 3.2:

(a) If $r \rightarrow_{i'}{}^* s$, $s \downarrow_{i'}$ and $s \overset{a}{\not\rightarrow}_{e'} s'$, then by definition of $g_{i'}$ $s \in Rt_g$. Distinguish whether (0) s' is the root of a derivative of g bisimilar to $\gamma(p_2{}')$ and $p_2 \xrightarrow{a} p_2{}'$ or (1) s' is the root of a derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^* p_2)$ and $p_1 \xrightarrow{a} p_1{}'$. (0) If s' is the root of a derivative of g bisimilar to $\gamma(p_2{}')$ and $p_2 \xrightarrow{a} p_2{}'$, then by definition of $g_{e'}$ $s \xrightarrow{a}_{e'} s'$. Contradiction.
(1) If s' is the root of a derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^* p_2)$ and $p_1 \xrightarrow{a} p_1{}'$, then by the operational semantics $p_1{}^* \epsilon \xrightarrow{a} p_1{}' \cdot p_1{}^* \epsilon$. By definition of $g_{i'}$ $s \xrightarrow{a}_{i'} s'$,

(b) Suppose $r \rightarrow_{i'}{}^+ s$, $s \not\downarrow_{i'}$ and $s \xrightarrow{a} s'$. By definition of g, $g_{i'}$ and the operational semantics we can assume that $s \not\equiv r$, $s' \not\equiv r$ are the roots of derivatives of g bisimilar to respectively $\gamma(p_1{}' \cdot p_1{}^* p_2)$, $\gamma(p_1{}'' \cdot p_1{}^* p_2)$, $p_1{}^* \epsilon \rightarrow^+ p_1{}' \cdot p_1{}^* \epsilon$, and $p_1{}' \cdot p_1{}^* p_2 \xrightarrow{a} p_1{}'' \cdot p_1{}^* p_2$. By definition of $g_{i'}$ $p_1{}' \cdot p_1{}^* p_2 \not\leftrightarrows p_1{}^* p_2$. This implies $p_1{}' \not\leftrightarrows \epsilon$. Proposition 2.7 implies that $p_1{}' \not\downarrow$ and hence $p_1{}' \xrightarrow{a} p_1{}''$. By the operational semantics $p_1{}' \cdot p_1{}^* \epsilon \xrightarrow{a} p_1{}'' \cdot p_1{}^* \epsilon$. By definition of $g_{i'}$ $s \xrightarrow{a}_{i'} s'$.

(C2) To prove that $g_{i'}$ and $\gamma(p_1{}^* \epsilon)$ are bisimilar, we present the relation $R_2 \subseteq R_{g_{i'}} \times R_{\gamma(p_1{}^* \epsilon)}$, which we prove to be a bisimulation. Let R_2 be the smallest superset of the tuple $\{(r, p_1{}^* \epsilon)\}$ of tuples of the form $(s, p_1{}' \cdot p_1{}^* \epsilon)$, where $s \not\equiv r$ is the root of a derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^* p_2)$ and $p_1{}^* \epsilon \rightarrow^+ p_1{}' \cdot p_1{}^* \epsilon$.

We check the demands of Definition 2.14. Suppose $(s, q) \in R$.

(a) ($\downarrow_{i'} \Rightarrow \downarrow$) Suppose $s \downarrow_{i'}$. By definition of $g_{i'}$ $s \in Rt_g$. Distinguish ($\frac{T}{F}$) whether or not $s = r$. (T) If $s = r$, then $q \equiv p_1{}^* \epsilon$. By the operational semantics $q \downarrow$.
(F) If $s \not\equiv r$, then s is the root of a derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^* p_2)$ and $p_1{}' \cdot p_1{}^* p_2 \leftrightarrows p_1{}^* p_2$. By Proposition 2.7 follows that $p_1{}' \leftrightarrows \epsilon$. By definition of R_2 $q \leftrightarrows p_1{}' \cdot p_1{}^* \epsilon$. By the operational semantics $q \downarrow$.
($\downarrow_{i'} \Leftarrow \downarrow$) Suppose $q \downarrow$. Distinguish ($\frac{T}{F}$) whether or not $q \equiv p_1{}^* \epsilon$. (T) If $q \equiv p_1{}^* \epsilon$, then by definition of R_2, $s = r$ and by definition of $g_{i'}$, $r \downarrow_{i'}$.
(F) If $q \not\equiv p_1{}^* \epsilon$, then $q \equiv p_1{}' \cdot p_1{}^* \epsilon$ and $p_1{}^* \epsilon \rightarrow^+ p_1{}' \cdot p_1{}^* \epsilon$. By Proposition 2.7 follows that $p_1{}' \leftrightarrows \epsilon$. By definition of R_2 $s \not\equiv r$ is the root of a derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^* p_2)$. This implies that $s \in Rt_g$. By definition of $g_{i'}$ $s \downarrow_{i'}$.

(b) $(\rightarrow_{i'} \Rightarrow \rightarrow)$ Suppose $s \xrightarrow{a}_{i'} s'$. Distinguish $(\frac{T}{F})$ whether or not $s = r$. (T) If $s = r$, then by definition of R_2, $q \equiv {p_1}^* \epsilon$. By definition of $g_{i'}$, $s' \neq r$ is the root of the derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^* p_2)$ and ${p_1}^* \epsilon \xrightarrow{a} {p_1}' \cdot {p_1}^* \epsilon$. By definition of R_2, $(s', {p_1}' \cdot {p_1}^* p_2) \in R_2$.
(F) If $s \neq r$, then by definition of $g_{i'}$ s, s' are the roots of the derivatives of g bisimilar to respectively $\gamma({p_1}' \cdot {p_1}^* p_2)$, $\gamma({p_1}'' \cdot {p_1}^* p_2)$, ${p_1}^* \epsilon \rightarrow^+ {p_1}' \cdot {p_1}^* \epsilon$ and ${p_1}' \cdot {p_1}^* \epsilon \xrightarrow{a} {p_1}'' \cdot {p_1}^* \epsilon$. By definition of R_2 $q \leftrightarrows {p_1}' \cdot {p_1}^* \epsilon$. This implies that $q \xrightarrow{a} q'$, such that $q' \leftrightarrows {p_1}'' \cdot {p_1}^* \epsilon$. By definition of R_2, $(s', q') \in R_2$.

(c) $(\rightarrow_{i'} \Leftarrow \rightarrow)$ Suppose $q \xrightarrow{a} q'$. Distinguish $(\frac{T}{F})$ whether or not $q \equiv {p_1}^* \epsilon$. (T) If $q \equiv {p_1}^* \epsilon$, then $q' \equiv {p_1}' \cdot {p_1}^* \epsilon$. By definition of R_2 follows that $s = r$. Let $s' \neq r$ be the root of a derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^* p_2)$. By definition of $g_{i'}$ $s \xrightarrow{a}_{i'} s'$. By definition of R_2 $(s', q') \in R_2$.
(F) If $q \not\equiv {p_1}^* \epsilon$, then ${p_1}^* \epsilon \rightarrow^+ q$, $q \equiv {p_1}' \cdot {p_1}^* \epsilon$ and $q' \equiv {p_1}'' \cdot {p_1}^* \epsilon$. By definition of $g_{i'}$ $s \neq r$ is the root of a derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^* p_2)$. Let $s' \neq r$ be the root of a derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^* p_2)$. By definition of $g_{i'}$ $s \rightarrow_{i'} s'$. By definition of R_2 $(s', q') \in R_2$.

By definition $(r, {p_1}^* \epsilon) \in R_2$ and hence $g_{i'} \leftrightarrows \gamma({p_1}^* \epsilon)$.

2. Now g is safely decomposable with the just established exit part $g_{e'}$ and iterative part $g_{i'}$: If $s \in R_{g_{i'}}$, then by definition of $g_{i'}$, $r = s$ or s is the root of a derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^* p_2)$ and ${p_1}^* \epsilon \rightarrow^+ {p_1}' \cdot {p_1}^* \epsilon$ so
   (a) by Proposition 2.7 $s \not\downarrow$, and
   (b) distinguish $(\frac{T}{F})$ whether or not $s = r$. (T) If $s = r$, then we are finished.
   (F) By the operational semantics ${p_1}' \cdot {p_1}^* \epsilon \rightarrow^* {p_1}'' \cdot {p_1}^* \epsilon$ and ${p_1}'' \leftrightarrows \epsilon$. By definition of $g_{i'}$ $s \rightarrow_{i'}{}^* s'$, where $s' \neq r$ is the root of a derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^* p_2)$. Obviously $s' \in Rt_g$ □

LEMMA 3.47. Let $p \in \mathbf{P}^*$. If $p^* \epsilon$ is a pure knotting and g is a rum of $\gamma(p^* \epsilon)$, then

1. $\gamma(p)$ is bisimilar to the unknotting $g_{\bar{k}}$ of g,
2. g is safely unknottable with $g_{\bar{k}}$.

Proof. Let $p^*\epsilon$ be and $g = (r, S, \downarrow, \rightarrow)$ be as in the premise [9].

1. We first present a process graph $g_{\bar{k}'}$, which we obtain by leaving out termination labels and transitions from g (A). We present a relation R which we prove to be a bisimulation relating $g_{\bar{k}'}$ and $\gamma(p)$ (B). Last we prove that $g_{\bar{k}'}$ is an unknotting of g (C) and hence *the* unknotting of g by Proposition 3.19.

   (A) Let $g_{\bar{k}'} = (r, S, \downarrow_{\bar{k}'}, \rightarrow_{\bar{k}'})$ be a process graph where $\downarrow_{\bar{k}'} = \downarrow - \{r\}$ and $\rightarrow_{\bar{k}'} \subseteq \rightarrow$ is the smallest set such that

   (a) if $p \xrightarrow{a} p'$ and $s \neq r$, then $r \xrightarrow{a}_{\bar{k}'} s$, where s is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$, and

   (b) if $p \rightarrow^+ p'$ and $p' \xrightarrow{a} p''$, then $s \xrightarrow{a}_{\bar{k}'} s'$, where $s \neq r$, $s' \neq r$ are the roots of derivatives of g bisimilar to respectively $\gamma(p' \cdot p^*\epsilon)$ and $\gamma(p'' \cdot p^*\epsilon)$.

   (B) Let $R \subseteq R_{g_{\bar{k}'}} \times R_{\gamma(p)}$ be the smallest superset of (r, p) of tuples of the form (s, p'), where $s \neq r$ is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$ and $p \rightarrow^+ p'$.

   We prove that R is a bisimulation between $g_{\bar{k}'}$ and $\gamma(p)$: Suppose $(s, q) \in R$.

   (a) $(\downarrow_{\bar{k}'} \Rightarrow \downarrow)$ Suppose $s \downarrow_{\bar{k}'}$. By definition of $g_{\bar{k}'}$ $s \in \downarrow - \{r\}$. This implies that $s \neq r$ is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$ and $p' \cdot p^*\epsilon \downarrow$. By the operational semantics $p' \downarrow$. By definition of R $q \leftrightarrows p'$. Hence $q \downarrow$.
   $(\downarrow_{\bar{k}'} \Leftarrow \downarrow)$ Suppose $q \downarrow$, then by Proposition 2.7 $q \leftrightarrows \epsilon$. By definition of R $s \neq r$ is the root of a derivative of g bisimilar to $\gamma(q \cdot p^*\epsilon)$. By the operational semantics $s \downarrow$. By definition of $g_{\bar{k}'}$, $s \downarrow_{\bar{k}'}$.

   (b) $(\rightarrow_{\bar{k}'} \Rightarrow \rightarrow)$ Suppose $s \xrightarrow{a}_{\bar{k}'} s'$. By definition of $g_{\bar{k}'}$, $s' \neq r$ is the root of a derivative of g bisimilar to $\gamma(p'' \cdot p^*\epsilon)$ and $p \rightarrow^* p' \xrightarrow{a} p''$. Distinguish $(\frac{T}{F})$ whether or not $s = r$. (T) If $s = r$, then $q \equiv p$. By definition of $g_{\bar{k}'}$, $p' \equiv p$. By definition of R $(s', p'') \in R$.
   (F) If $s \neq r$, then by definition of $g_{\bar{k}'}$ s is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$. By definition of R $(s', p'') \in R$.

   (c) $(\rightarrow_{\bar{k}'} \Leftarrow \rightarrow_p)$ Suppose $q \xrightarrow{a} q'$. Distinguish $(\frac{T}{F})$ whether or not $q \equiv p$. (T) If $q \equiv p$, then let $s' \neq r$ be the root of a

[9] Same as footnote 8.

derivative of g bisimilar to $\gamma(q' \cdot p^*\epsilon)$. By definition of $g_{\bar{k}'}$ $r \xrightarrow{a}_{k'} s'$. By definition of R $(s', q') \in R$.

(F) If $q \not\equiv p$, then by definition of R $s \neq r$ is the root of a derivative of g bisimilar to $\gamma(q \cdot p^*\epsilon)$. Let $s' \neq r$ be the root of a derivative of g bisimilar to $\gamma(q' \cdot p^*\epsilon)$. By definition of $g_{\bar{k}'}$ $s \xrightarrow{a}_{k'} s'$. By definition of R $(s', q') \in R$.

By definition $(r, p) \in R$ and hence $g_{\bar{k}'} \leftrightarrows \gamma(p)$.

(C) To verify that $g_{\bar{k}'}$ is an unknotting we verify the demands of Definition 3.16.

($\downarrow_{\bar{k}'} = \downarrow - \{r\}$) Immediate by definition of $g_{\bar{k}'}$.

($\rightarrow_{\bar{k}'} \subseteq \rightarrow$) Immediate by definition of $g_{\bar{k}'}$.

Now for the other demands of Definition 3.16:

(a) If $r \xrightarrow{a} s$, then by definition of g $p^*\epsilon \xrightarrow{a} p' \cdot p^*\epsilon$, s is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$ and $s \neq r$. By definition of $g_{\bar{k}'}$, $r \xrightarrow{a}_{k'} s$.

(b) If $s \rightarrow_{\bar{k}'}{}^* s'$, $s \xrightarrow{a} s'$ and $s \not\downarrow$, then by definition of $g_{\bar{k}'}$ $s \neq r$, $s' \neq r$ are the roots of derivatives of g bisimilar to respectively $\gamma(p' \cdot p^*\epsilon)$, $\gamma(p'' \cdot p^*\epsilon)$, $p \rightarrow^+ p'$, $p' \cdot p^*\epsilon \xrightarrow{a} p'' \cdot p^*\epsilon$ and $p' \cdot p^*\epsilon \not\downarrow$. By the operational semantics $p' \xrightarrow{a} p''$. By definition of $g_{\bar{k}'}$ $s \xrightarrow{a}_{\bar{k}'} s'$.

2. Now g is safely unknottable with $g_{\bar{k}'} \leftrightarrows \gamma(p)$. (1) By Proposition 2.8 $g_{\bar{k}'}$ is strongly normed. (2) ($\downarrow \subseteq Rt_g$) Suppose $s \downarrow$. Distinguish $(\frac{T}{F})$ whether or not $s = r$. (T) If $s = r$, we are finished.

(F) If $s \neq r$, then s is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$. By the operational semantics $p' \downarrow$. By Proposition 2.7 $p' \leftrightarrows \epsilon$. This implies that $s \in Rt_g$.

($\downarrow \supseteq Rt_g$) Suppose $s \in Rt_g$. Distinguish $(\frac{T}{F})$ whether or not $s = r$. (T) If $s = r$, then since $p^*\epsilon \downarrow$, $s \downarrow$.

(F) If $s \neq r$, then s is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$. By the assumption that $p^*\epsilon$ is a pure knotting $p' \leftrightarrows \epsilon$. By definition of g $s \downarrow$ □

Throughout this chapter we use primarily the symbol complexity as a measure of the size of regular expressions.

DEFINITION 3.48. The symbol complexity $\# : \mathbf{P}^*_{\delta,\epsilon} \to \mathsf{N}^+$ of a process expression is defined as follows. Let $p, q \in \mathbf{P}^*_{\delta,\epsilon}$, then

$$\begin{array}{lcll} \#(p) & = & 1, & p \in \{\delta, \epsilon\} \\ \#(a) & = & 2, & a \in Act, \\ \#(p \oplus q) & = & \#(p) + \#(q) + 1, & \oplus \in \{+, \cdot, ^*\}. \end{array}$$

LEMMA 3.49. Let $p \in \mathbf{P}^*_\epsilon$. If $p \xrightarrow{a} p'$ and $\#(p) \leq \#(p')$, then there are $p_1, p_2, p'' \in \mathbf{P}^*_\epsilon$ such that $p_1{}^*p_2 \xrightarrow{a} p''$, $\#(p_1{}^*p_2) \leq \#(p)$ and $p'' \leftrightarrows p'$.

Proof. We prove the result by a well-founded induction on the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that for all processes q with $\#(q) < \#(p)$, such that if $q \xrightarrow{a} q'$ and $\#(q) \leq \#(q')$, then there are q_1, q_2, q'' such that

1. $q_1{}^*q_2 \xrightarrow{a} q''$, and
2. $\#(q_1{}^*q_2) \leq \#(q)$ and $q'' \leftrightarrows q'$.

Distinguish the following cases in the Induction Step as to the structure of p.

1. If $p \equiv \epsilon$, then the operational semantics give that no transition is possible. If $p \equiv a$, for some $a \in Act$, only one transition is possible which decreases the symbol complexity. Contradiction,
2. If $p \equiv q_1 + q_2$, then the I.H. holds for q_1 and q_2, which have a smaller symbol complexity than p. Now use that the children of p are either children of q_1 or q_2,
3. If $p \equiv q_1 \cdot q_2$, distinguish $(\frac{T}{F})$ whether or not p' is a child of q_2. (T) If p' is a child of q_2, then we use that q_2 has as smaller symbol complexity than p. The result then follows by the I.H. [10].

   (F) If p' is not a child of q_2, by the operational semantics p' is of the form $q_1{}' \cdot q_2$ where $q_1 \xrightarrow{a} q_1{}'$. The I.H. also applies to q_1, and hence there are q_{11}, q_{12} such that $q_{11}{}^*q_{12} \xrightarrow{a} q_1{}''$, $\#(q_{11}{}^*q_{12}) \leq \#(q_1)$ and $q_1{}'' \leftrightarrows q_1{}'$. Now by the soundness of the axiom BKS2[11] the result follows with $p_1 \equiv q_{11}$, $p_2 \equiv q_{12} \cdot q_2$ and $p'' \equiv q_1{}'' \cdot q_2$,
4. If $p \equiv p_1{}^*p_2$, then the result follows immediately □

REMARK 3.50. In fact we use an even stronger result, which uses a similar proof. If $p \in \mathbf{P}^*$ in the above lemma, then $p_1{}^*p_2 \in \mathbf{P}^*$ as well.

[10] Notice that this is only possible if $q_1 \downarrow$, which is if $q_1 \in \mathbf{P}^*_\epsilon$.

[11] See the Introduction.

PROPOSITION 3.51. Let $p_1{}^*p_2 \in \mathbf{P}^*_\epsilon$ be an impure iterator. If $p_1{}^*p_2$ is returning, then either $p_1{}^*p_2$ is returning via a derivative of p_2, or every child of $p_1{}^*p_2$ is bisimilar to a derivative of p_2.

We can now prove the completeness of the criterion.

THEOREM 3.52. If $p \in \mathbf{P}^*$, then $\gamma(p)$ has the $\star$ property.

Proof. By the operational semantics of BPA$^*(Act)$, $\gamma(p)$ is not trivial for every $p \in \mathbf{P}^*$. Hence it is sufficient to prove that a rum of $\gamma(p)$ is of type 1,2 or 4.

We prove by a well-founded induction on the symbol complexity of p a slightly stronger result. Let the Induction Hypothesis (I.H.) be that for all $p' \in \mathbf{P}^*$, such that $\#(p') < \#(p)$, rum's of *derivatives* of $\gamma(p')$ are of type 1,2 or 4. Let $g = (r, S, \downarrow, \rightarrow)$ be a rum of $\gamma(p)$. Distinguish the following cases in the Induction Step (I.S.) as to the structure of p.

1. If $p \equiv a$ for some $a \in Act$, then $\gamma(a) = (a, S, \{\epsilon\}, \{a \xrightarrow{a} \epsilon\})$. This implies that $g \cong \gamma(p)$, since $\gamma(p)$ is a rum. We leave it to the reader to verify that g is of type 1.

   The only child and proper derivative of g is isomorphic to $\gamma(\epsilon) = (\epsilon, S, \{\epsilon\}, \emptyset)$, which is obviously of type 1 as well,
2. If $p \equiv p_1 + p_2$, then by the I.H. the rum's of derivatives of $\gamma(p_1)$ and $\gamma(p_2)$ are of type 1,2 or 4. By the assumption that $p \in \mathbf{P}^*$, g is terminating, not trivial and $r \not\downarrow$. Now distinguish $(\frac{T}{F})$ whether or not p is returning. (T) If p is returning, then g is bisimilar to a proper derivative of $\gamma(p_1)$ or $\gamma(p_2)$ and therefore g is of the requested type by the I.H..

   (F) If p is not returning, a root unwinding of a child of g is a rum by Lemma 2.27. Such a child is bisimilar to a rum of a child of $\gamma(p_1)$ or $\gamma(p_2)$, which is of the requested type by the I.H.. This implies that g is of type 1.

   Hence g is of type 1,2 or 4.

   The graph interpretations of rum's of proper derivatives of $\gamma(p)$ are of type 1,2 or 4, since the I.H. applies to the derivatives of $\gamma(p_1)$ and $\gamma(p_2)$.
3. If $p \equiv p_1 \cdot p_2$, then distinguish $(\frac{T}{F})$ whether or not [p is returning via a child p', such that $\#(p) > \#(p')$]. (T) If p is returning via child p', such that $\#(p) > \#(p')$, then g is bisimilar to a rum of a derivative of $\gamma(p')$. By the I.H. g is of type 1,2 or 4.

(F) If p is not [returning via a child p', such that $\#(p) > \#(p')$], then we reduce the question whether p is of type 1,2 or 4 to the question whether the derivatives of the rum of a graph interpretation of *another* process expression q is of type 1,2 or 4, such that p is bisimilar to a derivative of q: Distinguish (F . $\frac{T}{F}$) whether or not p is returning. (F.T) If p is returning, then by Lemma 3.49 and Remark 3.50 the next case in the I.S. with * as principal operator establishes that

(a) there is a $q \in \mathbf{P}^*$, such that a rum of $\gamma(q)$ is of type 1,2 or 4, and
(b) $\#(q) \leq \#(p)$, and
(c) p is returning via a derivative of q.

Assuming that we can prove the result for the case with * as principal operator, g is of type 1 2, or 4.

(F.F) If p is not returning, then by Proposition 2.27 a root unwinding $g_{c,ru}$ of each child of g is a rum. Furthermore $g_{c,ru}$ is of type 1,2 or 4: If $g_{c,ru}$ is bisimilar to a derivative $\gamma(p')$ of $\gamma(p)$, such that $\#(p') < \#(p)$, the I.H. establishes this result. If $g_{c,ru}$ is bisimilar to a derivative $\gamma(p')$ of $\gamma(p)$, such that $\#(p') \not< \#(p)$, we can reduce the question whether $g_{c,ru}$ is of type 1,2 or 4 in the same way as before to the question whether another process expression is of type 1,2 or 4.

Hence g is of type 1,2 or 4.

Now for the proper derivatives p' of p, distinguish ($\frac{T}{F}$) whether or not [p' is a derivative of a child p'' of p, such that $\#(p'') < \#(p)$]. (T) If $\#(p'') < \#(p)$], then by the I.H. a rum of $\gamma(p')$ is of type 1,2 or 4.

(F) If there is no child p'' of p, such that $p \to p'' \to^* p'$ and $\#(p'') < \#(p)$, then with the argument above it is established that a rum of $\gamma(p')$ is of type 1,2 or 4,

4. If $p \equiv p_1{}^*p_2$, then by Lemma 3.45 we can safely assume that $p_1{}^*\epsilon$ is a pure knotting. Distinguish ($\frac{T}{F}$) whether or not p is a pure iterator. For convenience we treat first the case (F) that p is an impure iterator. (F) Suppose p is an impure iterator. By Proposition 3.51 and the I.H. a rum of the graph interpretation of every derivative of p is of type 1,2 or 4.

(T) If p is a pure iterator, then by Lemma 3.46 g is safely decomposable in the exit part $g_e \leftrightarrows \gamma(p_2)$ and the iterative part $g_i \leftrightarrows \gamma({p_1}^*\epsilon)$.

The exit part g_e of g is a rum by Lemma 3.15. By the operational semantics of BPA$^*(Act)$, $\gamma(p_2)$ is not trivial and so g_e is not trivial. By the I.H. a rum of $\gamma(p_2)$ is of type 1,2 or 4 and so g_e is of type 1,2 or 4.

By Lemma 3.47 g_i is safely unknottable with $g_{i,\bar{k}} \leftrightarrows \gamma(p_1)$. By Lemma 3.15 g_i is a rum and by Lemma 3.24 the unknotting $g_{i,\bar{k}}$ of g_i is a rum too. By the operational semantics of BPA$^*(Act)$ $p_1 \not\leftrightarrows {p_1}^*\epsilon$ and hence $g_i \neq g_{i,\bar{k}}$. By the I.H. for p_1, $g_{i,\bar{k}}$ is of type 1,2 or 4 and so g_i is of type 3.

Together this implies that g is of type 2.

It remains to be proved that rum's of graph interpretations of proper derivatives of p are of type 1,2 or 4. If ${p_1}' \cdot {p_1}^*p_2$ is a derivative of p, such that $p_1 \rightarrow^+ {p_1}'$, then by the operational semantics of BPA$^*(Act)$ p is returning via ${p_1}' \cdot {p_1}^*p_2$ and ${p_1}' \cdot {p_1}^*p_2$ is returning. This implies that if a rum of $\gamma({p_1}' \cdot {p_1}^*p_2)$ is not of type 2, it is of type 4.

The proof is concluded with the proper derivatives of ${p_1}^*p_2$, which are also proper derivatives of p_2. By the I.H. rum's of their graph interpretations are of type 1,2 or 4 □

Collecting the results of this section we obtain.

COROLLARY 3.53. Let g be a finite process graph. Then g is a Σ(BPA$^*(Act)$) expressible process graph iff g has the $\star$ property.

In Section 5 we prove that the $\star$ property is decidable for finite process graphs.

COROLLARY 3.54. Corollary 3.53 generalizes to the unary interpretation of Kleene star.

Proof. It is enough to consider the following adaptation of the $\star$ property:

Let g be a finite process graph. Then g has the *unary* $\star$ *property* iff g has a non trivial typable rum.
□

4. A Criterion for Process Graphs expressible with Regular Expressions definable *with* ϵ: the $\star_\epsilon$ property

In this section we study the process graphs that are expressible with process expressions over $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$, i.e. with the extra constant ϵ. We develop the $\star_\epsilon$ property on finite process graphs, which we prove to correspond with expressibility of a regular expression with ϵ modulo bisimulation equivalence. The $\star_\epsilon$ property has a strong resemblance to the $\star$ property. It requires subtle changes to the previously seen auxiliary graph operations exit part, iterative part and unknotting, now called *ϵ-exit part*, *ϵ-iterative part* and *ϵ-unknotting*.

We have omitted most proofs, since they are repetitions of Section 3. We only present proofs if we feel that they are substantially more difficult than in the case without ϵ. For convenience we have put the number of the Propositions etc. in $\Sigma(\mathrm{BPA}^*(Act))$ in parentheses, if an equivalent fact is stated in Section 3.

Auxiliary graph operations for a criterion for $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressible process graphs

DEFINITION 4.1. *(3.2)* Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum.

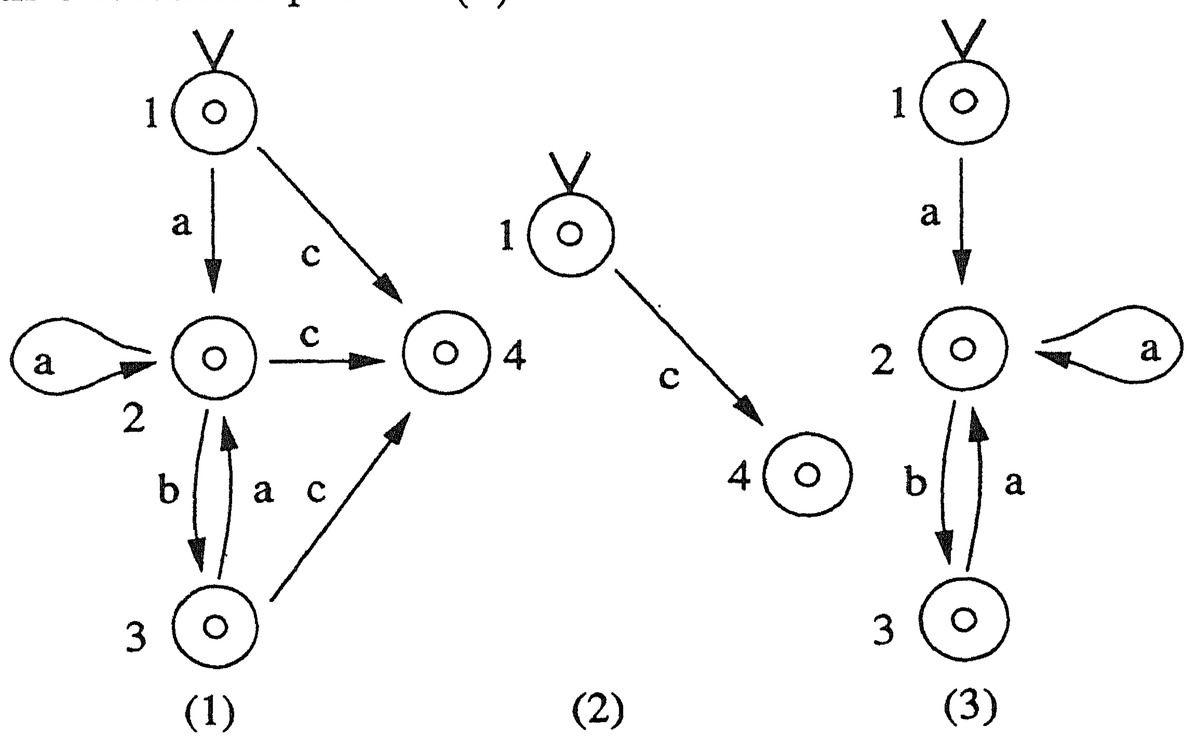
1. If $g_{\epsilon e} = (r, S, \downarrow_{\epsilon e}, \rightarrow_{\epsilon e})$ is a finite process graph where $\downarrow_{\epsilon e} \subseteq \downarrow$ and $\rightarrow_{\epsilon e} \subseteq \rightarrow$ are the smallest sets such that
   (a) if $r \xrightarrow{a} s$ and $s \not\rightarrow^* s'$, such that [whenever $r \xrightarrow{b} s''$, $s' \xrightarrow{b} s''$], then $r \xrightarrow{a}_{\epsilon e} s$, and
   (b) if $r \rightarrow_{\epsilon e}{}^+ s$ and $s \xrightarrow{a} s'$, then $s \xrightarrow{a}_{\epsilon e} s'$, and
   (c) if $r \rightarrow_{\epsilon e}{}^* s$ and $s \downarrow$, then $s \downarrow_{\epsilon e}$,

   then $g_{\epsilon e}$ is *an ϵ-exit part of g*.
2. If $g_{\epsilon i} = (r, S, \downarrow_{\epsilon i}, \rightarrow_{\epsilon i})$ is a finite process graph where $\downarrow_{\epsilon i} \supseteq Rt_g$ and $\rightarrow_{\epsilon i} \subseteq \rightarrow$ are the smallest sets such that
   (a) if $r \rightarrow_{\epsilon i}{}^* s$, $s \xrightarrow{a} s'$, $s \downarrow_{\epsilon i}$ and $r \not\xrightarrow{a}_{\epsilon e} s'$, then $s \xrightarrow{a}_{\epsilon i} s'$, and
   (b) if $r \rightarrow_{\epsilon i}{}^+ s$, $s \not\downarrow_{\epsilon i}$ and $s \xrightarrow{a} s'$, then $s \xrightarrow{a}_{\epsilon i} s'$, and
   (c) if $r \rightarrow_{\epsilon i}{}^* s$ and [whenever $r \xrightarrow{a} s'$, then $s \xrightarrow{a} s'$], then $s \downarrow_{\epsilon i}$,

   then $g_{\epsilon i}$ is *an ϵ-iterative part of g*.

REMARK 4.2. Notice that we can safely omit that $\downarrow_{\epsilon i} \supseteq Rt_g$.

EXAMPLE 4.3. In the picture below, (2) is an ϵ-exit part and (3) is an ϵ-iterative part of (1).



(1) (2) (3)

Notice that (1)–(3) are rum's of graph interpretations of respectively $(a\cdot(b+\epsilon))^*(c+\epsilon)$, $c+\epsilon$ and $(a\cdot(b+\epsilon))^*\epsilon$.

REMARK 4.4. The definition of an ϵ-exit and ϵ-iterative part is subtly different from Definition 3.2:

1. an ϵ-exit part can now have a root with a termination label, and
2. an ϵ-iterative part can have more termination labels than the set Rt_g.

PROPOSITION 4.5. *(3.5)* An ϵ-exit part and an ϵ-iterative part of a finite rum are finite rui process graphs.

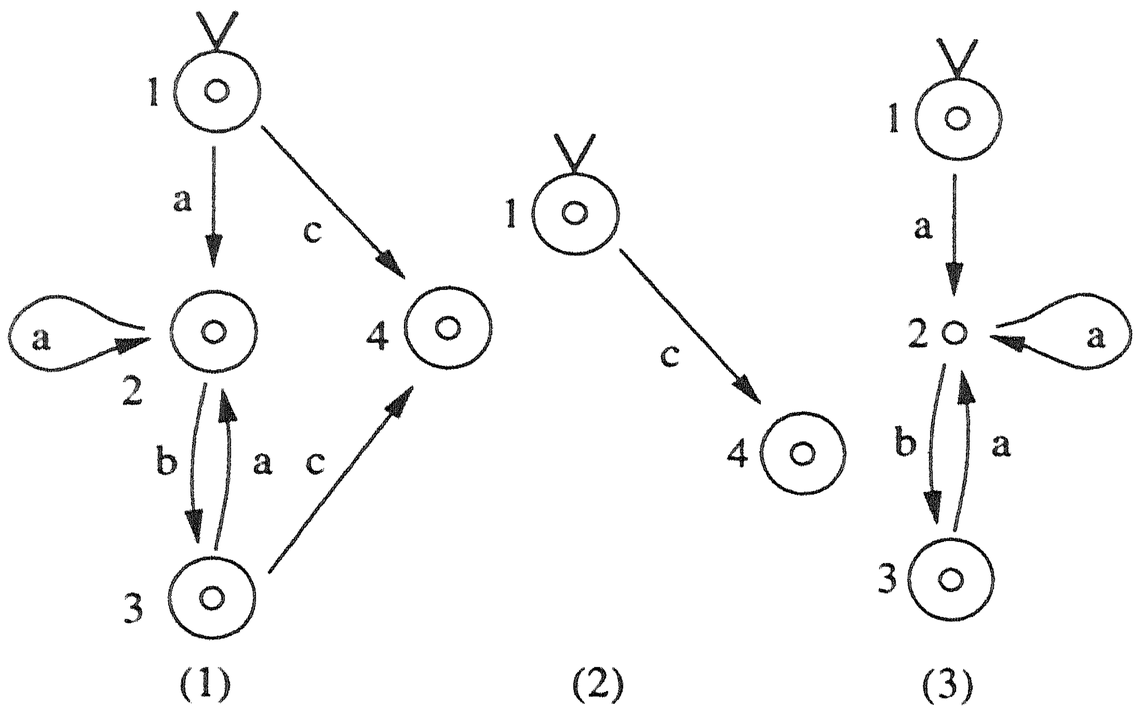
PROPOSITION 4.6. *(3.6)* An ϵ-exit part and an ϵ-iterative part of a finite rum are unique.

REMARK 4.7. Henceforth we speak of *the* ϵ-exit part and *the* ϵ-iterative part.

PROPOSITION 4.8. *(3.8)* Let g be a finite rum and $g_{\epsilon e}$ and $g_{\epsilon i}$ respectively the ϵ-exit and ϵ-iterative part of g. Then $R_g = R_{g_{\epsilon e}} \cup R_{g_{\epsilon i}}$.

DEFINITION 4.9. *(3.9)* Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum and $g_{\epsilon e} = (r, S, \downarrow_{\epsilon e}, \rightarrow_{\epsilon e})$ and $g_{\epsilon i} = (r, S, \downarrow_{\epsilon i}, \rightarrow_{\epsilon i})$ respectively the ϵ-exit and the ϵ-iterative part of g. If $g_{\epsilon i}$ is strongly normed and [whenever $s \in R_{g_{\epsilon i}}$, then $s\downarrow_{\epsilon i}$ and $r\downarrow_{\epsilon e}$ iff $s\downarrow$], then g is *ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e}$ and the ϵ-iterative part $g_{\epsilon i}$.*

EXAMPLE 4.10. The process graph (1) in Example 4.3 is ϵ-safely decomposable in the ϵ-exit part (2) and the ϵ-iterative part (3). The process graph (1) depicted below, is not ϵ-safely decomposable in (1) and (2).



(1) (2) (3)

(1) is *in*expressible over $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$; notice that rum's of graph interpretations of (2)–(3) are expressible as $\epsilon + c$, $(a \cdot (a^*b))^*\epsilon$.

REMARK 4.11. A rum can still be ϵ-safely decomposable if its ϵ-iterative part $g_{\epsilon i}$ is such that $\downarrow \cap R_{g_{\epsilon i}} \neq \emptyset$, whereas if g is safely decomposable and g_i is the iterative part, then $\downarrow \cap R_{g_i} = \emptyset$.

PROPOSITION 4.12. If g is safely decomposable in the exit part g_e and the non trivial iterative part g_i, then g is ϵ-safely decomposable with the ϵ-exit part g_e and the ϵ-iterative part g_i.

LEMMA 4.13. *(3.11)* Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If g is ϵ-safely decomposable in the ϵ-exit $g_{\epsilon e} = (r, S, \downarrow_{\epsilon e}, \rightarrow_{\epsilon e})$ and the ϵ-iterative part $g_{\epsilon e} = (r, S, \downarrow_{\epsilon i}, \rightarrow_{\epsilon i})$, then $R_{g_{\epsilon i}} \cap R_{g_{\epsilon e}} = \{r\}$, $\downarrow_{\epsilon e} \cap \downarrow_{\epsilon i} \subseteq \{r\}$ and $\rightarrow_{\epsilon e} \cap \rightarrow_{\epsilon i} = \emptyset$.

Proof. Let g, $g_{\epsilon e}$ and $g_{\epsilon i}$ be as in the premise. Suppose towards a contradiction that there is a $s' \neq r \in R_g$ such that $[s' \in R_{g_{\epsilon e}} \cap R_{g_{\epsilon e}}$ or $s' \in \downarrow_{\epsilon e} \cap \downarrow_{\epsilon i}$ or $s \xrightarrow{a} s' \in \rightarrow_{\epsilon e} \cap \rightarrow_{\epsilon i}]$. Observe first that if $s \rightarrow_{\epsilon e}{}^+ s'$, then there is no transition sequence $s' \rightarrow^* t$ such that whenever $r \xrightarrow{a} t'$, $t \xrightarrow{a} t'$. Observe second that if $r \rightarrow_{\epsilon i}{}^+ s'$, then by the assumption that g_i is strongly normed there is a transition sequence $s' \rightarrow_{\epsilon i}{}^* t''$ and $t'' \downarrow_{\epsilon i}$. By definition of an iterative part whenever $r \xrightarrow{a} t'''$, $t'' \xrightarrow{a} t'''$. Also by definition of an iterative part $s' \rightarrow^* t''$ and hence by definition of an

ϵ-exit part $s' \rightarrow^* {}_{\epsilon e}t''$. Contradiction with the definition of an ϵ-exit part $\square$

LEMMA 4.14. Let g be a finite rum. If g is ϵ-safely decomposable in the non trivial ϵ-exit part $g_{\epsilon e}$ and the ϵ-iterative part $g_{\epsilon i}$ and is returning, then

1. $r \rightarrow^* s \rightarrow^* s'$ for some $s' \in Rt_g$ iff $s \in R_{g_{\epsilon i}}$, and
2. if $s, s' \in R_{g_{\epsilon i}}$ and $s \xrightarrow{a} s'$, then $s \xrightarrow{a}_{\epsilon i} s'$, and
3. $s \in R_{g_{\epsilon i}}$ and there is a $t \in R_{g_{\epsilon e}}$ such that $s \xrightarrow{a} t$ iff $s \downarrow_{\epsilon i}$.

Proof. Let $g = (r, S, \downarrow, \rightarrow)$, $g_{\epsilon i} = (r, S, \downarrow_{\epsilon i}, \rightarrow_{\epsilon i})$ be as in the premise. From the assumption that g is returning follows that the ϵ-iterative part $g_{\epsilon i}$ is not trivial.

1. ($\Rightarrow$) Immediate by Definition 4.1 and Propositions 4.8, 4.13.
   ($\Leftarrow$) Suppose $s \in R_{g_{\epsilon i}}$. By the assumption that $g_{\epsilon i}$ is strongly normed, there is a s' such that $s \rightarrow_{\epsilon i}{}^* s'$ and $s' \downarrow_{\epsilon i}$. By definition of g $r \xrightarrow{a} t \rightarrow^* s''$ and $s'' \in Rt_g$. By definition of an ϵ-iterative part $r \rightarrow^* s$, $s \rightarrow^* s'$ and $s' \xrightarrow{a} t$. Hence $r \rightarrow^* s \rightarrow^* s''$ and $s'' \in Rt_g$.
2. Suppose $s, s' \in R_{g_{\epsilon i}}$ and $s \xrightarrow{a} s'$. By assumption $r \rightarrow_{\epsilon i}{}^* s$. By Lemma 4.13 $s' \notin R_{g_{\epsilon e}}$. This implies that $r \not\xrightarrow{a}_{\epsilon e} s'$. By definition of an ϵ-iterative part $s \xrightarrow{a}_{\epsilon i} s'$.
3. ($\Leftarrow$) Immediate by Definition 4.1 and the assumption that $g_{\epsilon e}$ is not trivial.
   ($\Rightarrow$) Suppose towards a contradiction that $s \in R_{g_{\epsilon i}}$, $t \in R_{g_{\epsilon e}}$ and $s \xrightarrow{a} t$, but $s \not\downarrow_{\epsilon i}$. By definition of an ϵ-iterative part $s \xrightarrow{a}_{\epsilon i} t$ and so $t \neq r \in R_{g_{\epsilon i}}$. Contradiction with Lemma 4.13 $\square$

PROPOSITION 4.15. *(3.14)* Let g be a finite rum. If g is ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e}$ and the ϵ-iterative part $g_{\epsilon i}$ and $g_{\epsilon e'} \cong g_{\epsilon e}$ is a finite process graph such that $R_{g_{\epsilon e'}} \cap R_g = \emptyset$, then $g_{\epsilon i} \cdot g_{\epsilon e'} \cong g$.

PROPOSITION 4.16. *(3.15)* Let g be a finite rum. If g is ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e}$ and the ϵ-iterative part $g_{\epsilon i}$, then $g_{\epsilon e}$ and $g_{\epsilon i}$ are rum's.

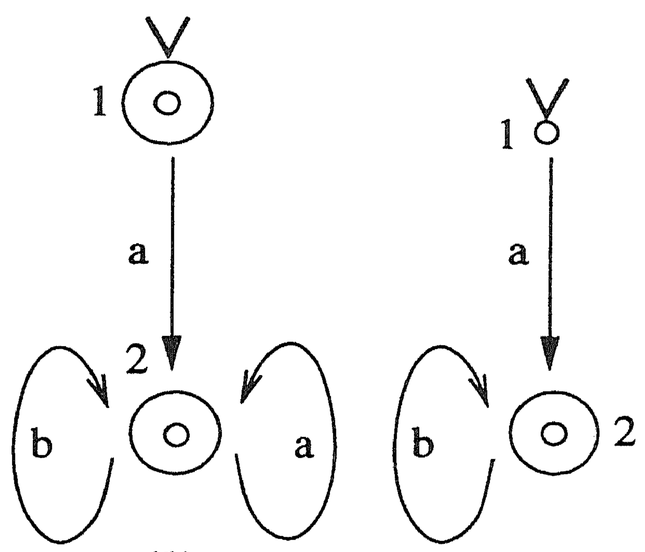
DEFINITION 4.17. *(3.16)* Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If $g_{\epsilon\bar{k}} = (r, S, \downarrow_{\epsilon\bar{k}}, \rightarrow_{\epsilon\bar{k}})$ is a process graph where $\downarrow_{\epsilon\bar{k}} = \downarrow - \{r\}$ and $\rightarrow_{\epsilon\bar{k}} \subseteq \rightarrow$ are the smallest sets such that

1. if $r \xrightarrow{a} s$, then $r \xrightarrow{a}_{\epsilon\bar{k}} s$, and

2. if $s \xrightarrow{a} s'$, $s \downarrow$ and $r \not\xrightarrow{a}_{\epsilon\bar{k}} s'$, then $s \xrightarrow{a}_{\epsilon\bar{k}} s'$, and
3. if $s \xrightarrow{a} s'$ and $s \not\downarrow$, then $s \xrightarrow{a}_{\epsilon\bar{k}} s'$,

then $g_{\epsilon\bar{k}}$ is *an ϵ-unknotting of g.*

EXAMPLE 4.18. The process graph depicted below by (2) is an ϵ-unknotting of (1).



(1) (2)

We leave it to the reader to verify that (2) is not an unknotting of (1).
Notice that (1)–(2) are rum's of graph interpretations of $a \cdot (a+b)^*\epsilon$ and $a \cdot (b^*\epsilon)$.

PROPOSITION 4.19. *(3.18)* Let g be a finite rum. If $g_{\epsilon\bar{k}}$ is an ϵ-unknotting of g, then $g_{\epsilon\bar{k}}$ is a finite rui process graph.

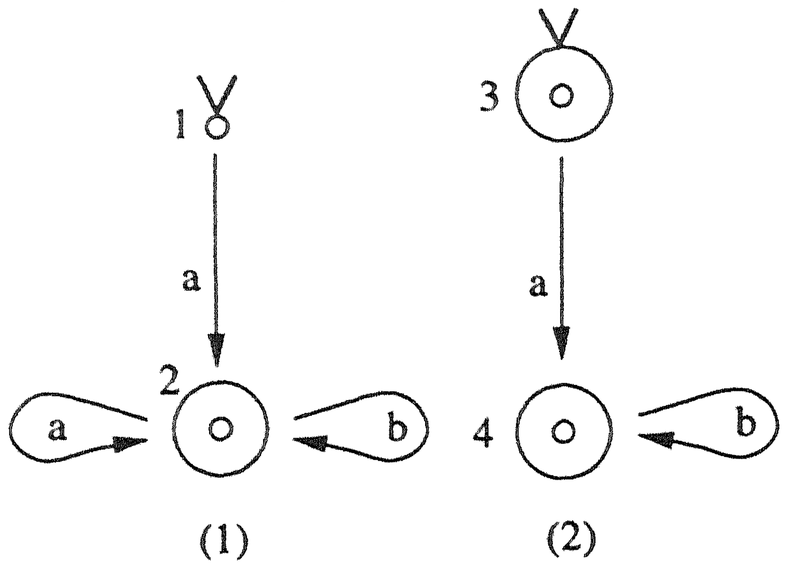
PROPOSITION 4.20. *(3.19)* An ϵ-unknotting of a finite rum is unique.

REMARK 4.21. Henceforth we speak of *the* ϵ-unknotting.

PROPOSITION 4.22. Let g be a finite rum. If $g_{\epsilon\bar{k}}$ is the ϵ-unknotting of g, then]whenever $g_{\epsilon\bar{k}}$ is strongly normed, g is strongly normed].

DEFINITION 4.23. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If $g_{\epsilon\bar{k}} = (r, S, \downarrow_{\epsilon\bar{k}}, \rightarrow_{\epsilon\bar{k}})$ is the ϵ-unknotting of g such that $g_{\epsilon\bar{k}}$ is strongly normed, $r \downarrow$ and [if $s \downarrow$ and $r \xrightarrow{a} s'$, then $s \xrightarrow{a} s'$], then g is *ϵ-safely unknottable with $g_{\epsilon\bar{k}}$.*

EXAMPLE 4.24. The process graph (1) in Example 4.18 is ϵ-safely unknottable with (2). The process graphs (1) and (2) depicted below are not ϵ-safely unknottable.



Notice that (1)–(2) are the rum's of graph interpretations of $a\cdot(a+b)^*\epsilon$, $\epsilon+a\cdot(b^*\epsilon)$.

PROPOSITION 4.25. If g is safely unknottable with the unknotting $g_{\bar{k}}$, then g is ϵ-safely unknottable with the ϵ-unknotting $g_{\bar{k}}$.

LEMMA 4.26. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If g is ϵ-safely unknottable with $g_{\epsilon\bar{k}}$ and returning, then for all $s \in R_g$ $s \rightarrow^* s'$ for some $s' \in Rt_g$.

Proof. Let g and s be as in the premise. By Proposition 4.22 g is strongly normed and so there is a t such that $s \rightarrow^* t$ and $t \downarrow$. By assumption that g is returning there is a $s' \neq r \in Rt_g$ and t' such that $r \xrightarrow{a} t'$ and $t' \rightarrow^* s'$. By the assumption that g is ϵ-safely unknottable $t \xrightarrow{a} t'$. Hence $s \rightarrow^* s'$ for some $s' \in Rt_g$ □

PROPOSITION 4.27. *(3.23)* Let g be a rum. If g is ϵ-safely unknottable and $g_{\epsilon\bar{k}}$ is an ϵ-unknotting of g, then the knotting of $g_{\epsilon\bar{k}}$ is g.

PROPOSITION 4.28. *(3.24)* Let g be a finite rum. If g is ϵ-safely unknottable with $g_{\epsilon\bar{k}}$, then $g_{\epsilon\bar{k}}$ is a rum.

A criterion for $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ *expressible process graphs*

We now present a classification of finite rum's in order to define the criterion that characterizes the $\Sigma(\mathrm{BPA}^*(Act))$ expressible process graphs.

DEFINITION 4.29. *(3.25)* Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum.

1. g is of type 1_ϵ iff
   (a) g is terminating, and
   (b) g is not returning, and
   (c) the root unwindings of all children of g, if any, are of type $1_\epsilon - 4_\epsilon$.
2. g is of type 2_ϵ iff g is returning and ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e}$ and the ϵ-iterative part $g_{\epsilon i}$ and
   (a) $g_{\epsilon e}$ is not trivial and is of type $1_\epsilon - 4_\epsilon$, and
   (b) $g_{\epsilon i}$ is not trivial and is of type 3_ϵ.
3. g is of type 3_ϵ iff g is returning and is ϵ-safely unknottable with $g_{\epsilon\bar{k}}$ of type $1_\epsilon, 2_\epsilon$ or 4_ϵ.
4. g is of type 4_ϵ iff g is returning via a derivative $g' \not\trianglelefteq g$, which has a root unwinding of type 2_ϵ or 3_ϵ.

LEMMA 4.30. *(3.29)* Let g be a finite rum.

1. If g is of type 2_ϵ, then g is not returning via a derivative which has a root unwinding $g' \not\trianglelefteq g$ of type 2_ϵ.
2. If g is of type 3_ϵ, then g has no derivative with a root unwinding $g' \not\trianglelefteq g$, which is of type 3_ϵ.
3. g is maximally of one type 1_ϵ–4_ϵ.

Proof. Let $g = (r, S, \downarrow, \rightarrow)$ be as in the premise.

1. Let g be of type 2_ϵ. Suppose towards a contradiction that g is returning via a derivative $g' \not\trianglelefteq g$ which has a root unwinding $g' = (r', S, \downarrow', \rightarrow')$ of type 2_ϵ. By supposition $R_g - \{r\} = R_{g'} - \{r'\}$ and there are $s \neq r \in Rt_g$ and $s' \neq r' \in Rt_{g'}$. By supposition g is ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e}$ and ϵ-iterative part $g_{\epsilon i} = (r, S, \downarrow_{\epsilon i}, \rightarrow_{\epsilon i})$ and similarly g' is ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e'}$ and ϵ-iterative part $g_{\epsilon i'} = (r', S, \downarrow_{\epsilon i'}, \rightarrow_{\epsilon i'})$. We prove only $R_{g_{\epsilon i}} - \{r\} \subseteq R_{g_{\epsilon i'}} - \{r'\}$, the symmetric inclusion has a similar proof. Suppose $t \neq r \in R_{g_i}$. By the supposition that g is returning via g', g' is also returning via g. This implies $s' \rightarrow^* s \rightarrow^* t \rightarrow^* s \rightarrow^* s'$. By Lemma 4.14 and since g' is rui $t \in R_{g_{\epsilon i'}} - \{r'\}$. By Lemmas 4.13 and 4.14 now follows $\{s \xrightarrow{a}_{\epsilon i} s' | s \neq r\} = \{s \xrightarrow{a}_{\epsilon i'} s' | s \neq r'\}$, $R_{g_{\epsilon e}} - \{r\} = R_{g_{\epsilon e'}} - \{r'\}$ and $\downarrow_{\epsilon i} - \{r\} = \downarrow_{\epsilon i'} - \{r'\}$. This implies that $g_{\epsilon i'}$ is a root unwinding of a derivative of $g_{\epsilon i}$. By supposition $g_{\epsilon i'}$ and $g_{\epsilon i}$ are of type 3_ϵ. Now use the proof for Part 2 to derive a contradiction.

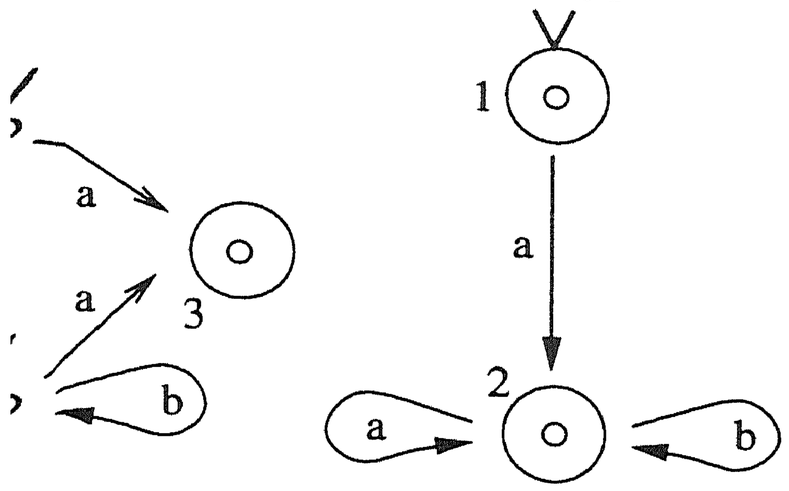
2. Let g be of type 3_ϵ and $g' = (r', S, \downarrow', \rightarrow') \not\leftrightarrows g$ be the root unwinding of a derivative of g. Suppose towards a contradiction that g' is of type 3_ϵ. By definition of type 3_ϵ both g and g' are returning. This implies that there are s, s' such that $s \neq r \in Rt_g$ and $s' \neq r' \in Rt_{g'}$. Also by definition of type 3_ϵ g and g' are ϵ-safely unknottable and so $s\downarrow$, $s'\downarrow$ and [whenever $s \xrightarrow{a} t$, $s' \xrightarrow{a} t$] and whenever $s' \xrightarrow{a} t'$, $s \xrightarrow{a} t'$. Hence s, s' are roots of bisimilar derivatives of g. Contradiction with the supposition that $g' \not\leftrightarrows g$.
3. Suppose g is of type 1_ϵ. By definition of type 1_ϵ g is not returning. If g is also of type 2_ϵ, 3_ϵ or 4_ϵ, then g is returning. Contradiction.

   Suppose g is of type 2_ϵ and ϵ-safely decomposable in the ϵ-exit part $g_{\epsilon e}$ and the ϵ-iterative part $g_{\epsilon i}$.
   (a) Suppose g is of type 3_ϵ. By Lemmas 4.26 and 4.14 $R_g = R_{g_{\epsilon i}}$. So by Proposition 4.13 $R_{g_{\epsilon e}} = \{r\}$. Contradiction with the definition of type 2_ϵ that $g_{\epsilon e}$ is not trivial.
   (b) Suppose g is of type 4_ϵ. Distinguish two cases (0) g is returning via a derivative which has a root unwinding $g' \not\leftrightarrows g$ of type 2_ϵ and (1) g is returning via a derivative which has a root unwinding $g' \not\leftrightarrows g$ of type 3_ϵ. (0) If g is returning via a derivative which has a root unwinding $g' \not\leftrightarrows g$ of type 2_ϵ, then Part 1 of this lemma establishes that g' is not of type 2_ϵ.
   (1) If g is returning via a derivative which has a derivative $g' \not\leftrightarrows g$ of type 3_ϵ, then with the same argument as in Part 3a we derive a contradiction.

   Suppose g is of type 3_ϵ. The same argument as for case 3a establishes that g cannot be of type 3_ϵ while being returning via derivative which has a root unwinding of type 2_ϵ. Part 2 of this lemma establishes that g cannot be of type 4_ϵ while being returning via derivative which has a root unwinding of type 3_ϵ □

REMARK 4.31. *(3.26)* Notice that the definitions of the $\star$ and $\star_\epsilon$ property are very similar. There are however a few differences: a rum can now be of type 1_ϵ if it is intermediately terminating at the root. We use this to include process graphs expressible with non returning expressions $p + \epsilon$. Furthermore a rum can also be of type 4_ϵ if it is a derivative of a rum of type 3_ϵ. We use this to include process graphs expressible with $p' \cdot p^*\epsilon$.

proviso that process graphs of type 2_ϵ and 3_ϵ are returning is ed to insure that the ϵ-types are unique.



(2)

. (1)–(2) are the rum's of the graph interpretations of respectively $a\cdot(a+$ $\epsilon+a\cdot(a+b)^*\epsilon$ (but also respectively of $(a\cdot(b+\epsilon))^*a$ and $(a\cdot(b+\epsilon))^*\epsilon$).

drop the demand for type 2_ϵ and type 3_ϵ that rum's are re- hen in the picture above (1) is of type 1_ϵ *and* type 2_ϵ, and (2) *and* type 3_ϵ, as the reader can verify.

n now present the criterion for the case with ϵ, which we have the $\star_\epsilon$ property.

ITION 4.32. *(3.30)* Let g be a finite process graph. g has perty iff g has a rum of type $1_\epsilon - 4_\epsilon$.

OSITION 4.33. Let g be a finite process graph. If g has the $\star$ then g has the $\star_\epsilon$ property.

stpone the proof that the $\star_\epsilon$ property is decidable to Section 5.

property identifies the process graphs expressible by $\Sigma(\mathrm{BPA}^*_\epsilon(Act)$

OSITION 4.34. *(3.34)* Let g be a rum and $h=(r,S,\downarrow_h,\rightarrow_h)$ ation of g. If g is ϵ-safely decomposable in the non trivial ϵ- $_{\epsilon e}$ and the non trivial ϵ-iterative part $g_{\epsilon i}$ and $h_{\epsilon e}=(r,S,\downarrow_{h_{\epsilon e}}$ d $h_{\epsilon i}=(r,S,\downarrow_{h_{\epsilon i}},\rightarrow_{h_{\epsilon i}})$ are minimizations of respectively $g_{\epsilon e}$ en

$|\rightarrow_{h_{\epsilon e}}|<|\rightarrow_h|$, $1\leq|\rightarrow_{h_{\epsilon i}}|<|\rightarrow_h|$, and
$|\geq 2$.

OSITION 4.35. *(4.35)* Let g be a finite rum and $h=$ $\rightarrow_h)$ a minimization of g. If g is returning and $g_{\epsilon\bar{k}}$ is the

ϵ-unknotting of g and $h_{\bar{k}} = (r_{h_{\bar{k}}}, S, \downarrow_{h_{\bar{k}}}, \rightarrow_{h_{\bar{k}}})$ a minimization of $g_{\bar{k}}$, then $| \rightarrow_h | > | \rightarrow_{h_{\epsilon\bar{k}}} |$.

PROPOSITION 4.36. *(3.37)* Let g be a finite rum. If g is ϵ-safely unknottable and $g_{\epsilon\bar{k}}$ is an ϵ-unknotting of g and $\epsilon \notin R_{g_{\epsilon\bar{k}}}$, then $g \leftrightarrows {g_{\epsilon\bar{k}}}^* \gamma(\epsilon)$.

THEOREM 4.37. *(3.39)* If g is a finite process graph and has the $\star_\epsilon$ property, then g is $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressible.

Proof. The proof is similar to that of Theorem 3.39. Assume without loss of generalization that g is a rum. The Induction Hypothesis (I.H.) is that all rum's whose minimizations have less transitions than a minimization of g and are of type $1_\epsilon - 4_\epsilon$ are $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressible. In the Induction Step we only treat the deviating case that g is of type 3_ϵ.

-If g is of type 3_ϵ, then by definition g is ϵ-safely unknottable with $g_{\epsilon\bar{k}}$, which is of type 1_ϵ, 2_ϵ or 4_ϵ. By Proposition 4.35 and the I.H. $g_{\epsilon\bar{k}} \leftrightarrows \gamma(p)$ for some $p \in \mathbf{P}^*_\epsilon$. By Propositions 4.36 and 2.39 g is expressible as $p^*\epsilon \in \mathbf{P}^*_\epsilon$ □

The graph interpretations of process expressions over $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ *have the* $\star_\epsilon$ *property*

The proof of completeness of the $\star_\epsilon$ property for $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressible process graphs identical in structure to the proof for the $\star$ property for $\Sigma(\mathrm{BPA}^*(Act))$ expressible process graphs. There is however one major difference: In $\mathrm{BPA}^*(Act)$ we could use that all process expressions $p^*\epsilon$ are pure knottings [12]. We prove that we can replace all *im*pure knottings with a bisimilar pure knotting with a less or equal symbol complexity.

First we prove that we can replace any process expression $p^*\epsilon \in \mathbf{P}^*_\epsilon$ with a bisimilar process expression $q^*\epsilon$ with the same or smaller symbol complexity such that $q \not\downarrow$. Second we prove that we can replace the thus obtained obtained process expressions $q^*\epsilon$ with a bisimilar pure knotting with a smaller or equal symbol complexity.

---

[12] All process expressions $p^*\epsilon$, where $p \in \mathbf{P}^*$ are *pure* knottings by Lemma 3.45.

LEMMA 4.38. Let $p \in \mathbf{P}^*_\epsilon$. If $p \downarrow$ and $p \not\leftrightarrows \epsilon$, then there is a $q \in \mathbf{P}^*_\epsilon$, such that $p^*\epsilon \leftrightarrows q^*\epsilon$, $q \not\downarrow$ and $\#(q) < \#(p)$.

Proof. Let p be as in the premise. We prove the result by a well-founded induction on the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that for all p', such that $\#(p') < \#(p)$ and $p' \downarrow$, there is a q' such that $p'^*\epsilon \leftrightarrows q'^*\epsilon$, $q' \not\downarrow$ and $\#(q') < \#(p')$. Distinguish the following cases in the Induction Step (I.S.).

1. If $p \equiv a$ for some $a \in Act$, then $p \not\downarrow$. Contradiction. If $p \equiv \epsilon$, then $p \leftrightarrows \epsilon$. Contradiction,
2. If $p \equiv p_1 + p_2$ and $p \downarrow$, then by the operational semantics $p_1 \downarrow$ or $p_2 \downarrow$. Distinguish $(\frac{T}{F})$ whether or not $[p_1 \leftrightarrows \epsilon$ or $p_2 \leftrightarrows \epsilon]$. (T) If $p_1 \leftrightarrows \epsilon$ or $p_2 \leftrightarrows \epsilon$, then assume without loss of generality that $p_1 \leftrightarrows \epsilon$. By the operational semantics $p^*\epsilon \leftrightarrows {p_2}^*\epsilon$. By the I.H. the result follows.

   (F) If not $[\ p_1 \leftrightarrows \epsilon$ or $p_2 \leftrightarrows \epsilon\ ]$, then assume without loss of generality that $p_1 \downarrow$. By the I.H. there is a ${p_1}'$ such that ${p_1}'^*\epsilon \leftrightarrows {p_1}^*\epsilon$, ${p_1}' \not\downarrow$ and $\#({p_1}') < \#(p_1)$ and hence $\#(({p_1}' + p_2)^*\epsilon) < \#(p)$. Since bisimulation is a congruence for the operators of $\mathrm{BPA}^*_\epsilon(Act)$ $p^*\epsilon \leftrightarrows ({p_1}' + p_2)^*\epsilon$. The result follows by the I.H..
3. If $p \equiv p_1 \cdot p_2$, then by the operational semantics $p_1 \downarrow$ and $p_2 \downarrow$. This implies that $p^*\epsilon \leftrightarrows (p_1 + p_2)^*\epsilon$. The result follows with the case in the I.S. with $+$ as principal operator.
4. If $p \equiv {p_1}^*p_2$, then by the operational semantics $p_2 \downarrow$. This implies also that $p^*\epsilon \leftrightarrows (p_1 + p_2)^*\epsilon$. The result follows with the case in the I.S. with $+$ as principal operator $\Box$

For the proof that we can replace an *im*pure knotting with a pure knotting with the same or smaller symbol complexity we use a "position" labeling map λ_e and an *un*labeling map $\bar{\lambda}$.

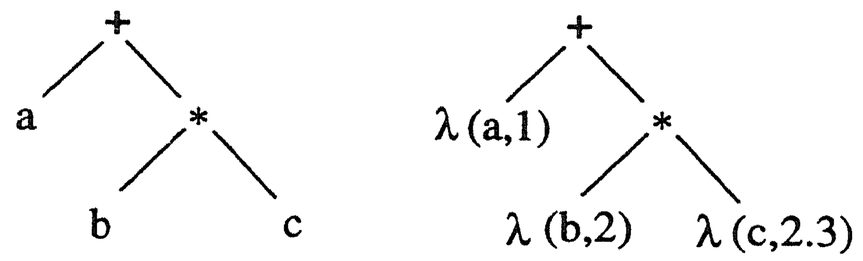
DEFINITION 4.39. Let $\lambda : Act \times \mathbf{N} \to Act$ be a map. Let $\lambda_{e,n,\lambda} : \mathbf{P}^*_\epsilon \times \mathbf{N} \times \mathbf{N} \to \mathbf{P}^*_\epsilon$, $\lambda_{e,\lambda} : \mathbf{P}^*_\epsilon \to \mathbf{P}^*_\epsilon$ be maps defined as follows.

$$\lambda_{e,\lambda}(p) = \lambda_{e,n,\lambda}(p,1,1)$$

$$\begin{aligned}
\lambda_{e,n,\lambda}(\epsilon, m, m') &= \epsilon \\
\lambda_{e,n,\lambda}(a, m, m') &= \lambda(a, m), \qquad a \in Act \\
\lambda_{e,n,\lambda}(p \oplus q, m, m') &= \lambda_{e,n,\lambda}(p, m, m'+1) \oplus \\
&\quad \lambda_{e,n,\lambda}(q, m.p_{m'}, m'+1)
\end{aligned}$$

where $p, q \in \mathbf{P}^*_\epsilon$, $\oplus \in \{+, \cdot, ^*\}$ and p_m is the m-th prime. The unlabeling map $\bar{\lambda} : \mathbf{P}^*_\epsilon \to \mathbf{P}^*_\epsilon$ is defined as $\bar{\lambda}(p) = \lambda_{e,\lambda'}(p)$, where $\lambda' : Act \times \mathbf{N} \to Act$ is a map such that $\lambda'(a, m) = a$.

REMARK 4.40. If $\lambda : Act \times \mathbf{N} \to Act$ is known from the context we write λ_e instead of $\lambda_{e,\lambda}$. Notice that $\lambda_e(p)$ is the process expression where the actions are labeled with a number denoting the "position" in the process expression: The parse tree of the process expression $a + (b + c)$ and $\lambda_e(a + (b + c)) = \lambda(a, 1) + (\lambda(b, 2) + \lambda(c, 6))$ is depicted as an example.



The "position" in the process expression can be read from the factorization in the second argument of λ.

LEMMA 4.41. Let $p \in \mathbf{P}^*_\epsilon$. If $\lambda_e(p) \to^* p' \xrightarrow{a} p''$, then a appears only once a subexpression of $\lambda(p)$.

Proof. The "position" of an action is unique □

DEFINITION 4.42. Let $p \in \mathbf{P}^*_\epsilon$. If $p^*\epsilon$ is an impure knotting, and a is an action such that

1. $\lambda_e(p^*\epsilon) \to^* p' \cdot \lambda_e(p^*\epsilon) \xrightarrow{a} p'' \cdot \lambda_e(p^*\epsilon)$ and $p' \xrightarrow{a} p''$ for some p', p'' and $p' \downarrow$, and
2. $\lambda_e(p^*\epsilon) \xrightarrow{b} p''' \cdot \lambda_e(p^*\epsilon)$ for some $b \in Act$, and
3. $\bar{\lambda}(a) = \bar{\lambda}(b)$ and $\bar{\lambda}(p'' \cdot \lambda_e(p^*\epsilon)) \leftrightarrows \bar{\lambda}(p''' \cdot \lambda_e(p^*\epsilon))$,

then a is an *impure knotting action of* $p^*\epsilon$, shortly an *impure knotting action.*

EXAMPLE 4.43. The process expression $p \equiv (a \cdot (a + \epsilon))^*\epsilon$ is an impure knotting. $\lambda_e(p)$ is $(\lambda(a,1) \cdot (\lambda(a,3) + \epsilon))^*\epsilon$ and an impure action of p is $\lambda(a,3)$.

PROPOSITION 4.44. Let $p \in \mathbf{P}^*_\epsilon$. If $p^*\epsilon$ is an impure knotting, then $p^*\epsilon$ has an impure knotting action.

LEMMA 4.45. Let $p \in \mathbf{P}^*_\epsilon$ and $a \in Act$. If there are $p', p'' \in \mathbf{P}^*$ such that $\lambda_e(p) \rightarrow^* p' \xrightarrow{a} p''$ and $p' \downarrow$, then whenever $\lambda_e(p) \equiv p_0 \rightarrow p_1 \ldots p_{n-1} \xrightarrow{a} p_n$ and there are no p_i, $i < n-1$ such that $p_i \xrightarrow{a} p_i'$, $p_{n-1} \downarrow$.

Proof. Let p and a be as in the premise. By Lemma 4.41 a appears only once in a subexpression of $\lambda_e(p)$, so if $p \equiv q_1 \oplus q_2$, then $\lambda_e(p) = \lambda_{e,n,\lambda}(q_1,1,2) \oplus \lambda_{e,n,\lambda}(q_2,2,2)$ for $\oplus \in \{+, \cdot, ^*\}$ by definition. The action a is a subexpression of $\lambda_e(q_1,1,2)$ or $\lambda_e(q_2,2,2)$ but not both. We prove the result by a well-founded induction on the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that for all q such that $\#(q) < \#(p)$, if there is a transition sequence $\lambda_e(q) \rightarrow^* q' \xrightarrow{a} q''$ and $q' \downarrow$, then whenever $\lambda_e(q) \equiv q_0 \rightarrow q_1 \ldots q_{n-1} \xrightarrow{a} q_n$ and there are no q_i, $i < n-1$ such that $q_i \xrightarrow{a} q_i'$, then $q_n \downarrow$. Distinguish the following cases in the Induction Step (I.S.).

1. If $p \equiv b, \epsilon$ for some $b \in Act$, then the premise does not hold,
2. If $p \equiv q_1 + q_2$, then assume without loss of generalization that a is a subexpression of $\lambda_{e,n,\lambda}(q_1,1,2)$. By the operational semantics and by assumption there is a transition sequence $\lambda_{e,n,\lambda}(q_1,1,2) \rightarrow^* p' \xrightarrow{a} p''$ and $p' \downarrow$. Since $\#(q_1) < \#(p)$ we can apply the I.H. to conclude that if $\lambda_{e,n,\lambda}(q_1,1,2) \equiv q_{1,0} \rightarrow q_{1,2} \ldots q_{1,n-1} \xrightarrow{a} q_{1,n}$ and there are no q_{1i}, $i < n-1$ such that $q_{1i} \xrightarrow{a} q_{1i}'$ for some $q_{1i}' \in \mathbf{P}^*_\epsilon$, then $q_{n-1} \downarrow$. The conclusion follows from the assumption that if $\lambda_e(p) \rightarrow^* p' \xrightarrow{a} p''$, then $\lambda_{e,n,\lambda}(q_1,1,2) \rightarrow^* p' \xrightarrow{a} p''$,
3. If $p \equiv q_1 \oplus q_2$, $\oplus \in \{\cdot, ^*\}$, then the case that a is a subexpression of $\lambda_{e,n,\lambda}(q_2,2,2)$ is identical to the previous case in the I.S.. If a is a subexpression of $\lambda_{e,n,\lambda}(q_2,1,2)$, then by the operational semantics and by assumption there is a transition sequence $\lambda_e(p) \rightarrow^* p' \xrightarrow{a} p''$ and $p' \downarrow$ such that $\lambda_{e,n,\lambda}(q_1,1,2) \rightarrow^* q' \xrightarrow{a} q''$, $p' \equiv q' \oplus \lambda_{e,n,\lambda}(q_2,2,2)$ and $q' \downarrow$. By the operational

semantics $\lambda_{e,n,\lambda}(q_2, 2, 2) \downarrow$. Suppose $\lambda_{e,n,\lambda}(q_1, 1, 2) \equiv q_{1,0} \rightarrow q_{1,2} \cdots q_{1,n-1} \xrightarrow{a} q_{1,n}$ and there are no q_{1i}, $i < n-1$ such that $q_{1i} \xrightarrow{a} q_{1i}'$ for some $q_{1i}' \in \mathbf{P}_\epsilon^*$ and $p_i \equiv q_{1,i} \oplus \lambda_{e,n,\lambda}(q_2, 2, 2)$. By the I.H. $q_{1,n-1} \downarrow$ and by the operational semantics $q_{1,n-1} \oplus \lambda_{e,n,\lambda}(q_2, 2) \downarrow$. Hence $p_{n-1} \downarrow$ □

LEMMA 4.46. Let $p \in \mathbf{P}_\epsilon^*$. If $p^*\epsilon$ is an impure knotting, $p \not\downarrow$, a is an impure knotting action of $p^*\epsilon$ and $\lambda_e(p^*\epsilon) \equiv p_0 \rightarrow p_1 \cdots p_{n-1} \xrightarrow{a} p_n$ and there are no p_i, $i < n-1$ such that $p_i \xrightarrow{a} p_i'$ for some $p_i' \in \mathbf{P}_\epsilon^*$, then there is an action $b \neq a \in Act$ such that

1. $\bar{\lambda}(a) = \bar{\lambda}(b)$, and
2. $\lambda_e(p) \xrightarrow{b} p'$, and
3. $\bar{\lambda}(p'') \leftrightarrows \bar{\lambda}(p' \cdot \lambda_e(p^*\epsilon))$.

Proof. Immediate by Definition 4.42 and Lemma 4.45 □

LEMMA 4.47. Let $p \in \mathbf{P}_{\delta,\epsilon}^*$. If p is strongly normed and has a single subexpression δ, then there is a $q \in \mathbf{P}_\epsilon^*$ such that $p \leftrightarrows q$ and $\#(q) < \#(p)$.

Proof. Let p be as in the premise. By assumption δ appears only once as a subexpression of p, so if $p \equiv p_1 \oplus p_2$, for $\oplus \in \{+, \cdot, ^*\}$, then δ is subexpression of p_1 or p_2 but not both.

We prove the result by an induction on the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that for all process expressions p' which are strongly normed, and have a single subexpression δ such that $\#(p') < \#(p)$, there is a $q' \in \mathbf{P}_\epsilon^*$ with $q' \leftrightarrows p'$ and $\#(q') < \#(p')$. Distinguish the following cases in the Induction Step (I.S.).

1. If $p \equiv a, \delta, \epsilon$ for some $a \in Act$, then the premise does not hold,
2. If $p \equiv p_1 \oplus p_2$ and $\oplus \in \{+, \cdot\}$, then assume without loss of generalization that δ is a subexpression of p_1. Since p is strongly normed, p_1 and p_2 are strongly normed. By the I.H. there is a $p_1' \in \mathbf{P}_\epsilon^*$ such that $\#(p_1') < \#(p_1)$ and $p_1' \leftrightarrows p_1$. Since bisimulation is a congruence $p_1' \oplus p_2 \leftrightarrows p_1 \oplus p_2$. Now apply the I.H. to $p_1' \oplus p_2$,
3. If $p \equiv p_1{}^* p_2$, then the case that δ is a subexpression of p_2 is identical to the I.S. for $+, \cdot$ as principal operator. If δ is a subexpression of p_1, then by Proposition 2.9 either p_1 is strongly normed or $p_1 \leftrightarrows \delta$. The case that p_1 is strongly normed is identical to

the I.S. for $+, \cdot$. If $p_1 \leftrightarrows \delta$, then by the operational semantics $p \leftrightarrows p_2$. Now apply the I.H. to p_2 $\square$

LEMMA 4.48. Let $p \in \mathbf{P}^*_\epsilon$. If $p \not\leftrightarrows \epsilon$, then there is a $q \in \mathbf{P}^*_\epsilon$, such that

1. $p^*\epsilon \leftrightarrows q^*\epsilon$, and
2. $q^*\epsilon$ is a pure knotting and $q \not\downarrow$, and
3. $\#(q^*\epsilon) \leq \#(p^*\epsilon)$.

Proof. Let p be as in the premise. We prove the result by a well-founded induction on the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that for all process expressions $p' \in \mathbf{P}^*_\epsilon$ such that $\#(p') < \#(p)$ and $p' \not\leftrightarrows \epsilon$, there is a $q' \in \mathbf{P}^*_\epsilon$, such that

1. $p'^*\epsilon \leftrightarrows q'^*\epsilon$, and
2. $q'^*\epsilon$ is a pure knotting and $q' \not\downarrow$, and
3. $\#(p'^*\epsilon) \leq \#(q'^*\epsilon)$.

Distinguish the following cases in the Induction Step (I.S.).

1. If $p \downarrow$, then the result follows by Lemma 4.38 and the I.H.,
2. If $p^*\epsilon$ is an impure knotting and $p \not\downarrow$, then by Proposition 4.44 $p^*\epsilon$ has an impure knotting action a. Let $C[\cdot]$ be the unary context such that $\lambda_e(p^*\epsilon) = C[a]^*\epsilon$. By Lemma 4.41 a appears only once a subexpression of $\lambda_e(p^*\epsilon)$. Now an important equivalence that holds is $\bar{\lambda}(C[a])^*\epsilon \leftrightarrows \bar{\lambda}(C[\delta \cdot a])^*\epsilon$. Basically this is a statement of the fact that the specific action a does not have to be used anymore: whenever there is choice between a transition *with* an impure knotting action a by Lemma 4.46 another transition can be chosen that goes to a bisimilar process expression, *without* using an impure knotting action. So in the process expression $p^*\epsilon$ the subexpression starting with a can be safely "inactivated" by prefixing δ.

   We assume without further proof that $\bar{\lambda}(C[\delta]) \leftrightarrows \bar{\lambda}(C[\delta \cdot a])$. By Proposition 2.8 $p^*\epsilon$ is strongly normed. Since $p^*\epsilon \not\leftrightarrows \epsilon$, $\bar{\lambda}(C[\delta]) \not\leftrightarrows \delta$ and and so by Proposition 2.9 $\bar{\lambda}(C[\delta])$ is strongly normed. By Lemma 4.47 there is a $q \in \mathbf{P}^*_\epsilon$, such that $\#(q) < \#(C[\delta])$ and $q \leftrightarrows \bar{\lambda}(C[\delta])$. Since bisimulation is a congruence $p^*\epsilon \leftrightarrows q^*\epsilon$. By the I.H. the result follows since $\#(q^*\epsilon) < \#(p^*\epsilon)$ $\square$

LEMMA 4.49. *(3.46)* Let $p_1{}^*p_2 \in \mathbf{P}^*_\epsilon$. If $p_1{}^*p_2$ is a pure iterator, $p_1{}^*\epsilon$ is a pure knotting and g is a rum of $\gamma(p_1{}^*p_2)$, then

1. the ϵ-exit part $g_{\epsilon e}$ of g is bisimilar to $\gamma(p_2)$ and the ϵ-iterative part $g_{\epsilon i}$ of g is bisimilar to $\gamma(p_1{}^*\epsilon)$, and
2. g is ϵ-safely decomposable in $g_{\epsilon e}$ and $g_{\epsilon i}$.

Proof. Let $g_{\epsilon e'} = (r, S, \downarrow_{\epsilon e'}, \rightarrow_{\epsilon e'})$ be the process graph where $\downarrow_{\epsilon e'} \subseteq \downarrow$ and $\rightarrow_{\epsilon e'} \subseteq \rightarrow$ are the smallest sets such that

1. if $p_2 \rightarrow^* p_2{}'$ and $p_2 \downarrow$, then $s \downarrow_{\epsilon e'}$, where s is the root of a derivative of g bisimilar to $\gamma(p_2{}')$, and
2. If $p_2 \xrightarrow{a} p_2{}'$, then $r \xrightarrow{a}_{\epsilon e'} s'$, where s is the root of the derivative of g bisimilar to $\gamma(p_2{}')$, and
3. If $p_2 \rightarrow^+ p_2{}'$ and $p_2{}' \xrightarrow{a} p_2{}''$, then $s \xrightarrow{a}_{\epsilon e'} s'$, where s, s' are the roots of the derivatives of g bisimilar to respectively $\gamma(p_2{}')$ and $\gamma(p_2{}'')$, and

let $g_{\epsilon i'} = (r, S, \downarrow_{\epsilon i'}, \rightarrow_{\epsilon i'})$ be the process graph where $\downarrow_{\epsilon i'}$ and $\rightarrow_{\epsilon i'} \subseteq \rightarrow$ are the smallest sets such that

1. $r \downarrow_{\epsilon i'}$, and
2. If $p_1{}^*\epsilon \rightarrow^+ p_1{}' \cdot p_1{}^*\epsilon$ and $p_1{}' \cdot p_1{}^*\epsilon \downarrow$, then $s \downarrow_{\epsilon i'}$, where $s \neq r$ is the root of the derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^*p_2)$, and
3. If $p_1{}^*\epsilon \xrightarrow{a} p_1{}' \cdot p_1{}^*\epsilon$, then $r \xrightarrow{a}_{\epsilon i'} s$, where $s \neq r$ is the root of the derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^*p_2)$, and
4. If $p_1{}^*\epsilon \rightarrow^+ p_1{}' \cdot p_1{}^*\epsilon$ and $p_1{}' \cdot p_1{}^*\epsilon \xrightarrow{a} p_1{}'' \cdot p_1{}^*\epsilon$, then $s \xrightarrow{a}_{\epsilon i'} s'$, where $s \neq r$, $s' \neq r$ are the roots of the derivatives of g bisimilar to respectively $\gamma(p_1{}' \cdot p_1{}^*p_2)$, $\gamma(p_1{}'' \cdot p_1{}^*p_2)$.

We use without further proof that $g_{\epsilon e'} \leftrightarrows \gamma(p_2)$ and $g_{\epsilon i'} \leftrightarrows \gamma(p_1{}^*\epsilon)$ are respectively the ϵ-exit part and ϵ-iterative part of g.

We only present the proof of (2).

By Proposition 2.8 $g_{\epsilon i'}$ is strongly normed.

Suppose $s \in R_{g_{\epsilon i'}}$. By definition of $g_{\epsilon i'}$, $s = r$ or s is the root of a derivative of g bisimilar to $\gamma(p_1{}' \cdot p_1{}^*p_2)$ and $p_1 \cdot p_1{}^*\epsilon \rightarrow^+ p_1{}' \cdot p_1{}^*\epsilon$.

($s \downarrow_{\epsilon i'}$ and $r \downarrow_{\epsilon e'} \Rightarrow s \downarrow$) Suppose $s \downarrow_{\epsilon i'}$ and $r \downarrow_{\epsilon e'}$. By definition of $g_{\epsilon e'}$ this implies that $p_1{}^*p_2 \downarrow$. Distinguish $(\frac{T}{F})$ whether or not $s = r$.
(T) If $s = r$, then by definition of g $s \downarrow$.

(F) If $s \neq r$, then by definition of $g_{\epsilon i'}$ $p_1{}' \cdot p_1{}^*\epsilon \downarrow$. By the operational semantics $p_1{}' \downarrow$ and hence $p_1{}' \cdot p_1{}^*p_2 \downarrow$. By definition of g $s \downarrow$.

($s \downarrow_{\epsilon i'}$ and $r \downarrow_{\epsilon e'} \Leftarrow s \downarrow$) Distinguish ($\frac{T}{F}$) whether or not $s = r$. (T) If $s = r$, then by definition of $g\ {p_1}^* p_2 \downarrow$. By the operational semantics $p_2 \downarrow$. By definition of $g_{\epsilon e'}\ r \downarrow_{\epsilon e'}$. By definition of $g_{\epsilon i'}$, $s \downarrow_{\epsilon i'}$.

(F) If $s \neq r$, then by the operational semantics ${p_1}' \cdot {p_1}^* p_2 \downarrow$. By the operational semantics ${p_1}' \downarrow$ and $p_2 \downarrow$, and so ${p_1}' \cdot {p_1}^* \epsilon \downarrow$. By definition of $g_{\epsilon e'}\ r \downarrow_{\epsilon e'}$ and by definition of $g_{\epsilon i'}\ s \downarrow_{\epsilon i'}$ □

LEMMA 4.50. *(3.47)* Let $p \in \mathbf{P}^*_\epsilon$. If $p^*\epsilon$ is a pure knotting and g a rum of $\gamma(p^*\epsilon)$, then

1. $\gamma(p)$ is bisimilar to an ϵ-unknotting $g_{\epsilon\bar{k}}$ of g,
2. g is ϵ-safely unknottable with $g_{\epsilon\bar{k}}$

Proof. Let $g_{\epsilon\bar{k}'} = (r, S, \downarrow_{\epsilon\epsilon\bar{k}'}, \rightarrow_{\epsilon\epsilon\bar{k}'})$ be a process graph where $\downarrow_{\bar{k}'} = \downarrow - \{r\}$ and $\rightarrow_{\epsilon\bar{k}'} \subseteq \rightarrow$ is the smallest set such that

1. if $p \xrightarrow{a} p'$ and $s \neq r$, then $r \xrightarrow{a}_{\epsilon\bar{k}'} s$, where s is the root of a derivative of g bisimilar to $\gamma(p' \cdot p^*\epsilon)$, and
2. if $p \rightarrow^+ p'$ and $p' \xrightarrow{a} p''$, then $s \xrightarrow{a}_{\epsilon\bar{k}'} s'$, where $s \neq r$, $s' \neq r$ are the roots of derivatives of g bisimilar to respectively $\gamma(p' \cdot p^*\epsilon)$ and $\gamma(p'' \cdot p^*\epsilon)$

We use without proof that $g_{\epsilon\bar{k}'}$ is the ϵ-unknotting of g and present only the proof of (2). By Proposition 2.8 p and so $g_{\epsilon\bar{k}'}$ is strongly normed. Now suppose $s \downarrow$ and $r \xrightarrow{a} s'$. We only treat the case that $s \neq r$. By definition of g, s is the root of the derivative of $\gamma(p^*\epsilon)$ bisimilar to $\gamma(p' \cdot p^*\epsilon)$, $p \rightarrow^+ p'$ and $p' \cdot p^*\epsilon \downarrow$. Similarly s' is the root of the derivative of $\gamma(p^*\epsilon)$ bisimilar to $\gamma(p'' \cdot p^*\epsilon)$, $p \xrightarrow{a} p''$ and since g is rui $s' \neq r$. By the operational semantics $p' \downarrow$ and so $p' \cdot p^*\epsilon \xrightarrow{a} p'' \cdot p^*\epsilon$. By definition of $g\ s \xrightarrow{a} s'$ □

THEOREM 4.51. *(3.52)* If $p \in \mathbf{P}^*_\epsilon$, then $\gamma(p)$ has the $\star_\epsilon$ property.

Proof. We prove the result by a well-founded simultaneous induction on the number of transitions m in a minimization of $\gamma(p)$ and the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that for all process expressions $p' \in \mathbf{P}^*_\epsilon$ with minimizations with m' transitions such that the tuple $(m', \#(p'))$ is lexicographically smaller than $(m, \#(p))$, a rum of $\gamma(p')$ is of type $1_\epsilon - 4_\epsilon$. Let g be a rum of $\gamma(p)$. In the Induction Step we only treat the case that $p \equiv {p_1}^*\epsilon$, the proof for the other cases is similar to the proof of Theorem 3.52. By Lemma

4.48 we can safely assume that ${p_1}^*\epsilon$ is a pure knotting.

If $p \equiv {p_1}^*\epsilon$ for some $p_1 \in \mathbf{P}^*_\epsilon$, then distinguish $(\frac{T}{F})$ whether or not $p \leftrightarrows p_1$. (T) If $p \leftrightarrows p_1$, then any derivative of g is of type $1_\epsilon - 4_\epsilon$ by the I.H..

(F) If $p_1 \not\leftrightarrows {p_1}^*\epsilon$, then distinguish $(\frac{T}{F})$ whether or not g is returning. (F.T) If g is returning, then by Lemma 4.50 g is ϵ-safely unknottable with $g_{\epsilon\bar{k}} \leftrightarrows \gamma(p_1)$. By Proposition 4.28 $g_{\epsilon\bar{k}}$ is a rum. By Lemma 4.35 a minimization of $g_{\epsilon\bar{k}}$ has $m' < m$ transitions and so $(m', \#({p_1}')) < (m, \#({p_1}^*\epsilon))$. By the I.H. for p_1, $g_{\epsilon\bar{k}}$ is of type $1_\epsilon, 2_\epsilon$ or 4_ϵ. Hence g is of type 3_ϵ.

(F.F) If g is not returning, then each child $g_c \leftrightarrows \gamma({p_1}' \cdot {p_1}^*\epsilon)$ of g such that ${p_1}^*\epsilon \xrightarrow{a} {p_1}' \cdot {p_1}^*\epsilon$ has $m' < m$ transitions and is a minimization. By Proposition 2.27 a root unwinding of each child g_c is a rum and of $1_\epsilon - 4_\epsilon$ by the I.H.. $g_{\epsilon\bar{k}}$ is strongly normed by the assumption that g is ϵ-safely unknottable with $g_{\epsilon\bar{k}}$. By Proposition 4.22 g is strongly normed (which is even stronger than terminating.) This implies that g is of type 1_ϵ.

Now for the proper derivatives of ${p_1}^*\epsilon$ of the form ${p_1}'' \cdot {p_1}^*\epsilon$ with $p \rightarrow^+ p'$, let g'' be a rum of $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$. Distinguish $(\frac{T}{F})$ whether or not ${p_1}^*\epsilon$ is returning.

(T) If ${p_1}^*\epsilon$ is returning, then g is of type 3_ϵ as proved above and $p' \in \mathbf{P}^*_\epsilon$ is strongly normed by Proposition 2.8. This implies that g'' is returning via a derivative bisimilar to g. So

1. either g'' is of type 2_ϵ or 3_ϵ, in which case we are finished, or
2. g'' is not of type $1_\epsilon - 3_\epsilon$ and hence of type 4_ϵ.

(F) If ${p_1}^*\epsilon$ is not returning, then g'' has a minimization with $m' < m$ transitions and so $(m', \#({p_1}'' \cdot {p_1}^*\epsilon)) < (m, \#({p_1}^*\epsilon))$. By Proposition 2.27 a root unwinding of the derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$ is a rum. By the I.H. g'' is of type $1_\epsilon - 4_\epsilon$ □

Collecting the results of this section we obtain.

COROLLARY 4.52. *(3.53)* Let g be a finite process graph. g is a $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressible process graph iff g has the $\star_\epsilon$ property.

COROLLARY 4.53. *(3.54)* Corollary 3.53 generalizes to the unary interpretation of Kleene star.

5. Decidability of Expressibility and the ⋆ properties

The last results of the previous two sections show that the classes of process graphs which have the ⋆ (and $\star_\epsilon$) property correspond to the process graphs expressible with regular expression without (and with) ϵ. This implies that $\Sigma(\mathrm{BPA}^*(Act))$ expressibility is decidable iff the ⋆ property is decidable. A straight forward method to prove decidability of the ⋆ property involves establishing that the definitions of an exit part, iterative part etc. define *effective* relations. Each of these relations can be proved to be effective by showing that they are based on computable constraints on a *finite* set of transitions and/or termination labels.

We use another method to prove decidability of the ⋆ property, which uses an upper bound argument as in Chapter 2 and permits an approximation of the complexity of the size of a regular expression expressing a process graph. We prove that process graphs which have the ⋆ property can be expressed with a regular expression with a *lower bounded* and *upper bounded* size. Since there are only finitely many regular expressions in between the two bounds, and bisimulation equivalence of the graph interpretation of a regular expression and a finite process graph is decidable, the ⋆ and $\star_\epsilon$ property are decidable and hence $\Sigma(\mathrm{BPA}^*(Act))$ and $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressibility is decidable.

Lower bounds on regular expressions expressing a process graph

First we prove that there is an upper bound for the complexity of a regular expression in terms of the number of states and second in terms of the number of transitions.

In [BBP94] it was already proved that process expressions in ACP* have finitely many derivatives. We can make a more precise statement for the class of process expressions over $\Sigma(\mathrm{BPA}^*_{\delta,\epsilon}(Act))$.

PROPOSITION 5.1. Let g be a finite rui process graph and g_k be the knotting of g. Then $R_g = R_{g_k}$.

LEMMA 5.2. Let $g_1 = (r_1, S, \downarrow_1, \rightarrow_1)$ and $g_2 = (r_2, S, \downarrow_2, \rightarrow_2)$ be finite rui process graphs such that $R_{g_1} \cap R_{g_2} = \emptyset$. Then for $\oplus \in \{+, \cdot, ^*\}$, $R_{g_1 \oplus g_2} \subseteq R_{g_1} \cup (R_{g_2} - r_2)$.

Proof. Immediate by Proposition 5.1 and Definition 2.35 □

LEMMA 5.3. Let $p \in \mathbf{P}^*_{\delta,\epsilon}$ and g a minimization of $\gamma(p)$. Then $R_g < \#(p)$.

Proof. Let p be as in the premise and g a minimization of $\gamma(p)$. We prove the result by a well-founded induction on the symbol complexity of p. Let the Induction Hypothesis (I.H.) be that if $p' \in \mathbf{P}^*_{\delta,\epsilon}$, $\#(p') < \#(p)$ and g' is a minimization of $\gamma(p')$, then $|R_{g'}| < \#(p')$. Distinguish the following cases in the Induction Step.

1. If $p \equiv a$ for some $a \in Act$ or $p \equiv \delta$ or $p \equiv \epsilon$, then the conclusion follows immediately,
2. If $p \equiv p_1 \oplus p_2$ for $\oplus \in \{+, \cdot, ^* \}$, then let g_1, g_2 be minimizations of of respectively $\gamma(p_1)$ and $\gamma(p_2)$. By Definition 2.35 and Lemma 5.2 a minimization of $\gamma(p_1 \oplus p_2)$ has at most $|R_{g_1}| + |R_{g_2}| + 1$ states. By the I.H. the result follows □

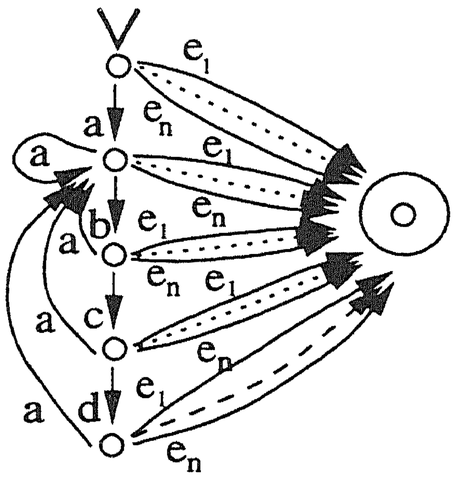
LEMMA 5.4. Let $p \in \mathbf{P}^*$ and $g = (r, S, \downarrow, \rightarrow)$ be a minimization of $\gamma(p)$. Then $|\rightarrow| < \#(p)$.

Proof. Let p and g be as in the premise. We prove the result by a well-founded induction on $\#(p)$. Let the Induction Hypothesis (I.H). be that all minimizations g', which are expressible with a $p' \in \mathbf{P}^*$ such that $\#(p') < \#(p)$, have $m' + n' \leq \#(p')$, where m' is the total number of transitions and n' the number of transitions of the root of g' immediately to a child. Distinguish the following cases in the Induction Step.

1. If $p \equiv a$ for some $a \in Act$, then the result follows immediately,
2. If $p \equiv p_1 \oplus p_2$ for $\oplus \in \{+, \cdot, ^* \}$, then let g_1 with m_1 transitions and n_1 "initial transitions", and the same for g_2 with m_2 and n_2 be rum's of respectively $\gamma(p_1)$ and $\gamma(p_2)$ such that $R_{g_1} \cap R_{g_2} = \emptyset$. By Definition 2.35 g has less transitions than $g_1 \oplus g_2$ for $\oplus \in \{+, \cdot, ^* \}$. Now observe that since $p_1 \in \mathbf{P}^*$, g_1 has an *unique* termination label and by Definition 2.35 the knotting of g_1 has *two* termination labels. Hence by Definition 2.35 $g_1 + g_2$ has $m_1 + m_2$ transitions, $g_1 \cdot g_2$ has at most $m_1 + n_2 + m_2$ transitions and ${g_1}^* g_2$ at most $m_1 + n_1 + n_2 + m_2$ transitions. By the I.H. the result follows.

Since $p \in \mathbf{P}^*$, it has a non zero number of "initial" transitions and so $\#(p) > |\rightarrow|$ □

REMARK 5.5. We do not have the same result for the regular expressions with ϵ: A rum (and a minimization) of the graph interpretation of $p \equiv (a \cdot (\epsilon + b \cdot (\epsilon + c.(\epsilon + d))))^*(e_1 + ... + e_n)$ for $n > 11$ has more transitions than $\#(p)$ as can be verified with the picture below.



Notice that $\#(p)$ is $18 + 3.n$ and the number of transitions of a rum and minimization are respectively $8 + 5.n$ and $7 + 4.n$ for $n > 1$.

We can give a lower bound in terms of the number of transitions for the graph interpretation of regular expressions with ϵ, if we use an adaptation of the symbol complexity.

DEFINITION 5.6. The lower bound symbol complexity $\#_< : \mathbf{P}^*_{\delta,\epsilon} \to \mathsf{N}^+$ of a process expression is defined as follows. Let $p, q \in \mathbf{P}^*_{\delta,\epsilon}$, then

$$\begin{array}{lcll} \#_<(p) & = & 1, & p \in \{\delta, \epsilon\} \\ \#_<(a) & = & 2, & a \in Act, \\ \#_<(p \ + \ q) & = & \#_<(p) + \#_<(q) + 1 & \\ \#_<(p \ . \ q) & = & \#_<(p).\#_<(q) + \#_<(p) & \\ \#_<(p \ ^* \ q) & = & \#_<(p).\#_<(q) + 2.\#_<(p) & \end{array}$$

PROPOSITION 5.7. Let g be a finite rui process graph. The knotting of g has less or equal twice the number of transitions of g.

LEMMA 5.8. Let $g_1 = (r_1, S, \downarrow_1, \rightarrow_1)$ and $g_2 = (r_2, S, \downarrow_2, \rightarrow_2)$ be finite rui process graphs such that $R_{g_1} \cap R_{g_2} = \emptyset$ and $g_\oplus = (r_1, S, \downarrow_\oplus, \rightarrow_\oplus) = (r_1, S, \downarrow_1, \rightarrow_1) \oplus (r_2, S, \downarrow_2, \rightarrow_2)$ for $\oplus \in \{+, \cdot, ^*\}$. Then

1. $| \rightarrow_+ | = | \rightarrow_1 | + | \rightarrow_2 |$, and
2. $| \rightarrow_. | \leq | \rightarrow_1 |.| \rightarrow_2 | + | \rightarrow_1 |$, and
3. $| \rightarrow_* | \leq | \rightarrow_1 |.| \rightarrow_2 | + 2.| \rightarrow_1 |$.

Proof. Immediate by Proposition 5.7 and Definition 2.35 □

LEMMA 5.9. Let $g = (r, S, \downarrow, \rightarrow)$ be a rum. If g is expressible with $p \in \mathbf{P}^*_{\delta,\epsilon}$, then $| \rightarrow | < \#_<(p)$.

Proof. Let g be as in the premise. We prove the result by a well-founded induction on $\#_<(p)$. Let the Induction Hypothesis (I.H) be that if $\#_<(q) < \#_<(p)$, then a rum of $\gamma(q)$ has less than $\#_<(q)$ transitions. Distinguish the following cases in the Induction Step.

1. If $p \equiv a$ for some $a \in Act$ or $p \equiv \delta$ or $p \equiv \epsilon$ the conclusion follows immediately,
2. If $p \equiv p_1 \oplus p_2$ with $\oplus \in \{+, \cdot, ^*\}$, then by the I.H. rum's of $\gamma(p_1)$ and $\gamma(p_2)$ have respectively $m_1 < \#_<(p_1)$, $m_2 < \#_<(p_2)$ transitions. By Proposition 5.8 the conclusion follows □

Process graphs with the $\star$ or $\star_\epsilon$ property can be expressed with regular expressions with an upper bounded size

The proofs of Theorem 3.39 and Theorem 4.37 give a method to construct a regular expression from a process graph which has the $\star$ or $\star_\epsilon$ property. For process graphs of type $1_{(\epsilon)}$–$3_{(\epsilon)}$ the construction is clear cut in terms of the regular expressions for sub graphs. This allows for an effective upper bound on the size of the regular expression expressing a finite process graph.

In terms of the afore mentioned proofs we know only that process graphs of type $4_{(\epsilon)}$ are *derivatives* of a process graph of another type. So the size of the expressing regular expression is only estimated as the size of one derivative of *another* derivative.

We present the proof that a regular expression expressing a process graph of type $4_{(\epsilon)}$, can be expressed with a regular expression expressing the derivative of type $2_{(\epsilon)}$ or $3_{(\epsilon)}$ *prefixed* with a process expression based on a subgraph. This allows an successful analysis of the size of the regular expression expressing a process graphs of type $4_{(\epsilon)}$.

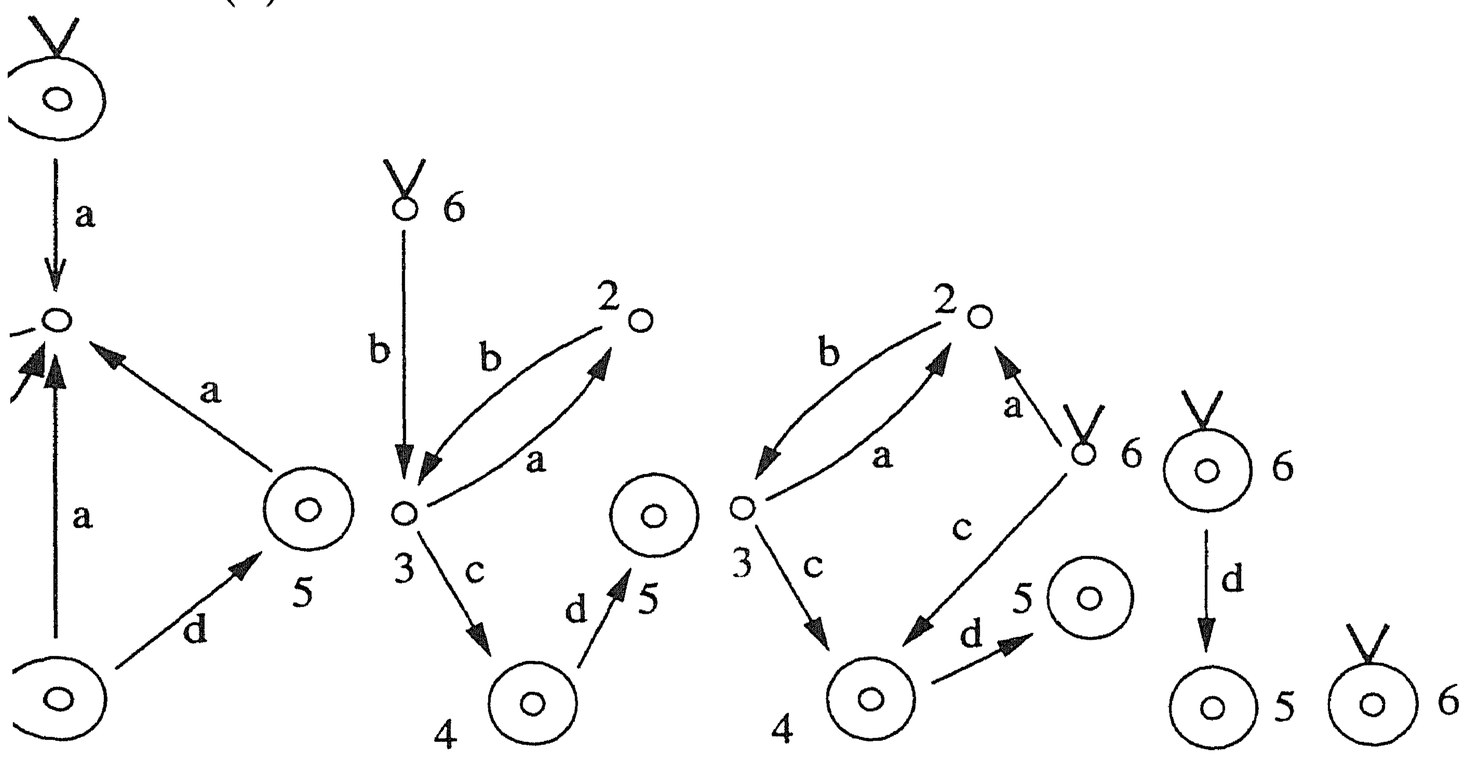
We start with the definition of a subgraph expressing the "prefix". Second we prove that the prefix is indeed bisimilar to a graph interpretation of the subgraph. Third we prove that we can give an upper bound for the symbol complexity of the smallest regular expression expressing a finite process graph with the $\star$ or $\star_\epsilon$ property.

DEFINITION 5.10. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite rum. If $g_s = (r_s, S, \downarrow_s, \rightarrow_s)$ is a process graph, where $r_d \in R_g$ and $r_s \in S - R_g$ is an unreachable state of g and $\downarrow_s$, $\rightarrow_s$ are the smallest sets such that

1. if $r_d \notin Rt_g$, $r_d \xrightarrow{a} s$ and [if $r_d \downarrow$, then $r \overset{a}{\not\rightarrow} s$], then $r_s \xrightarrow{a}_s s$,
2. if $r_s \rightarrow_s{}^+ s$, $s \notin Rt_g$, $s \xrightarrow{a} s'$ and [if $s \downarrow$, then $r \overset{a}{\not\rightarrow} s$], then $s \xrightarrow{a}_s s'$,

$r_d\downarrow$, then $r_s\downarrow_s$,
$r_s{\rightarrow_s}^+ s$ and $s\downarrow$, then $s\downarrow_s$,
is *the root unwinding with* r_s *of the root search graph from* r_d
oot original of g, shortly *a root search graph from* r_d *in* g.

MPLE 5.11. In the picture below, (2)–(5) are the root search in (1) from respectively states 2–5. (5) is also a root search om 1 in (1).



(1) (2) (3) (4) (5)

at (1)–(5) are rum's of graph interpretations of respectively $(a\cdot b\cdot(a\cdot b)^*(c\cdot\epsilon, b\cdot(a\cdot b)^*c\cdot(d+\epsilon)$, $(a\cdot b)^*c\cdot(d+\epsilon)$, $d+\epsilon$ and ϵ.

MA 5.12. Let g be a finite rum and $s\in R_g$. A root search om s in g is rui.

f. No root incoming transitions are added by taking a root raph □

POSITION 5.13. Let g be a finite rum and $s\in R_g$. Then the rch trees of g are unique up to isomorphism.

ARK 5.14. Henceforth we speak of *the* root search graph.

MA 5.15. Let g be a finite rum and $s_d\in R_g$. Then the root ee from s_d in g is a rum.

f. Let $g=(r,S,\downarrow,\rightarrow)$ and s_d be as in the premise. We only For the case that $s_d\notin Rt_g$. Let r_s be the root of the root search om s_d in g.

Suppose towards a contradiction that g_s is not a rum, i.e. there is a rum $g_c = (r_c, S, \downarrow_c, \rightarrow_c)$ such that $g_c \leftrightarrows g_s$. Assume without loss of generalization that $R_{g_c} \cap R_g = \emptyset$. Let $\varphi_{s,c} : R_{g_s} \rightarrow R_{g_c}$ be the map defined as follows

$$\begin{array}{lcll} \varphi_{s,c}(r_s) & = & r_c & \\ \varphi_{s,c}(s) & = & t, & \text{where } s \neq r_s \text{ and } t \neq r_c \text{ are respectively} \\ & & & \text{roots of bisimilar derivatives} \\ & & & \text{of } g_c \text{ and } g_s. \end{array}$$

Let $\varphi : R_g \rightarrow (R_g - (R_s \cup \{s_d\})) \cup R_{g_c}$ be the map defined as follows

$$\begin{array}{lcll} \varphi(s) & = & s, & \text{if } s \in R_g - (R_{g_s} \cup \{s_d\}) \\ \varphi(s) & = & \varphi_{s,c}(s), & \text{if } s \neq s_d \text{ and } s \in R_{g_s}. \end{array}$$

We will prove that g and $g_\varphi = (\varphi(r), S, \{\varphi(s) \mid s \downarrow\}, \{\varphi(s) \xrightarrow{a} \varphi(s') \mid s \xrightarrow{a} s'\})$ are bisimilar. Let $R \subseteq R_g \times R_{g_\varphi}$ be the set of tuples $(s, \varphi(s))$ for $s \in R_g$. We check the demands of Definition 2.14 to prove that R is a bisimulation relating g and g_φ.

1. $(\downarrow \Rightarrow \downarrow_\varphi)$ Suppose $s \downarrow$. By definition of g_φ, $\varphi(s) \downarrow_\varphi$.

   $(\downarrow \Leftarrow \downarrow_\varphi)$ Suppose $\varphi(s) \downarrow$. Distinguish $(\frac{T}{F})$ whether or not $\varphi(s) = s$. (T) If $\varphi(s) = s$, then by definition of φ $s \in R_g - (R_{g_s} \cup \{s_d\})$. By definition of g_φ $s \downarrow$.

   (F) If $\varphi(s) \neq s$, then by definition of φ, $s \in R_{g_s} \cup \{s_d\}$. By definition of bisimulation and definition of $\varphi_{s,c}$ and φ either $s = s_d$, $s_d \downarrow$ and g_s is returning, or $s \neq s_d \in R_{g_s} - s_d$. By definition of a root search graph in both cases $s \downarrow$.
2. $(\rightarrow \Rightarrow \rightarrow_\varphi)$ Suppose $s \xrightarrow{a} s'$. By definition of g_φ, $\varphi(s) \xrightarrow{a} \varphi(s')$ and by definition of R, $(s', \varphi(s')) \in R$.
3. $(\rightarrow \Leftarrow \rightarrow_\varphi)$ Suppose $\varphi(s) \xrightarrow{a} \varphi(s')$. By definition of φ and $\varphi_{s,c}$, s and t are roots of bisimilar derivatives of g and similarly s' and t'. This implies by definition of bisimulation that $s \xrightarrow{a} s''$ for some $s'' \in R_g$, such that s'' and t' are roots of bisimilar derivatives of g. By definition of φ, $\varphi(s'') = \varphi(s')$. By definition of R $(s'', \varphi(s')) \in R$.

Obviously $(r, \varphi(r)) \in R$ and hence $g \leftrightarrows g_\varphi$.

Now by definition of φ, g_φ has less states than g. Also g_φ is rui, since no root incoming transitions are added in the definition going from g to g_φ. This implies that g is not a rum. Contradiction with the assumption and hence g_s is a rum $\square$

LEMMA 5.16. Let g be a finite rum with root r and $r_d \in R_g$. If g is returning, then a minimization of the root search graph from r_d in g has less transitions than a minimization of g.

Proof. Let g be as in the premise. By the assumption that g is returning, there are s, s' such that $s \neq r \in Rt_g$ and $s \xrightarrow{a} s'$. By definition of a root search graph, $s \xrightarrow{a} s'$ is not a transition of the root search graph. Hence a minimization of a root search graph has less transitions than a minimization of g □

LEMMA 5.17. Let ${p_1}^*\epsilon \in \mathbf{P}^*_\epsilon$ be a pure knotting and g a rum of $\gamma({p_1}^*\epsilon)$. If ${p_1}^*\epsilon \rightarrow^+ {p_1}' \cdot {p_1}^*\epsilon$, ${p_1} \rightarrow^+ {p_1}'$ and the derivative of of g with root r_d is bisimilar to $\gamma({p_1}' \cdot {p_1}^*\epsilon)$, then the root search graph from r_d in g is bisimilar to $\gamma({p_1}')$.

Proof. Let ${p_1}^*\epsilon$ and $g = (r, S, \downarrow, \rightarrow)$ be as in the premise. Let $p \rightarrow^+ {p_1}'$ and r_d be the root of a derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^*\epsilon)$.

We start with a construction of a process graph $g_{s'}$ by leaving out transitions and transition labels from g (A). Next we prove that the thus obtained process graph is the root search graph from r_d in g (B). Last we prove that the process graph $g_{s'}$ is bisimilar to $\gamma({p_1}')$ (C).

(A) Let $r_{s'} \in S - R_g$ and $g_{s'} = (r_{s'}, S, \downarrow_{s'}, \rightarrow_{s'})$ be the process graph where $\downarrow_{s'}$, $\rightarrow_{s'}$ are the smallest sets such that

1. if ${p_1}' \xrightarrow{a} {p_1}''$, then $r_{s'} \xrightarrow{a}_{s'} s$, where $s \neq r$ is the the root of the derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$,
2. If ${p_1}' \rightarrow^+ {p_1}''$ and ${p_1}'' \xrightarrow{a} {p_1}'''$, then $s \xrightarrow{a}_{s'} s'$, where $s, s' \neq r$ are the roots of the derivatives of g bisimilar to respectively $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$ and $\gamma({p_1}''' \cdot {p_1}^*\epsilon)$,
3. if ${p_1}' \downarrow$, then $r_{s'} \downarrow_{s'}$,
4. if ${p_1}' \rightarrow^+ {p_1}''$, then $s \downarrow_{s'}$, where $s \neq r$ is the root of the derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$.

(B1) To prove that $g_{s'}$ is indeed a root search graph from r_d in g we check the demands of Definition 5.10.

1. By definition of g follows that r_d is the root of a derivative of g bisimilar to $\gamma({p_1}' \cdot {p_1}^*\epsilon)$. Now suppose $r_d \notin Rt_g$, $r_d \xrightarrow{a} s$ and if $r_d \downarrow$, then $r \xrightarrow{a}\!\!\!\!\!\!/$. Since ${p_1}^*\epsilon$ is a pure knotting and by definition of g, there is a ${p_1}''$ such that ${p_1}' \xrightarrow{a} {p_1}''$ and s is the root of the

derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$. By definition of $g_{s'}$, $r_s \xrightarrow{a} s$.

2. If $r_{s'} \rightarrow_{s'}{}^+ s$, $s \notin Rt_g$, $s \xrightarrow{a} s'$ and [if $s \downarrow$, then $r \xrightarrow{a}\!\!\!\!\!\!\rightarrow s'$], then by definition of $g_{s'}$ and g, ${p_1}' \rightarrow^+ {p_1}''$ and ${p_1}'' \xrightarrow{a} {p_1}'''$, such that s, s' are the roots of derivatives of g bisimilar to respectively $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$ and $\gamma({p_1}''' \cdot {p_1}^*\epsilon)$. By definition of $g_{s'}$, $s \xrightarrow{a}_{s'} s'$,
3. If $r_d \downarrow$, then by definition of g this implies that ${p_1}' \cdot {p_1}^*\epsilon \downarrow$. By the operational semantics follows that ${p_1}' \downarrow$. By definition of $g_{s'}$, $r_s \downarrow_{s'}$,
4. If $r_{s'} \rightarrow_{s'}{}^+ s$ and $s \downarrow$, then by definition of $g_{s'}$, ${p_1}' \rightarrow^+ {p_1}''$ such that $s \neq r$ is the root of a derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$ and ${p_1}'' \cdot {p_1}^*\epsilon \downarrow$. By the operational semantics ${p_1}'' \downarrow$. By definition of $g_{s'}$, $s \downarrow_{s'}$.

(C) To prove that $g_{s'}$ and $\gamma({p_1}')$ are bisimilar, we present the relation $R \subseteq R_{g_{s'}} \times R_{\gamma({p_1}')}$, which we prove to be a bisimulation. Let R be the smallest superset of the tuple $(r_{s'}, {p_1}')$, of tuples of the form $(s, {p_1}'')$, where $s \neq r$ is the root of the derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$ and ${p_1}' \rightarrow^+ {p_1}''$.

We check the demands of Definition 2.14. Suppose $(s, q) \in R$.

1. ($\downarrow_{s'} \Rightarrow \downarrow$) Suppose $s \downarrow_{s'}$. Distinguish $(\frac{T}{F})$ whether or not $s = r_{s'}$. (T) If $s = r_{s'}$, then by definition of $g_{s'}$, ${p_1}' \downarrow$. By definition of R, $q \equiv {p_1}'$ and hence $q \downarrow$.

   (F) If $s \neq r_{s'}$, then by definition of $g_{s'}$, $s \neq r$ is the root of the derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$, such that ${p_1}'' \downarrow$ and ${p_1}' \rightarrow^+ {p_1}''$. By definition of R follows that $q \leftrightarrows {p_1}''$ and hence $q \downarrow$.

   ($\downarrow_{s'} \Leftarrow \downarrow$) Suppose $q \downarrow$. By assumption ${p_1}' \rightarrow^* q$ and $q \downarrow$. Distinguish $(\frac{T}{F})$ whether or not $q \equiv {p_1}'$. (T) If $q \equiv {p_1}'$, then by definition of R, $s = r_{s'}$. By definition of $g_{s'}$, $s \downarrow_{s'}$.

   (F) If $q \not\equiv {p_1}'$, then by definition of R, $s \neq r$ is the root of the derivative of g bisimilar to $\gamma(q \cdot {p_1}^*\epsilon)$. By definition of $g_{s'}$, $s \downarrow_{s'}$.
2. ($\rightarrow_{s'} \Rightarrow \rightarrow$) Suppose $s \xrightarrow{a}_{s'} s'$. Distinguish $(\frac{T}{F})$ whether or not $s = r_{s'}$. (T) If $s = r_{s'}$, then by definition of $g_{s'}$, $s' \neq r$ is the root of the derivative of g bisimilar to $\gamma({p_1}'' \cdot {p_1}^*\epsilon)$ for some ${p_1}''$ such that ${p_1}' \xrightarrow{a} {p_1}''$. By definition of R $(s', {p_1}'') \in R$.

(F) If $s \neq r_{s'}$, then by definition of $g_{s'}$, the states $s, s' \neq r$ are the roots of the derivatives of g bisimilar to respectively $\gamma(p_1'' \cdot p_1{}^*\epsilon)$ and $\gamma(p_1''' \cdot p_1{}^*\epsilon)$. By definition of R $q \leftrightarrows p_1''$ and by definition of bisimulation there is a q' such that $q \xrightarrow{a} q'$ and $q' \leftrightarrows p_1'''$. By definition of R $(s', q') \in R$.

3. $(\rightarrow_{s'} \Leftarrow \rightarrow)$ Suppose $q \xrightarrow{a} q'$. Distinguish $(\frac{T}{F})$ whether or not $q \equiv p_1'$. (T) If $q \equiv p_1'$, then by definition of R, $s = r_{s'}$. By definition of $g_{s'}$, this implies that $r_{s'} \xrightarrow{a}_{s'} s'$, where $s' \neq r$ is the root of the derivative of g bisimilar to $\gamma(q' \cdot p_1{}^*\epsilon)$. By definition of R this implies $(s', q') \in R$.

   (F) If $q \not\equiv p_1'$, then by definition of R, s is the root of a derivative of g bisimilar to $\gamma(q \cdot p_1{}^*\epsilon)$ and $p_1 \rightarrow^+ q$. By definition of $g_{s'}$, this implies $s \xrightarrow{a}_{s'} s'$, where $s' \neq r$ is the root of the derivative of g bisimilar to $\gamma(q' \cdot p_1{}^*\epsilon)$. By definition of R, this implies that $(s', q') \in R$.

By definition $(r_{s'}, p_1') \in R$ and hence $g_{s'} \leftrightarrows \gamma(p_1')$ □

THEOREM 5.18. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite process graph. If g has the $(\star)$ $\star_\epsilon$ property, then g is expressible as $\gamma(p)$, such that $(p \in \mathbf{P}^*)$ $p \in \mathbf{P}^*_\epsilon$ and $\#(p) \leq (|\rightarrow| + 1)^{(|\rightarrow|+1)}$.

Proof. We only prove it for the $\star_\epsilon$ property. Assume without loss of generalization that g is a rum. Let g be as in the premise and let m be the number of transitions of a minimization of g. Assume without loss of generalization that g is a rum. By Theorem 4.37 it remains to be proved that there is an expressing process expression $p \in \mathbf{P}^*_\epsilon$ such that $\#(p) \leq (|\rightarrow| + 1)^{(|\rightarrow|+1)}$. We prove the result using the proof of Theorem 4.37 and a well-founded induction on the number m. Let the Induction Hypothesis (I.H.) be that all rum's g', which

1. have minimizations with $m' < m$ transitions, and
2. have the $\star_\epsilon$ property,

can be expressed with a process expression $p' \in \mathbf{P}^*_\epsilon$, such that $\#(p') \leq (m' + 1)^{(m'+1)}$.

Distinguish the following cases in the Induction Step (I.S.) as to the type of g.

1. If g is of type 1_ϵ, then by the corresponding case in the proof of Theorem 3.39 g is expressible as $p \equiv \sum_{i=1}^{k} a_i \cdot p_i + \sum_{j=1}^{l} a_i[+\epsilon]$, such that the number of children of g is $c = k + l$. Notice that

because g is not returning, a minimization of a child of g has at most $m - c$ transitions. Distinguish the following cases.

(a) If $m < c$, then g has more children than transitions. Contradiction,

(b) If $m = 0$, then g is trivial and expressible with $p \equiv \epsilon$. By definition of symbol complexity $\#(\epsilon) = 1 \leq (0+1)^{(0+1)}$,

(c) If $m = c$ and $m \geq 1$, then g is expressible as $p \equiv \sum_{i=1}^{m} a_i[+\epsilon]$, where $a_i \in Act$. By definition of symbol complexity $\#(p) \leq m.2 + (m-1) + 1 = 3.m \leq (m+1)^{(m+1)}$,

(d) If $1 \leq c < m$, then

$$
\begin{array}{lcl}
\#(\sum_{i=1}^{k} a_i \cdot p_i + \sum_{j=1}^{l} a_i[+\epsilon]) & \leq & \{ \text{ I.H. } \} \\
\sum_{i=1}^{c}(3 + ((m-c)+1)^{(m-c)+1}) + c - 1[+1] & < & \\
c.(4 + ((m-c)+1)^{(m-c)+1}) & < & \\
4.c.((m-c)+1)^{(m-c)+1} & \leq & \{m > c \geq 1\} \\
(m+1)^{c}.(m+1)^{(m-c)+1} & & \\
(m+1)^{m+1}. & &
\end{array}
$$

Notice that we even have for $m \geq 2$ that g can be expressed with a p such that $\#(p) \leq (m+1)^{m+1} - 2$.

2. If g is of type 2_ϵ, then by the corresponding case in Theorem 4.37 $g \leftrightarrows \gamma(p_1{}^*p_2)$ is ϵ-safely decomposable in the non trivial ϵ-exit part $g_{\epsilon e} \leftrightarrows \gamma(p_2)$ and the not trivial ϵ-iterative part $g_{\epsilon i} \leftrightarrows \gamma(p_1{}^*\epsilon)$. By Lemma 4.34 $m \geq 2$ and minimizations of $g_{\epsilon e}$ and $g_{\epsilon i}$ have less than m transitions. Then

$$
\begin{array}{lcl}
\#(p_1{}^*p_2) & = & \\
\#(p_1{}^*\epsilon) + \#(p_2) - \#(\epsilon) & \leq & \{ \text{ I.H } \} \\
m^m + m^m - 1 & < & \\
2.m^m & < & \{m \geq 2\} \\
(m+1)^{m+1}. & &
\end{array}
$$

3. If g is of type 3_ϵ, then by the corresponding case in the proof of Theorem 4.37 $g \leftrightarrows \gamma(p^*\epsilon)$ is ϵ-safely unknottable with $g_{\epsilon\bar{k}} \leftrightarrows \gamma(p)$, which is of type 1_ϵ, 2_ϵ or 4_ϵ. By Lemma 4.35 follows that $m \geq 1$. A minimization of $g_{\epsilon\bar{k}}$ has less than m transitions if $g_{\epsilon\bar{k}}$ is returning, else $g_{\epsilon\bar{k}}$ is of type 1_ϵ and each child of $g_{\epsilon\bar{k}}$ has less than m transitions. With the I.H. and the case for process graphs of type 1_ϵ $\#(p) \leq (m+1)^{m+1} - 2$. Then $\#(p^*\epsilon) \leq (m+1)^{m+1} - 2 + 2 = (m+1)^{m+1}$.

4. If g is of type 4_ϵ, then by the corresponding case in the proof of Theorem 4.37 (0) $g \leftrightarrows \gamma({p_1}' \cdot {p_1}^* p_2)$ or (1) $g \leftrightarrows \gamma({p_1}' \cdot {p_1}^* \epsilon)$ is expressible as a proper derivative (0) of $g_d \leftrightarrows \gamma({p_1}^* p_2)$ of type 2_ϵ or (1) of a rum $g_d \leftrightarrows \gamma({p_1}^* \epsilon)$ of type 3_ϵ. By Proposition 3.38 a minimization of g has the same number of transitions as a minimization of g_d and $m \geq 2$. With the previous cases for process graphs of type (0) 2_ϵ or (1) 3_ϵ $\#(p) \leq 2.m^m$ where (0) $p \equiv {p_1}^* p_2$ or (1) $p \equiv {p_1}^* \epsilon$.

   By Lemma 4.48 we can safely assume that ${p_1}^* \epsilon$ is a pure knotting. This implies by Proposition 5.17 that $\gamma({p_1}')$ is bisimilar to the root search graph from r_d in the ϵ-iterative part of a rum g' of $\gamma({p_1}^* \epsilon)$, where r_d is the root of the derivative of g' bisimilar to $\gamma({p_1}' \cdot {p_1}^* \epsilon)$. If $g \leftrightarrows g'$, then by assumption a minimization of g' has m transitions, else g' is the iterative part of g and by Lemma 4.34 a minimization has less than m transitions. By Proposition 5.16 a minimization of the root search graph $g_s \leftrightarrows \gamma({p_1}')$ from r_d in the (0) ϵ-iterative part of g_d or (1) in g has *less* than m transitions. For the symbol complexity of the process expression p expressing g holds

$$\begin{array}{lcl} \#({p_1}' \cdot p) & \leq & \{ \text{ I.H. } \} \\ 2.m^m + 1 + m^m & = & \\ 3.m^m + 1 & \leq & \{ \, m \geq 2 \, \} \\ (m+1)^{m+1} \; \square & & \end{array}$$

COROLLARY 5.19. It is decidable whether a finite process graph has the $\star$ or $\star_\epsilon$ property.

Proof. Let g be a finite process graph. Obviously g has more or an equal number of transitions m than any of its minimizations. List all process expressions with a symbol complexity of less than $(m+1)^{m+1}$. If g has the $\star$ or $\star_\epsilon$ property, then by Theorem 5.18 there is a process expression in this list whose graph interpretation is bisimilar to g.

   If g does not have the $\star$ or $\star_\epsilon$ property, then by Corollary 3.53 there is no expressing process expression for g in the listing. Now the listing has a bounded length. It is folklore that bisimulation equivalence of finite process graphs is decidable and that graph interpretation of a process expression over $\Sigma(\mathrm{BPA}^*(Act))$ can be effectively computed [Mil84]. Hence the $\star$ and $\star_\epsilon$ property is decidable $\square$

COROLLARY 5.20. Let $g = (r, S, \downarrow, \rightarrow)$ be a finite process graph. If g is expressible with $p \in \mathbf{P}^*$ (or $p \in \mathbf{P}^*_\epsilon$), then there is a $q \in \mathbf{P}^*$ (or $q \in \mathbf{P}^*_\epsilon$) such that $q \leftrightarrows p$ and $\#(q) \leq (| \rightarrow | + 1)^{|\rightarrow|+1}$.

COROLLARY 5.21. $\Sigma(\mathrm{BPA}^*(Act))$ and $\Sigma(\mathrm{BPA}^*_\epsilon(Act))$ expressibility of finite process graphs is decidable.

Proof. Use an upper bound argument with Corollary 5.20 □

EXAMPLE 5.22. The results of this section imply that if the process graph of Example 1.1 is to be expressible, the expressing regular expression with or without ϵ has a symbol complexity (#) more or equal than 2, lower bound symbol complexity ($\#_<$) of more than 2 and an upper bound symbol complexity (#) less or equal 9. Obviously we can restrict the search to regular expressions which have at least a and b actions and are strongly normed. We have to check whether in the finite list $a+b$, $b+a$, $a \cdot b$, $b \cdot a$, a^*b, b^*a, $a+b+\epsilon$ ... there is a regular expression with a bisimilar graph interpretation.

6. Identifying Expressible Process Graphs

The results in this chapter suggest *two* algorithms for deciding expressibility of finite process graphs with a regular expression and computation of such a regular expression if it exists. Intuitively the most simple algorithm is to list consecutively all regular expressions as we did in Example 5.22. Once a regular expression with a graph interpretation bisimilar to a process graph is found, it is expressible. This algorithm is effective since regular expressions expressing a regular process graph have a lower bound on the size (Lemmas 5.3, 5.4 and 5.9), but more important have a maximum symbol complexity (denoted by #) [13] by Corollary 5.20 of $(m+1)^{m+1}$, where m is the number of transitions of g. This algorithm also produces the *smallest* regular expression which expresses a process graph.

COROLLARY 6.1. Let g be a finite process graph. A process expression $p \in \mathbf{P}^*$ or $p \in \mathbf{P}^*_\epsilon$ expressing g with a smallest symbol complexity is effectively computable, *if it exists*.

We leave the precise estimate of the complexity of this algorithm to others, but we expect it to be exponential. An experimental implementation of an algorithm which is based directly on the $\star$ and $\star_\epsilon$ property is more promising in this aspect. From the inductive proof of Theorem 5.18 an algorithm can be extracted which checks whether a process graph has the $\star$ or $\star_\epsilon$ property and spews out an expressing regular expression. This is algorithm is effective since the $\star$ and $\star_\epsilon$ property correspond to the property that a finite process graph can be expressed with a regular expression without and with ϵ (Corollaries 3.52 and 4.51). This shows in a twinkle of an eye that the suspected process graphs (1) in Example 1.1[14] and (2) in Example 1.2 of the Introduction are indeed *in*expressible. A prototype (executable) specification in ASF+SDF [vDHK96] can be found at `http://www.cwi.nl/~doeko`. The regular expressions expressing strongly normed rum's in this chapter are the same as found with this specification.

---

[13] The symbol complexity as defined in Definition 3.48 counts 2 for every action and 1 for every non action symbol.

[14] We leave it as an exercise to the reader to verify that it is even the *smallest* inexpressible process graph, if we measure a process graph by the sum of its reachable states and transitions.

CONJECTURE 6.2. The algorithm based on the proof of Theorem 5.18 computing the regular expression expressing a rum g has a polynomial time complexity in the size of g.

Milner's open question and richer classes of regular expressions

The theory developed in this chapter provides an answer to the second open question of Robin Milner in [Mil84] "What structural property of finite charts is necessary and sufficient for star behaviours ?", if star behavior is understood as the the behavior of regular expressions in the usual meaning.

In most of the literature regular expressions are usually treated *without* a constant δ, denoting some deadlocked process. In [Mil84] star behavior is defined as the behavior of a super class of the regular expressions and a kind of δ (which is denoted by $\emptyset$) that satisfies the usual axioms for δ, (A6) $\delta \cdot x = x$ and (A7) $x + \delta = x$ and ϵ (denoted by $\emptyset^*$), that satisfies (A8) $x \cdot \epsilon = x$, where the axiom names are taken from [BW90].

An interesting question is whether the techniques described in this chapter can be used for "richer" classes of regular expressions. We might for instance add the parallel operator or the special constant δ from ACP, which denotes the deadlocked process. We leave the first question to others and ponder briefly on the question of expressibility of regular expressions with δ.

In the proof of the decidability of expressibility with regular expressions we were able to formulate conditions on regular expressions which allows for a close correspondence between a process graph and an expressing regular expression, i.e.

1. Given some condition on the regular expression ${p_1}^* p_2$, the graph interpretations $\gamma({p_1}^* \epsilon)$ and $\gamma({p_1}^* p_2)$ are bisimilar to respectively the ϵ-*exit part* and ϵ-*iterative part* of a rum g of the graph interpretation $\gamma({p_1}^* p_2)$ (Lemmas 3.46 and 4.49),
2. Given some condition on the regular expression $p^* \epsilon$, the graph interpretation $\gamma(p)$ is bisimilar to the ϵ-*unknotting* of a rum g of $\gamma(p^* \epsilon)$ (Lemmas 3.47 and 4.50),
3. Given some condition on the regular expression ${p_1}' \cdot {p_1}^* \epsilon$, where ${p_1}'$ is a derivative of p_1, the graph interpretation $\gamma({p_1}')$ is bisimilar

to a *root search tree* of a rum of the graph interpretation $\gamma({p_1}' \cdot {p_1}^*\epsilon)$ (Lemma 5.17).

Unfortunately we were not able to formulate a condition on regular expressions and a new graph operation to the extent that $\gamma(p)$ is bisimilar to the operation on a rum of $\gamma(p^*\delta)$. For us this means that we cannot prove completeness of a new "$\star_\delta$" property for $\Sigma(\mathrm{BPA}^*_\delta(Act))$ expressible process graphs. We have the impression that the structure of regular expressions with ϵ remains visible in process graphs, whereas it is lost with δ: in the context with δ the difference between termination and non termination is lost, e.g. the regular expressions $(a^i)^*\epsilon$ are not bisimilar, but the regular expressions $a^i\ {}^*\delta$ are all bisimilar (and hence have the same rum's) as i varies.

Better criteria for (in)expressibility ?

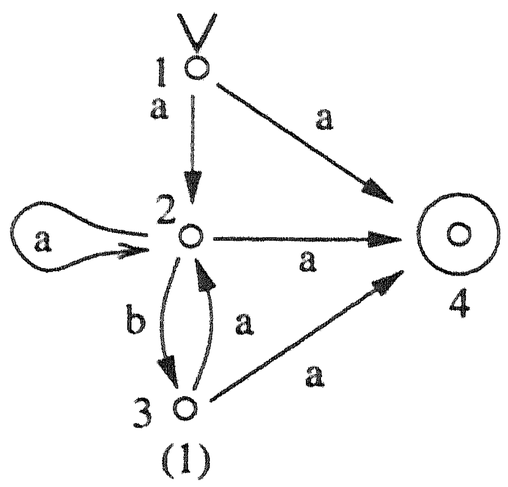
The $\star$ and $\star_\epsilon$ property were devised with the idea that an expressible process graph can somehow be constructed. They can also be seen as *recipes* to construct a regular expression expressing a process graph (Remarks 3.26 and 4.31). The proof that the process graph which have the $\star$ and $\star_\epsilon$ property are all expressible process graphs is much more involved (Theorems 3.52 and 4.51).

A tempting question is whether there are more direct or *easier* criteria on process graphs for (in)expressibility. Since the $\star$ and $\star_\epsilon$ property are effective, we can verify (in)expressibility for every individual graph, but one might ask for a more intuitive classification.

The problem can be illustrated with the (technical) question which process graphs are expressible with regular expressions with ϵ, but not without ϵ. As an exercise we will investigate whether *non* intermediately terminating process graphs which are expressible with a regular expression *with* ϵ are inexpressible with a regular expression *without* ϵ (which we tried as a distinguishing criterion for *recursive* process expressions in Section 5 of Chapter 1).

Clearly all process graphs which allow intermediate termination, i.e. there is a choice between immediate termination and an action, cannot be expressed with a regular expression without ϵ. However not all non-intermediately terminating finite process graphs which are expressible with a regular expression *with* ϵ are expressible *without* ϵ, as the example below shows.

EXAMPLE 6.3. The process graph (1) depicted below is a (root unwinding of a minimization of a) graph interpretation of $(a \cdot (b + \epsilon))^* a$.



(1)

(1) is not $\Sigma(\mathrm{BPA}^*(Act))$ expressible, since it does not have the $\star$ property. Of course it has the $\star_\epsilon$ property.

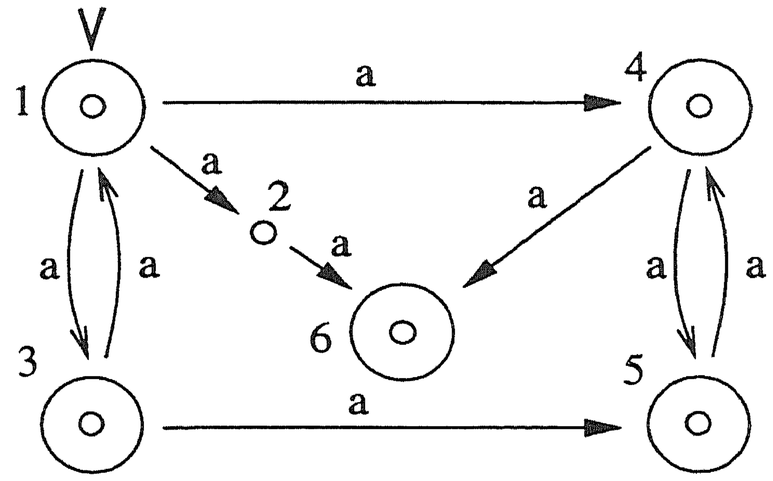
The example above is even more interesting since it shows that the so-called exit property as proved for ACP* in [BBP94, Bos95] does not hold in the setting with ϵ. The exit property states that, roughly spoken, every cycle can only be exit-ed with a transition ending in a terminated state at one state modulo bisimulation.

With the techniques developed in this chapter we can prove a stronger version of the exit property, which implies the inexpressibility of the process graph in Example 6.3 with a regular expression without ϵ.

PROPOSITION 6.4. *(Cluster Exit Property)* Let $g = (r, S, \downarrow, \rightarrow)$ be a finite process graph which is minimal modulo bisimulation and R_g the set of reachable states of g, i.e. $\{s | r \rightarrow^* s\}$. Let $C \subseteq R_g$ be a maximal subset such that for every two states $s, t \in C$ with $s \neq t$, $s \rightarrow^+ t$. Then

- If there are states s, s' in C and $t, t' \in R_g - C$ such that $s \xrightarrow{a} t$ and $s' \xrightarrow{b} t'$ for some $a, b \in Act$, then g is *in*expressible with a regular expression without ϵ,
- If there are states s, s' in C and $t, t' \in R_g - C$ such that $s \xrightarrow{a} t$, $s' \xrightarrow{a} t'$ and $s \not\xrightarrow{a} t'$ for some $a, b \in Act$, then g is *in*expressible with a regular expression with ϵ.

EXAMPLE 6.5. The cluster exit property also implies the inexpressibility of the process graph [15] depicted below.



REMARK 6.6. The cluster exit property does not not give a decision on e.g. the inexpressible process graph of Example 1.1, (2) in Example 1.2, (3) in Example 2.31 and (1) in Example 4.10. Therefore it is not a *complete* criterion for *in*expressibility of finite process graphs, such as the $\star$ and $\star_\epsilon$ property.

Acknowledgements. This chapter has been greatly influenced by the ideas of Alban Ponse. Didier Caucal, David Griffioen, Jan-Friso Groote, Wan Fokkink, Bas Luttik and Jos Egger have supplied valuable comments on draft versions of this chapter.

[15] It is presented in [Bos95] to prove that regular expressions *with* merge are more expressible than regular expressions *without* the merge. It is expressible with $(a \cdot a)^* a || a$ (assuming the "standard" operational semantics for the merge).

CHAPTER 4

Term Rewriting Properties of SOS Axiomatizations

In [ABV94] two strategies are presented to produce axiomatizations of strong bisimulation equivalence for languages whose operational semantics can be expressed in the GSOS format of [BIM95]. In [ABV94] it is stated that if the GSOS systems satisfy certain finiteness conditions, then one of these axiomatizations is strongly normalizing and confluent. We show that their claim as a whole is wrong, but prove confluency and weak normalization by presenting a normalizing rewrite strategy. We can however prove strong normalization for the axiomatizations of a decidable class of such systems. The analysis of the term rewriting properties of the axiomatizations is modulo the associativity and commutativity of the choice operation.

1. Introduction

Recent years have seen an enormous interest in formulating operational semantics in the style of Plotkin's Structured Operational Semantics (SOS) [Plo81]. Several formats have emerged which allow the description of various programming languages and the properties of these have been studied extensively [BIM95], [Sim85],[GV92],[Ver95]. One of these formats is the GSOS format in which the operational semantics of many process calculi of interest can be expressed [BIM95].

In [ABV94] two strategies are presented for automatically deriving an axiomatization for bisimulation equivalence for languages whose operational semantics is expressed in the GSOS format of [BIM95]. This research gives an unambiguous method for generating sound and complete axiom systems. The axiomatizations produced by the strategies are often rather close to existing "human invented" axiomatizations. E.g. the strategies find the left-merge which was conjectured to be

an essential operation for axiomatization of the behavior of merge in several process algebras [BK92], and which was later proved in [Mol89].

A practical application of this theory lies in the simplification of process expressions with the axioms, i.e. how certain operations can be eliminated. Term rewriting of SOS-style axiomatizations is of interest for simplifying process terms as e.g. is done in the Process Algebra Manipulator [Lin93] and the ECRIN-tool [MdSV89].

The subject of this chapter is a claim of the authors of [ABV94] that the axiomatizations produced by their so-called *alternative strategy* have nice term rewriting properties: for a class of GSOS systems in which only finite, non-cyclic transition systems can be expressed the axiomatizations are supposed to be strongly normalizing and confluent on closed terms. We show that their claim as a whole is wrong, but prove weak normalization and confluency. Furthermore we prove strong normalization for the axiomatizations of a decidable class of GSOS systems.

We start with a description of the axiomatization of the output of the alternative strategy and some general term rewriting theory in Section 2. In Section 3 we prove that if we orient the axiomatization produced by the alternative strategy, there is a weakly normalizing rewrite strategy and confluency. In Section 4 we prove that for a decidable class of axiomatizations we even have strong normalization.

2. The Axiomatization of a GSOS system as a TRS

In this chapter we investigate the term rewriting properties of the axiomatizations produced by alternative strategy of [ABV94] for GSOS systems. We combine three different worlds, namely (1) term rewriting systems (TRS's) obtained from (2) axiomatizations produced by the alternative strategy for (3) GSOS systems. The first two are familiar to most computer scientists, whereas the last is known in the process algebraic community.

The focus of this chapter is on term rewriting systems. For technical convenience we introduce a Term Deduction System (TDS) as advocated in [FV95, dV97] to be a unifying framework for the three worlds. We present a term rewriting system, axiomatization and a GSOS system to be instances of a Term Deduction System which allows us to give their connections in a convenient way.

2.1. TRS, Axiomatization and GSOS systems as TDS's. We assume a finite set Act of *actions* with typical elements a, b, c with the usual annotations and a countable (infinite) set of variables Var disjoint from Act, with typical elements x, y, z. A *signature* Σ consists of a set of *operation symbols*, typically f, g and an *arity function* $ar : \Sigma \to \mathbf{N}$ which assigns the *arity* to each operation, i.e., the number of arguments. An operation symbol with arity 0 is called a *constant* symbol. The set of *open terms over* Σ, $\mathbb{T}(\Sigma, Var)$ with typical elements P, Q is the least superset of Var such that if $ar(f) = n > 0$ and $P_1, ..., P_n \in \mathbb{T}(\Sigma, Var)$, then $f(P_1, ..., P_n) \in \mathbb{T}(\Sigma, Var)$. The set of *closed terms over* Σ, $T(\Sigma)$ with typical elements p, q, s, t is the largest subset of $\mathbb{T}(\Sigma, Var)$ of terms which do not contain any variables. $var(P)$ is the set of variables occurring in term P. A Σ-*context* $C[\vec{x}] \in \mathbb{T}(\Sigma, Var)$ is a term in which at most the variables $\vec{x}$ appear. A *substitution* σ is a map from the set of variables into the set of terms over a given signature. This map can easily be extended to the set of all terms by substituting for each variable occurring in an open term its σ-image.

DEFINITION 2.1. A *Term Deduction System*, shortly *TDS*, is a tuple (Σ, D) with Σ a signature and D a set of deduction rules. The set $D = D(T_{\mathsf{R}})$ is parameterized with the set of *relation symbols* T_{R}. Let $\vec{P} = P_1, ..., P_n \in \mathbb{T}(\Sigma, Var)$ and $\mathsf{R} \in T_{\mathsf{R}}$. We call $\mathsf{R}(\vec{P})$ a *positive formula* and $\neg\mathsf{R}(\vec{P})$ a *negative formula.* Most of the time we write formulas with an infix notation for relations and keep the set T_{R} implicit.

A deduction rule $d \in D$ has the form

$$\frac{H}{C}$$

with H a set of formulas and C a single positive formula. We call the elements of H *hypotheses* of d and the formula C its *conclusion.* If the set of hypotheses of a deduction rule is empty, then we call such a rule an *axiom.* If no confusion can arise, we denote an axiom only by its conclusion. The notion "substitution" extends to formulas and deduction rules as expected.

DEFINITION 2.2. A *Term Rewriting System* $\Re = (\Sigma, R)$, shortly *TRS* is a TDS where R is a set of rewrite rules, typically r. *Rewrite rules* are axioms where the conclusion is formed by a formula $lhs \to rhs$ with $lhs, rhs \in \mathbb{T}(\Sigma, Var)$ and $\to$ a binary relation symbol. We suppose that $lhs \notin$ Var and that $var(rhs) \subseteq var(lhs)$. The *one step rewrite*

relation $\rightarrow_R^1$ is the smallest relation on terms closed under substitution and contexts which is based on $\rightarrow$. If $s \in T(\Sigma)$ and there is no s' such that $s \rightarrow_R^1 s'$, then s' is *in normal form with respect to* $\rightarrow_R^1$, shortly *normal form*. The rewrite relation $\rightarrow_R$ is the transitive closure of the one step relation $\rightarrow_R^1$.

DEFINITION 2.3. An *axiomatization* $E = (\Sigma, R)$ is a TRS with the special property that for each axiom $lhs \rightarrow rhs \in R$, there is an axiom $rhs \rightarrow lhs \in R$. For convenience we write both rules as one, i.e., we summarize $lhs \rightarrow rhs$ and $rhs \rightarrow lhs$ as $lhs = rhs$.

DEFINITION 2.4. Let $\Re_1 = (\Sigma, R_1)$ be a TRS and $E = (\Sigma, R_2)$ an axiomatization. The one step rewriting relation $\rightarrow_{R_1/R_2}^1$ is defined as $\rightarrow_{R_2} \circ \rightarrow_{R_1}^1 \circ \rightarrow_{R_2}$. The rewrite relation $\rightarrow_{R_1/R_2}$ is the transitive closure of the one step relation $\rightarrow_{R_1/R_2}^1$. We refer to $\rightarrow_{R_1/R_2}$ also as the rewrite relation $\rightarrow_{R_1}$ *modulo* R_2. The notion of "normal form" extends to rewriting modulo a set of rules as expected.

DEFINITION 2.5. Let $\Re_1 = (\Sigma, R_1)$ be a TRS, $E = (\Sigma, R_2)$ an axiomatization and $s \in T(\Sigma)$. If there is a sequence $s \rightarrow_{R_1/R_2} s_1 \ldots \rightarrow_{R_1/R_2} s_n$, such that s_n is in normal form, then s is *weakly normalizing*. If for every $s \in T(\Sigma)$, s starts a weakly normalizing sequence, then $\Re$ is *weakly normalizing*.

If s starts only weakly normalizing sequences, then s is strongly normalizing. If for every $s \in T(\Sigma)$, s is strongly normalizing, then $\Re$ is *strongly normalizing*.

If for all s_0, s_1 such that $s \rightarrow_{R_1/R_2} s_0$ and $s \rightarrow_{R_1/R_2} s_1$, there is a term s' such that $s_0 \rightarrow_{R_1/R_2} s'$ and $s_1 \rightarrow_{R_1/R_2} s'$, then $\rightarrow_{R_1/R_2}$ is *Church-Rosser* or *confluent*.

DEFINITION 2.6. A *GSOS system* $G = (\Sigma, R)$ is a TDS where R is a set of GSOS rules. Let $\cdot \stackrel{\cdot}{\rightarrow} \cdot \subseteq \mathbb{T}(\Sigma, \mathrm{Var}) \times Act \times \mathbb{T}(\Sigma, \mathrm{Var})$ be a ternary relation, called a *transition* relation and $\cdot \stackrel{\cdot}{\rightarrow} \subseteq \mathbb{T}(\Sigma, \mathrm{Var}) \times Act$ be a binary relation which denotes that a process can perform an action. As usual we write the negative formula $\neg(x \xrightarrow{a})$ as $x \not\xrightarrow{a}$. A *GSOS rule* ρ is a deduction rule of the form

$$\frac{\bigcup_{i=1}^{ar(f)} \{x_i \xrightarrow{a_{ij}} y_{ij} | 1 \leq j \leq m_i\} \cup \bigcup_{i=1}^{ar(f)} \cup \{x_i \not\xrightarrow{b_{ik}} | 1 \leq k \leq n_i\}}{f(x_1, \ldots, x_{ar(f)}) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

where a_{ij}, b_{ij} and $c \in Act$, all variables x_i and y_i are distinct, $m_i, n_i \geq 0$ and $\vec{x} \cup \vec{y} \subseteq \bigcup x_i \cup \bigcup y_i$. The operation symbol f is a *GSOS operation* and the *principal operation of* ρ, $f(\vec{x})$ the *source of* ρ and the term $C[\vec{x}, \vec{y}]$ is the *target of* ρ. If, for some i, $m_i > 0$, then ρ *tests* its i-th argument *positively*. If, for some i, $n_i > 0$, then ρ *tests* its i-th argument *negatively*. f *tests* its i-th argument *positively* (*negatively*) if there is a GSOS rule $\rho \in R$ with $m_i > 0$ ($n_i > 0$).

2.2. GSOS rules and Bisimulation. In this section we present the formats of GSOS rules defined in [ABV94]. Most examples of GSOS systems stem for this paper. We end with a definition of bisimulation over GSOS systems.

This section can safely be skipped by readers only interested in rewriting with axiomatizations. Nevertheless we present it to grasp the rationale of [ABV94] for the axiomatization used for a GSOS system.

EXAMPLE 2.7. Let Σ be the signature which contains only the priority operation Θ taken from [BBK86]. The deduction rule

$$\frac{x \xrightarrow{a} x' \quad x \xrightarrow{b}\!\!\!\!\!\!/\;\;}{\Theta(x) \xrightarrow{a} \Theta(x')}$$

where $a, b \in Act$ is a GSOS rule.

REMARK 2.8. Notice that the argument of Θ is tested both positively and negatively.

EXAMPLE 2.9. Let G be the TDS with signature $\{+\}$ and two rules

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

for some fixed $a \in Act$. Then G is a GSOS system.

DEFINITION 2.10. Let $G = (\Sigma_G, R_G)$ be a GSOS system and $\rho \in R_G$. Let ρ be of the form

$$\frac{\{x_i \xrightarrow{a_i} y_i | i \in I\} \cup \{x_i \xrightarrow{b_{ij}}\!\!\!\!\!\!/\;\;| i \in K, 1 \leq j \leq n_i\}}{f(x_1, ..., x_l) \xrightarrow{c} C[\vec{x}, \vec{y}],}$$

where $I \cap K = \emptyset$ and $I \cup K = \{1, .., l\}$ and x_i does not appear in $C[\vec{x}, \vec{y}]$, then ρ is *smooth*. An operation f is *smooth* iff every rule $\rho \in R_G$ with f as principal operation is smooth. G is *smooth* iff every $\rho \in R_G$ is smooth.

EXAMPLE 2.11. The rule in Example 2.7 for Θ is an example of a non smooth rule. The rules in the GSOS system G of Example 2.9 are examples of smooth GSOS rules and G is a smooth GSOS system.

DEFINITION 2.12. Let $G = (\Sigma_G, R_G)$ be a GSOS system and $f \in \Sigma_G$ be a smooth operation. If for each i-th argument of f either all rules in R_G with principal operation f test the i-th argument positively, or none of them does, and for each pair of different rules in R_G there is an argument which is tested positively in both rules, but with a different action, then f is *distinctive.*

EXAMPLE 2.13. The operation Θ over the GSOS system in Example 2.9 and $+$ over the GSOS system in Example 2.11 are not distinctive. The GSOS system with signature $\{\mathbf{0}, a\cdot, \between, \|\}$ which describes action prefixing, left-merge and merge respectively for a single action $a \in Act$ is given by

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

and describes all operations except $\|$ as distinctive operations.

DEFINITION 2.14. Let $G = (\Sigma_G, R_G)$ be a GSOS system and $\rho \in R_G$ with principal operation f be a smooth GSOS rule as in Definition 2.10. If there is no argument i of f such that f is tested negatively and x_i occurs in the target $C[\vec{x}, \vec{y}]$, then ρ is *discarding*. G is *discarding* iff every $\rho \in R_G$ is discarding.

EXAMPLE 2.15. All rules in previous examples, except the rule for Θ in Example 2.7 are discarding. The rule

$$\frac{x \not\xrightarrow{a}}{f(x) \xrightarrow{a} x}$$

is smooth, distinctive, but not discarding.

DEFINITION 2.16. Let $G = (\Sigma_G, R_G)$ be a GSOS system. The transition relation $\rightarrow_G \subseteq T(\Sigma) \times Act \times T(\Sigma)$ over G is the set of positive transition formulas or *transitions* provable by the rules of R_G for closed terms.

EXAMPLE 2.17. Let G be the GSOS system of Example 2.13. Then $\rightarrow_G$ contains the transitions $a \cdot \mathbf{0} \xrightarrow{a} \mathbf{0}$ and $(a \cdot \mathbf{0}) \between \mathbf{0} \xrightarrow{a} \mathbf{0} \| \mathbf{0}$, since there

are proofs

$$\frac{}{a \cdot \mathbf{0} \xrightarrow{a} \mathbf{0}} \qquad \frac{\dfrac{}{a\cdot\mathbf{0} \xrightarrow{a} \mathbf{0}}}{(a \cdot \mathbf{0}) \between \mathbf{0} \xrightarrow{a} \mathbf{0} \| \mathbf{0}}$$

of these transitions. "Proofs" are the expected notion, see [ABV94] for the exact definition.

DEFINITION 2.18. Let $G = (\Sigma_G, R_G)$ be a signature and $\rightarrow_G$ the transition relation over G.

1. Let $Bis \subseteq T(\Sigma) \times Act \times T(\Sigma)$ be a relation satisfying for every $(p, q) \in Bis$,
   (a) If $p \xrightarrow{a}_G p'$, then there is a q' such that $q \xrightarrow{a}_G q'$ and $(p', q') \in Bis$, and
   (b) If $q \xrightarrow{a}_G q'$, then there is a p' such that $p \xrightarrow{a}_G p'$ and $(p', q') \in Bis$,

   then Bis is a *bisimulation over* G.
2. If there exists a bisimulation Bis over G, such that $(p, q) \in Bis$, then p and q are *bisimilar over* G, shortly $p \leftrightarroweq_G q$.

A special GSOS system we use throughout this chapter is for a fragment of CCS [Mil86] which describes the finite trees.

DEFINITION 2.19. $G_{\text{FINTREE}} = (\Sigma_{\text{FINTREE}}, R_{\text{FINTREE}})$ is the GSOS system where $\Sigma_{\text{FINTREE}} = \{+, \mathbf{0}\} \cup PreAct$ and R_{FINTREE} is the set of rules of the form

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

with $a \in Act$ and $a\cdot \in PreAct$ a prefixing action.

DEFINITION 2.20. Let $G = (\Sigma_G, R_G)$ be a GSOS system. The GSOS system $H = (\Sigma_H, R_H)$ is a *disjoint extension* of G iff $\Sigma_G \subset \Sigma_H$ and R_H introduces no new rules for the operators in Σ_G.

In the remainder of this chapter we only study disjoint extensions of G_{FINTREE}.

DEFINITION 2.21. A GSOS system (Σ, R) is *semantically well-founded* iff there is no infinite sequence $p_0 \xrightarrow{a_0}_G p_1 \ldots$ with $p_i \in T(\Sigma)$.

Without proof we state the following.

PROPOSITION 2.22. Let $G = (\Sigma_G, R_G)$ be a disjoint extension of G_{FINTREE}. If for every $p \in T(\Sigma_G)$ there exists a $q \in T(\Sigma_{\text{FINTREE}})$ such that $p \leftrightarrows_G q$, then G is *semantically well-founded.*

Semantic well-foundedness is however an undecidable property. It is possible to express so-called 2-counter machines in GSOS, that are known to be Turing powerful [Vaa93].

2.3. Axioms for a GSOS system. In [ABV94] two algorithms are presented which produce for semantically well-founded GSOS systems an axiomatization which is sound and complete with respect to bisimulation equivalence. In this section we focus on the so-called *alternative strategy* as described in [ABV94], of which the authors believed that the output has nice term rewriting properties. We do not give the modus operandi of the algorithm, but present only the axiomatization that is produced. We describe only the form of different axioms in the output, which is sufficient to verify the term rewriting properties. At the end of this section we present an elaborate example of an output of the alternative strategy.

The alternative strategy produces for a GSOS system $G = (\Sigma_G, R_G)$, which disjointly extends G_{FINTREE}, a new GSOS system which extends G and an axiomatization $E_{G'} = (\Sigma_{G'}, R_{G'})$ on which our attention is focused.

DEFINITION 2.23. *FINTREE* is the axiomatization given by Σ_{FINTREE} and the rules

$$x + y = y + x \tag{1}$$
$$(x + y) + z = x + (y + z) \tag{2}$$
$$x + x = x \tag{3}$$
$$x + \mathbf{0} = x \tag{4}$$

Besides FINTREE, the output of the alternative strategy has axioms for the GSOS operations in $\Sigma_{G'} - \Sigma_{\text{FINTREE}}$. These are divided in three classes: [smooth-discarding-distinctive], [smooth-discarding and non distinctive] and [non smooth and discarding] with respect to G'. Only for the [smooth-discarding-distinctive] operations the strategy can generate the axioms describing their behavior. The other operations are linked with copying and distinctifying axioms to [smooth-discarding-distinctive] operations.

Each *non* [smooth-discarding-distinctive] operation is linked in the output to either a [smooth-discarding and non distinctive] or a [smooth-discarding-distinctive] operation with a so-called copying axiom. "Copying" refers to the possible multiplication of arguments going from left-to-right.

DEFINITION 2.24. Let $f \in \Sigma_{G'}$ be a non [smooth and discarding] operation and $A \in R_{G'}$ an axiom of the form

$$f(x_1, ..., x_{ar(f)}) \quad = \quad f^c(x_1', ..., x_{ar(f^c)}'), \tag{5}$$

where $x_i' \equiv x_j$ for some j, and f^c is a [smooth-discarding-distinctive] or [smooth-discarding non distinctive] operation. Then A is called a *copying* axiom.

In the proofs in subsequent sections we will use the maximum number an argument is copied by a copy axiom.

DEFINITION 2.25. Let $cf_{G'} : \Sigma_{G'} \times \mathbf{N}^+ \to \mathbf{N}^+$ be a function defined as follows. Let $f \in \Sigma_{G'}$ and $i \in \mathbf{N}^+$. If f has no copy axiom in $R_{G'}$ or $i > ar(f)$ then $cf(f,i) = 1$. If f has a copy axiom and $i \leq ar(f)$ then $cf(f,i) = n$, if the argument x_i occurs n times in $f^c(x_1', ..., x_{ar(f^c)}')$. The *maximum copy factor* $cf_{G'}$ is defined as $max(\{cf(f,i) | f \in \Sigma_{G'}, i \in \mathbf{N}^+\})$.

Second all [smooth-discarding non distinctive] operations are linked to [smooth-discarding-distinctive operations] with so-called distinctifying axioms.

DEFINITION 2.26. Let $f \in \Sigma_{G'}$ be a [smooth-discarding non distinctive] operation and $A \in R_{G'}$ an axiom of the form

$$f(x_1, ..., x_{ar(f)}) \quad = \quad \sum_{i=1}^{p} f_i(x_1, ..., x_{ar(f)}), \tag{6}$$

where all f_i's are [smooth-discarding-distinctive] operations. Then A is called a *distinctifying* axiom.

In proofs in subsequent sections we need the (maximum) distinctivity factor of operations.

DEFINITION 2.27. Let $df_{G'} : \Sigma_{G'} \to \mathbf{N}^+$ be a function defined as follows. If f has no distinctivity axiom in $R_{G'}$, then $df_{G'}(f) = 1$. If $f(x_1, ..., x_{ar(f)}) = \sum_{i=1}^{n} f_i(x_1, ..., x_{ar(f)})$ is a distinctivity axiom, then

$df_{G'}(f) = n$. The *maximum distinctivity factor* $df_{G'}$ is defined as $max(\{df_{G'}(f) | f \in \Sigma_{G'}\})$.

Finally for all (introduced) [smooth-discarding-distinctive] operations, axioms are generated describing their interaction with the FIN-TREE operations. Intuitively the action axiom describes the result after the process has taken one step.

DEFINITION 2.28. Let $f \in \Sigma_{G'}$ be a [smooth-discarding-distinctive] operation and $A \in R_{G'}$ an axiom of the form

$$f(P_1, ..., P_{ar(f)}) = a \cdot C[x_1, ..., x_{ar(f)}], \tag{7}$$

where P_i is of the form $a_i \cdot x_i$, x_i or $\mathbf{0}$ and x_i appears only in $C[x_1, ..., x_{ar(f)}]$ if $P_i \not\equiv \mathbf{0}$. Then A is called an *action* axiom.

The distributivity axiom describes the interaction between GSOS operations and the $+$ operation.

DEFINITION 2.29. Let $f \in \Sigma_{G'}$ be a [smooth-discarding-distinctive] operation and $A \in R_{G'}$ an axiom of the form

$$\begin{aligned} f(x_1, ..., x_i + y_i, ..., x_{ar(f)}) = {} & f(x_1, ..., x_i, ..., x_{ar(f)}) \\ & + f(x_1, ..., y_i, ..., x_{ar(f)}). \end{aligned} \tag{8}$$

Then A is called a *distributivity* axiom.

The inaction axiom identifies the GSOS terms which have no behavior with the constant $\mathbf{0}$.

DEFINITION 2.30. Let $f \in \Sigma_{G'}$ be a [smooth-discarding-distinctive] operation and $A \in R_{G'}$ an axiom of the form

$$f(P_1, ..., P_{ar(f)}) = \mathbf{0}, \tag{9}$$

where P_i is of the form $a_i \cdot x_i$, $b_i \cdot x_i + y_i$, x_i or $\mathbf{0}$. Then A is called an *inaction* axiom.

The peeling axiom "peels" of parts of a term, which cannot influence its behavior in any way.

DEFINITION 2.31. Let f be a [smooth-discarding-distinctive] operation and $A \in R_{G'}$ an axiom of the form

$$f(P_1, ..., b_i.x_i + y_i, ..., P_{ar(f)}) \quad = \quad f(P_1, ..., y_i, ..., P_{ar(f)}) \tag{10}$$

where P_j is of the form $a_j \cdot x_j$ or x_j. Then A is called a *peeling* axiom.

REMARK 2.32. The attentive reader may have noticed that we have specified in some detail what the syntactical form is of the different axioms of the axiomatization. We have not specified which axioms are actually *included* in the axiomatization produced by the alternative strategy. A precise description can be found in the original paper. Furthermore in [ABV94] is proved that all axioms are sound with respect to bisimulation equivalence.

A full example of an output of the alternative strategy

EXAMPLE 2.33. Let $G = (\Sigma_G, R_G)$ be the GSOS system with $\Sigma_G = \{\mathbf{0}, +, a\cdot, \Theta, \|\}$ and a collection of rules of previous examples of Section 2.2

$$\frac{}{a \cdot x \xrightarrow{a} x}$$

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

$$\frac{x \xrightarrow{a} x' \;\; x \overset{b}{\not\longrightarrow}}{\Theta(x) \xrightarrow{a} \Theta(x')}$$

$$\frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'}$$

which is a disjoint extension of G_{FINTREE} for some $a, b \in Act$ as the reader can verify. The alternative strategy first outputs the FINTREE axioms. Next it outputs the the copying axiom

$$\Theta(x) = \Theta^c(x, x)$$

for the (non [smooth and discarding]) rule for the priority operation Θ and extends the GSOS system with the ([smooth and discarding]) rule

$$\frac{x \xrightarrow{a} x' \;\; y \overset{b}{\not\longrightarrow}}{\Theta^c(x, y) \xrightarrow{a} \Theta(x').}$$

For the ([smooth-discarding non distinctive]) rules for the merge operation $\|$ it adds the distinctifying axiom

$$x \| y = x \|_1 y + x \|_2 y$$

and two ([smooth-discarding-distinctive]) rules

$$\frac{x \xrightarrow{a} x'}{x \|_1 y \xrightarrow{a} x' \| y} \qquad \frac{y \xrightarrow{a} y'}{x \|_2 y \xrightarrow{a} x \| y'.}$$

Last it adds for the ([smooth-distinctive-discarding]) operations $\mathbf{0}$, Θ^c, $\|_1$ and $\|_2$ the action axioms

$$\begin{aligned} \Theta^c(a \cdot x, \mathbf{0}) &= a \cdot \Theta(x) \\ (a \cdot x) \|_1 y &= a \cdot (x \| y) \\ x \|_2 (a \cdot y) &= a \cdot (x \| y), \end{aligned}$$

the distributivity axioms

$$\begin{aligned}\Theta^c(x+y,z) &= \Theta^c(x,z)+\Theta^c(y,z)\\ (x+y)\|_1 z &= x\|_1 z + y\|_1 z\\ x\|_2(y+z) &= x\|_2 y + x\|_2 z,\end{aligned}$$

the inaction axioms

$$\begin{aligned}\mathbf{0}\|_1 y &= \mathbf{0}\\ x\|_2\mathbf{0} &= \mathbf{0}\\ \Theta^c(\mathbf{0},x) &= \mathbf{0}\\ \Theta^c(a\cdot x, b\cdot y+z) &= \mathbf{0}\end{aligned}$$

and the peeling axiom

$$\Theta^c(a\cdot x, a\cdot y+z) = \Theta^c(a\cdot x, z).$$

2.4. Term Rewriting with Axioms of a GSOS system. Rewriting with axioms starts with orienting the axioms of the previous section from left-to-right, or right-to-left to which we refer as *rulifying*. The TRS's we consider consist of the axioms 3...10 of the previous section oriented from left-to-right. We do not include an associativity and commutativity rewrite rule, for reasons we will explain later. The rewrite rules will be named after the axioms, i.e. *action* rewrite rules, if they stem from action axioms, *copy* rewrite rules, if they stem from copy axioms, etc.

For obvious reasons we have not oriented the FINTREE axioms $x+\mathbf{0}=x$ and $x+x=x$ as $x \to x+\mathbf{0}$ and $x \to x+x$, because then all terms lose the strong normalization property, e.g. $s \to s+\mathbf{0} \to (s+\mathbf{0})+\mathbf{0}$ ... . To maintain confluency we now have to introduce "extra" inaction and peeling rewrite rules, mimicking the effect of the combination of the two FINTREE and inaction and peeling rewrite rules, without unnecessary spoiling of rewriting properties.

DEFINITION 2.34. Let $f \in \Sigma_{G'}$ be a [smooth-discarding-distinctive] operation [1]. If f has an (extra) inaction rewrite rule of the form

$$f(P_1,...,b_i\cdot x_i+y_i,...,P_{ar(f)}) \;\to\; \mathbf{0}$$

then

$$f(P_1,...,b_i\cdot x_i,...,P_{ar(f)}) \;\to\; \mathbf{0}$$

[1] We chose to be informal and not use a inductive definition or fixed point construction to define the extra inaction and peeling rewrite rules.

is an *extra inaction* rewrite rule. If f has a (extra) peeling rewrite rule of the form

$$f(P_1, ..., b_i \cdot x_i + y_i, ..., P_{ar(f)}) \rightarrow f(P_1, ..., y_i, ..., P_{ar(f)})$$

then

$$f(P_1, ..., b_i \cdot x_i, ..., P_{ar(f)}) \rightarrow f(P_1, ..., \mathbf{0}, ..., P_{ar(f)})$$

is an *extra peeling* rewrite rule.

EXAMPLE 2.35. The extra inaction and peeling rules for the oriented axiomatization of Example 2.33 consist of the extra inaction rule

$$\Theta^c(a \cdot x, b \cdot x) \rightarrow \mathbf{0}$$

and extra peeling rule

$$\Theta^c(a \cdot x, a \cdot y) \rightarrow \Theta^c(a \cdot x, \mathbf{0}).$$

The working TRS we assume in this chapter is an axiomatization produced by the alternative strategy oriented from left-to-right as indicated above *with* the extra inaction and peeling rewrite rules defined in this section. We refer to this TRS as the *rulified axiomatization of the alternative strategy*, shortly the *rulified axiomatization* which is typically $\Re_{G'} = (\Sigma_{G'}, R_{G'})$. We also use $\Re_{G'}$ with and without the action rules, which we refer to as the (TRS of the) *(non) action rewrite rules.*

We have not oriented the associativity and commutativity axiom for the FINTREE operation $+$ for the simple reason that they cannot be oriented without losing the normalizing property: $s+t \rightarrow t+s \rightarrow s+t$ ... Therefore we cannot use "ordinary" term rewriting, but have to use the more complex rewriting modulo the commutativity of the $+$ operator. However, it is well-known that this is not sufficient for the associativity still spoils the normalizing property and so we have to work modulo the associativity of the $+$ as well. As usual we denote the commutativity and associativity axioms by *ac*.

In this chapter we exclusively use the rewrite relation $\rightarrow_{R_{G'}}$ modulo the associativity and commutativity axiom for the $+$ operation as defined in Definition 2.4, which we refer to simply as $\rightarrow$.

In this chapter we have made the deliberate choice to specify the form of the rewrite rules in the rulified axiomatization, and not *which rewrite rules are in it* as noted in Remark 2.32. We have tried to

repeat as little as possible from [ABV94], to retain readability and avoid overlap. Consequence of this choice is that we cannot reason about the precise content of a rulified axiomatization. We have distilled two properties, that we cannot prove here, but follow immediately from [ABV94] and the definition of the rulified axiomatization. It is essential in the proof of the existence of normalizing strategy and confluency in the next section (Lemma 3.13 and Corollary 3.17).

PROPOSITION 2.36. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. Let $f \in \Sigma_{G'}$ be a non FINTREE operation and $p_1, ..., p_{ar(f)} \in T(\Sigma_{G'})$. Then $f(p_1, ..., p_{ar(f)})$ is a redex.

PROPOSITION 2.37. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. Let $p \in T(\Sigma_{G'})$. If $p \to q$, $p \to q'$ and $q, q' \in T(\Sigma_{\text{FINTREE}})$, then $q =_{\text{FINTREE}} q'$.

3. Confluency and Weak Normalization

In [ABV94] is claimed that the rulified axiomatization has nice term rewriting properties. The authors conjecture that the axiomatization is strongly normalizing and confluent for the (undecidable) class of GSOS systems which are so-called *semantically well-founded.* The next small example disproves their claim: it shows a semantically well-founded GSOS system with an axiomatization produced by the alternative strategy which is not strongly normalizing. However, we prove that a normalizing strategy exists which implies weak normalization and confluency.

EXAMPLE 3.1. Suppose G is the GSOS system which is the disjoint extension of FINTREE with two operations f and g and only one rule

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} g(f(a \cdot y)).}$$

Then the action rule $f(a \cdot y) \to a \cdot g(f(a \cdot y))$ and inaction rule $g(x) \to \mathbf{0}$ are obtained by rulifying the axioms produced by the alternative strategy. G is semantically well-founded, as can be verified easily, but $f(a \cdot \mathbf{0})$ is not strongly normalizing modulo ac, because

$$f(a \cdot \mathbf{0}) \to a \cdot g(f(a \cdot \mathbf{0})) \to a \cdot g(a \cdot g(f(a \cdot \mathbf{0})))...$$

The following strategy however is normalizing for the rulified axiomatization of a semantically well-founded GSOS system.

DEFINITION 3.2. The strategy *Normalize* contracts redexes in the following way.

1. Contract all non action redexes.
2. Contract the outermost action redex surrounded [2] by the least number of action prefixing operations.
3. Repeat until no redexes are left.

REMARK 3.3. The strategy is *non deterministic*, e.g. the non action redexes can be contracted in any order. Furthermore the strategy is Markov, in the sense that there is no need to keep the history of the reduction.

Normalize reduces the weight with application of non action rules

To prove that Normalize is indeed normalizing, we first prove that all non action rules decrease the weight of terms of some weight map. The weight map was inspired by the one used in [AB90].

DEFINITION 3.4. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. Let $\hat{} : \Sigma_{G'} \to \Sigma_{G'}$ be the origin map which is defined as $\hat{f^c} = \hat{f}_i = \hat{f} = f \in \Sigma_{G'}$.

EXAMPLE 3.5. In the rulified axiomatization of Example 2.33 $\hat{\Theta}^c = \Theta$, $\hat{\|}_1 = \hat{\|}_2 = \|$ and $\hat{+} = +$.

DEFINITION 3.6. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. Let $\bar{\imath} : \Sigma_{G'} \times \mathbf{N}^+ \to \mathbf{N}^+$ be a map which assigns the maximum copy factor $cf_{G'}$ to each [smooth-distinctive-discarding] f and i such that the i-th argument of f in the left-hand side of some action rule for $f \in R_{G'}$ is x_i, else 1.

EXAMPLE 3.7. In the rulified axiomatization of Example 2.33 the maximum copy factor is 2 and hence $\bar{\imath}(\Theta^c, 1) = 2$ and $\bar{\imath}(\Theta^c, 2) = 1$. For all other operations f of the signature of the rulified axiomatization $\bar{\imath}(f, i) = 1$ for $i \in \mathbf{N}^+$.

---

[2] We say that a sub term or a redex q' of a term q is *surrounded* by an action prefixing operation iff q has a sub term of the form $a \cdot C[t']$, $C[]$ possibly the trivial context.

DEFINITION 3.8. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization and $w : \Sigma_{G'} \to \mathbf{N}$ a map. The weight map $|-|_w : T(\Sigma_{G'}) \to \mathbf{N}$ is defined as follows. Let $f \in \Sigma_{G'}$ and $\vec{p} = p_1, p_2 \ldots, p_{ar(f)} \in T(\Sigma_{G'})$, then

$$
\begin{array}{lcll}
|\mathbf{0}|_w & = & 2 & \\
|p_1 + p_2|_w & = & |p_1|_w + |p_2|_w & \\
|a \cdot p_1|_w & = & (cf_{G'} + 2).|p_1|_w & \\
|f(\vec{p})|_w & = & 1 + |f^c(\vec{p'})|_w & f \text{ has a copy rule in } R_{G'} \\
|f(\vec{p})|_w & = & 1 + \sum_{i=1}^{df_{G'}(f)} |f_i(\vec{p})|_w & f \text{ has a distinctifying rule in } R_{G'} \\
|f|_w & = & w(\hat{f}) & f \not\equiv \mathbf{0} \text{ is a constant} \\
|f(\vec{p})|_w & = & w(\hat{f}) \sum_{i=1}^{ar(f)} \bar{\imath}(f,i).|p_i|_w & \textit{otherwise},
\end{array}
$$

where $\vec{p'} = p_1', \ldots, p_{ar(f^c)}'$ and $p_i' = p_j$ for some $1 \leq j \leq ar(f)$.

REMARK 3.9. That $|-|_w$ is well-defined can be verified with an argument using that the weight of a copy (or a distinctifying redex) is calculated from a redex which is no longer a copy redex (and no longer a distinctifying redex.)

The reader may be blurred by the abundance of implicit maps in the definition of $|-|_w$. The maps $\hat{}, \bar{\imath}$ and $df_{G'}$ and constant $cf_{G'}$ are determined *solely* by the rulified axiomatization, as one can easily verify. The definition of $|-|_w$ is much simpler, if we choose to prove normalization alone. We use its full strength in proving strong normalization in Section 4, where we formulate some extra conditions on the rulified axiomatizations and the map w.

As we will see w can be chosen in such a way that all non action rewrite rules respect $|-|_w$. In this section we prove that weight maps $|-|_{w_0}$, such that $w_0(f) \geq 3$ for all $f \in \Sigma_{G'} - \Sigma_{\text{FINTREE}}$ are sufficient to prove that the strategy in Definition 3.2 is normalizing.

LEMMA 3.10. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. If $p \neq \mathbf{0} \in T(\Sigma_{G'})$ then $|p|_{w_0} > 2$.

Proof. Trivial, since $w_0(f) \geq 3$ for all non FINTREE operations □

LEMMA 3.11. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. If p rewrites to q with a non action rewrite rule then $|p|_{w_0} > |q|_{w_0}$.

Proof. Let $p \equiv C[p'] \in T(\Sigma_{G'})$, where p' is a non action redex. The proof is by an straight forward induction on the complexity of $C[]$. We

present one case of the Base Step, which explains the use of w_0.

-Suppose $p \equiv f(p_1, ..., p_{ar(f)})$ is an (extra) inaction redex. Notice that since f has an (extra) inaction redex, f is [smooth-discarding-distinctive] and has no copy or distinctifying axiom. Distinguish $(\frac{T}{F})$ whether or not $ar(f) = 0$. (T) If $ar(f) = 0$, then the conclusion follows immediately, since $|f|_{w_0} = w_0(f) \geq 3 > 2 = |\mathbf{0}|_{w_0}$.

(F) If $ar(f) > 0$, then

$$\begin{array}{lcl} |f(p_1, ..., p_{ar(f)})|_{w_0} & \geq & \{\bar{\imath}(f, i) \geq 1 \text{ for all } i\} \\ w_0(\hat{f})^{\sum_{i=1}^{ar(f)} |p_i|_{w_0}} & > & \{w_0(f) \geq 3, \text{Lemma 3.10}\} \\ 2 & = & \\ |\mathbf{0}|_{w_0} \square & & \end{array}$$

LEMMA 3.12. The TRS of non action rewrite rules of a rulified axiomatization is strongly normalizing modulo ac for closed terms.

Proof. It is a straight-forward induction proof to verify that for all p' such that $p \rightarrow_{ac} p'$ holds $|p'|_{w_0} = |p|_{w_0}$. Now with Lemma 3.11 the result follows $\square$

Normalize is head normalizing

Second we prove the head normalization property[3] of Normalize.

LEMMA 3.13. Normalize is head normalizing on closed terms for the rulified axiomatization.

Proof. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization and $p \in T(\Sigma_{G'})$. We prove by an induction on the complexity of p that Normalize is head normalizing for p. The Base Step is trivial. Let the Induction Hypothesis (I.H) be that Normalize is head normalizing for $p_1, ..., p_{ar(f)}$. Let $p \equiv f(p_1, ..., p_{ar(f)})$ and the head symbol f be the only *marked* symbol. The (marked) copies of this f will be introduced solely by copy, distinctifying or distributivity rewrite rules.

First suppose that during the reduction an *infinite* number of marked fs is created. From Lemma 3.12 we get that after a finite number of rewrite steps all marked f's present in the reduct can no longer be head symbols of copy or distinctifying redexes. So to create an infinite number of marked f's an infinite number of distributivity redexes are contracted. With König's Lemma this implies that at least one of the arguments $p_1, ..., p_{ar(f)}$ has an infinite reduction, which is *not* head normalizing. Contradiction with the I.H.. Thus the number of marked f's created during the reduction with Normalize is *finite.*

Now suppose the number of marked f's created is finite. At any time in the reduction, the marked f's have a finite number of arguments, derived from $p_1, ..., p_{ar(f)}$. By the I.H. these arguments (or their derivations, to be more precise) are reduced to head normal form. Consequence of Proposition 2.36 is that after finitely many rewrite steps the reduct is a sum of terms which are action or (extra) inaction redexes. With Normalize all (extra) inaction redexes are rewritten to the head normal form $\mathbf{0}$. The order in which action redexes are chosen in step 2 of Normalize guarantees that all outermost action redexes are rewritten to head normal form $\Box$

Normalize is indeed normalizing

Third a recursive application of the argument that Normalize is head normalizing implies normalization.

THEOREM 3.14. The strategy Normalize is normalizing for the rulified axiomatization of a semantically well-founded GSOS system.

Proof. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization and $p \in T(\Sigma_{G'})$. Suppose towards a contradiction that there is an *infinite* ac reduction with the strategy Normalize of p. Then by Lemma 3.13 we know that p is reduced to the form $\sum_{i=1}^{n} a_i \cdot p_i$, where the p_i's are reduced to the same form ad infinitum. Now the reduct is not bisimilar to a FINTREE term. Contradiction □

EXAMPLE 3.15. As opposed to the reduction in Example 3.1 the contraction of the outermost inaction redex is not postponed indefinitely long by the strategy Normalize.

$$\begin{array}{lcl} f(a \cdot \mathbf{0}) & \rightarrow & \{\text{action}\} \\ a \cdot g(f(a \cdot \mathbf{0})) & \rightarrow & \{\text{inaction}\} \\ a \cdot \mathbf{0}. & & \end{array}$$

PROPOSITION 3.16. If $p, q \in T(\Sigma_{\text{FINTREE}})$ and $p \leftrightarrows_{G_{\text{FINTREE}}} q$ and are in normal form with respect to the FINTREE rewrite rules[4], then $p =_{ac} q$.

COROLLARY 3.17. The rulified axiomatization of a semantically well-founded GSOS system is weakly normalizing and Church-Rosser on closed terms.

Proof. Weak normalization follows immediately from the existence of a normalizing strategy in Theorem 3.14. Now for the proof of Church-Rosser, suppose that $p \rightarrow p'$, $p \rightarrow p''$ and that p', p'' are in normal form. By Proposition 2.36 $p', p'' \in T(\Sigma_{\text{FINTREE}})$. The result follows immediately from Propositions 2.37 and 3.16 □

[4]i.e. the axioms $x+x = x$ and $x+\mathbf{0} = \mathbf{0}$ oriented from left-to-right as described at the beginning of Section 2.4.

4. A Strongly Normalizing Subclass

In this section we prove that a decidable class of rulified axiomatizations is strongly normalizing. The definition of the class of *copy safe* and *syntactically well-founded* rulified axiomatizations[5] is rather technical. The class encompasses the rulified axiomatizations for most frequently seen operators seen in the literature, such as $\Theta, \|, +$ etc.

DEFINITION 4.1. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. If for every action rule $lhs \to rhs \in R_{G'}$ holds that every variable in rhs occurs only once, then $lhs \to rhs$ is *linear*. If every action rule in $R_{G'}$ is linear, then $\Re_{G'}$ is *linear*.

EXAMPLE 4.2. The rulified axiomatization of Example 2.33 is linear.

Let G be the GSOS system which disjointly extends G_{FINTREE} with the operator *twice* and rule

$$\frac{}{twice(x) \xrightarrow{a} x + x.}$$

The rulified axiomatization $\Re_{G'}$ consists amongst others of the the action rule $twice(x) = a \cdot (x + x)$, which is not linear.

PROPOSITION 4.3. It is decidable whether a rulified axiomatization is linear.

DEFINITION 4.4. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a linear rulified axiomatization and $w : \Sigma_{G'} \to \mathbf{N}$ a map.

1. Let $r^c : g(\vec{x}) \to g^c(\vec{x'}) \in R_{G'}$ be a copy rule. The argument $x_i \in \bigcup \vec{x}$ is a *multiplied argument* if x_i appears more than once in $g^c(\vec{x'})$.
2. $\Re_{G'}$ is *copy safe* if there is no action rule $r^a : f(\vec{P}) \to a \cdot g(\vec{x'}) \in R_{G'}$ with $w(f) = w(g)$ such that the i-th argument of g is multiplied by a copy rule.

EXAMPLE 4.5. The rulified axiomatization of Example 2.33 is copy safe.

[5] This class is inspired by the definition of a linear and syntactically well-founded GSOS system in [ABV92, ABV94], which is proved to define semantically well-founded GSOS systems. The definition of syntactic well-foundedness in this chapter follows [ABV92].

Let G be the GSOS system which disjointly extends G_{FINTREE} with the operator f and the rule

$$\frac{x \to x'}{f(x) \xrightarrow{a} f(x).}$$

The rulified axiomatization $\Re_{G'}$ consists amongst others of the copying rule $r^c : f(x) \to f^c(x,x)$ and the action rule $r^a : f^c(a \cdot x, y) \to a \cdot f(y)$. Then the argument y of $f(y)$ is multiplied by r^c. Hence $\Re_{G'}$ is not copy safe.

PROPOSITION 4.6. It is decidable whether a rulified axiomatization is copy safe.

PROPOSITION 4.7. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a copy safe rulified axiomatization with map $w : \Sigma_{G'} \to \mathbf{N}$. Let $r^a : f(\vec{P}) \to a \cdot g(\vec{x'}) \in R_{G'}$ be an action rule and $r^c : g(\vec{x}) \to g^c(\vec{y}) \in R_{G'}$ a copy rule. If $f(\vec{P})$ rewrites consecutively with r^a and r^c to $g^c(\vec{y'})$, then for each variable x_i that appears in $f(\vec{P})$ holds

1. either x_i does not appear in $g^c(\vec{y'})$, or
2. x_i appears only once in $g^c(\vec{y'})$, or
3. x_i appears maximally $cf_{G'}$ times in $g^c(\vec{y'})$ and the copies in $g^c(\vec{y'})$ are not multiplied by r^c.

DEFINITION 4.8. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization. If there exist maps $w : \Sigma_G \to \mathbf{N}$ and $W_w : \mathbb{T}(\Sigma_{G'}, Var) \to \mathbf{N}$ defined by

$$\begin{aligned} W_w(x) &= 0 \\ W_w(f(\vec{P})) &= w(f) + W_w(P_1) + \ldots + W_w(P_{ar(f)}) \end{aligned}$$

such that

1. if $f, g \in \Sigma_{G'}$ and $\hat{g} = \hat{f}$, then $w(g) = w(f)$, and
2. for every action prefixing operation $a\cdot \in \Sigma_{G'}$ holds $w(a\cdot) \geq 1$, and
3. for each action rule of the form $f(P_1, \ldots, P_{ar(f)}) \to a \cdot C[x_1, \ldots, x_{ar(f)}]$ holds
   (a) $W(C[\vec{x}, \vec{y}]) \leq w(f)$, if there is a P_i of the form $a_i \cdot x_i$, and
   (b) otherwise $W(C[\vec{x}, \vec{y}]) < w(f)$,

then $\Re_{G'}$ is a *syntactically well-founded with* w.

LEMMA 4.9. It is decidable whether a rulified axiomatization is syntactically well-founded.

Proof. The check for syntactic well-foundedness comes down to solving a linear system of Diophantine equations, which is known to be decidable [PS82] □

EXAMPLE 4.10. The rulified axiomatization in Example 2.33 is syntactically well-founded with the map w, where $w(f) = 1$ for $f \in \Sigma_G$.

The rulified axiomatization in Example 3.1 is not syntactically well-founded.

In Lemma 3.11 we saw that the non action rules respect a weight map $|-|_{w_0}$, where $w_0 : \Sigma_{G'} \to \mathbf{N}^+$ is a map such that $w_0(f) \geq 3$ for all non FINTREE operations. We set out to prove that *with* the proviso of linearity, copy safety and syntactic well-foundedness, action rules diminish the weight as well. However, in general $|-|_{w_0}$ is *not* respected by action rewrite rules. Due to the possible nested use of operation symbols in the right-hand side of action rewrite rules, there is an exponentiation, which spoils a decreasing of weight. To overcome this problem, we will prove that based on the map w_0 a new "exponentiation resistant" map e_w can be constructed, so that all rewrite rules respect $|-|_{e_w}$. The idea is that we compute for all operations in the left-hand side of action rules a new value, starting with the minimal weight of the right-hand side. The maximum of the right-hand sides is then assigned. Because the values of operations depend on each other, we calculate the values recursively.

The map nf calculates the maximum values of the right-hand sides with a minimal filling (i.e. **0**'s) for a given weight map.

DEFINITION 4.11. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization and $w : \Sigma_{G'} \to \mathbf{N}$ a map. Let $nf_w : \Sigma_{G'} \to \mathbf{N}$ be a map defined as follows. Let $f \in \Sigma_{G'}$, then

$$nf_w(f) \quad = \quad max(\{w(f)\} \cup \{|rhs[\vec{x} := \vec{\mathbf{0}}]|_w | rhs \in Rhs_f\})$$

where Rhs_f is defined as

$$\{rhs \quad | \quad r : lhs \to rhs \in R_{G'} \text{ is an action rule } \& \text{ the principal operation of } lhs \text{ is } g \ \& \ \hat{g} = f\}.$$

DEFINITION 4.12. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization and $w : \Sigma_{G'} \to \mathbf{N}$ a map. The maps $e_{w,0}, ..., e_{w,n} : \Sigma_{G'} \to \mathbf{N}$ are defined inductively as follows. Let $f \in \Sigma_{G'}$, then $e_{w,0}$ is defined as

$$e_{w,0}(f) = (cf_{G'} + 2).df_{G'}.3.w(f)$$

and $e_{w,i+1}$ as

$$\begin{array}{lcll} e_{w,i+1}(f) & = & e_{w,i}(f) & \text{if } w(f) \leq i \\ e_{w,i+1}(f) & = & 1 + max(\{nf_{e_{w,i}}(g) | w(g) = w(f)\}) & otherwise. \end{array}$$

Then $e_w : \Sigma_{G'} \to \mathbf{N}$ is defined as $e_{w,n}$, where n is the maximum of the set $\{w(f) | f \in \Sigma_{G'}\}$.

REMARK 4.13. To see that e_w is well-defined it is sufficient to recall that $\Sigma_{G'}$ is finite.

PROPOSITION 4.14. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a rulified axiomatization and $w : \Sigma_{G'} \to \mathbf{N}$ a map such that $w(f) \geq 1$ for all $f \in \Sigma_{G'}$. Then

1. $e_w(\hat{f})^2 > (1 + cf_{G'}).(2 + cf_{G'}).(1 + df_{G'})$, and
2. if $f(\vec{x}) \to rhs \in R_{G'}$ is an action rule and rhs contains no operation g such that $w(g) = w(f)$, then $e_w(f) \geq 1 + |rhs[\vec{x} := \vec{\mathbf{0}}]|_{e_w}$.

The proof that action rules respect $| - |_{e_w}$ is very detailed. For clarity we have extracted the following technical fact needed in the proof.

LEMMA 4.15. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a linear and copy safe rulified axiomatization with map $w : \Sigma_{G'} \to \mathbf{N}$ such that $w(f) \geq 1$ for $f \in \Sigma_{G'}$. Let $C[]$ be an n-ary context with $n \geq 1$. Let $p_1, ..., p_n \in T(\Sigma_{G'})$, then $|C[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'} \cdot \sum_{i=1}^{n} |p_i|_{e_w}} > |a \cdot C[p_1, ..., p_n]|_{e_w}$.

Proof. We prove the result by an induction on the complexity of $C[]$. Let $C[]$ be an n-ary context with $n \geq 1$ and $p_1, ..., p_n \in T(\Sigma_{G'})$. Let the Induction Hypothesis (I.H.) be that if $C_i[]$ is an $k \leq n$-ary context of smaller complexity then $C[]$, then $|C_i[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'} \cdot \sum_{i=1}^{n} |p_i|_{e_w}} > |a \cdot C_i[p_1, ..., p_n]|_{e_w}$. Distinguish the following cases in the Induction.

1. *Base Step.* Let $C[]$ be the trivial context. Then

$$\begin{array}{lll} (1+|\mathbf{0}|_{e_w})^{cf_{G'}.|p_1|_{e_w}} & > & \{\text{Lemma 3.10}\} \\ (cf_{G'}+2).|p_1|_{e_w} & = & \\ |a\cdot p_1|_{e_w}. & & \end{array}$$

2. *Induction Step.* Let $C[p_1,...p_n] \equiv f(C_1[p_1,...p_n],...,C_{ar(f)}[p_1,...p_n])$ and let the Induction Hypothesis (I.H.) hold for $C_1[],...,C_{ar(f)}[]$. Now distinguish three cases.
   (a) Suppose $f \equiv a\cdot$. Then

$$\begin{array}{lll} |a\cdot C_1[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'}.|p_1|_{e_w}} & > & \\ (cf_{G'}+2).|C_1[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'}.|p_1|_{e_w}} & > & \{\text{I.H.}\} \\ (cf_{G'}+2).|a\cdot C_1[\vec{\mathbf{0}}]|_{e_w} & = & \\ |a\cdot a\cdot C_1[\vec{\mathbf{0}}]|_{e_w}. & & \end{array}$$

   (b) Suppose $f \equiv +$. Then

$$\begin{array}{lll} |C_1[\vec{\mathbf{0}}]|_{e_w}+|C_2[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'}\cdot\sum_{i=1}^n|p_i|_{e_w}} & > & \{\text{Binomium}\} \\ |C_1[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'}\sum_{i=1}^n|p_i|_{e_w}}+|C_2[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'}\cdot\sum_{i=1}^n|p_i|_{e_w}} & > & \{\text{I.H}\} \\ |a\cdot C_1[p_1,...,p_n]|_{e_w}+|a\cdot C_2[p_1,...,p_n]|_{e_w} & > & \\ |a\cdot(C_1[p_1,...,p_n]+C_2[p_1,...,p_n])|_{e_w}. & & \end{array}$$

   (c) Suppose $f \notin \{a\cdot,+\}$. Then

$$\begin{array}{lll} |f(C_1[\vec{\mathbf{0}}],...,C_{ar(f)}[\vec{\mathbf{0}}])|_{e_w}{}^{cf_{G'}\cdot\sum_{i=1}^n|p_i|_{e_w}} & > & \\ (e_w(\hat{f})^{\sum_{i=1}^n|C_i[\vec{\mathbf{0}}]|_{e_w}})^{cf_{G'}\cdot\sum_{i=1}^n|p_i|_{e_w}} & > & \\ e_w(\hat{f})^{\sum_{i=1}^n|C_i[\vec{\mathbf{0}}]|_{e_w}{}^{cf_{G'}\cdot\sum_{i=1}^n|p_i|_{e_w}}} & > & \{\text{I.H.}\} \\ e_w(\hat{f})^{\sum_{i=1}^n|a\cdot C_i[\vec{\mathbf{0}}]|_{e_w}} & > & \\ e_w(\hat{f})^{(cf_{G'}+2)\cdot\sum_{i=1}^n|C_i[p_1,...,p_n]|_{e_w}} & > & \\ (cf_{G'}+2).(e_w(\hat{f})^{cf_{G'}\cdot\sum_{i=1}^n|C_i[p_1,...,p_n]|_{e_w}} & \geq & \{\text{Propositions 4.7,4.14}\} \\ |a\cdot f(C_1[p_1,...,p_n],...,C_{ar(f)}[p_1,...,p_n])|_{e_w} & & \end{array}$$

□

LEMMA 4.16. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a linear, copy safe and syntactically well-founded rulified axiomatization with map $w : \Sigma_{G'} \to \mathbf{N}$ where $w(f) \geq 1$ for $f \in \Sigma_{G'}$ and $p, q \in T(\Sigma)$. If p rewrites to q with an action rule of $R_{G'}$, then $|p|_{e_w} > |q|_{e_w}$.

Proof. Let $p \equiv C[p']$ where p' is an action redex. The result follows with an induction on the complexity of $C[]$. We present only the Base

Step, since the rest of the induction is straight forward. Let $lhs \to rhs \in R_{G'}$ be an action rewrite rule so that $\sigma(lhs) = p$ for some closed substitution $\sigma : \text{Var} \to T(\Sigma)$. By definition of action axioms and hence of action rewrite rules, lhs is of the form

$$f(P_1, ..., P_{ar(f)}),$$

for some $f \in \Sigma_{G'}$, where P_i is of the form $a_i \cdot x_i$, x_i or $\mathbf{0}$.

Let $I_{a \cdot x}$ be the set of the indexes of the arguments of lhs of the form $a_i \cdot x_i$ and similarly I_x the set of the indexes of the arguments of lhs of the form x_i. I_{rhs} is the subset of indexes of the variables of lhs appearing in rhs. Distinguish $(\frac{T}{F})$ whether or not rhs contains an operation g such that $w(g) = w(f)$. (T) Suppose rhs contains an operation g such that $w(g) = w(f)$. Because $\Re_{G'}$ is syntactically well-founded, rhs is of the form $a \cdot g(\vec{x'})$, where $\bigcup \vec{x'} \subseteq var(lhs)$ and there is *at least one* P_i of the form $a_i \cdot x_i$. Let $p_1, ..., p_{ar(f)} \in T(\Sigma_{G'})$, then

$$\begin{array}{ll}
|\sigma(f(P_1, ..., P_{ar(f)}))|_{e_w} & \geq \\
e_w(\hat{f})^{\sum_{i \in I_{a \cdot x}} (cf_{G'}+2) \cdot |p_i|_{e_w} + \sum_{i \in I_x} cf_{G'} \cdot |p_i|_{e_w}} & > \\
e_w(\hat{f})^2 . e_w(\hat{f})^{\sum_{i \in (I_{a \cdot x} \cup I_{a \cdot x}) \cap I_{rhs}} cf_{G'} \cdot |p_i|_{e_w}} & = \\
e_w(\hat{f})^2 . e_w(\hat{g})^{\sum_{i \in (I_{a \cdot x} \cup I_x) \cap I_{rhs}} cf_{G'} \cdot |p_i|_{e_w}} & \geq \{\text{Propositions 4.7,4.14}\} \\
|\sigma(a \cdot g(\vec{x'}))|_{e_w}. &
\end{array}$$

(F) Suppose rhs contains *no* operation g such that $w(g) = w(f)$. Because $\Re_{G'}$ is syntactically well-founded, rhs contains *only* operations g such that $w(g) < w(f)$. Let $p_1, ..., p_{ar(f)} \in T(\Sigma_{G'})$, then

$$\begin{array}{ll}
|\sigma(f(P_1, ..., P_{ar(f)}))|_{e_w} & \geq \\
e_w(\hat{f})^{\sum_{i \in I_{a \cdot x}} |a_i \cdot p_i|_{e_w} + \sum_{i \in I_x} cf_{G'} \cdot |p_i|_{e_w}} & \geq \\
e_w(\hat{f})^{cf_{G'} \cdot \sum_{i \in (I_{a \cdot x} \cup I_x) \cap I_{rhs}} |p_i|_{e_w}} & > \{\text{Proposition 4.14, Lemma 4.15}\} \\
|\sigma(rhs[x_1, ..., x_{ar(f)}])|_{e_w} \square &
\end{array}$$

Now we have a weight map which is respected by all rewrite rules.

LEMMA 4.17. Let $\Re_{G'} = (\Sigma_{G'}, R_{G'})$ be a linear, copy safe and syntactically well-founded rulified axiomatization with map $w : \Sigma_{G'} \to \mathbf{N}$ where $w(f) \geq 1$ for $f \in \Sigma_{G'}$ and $p, q \in T(\Sigma)$. If p rewrites to q with a rewrite rule of $R_{G'}$, then $|p|_{e_w} > |q|_{e_w}$.

Proof. By definition of e_w, $e_w(f) \geq 3$ for all f. Now use Lemmas 3.11 and 4.16 $\square$

This results extends to rewriting modulo associativity and commutativity of the + using the argument of Lemma 3.12.

COROLLARY 4.18. Let $\Re_G = (\Sigma_{G'}, R_{G'})$ be a linear, copy safe and syntactically well-founded rulified axiomatization with map w where $w(f) \geq 1$ for all $f \in \Sigma_{G'}$. Then $\Re_{G'}$ is strongly normalizing modulo ac on closed terms.

Although we have no proof for this at the moment, we think that the demand $w(f) \geq 1$ can be dropped. Unfortunately our method of proving a decreasing of weight for action rewrite rules then fails: using 0 as a base in the exponentiation spoils the argument in Lemma 4.16.

EXAMPLE 4.19. The condition $w(f) \geq 1$ for all $f \in \Sigma_{G'}$ of the rulified axiomatization $\Re_{G'}$, *excludes* the GSOS system G which is a disjoint extension of G_{FINTREE} with the rule

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} f(f(y))}$$

where $a \in Act$ is an action. Then the rulified axiomatization is linear, copy safe and syntactically well-founded and consists of the FINTREE rules plus

$$\begin{aligned} f(a \cdot x) &= a \cdot f(f(x)) \\ f(x_1 + x_2) &= f(x_1) + f(x_2) \\ f(\mathbf{0}) &= \mathbf{0}. \end{aligned}$$

We leave the proof that $\Re_{G'}$ is strongly normalizing to the reader.

CONJECTURE 4.20. The rulified axiomatization of a linear, copy safe and syntactically well-founded GSOS system is strongly normalizing modulo ac for closed terms.

Acknowledgements I thank Frits Vaandrager for the idea, critical reading and the stimulating discussions and Wan Fokkink for careful proof-reading on [Bos94], which forms the basis of this chapter.

Bibliography

[AB90] G.J. Akkerman and J.C.M. Baeten. Term rewriting analysis in process algebra. Report P9006, Programming Research Group, University of Amsterdam, 1990.

[ABV92] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. In *Proceedings 7^{th} Annual Symposium on Logic in Computer Science,* Santa Cruz, California, pages 113–124. IEEE Computer Society Press, 1992.

[ABV94] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, May 1994.

[AFIa] L. Aceto, W.J. Fokkink, and A. Ingólfsdóttir. A menagerie of non-finitely based process semantics over BPA*: from ready simulation to completed traces. *Mathematical Structures in Computer Science.* To appear.

[AFIb] L. Aceto, W.J. Fokkink, and A. Ingólfsdóttir. On a question of A. Salomaa: the equational theory of regular expressions over a singleton alphabet is not finitely based . *Theoretical Computer Science.* To appear.

[BBK86] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

[BBK87a] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Conditional axioms and α/β calculus in process algebra. In M. Wirsing, editor, *Formal Description of Programming Concepts – III, Proceedings of the 3^{th} IFIP WG 2.2 working conference,* Ebberup 1986, pages 53–75, Amsterdam, 1987. North-Holland.

[BBK87b] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE conference,* Eindhoven, *Vol. II (Parallel Languages)*, volume 259 of *Lecture Notes in Computer Science*, pages 94–113. Springer-Verlag, 1987.

[BBK87c] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.

[BBK93] 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.

[BBP94] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[BCS95] O. Burkart, D. Caucal, and B. Steffen. An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes. In Jîrí Wiedermann and Petr Hájek, editors, *MFCS '95*, volume 969 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 1995.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation Collapse and the Process Taxonomy. In Martin Wirsing and Maurice Nivat, editors, *Proceedings of 5th AMAST Conference*, volume 1101 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[BG96] D.J.B. Bosscher and W.O.D. Griffioen. Regularity for a large Class of Context-free Processes is decidable. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP '96*, volume 1099 of *Lecture Notes in Computer Science*, pages 182–193. Springer-Verlag, 1996.

[BH64] Y. Bar-Hillel. *Language and Information*. Series in Logic. Addison-Wesley, 1964.

[BHPS61] Y. Bar-Hillel, M. Perles, and E. Shamir. On Formal Properties of Simple Phrase Structure Grammars. *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–177, 1961.

[BIM95] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995.

[BK84a] 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 ICALP*, Antwerpen, volume 172 of *Lecture Notes in Computer Science*, pages 82–95. Springer-Verlag, 1984.

[BK84b] J.A. Bergstra and J.W. Klop. Fair FIFO queues satisfy an algebraic criterion for protocol correctness. Report CS-R8405, CWI, Amsterdam, 1984.

[BK84c] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[BK92] J.A. Bergstra and J.W. Klop. A convergence theorem in process algebra. In J.W. de Bakker and J.J.M.M. Rutten, editors, *Ten Years of Concurrency Semantics*. Selected Papers of the Amsterdam Concurrency Group, pages 164–195. World Scientific, 1992.

[BK94] J.A. Bergstra and P. Klint. The Toolbus - a Component Interaction Architecture. Research Report P9408, University of Amsterdam, March 1994.

[Bos94] D.J.B. Bosscher. Term Rewriting Properties of GSOS Axiomatizations. In Masami Hagiya and John C. Mitchell, editors, *Proceedings of TACS '94*, volume 789 of *Lecture Notes in Computer Science*, pages 425–439. Springer-Verlag, 1994.

[Bos95] Joris Boselie. Expressiveness Results for Process Algebra with Iteration. Master's thesis, University of Amsterdam, July 1995.

[BV95] J.C.M. Baeten and C. Verhoef. Concrete Process Algebra. In S. Abramsky, Dov M. Gabbay, and T.S. Maibaum, editors, *Handbook of Logic in Computer Science, Semantic Modeling*, volume 4, pages 149–268, 1995.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[Cau90] D. Caucal. Graphes canoniques de graphes algébriques. *Theoretical Informatics and Applications*, 24(4):339–352, 1990.

[CEW58] I.M. Copi, C.C. Elgot, and J.B. Wright. Realization of events by logivcal nets. Journal of the ACM, 5:181–196, 1958.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation is Decidable for all Context-free Processes. In W.R. Cleaveland, editor, *Proceedings of CONCUR 92*, volume 630 of *Lecture Notes in Computer Science*, pages 138–147. Springer-Verlag, 1992.

[dV97] P.R. d'Argenio and C. Verhoef. A General Conservative Extension Theorem in Process Algebra with Inequalities. *Theoretical Computer Science*, 2(177):351–380, 1997.

[Eng85] J. Engelfriet. Determinacy → (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36(1):21–25, 1985.

[Fok96] W.J. Fokkink. On the Completeness of the Equations for Kleene Star in Bisimulation. In Martin Wirsing and Maurice Nivat, editors, *Proceedings of 5th AMAST Conference*, volume 1101 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 1996.

[Fok97] W.J. Fokkink. Axiomatizations for the perpetual loop in process algebra. In *Proceedings of ICALP '97*, Lecture Notes in Computer Science. Springer-Verlag, 1997. To appear.

[FV95] W. Fokkink and C. Verhoef. A conservative look at term deduction systems with variable binding. Technical Report 105, Utrecht Research Institute for Philosophy, September 1995. Submitted for Publication.

[FZ94] W.J. Fokkink and H. Zantema. Basic process algebra with iteration: Completeness of its equational axioms. *The Computer Journal*, 37(4):259–267, 1994.

[GH94] J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Control*, 115(2):354–371, 1994.

[Gla90] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, Free University, Amsterdam, 1990.

[Gla93] R.J. van Glabbeek. The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.

[Gro92] J.F. Groote. A short proof of the decidability of bisimulation for normed BPA-processes. *Information Processing Letters*, 42:167–171, 1992.

[GV92] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992.

[GW89] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.

[HJM94] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. Technical Report R94:08, SICS, 1994.

[HM96] Y. Hirshfeld and F. Moller. Decidability Results in Automata and Process Theory. In Faron Moller and Graham Birtwistle, editors, *Logics for*

Concurrency: Automata vs Structure, volume 1043 of *Lecture Notes in Computer Science*, pages 102–148, 1996. Previously presented as lecture notes at the VIII-th Banff Higher Order Workshop *"Theories of Concurrency: Structure vs Automata "* in 1994 .

[Hol89] Uno Holmer. Translating Static CCS Agents into Regular Form. PMG report 51, Department of Computer Science, Chalmers University of Technology and the University of Göteborg, 1989.

[HS91] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, pages 376–386. IEEE Computer Society Press, 1991.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Hüt91] Hans Hüttel. Silence is Golden: Branching Similarity is Decidable for Context-Free Processes. In K. G. Larsen and A. Skou, editors, *Proceedings of CAV '91*, volume 575 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.

[Kle56] S.C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[Kru95] A. Kručera. Deciding Regularity in Process Algebras. Technical Report RS-95-52, BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation), 1995.

[Lin93] Huimin Lin. PAM: A Process Algebra Manipulator (Version 1.0). Report 4/93, Computer Science, University of Sussex, Brighton, February 1993.

[LP79] Harry R. Lewis and Christos R. Papidimitriou. *Elements of the Theory of Computation.* Prentice-Hall International, 1979.

[MdSV89] E. Madelaine, R. de Simone, and D. Vergamini. *ECRINS V2-1, USERS MANUAL*, 1989.

[Mil84] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[Mil86] R. Milner. Process constructors and interpretations. In H.-J. Kugler, editor, *IFIP Information processing 86*, pages 507–514. North-Holland, 1986.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[MM94] S. Mauw and H. Mulder. Regularity of BPA-Systems is Decidable. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR'94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 34–47. Springer-Verlag, 1994.

[Mol89] F. Moller. *Axioms for concurrency.* PhD thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh, 1989.

[MY60] R. McNaughton and H. Yamada. Regular Expressions and State Graphs for Automata. *IEEE Transactions on Electronic Computers*, EC-9(1):39–47, 1960.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pon94] A. Ponse. Process algebra and dynamic logic. In J. van Eijck and A. Visser, editors, *Logic and Information Flow*, pages 125 – 148. MIT Press, 1994.

[Pos46] E.L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity*. Prentice-Hall International, Englewood Cliffs, 1982.

[RS59] M.O Rabin and D. Scott. Finite Automata and Their Decisions Problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.

[Sch63] M.P. Schutzenberger. On Context-free Languages and Pushdown Automata. *Information and Control*, 6(3):246–264, 1963.

[Sim85] R. de Simone. Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Vaa93] F.W. Vaandrager. Expressiveness results for process algebras. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Semantics: Foundations and Applications,* Beekbergen, The Netherlands, June 1992, volume 666 of *Lecture Notes in Computer Science*, pages 609–638. Springer-Verlag, 1993.

[vDHK96] Arie van Deursen, Jan Heering, and Paul Klint, editors. *Language Prototyping. An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing.* World Scientific, 1996.

[Ver95] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2:274–302, 1995.

[Vra97] J.L.M. Vrancken. The algebra of communicating processes with empty process. *Theoretical Computer Science*, 177(2):287–328, 1997.

[Wat95] Bruce W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms.* PhD thesis, Eindhoven University of Technology, 1995.