Choice Quantification in Process Algebra

IPA Dissertation Series

2002-04

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed by the Netherlands Organisation for Scientific Research (NWO; project 612-33-008; 1996–2000), and by the Centre for Mathematics and Computer Science (CWI; 2000–2001).

Choice Quantification in Process Algebra

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. mr. P.F. van der Heijden
ten overstaan van
een door het college voor promoties ingestelde commissie,
in het openbaar te verdedigen
in de Aula der Universiteit
op woensdag 3 april 2002, te 14.00 uur

door

Sebastiaan Pascal Luttik

geboren te Zutphen

Promotoren: prof. dr. J.A. Bergstra
prof. dr. ir. J.F. Groote

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

ISBN 90-90156-24-0
NUGI 855
IPA Dissertation Series 2002-04

Typeset with LaTeX 2ε
Printed by Thela Thesis, Amsterdam
Cover design by Simona Orzan

Author's address:
CWI
P.O. Box 94079
1098 SJ Amsterdam
The Netherlands
Bas.Luttik@cwi.nl

Contents

Preface

When I started work as an *onderzoeker in opleiding*, affiliated with the CWI and the University of Amsterdam, my first project was to specify a part of the IEEE 1394 (see Luttik, 1997). It was my first encounter with the process specification language μCRL, and with the *choice quantifier* (also referred to as 'sum operator'). Syntactically, μCRL is an extension of the algebraic process theory ACP, which I had come across before, and, indeed, the choice quantifier reminded me of the notation sometimes used in ACP specifications to abbreviate large alternative compositions. However, in ACP, this notation is informal, and its use is explicitly restricted to cases in which the abbreviated alternative composition is finite. In contrast, the choice quantifier belongs to the official syntax of μCRL, and it may refer to an infinite alternative composition.

Since my first contact with μCRL, I have been interested in choice quantification, and especially in its mathematical theory. Jan Friso Groote and I tried to axiomatise choice quantification in the context of a finite fragment of μCRL in several semantic settings. First, we investigated the completeness of a set of equational axioms with strong bisimulation as a semantics (Groote and Luttik, 1998a). Our main conclusion was that a complete set of axioms could not be found in general, because for certain data, the associated notion of bisimulation was too complex to have an axiomatisation. We also formulated general restrictions on the data, under which our set of axioms was complete. Then, we extended our results to a setting with branching bisimulation as a semantics (Groote and Luttik, 1998b), and I extended this result further, to settings with weak-, delay- and η-bisimulation as semantics (Luttik, 1999a).

In my view, a drawback of the axiomatisations we had found was that they treat choice quantification as a binder, a construction that relies on the syntactic structure of its argument. As such, our axiomatisations could not be viewed at the same time as an abstract algebraic definition of the mathematical notion that choice quantification refers to. In other words, choice quantification had in our theory not the same semantic status as the operations of a purely algebraic theory such as ACP. Building on the techniques of algebraic logic, I therefore proposed an alternative treatment of choice quantification, which abstracts from the syntactic aspect of choice quantification (Luttik, 1999b).

Apart from conducting the above mentioned investigations, I participated in other research. Eelco Visser and I coauthored a paper on the specification of rewriting strategies (Luttik and Visser, 1997). Together with Piet Rodenburg and Rakesh Verma, I wrote a paper on correctness criteria for transformations of rewrite systems (Luttik *et al.*, 1998). Wan Fokkink and I proved that a finite ω-complete specification of interleaving is obtained by adding to the algebraic theory

PA of Bergstra and Klop (1985) a well-known axiom for *standard concurrency* and the equations generated by a new axiom schema (Fokkink and Luttik, 2000). And I wrote a short note about unique decomposition of processes with respect to parallel composition (Luttik, 2000).

When time had come to present a dissertation, I could have chosen to just put all my papers together. Clearly, given the diversity of subjects, this would have resulted in a very fragmented account. Instead, I preferred to try and write a coherent report of my study of choice quantification, the main theme of my research thus far. One of the things I had learned, was that the mathematical definitions underlying μCRL are considerably more complex than those underlying, e.g., the algebraic theory ACP, which is firmly founded on the standard theory of universal algebra. In fact, I found that the mathematical basis for μCRL had not been defined in sufficiently precise detail.

This dissertation, then, is concerned with the mathematical theory of choice quantification, with a bias towards an algebraic approach. It is organised as follows. Chapter 1 explains the advantages of using choice quantification in a process specification, and briefly touches on the subjects of the later chapters. Chapter 2 explores the semantic connection between μCRL and ACP, providing an abstract algebraic definition of infinite sums in basic process algebras with deadlock. Chapter 3 defines the fragment of μCRL which is the main focus of the rest of the book. It establishes a connection with the structures discussed in Chapter 2; in particular, it explains how choice quantification relates to alternative composition.

Chapter 4 demonstrates a correspondence between choice quantification in μCRL and quantification in first-order logic. It considerably improves on the first part of (Groote and Luttik, 1998a), showing that, with respect to the data inside μCRL expressions, choice quantification can simulate both universal and existential quantification of first-order logic. We put this in perspective by showing that the input prefix mechanism of value-passing CCS can only simulate universal quantification.

The results of Chapter 4 motivate the restrictions imposed on the data domain in later chapters. Chapter 5 discusses a sound and complete deductive system, and is based on the second part of (Groote and Luttik, 1998a). Chapter 6 is based on (Luttik, 1999a), and presents an alternative to the deductive system discussed in Chapter 5; this alternative is more attractive from an algebraic point of view. It is shown that the systems of Chapters 5 and 6 are equivalent in expressive and deductive power. Chapter 7 presents the conclusions.

Acknowledgments

My supervisors, Jan Bergstra and Jan Friso Groote, always expressed their confidence in me, for which I am very grateful. Jan had time for me whenever I wanted to discuss my work or my personal situation. He always gave inspiring and valuable advice. Jan Friso made his 'theme' at the CWI into a stimulating research environment, in which I had the freedom to do the research that I wanted to do. His enthusiasm was of great support to me.

Piet Rodenburg played an invaluable part in my development as a researcher. I could always drop by his office to ask him a question, to test an idea for a proof, or

just to chat about one thing or another. He taught me a lot about logic, algebra, and science in general. His proof reading ability is superhuman.

My cooperation with Wan Fokkink was most pleasant, and when he succeeded Jan Friso as theme leader, he turned out to be an excellent boss as well, providing me, both literally and figuratively, with the room in which I could write this book. The circumstances in my last six months at the CWI, in my corner on the third floor, were perfect. I could close my door and concentrate on the writing. But whenever I felt like it, I could swing it open again, to find in Jaco van de Pol an enthusiastic neighbour, ready to exchange ideas. He read and commented on large parts of this dissertation.

I thank the members of the reading committee, Maarten Boasson, Jan van Eijck, Wan Fokkink, Paul Klint, Kees Middelburg, Piet Rodenburg and Davide Sangiorgi for reviewing the manuscript and for their comments. A last minute discussion with Jan van Eijck enabled me to improve the presentation.

I wish to express my gratitude to all the participants of PAM, and especially to top speakers Sjouke Mauw and Vincent van Oostrom, who were always willing and able to fill gaps in my program. Vincent was usually there when I spent the weekend at the CWI, and I have benefited from our long conversations. I wish to extend my gratitude to all the former colleagues of the CWI, and in particular to Doeko Bosscher, David Griffioen, Alban Ponse, Michel Reniers, Judi Romijn, Yaroslav Usenko and Mark van der Zwaag.

Jan Willem Klop and Roel de Vrijer were so kind as to enable me to complete this book at the VU. My new colleagues there, and especially my new office mate Mirna Bognar, made me quickly feel at home.

The cooperation with Eelco Visser, at the end of my first year, has meant a lot to me. He has become a good friend, and has been very encouraging ever since.

The most important source of relaxation in my life is jazz music; I played the piano in quite a few (big) bands, and I wish to thank my fellow musicians. I am most grateful to all my friends, and in particular to Anna, Eric, Floortje, Ingmar, Luuk, Ramin, Rob and Sanne. Simona's presence and support made a great difference to me and, indeed, to this book; mulţumesc frumos!

My final words of thanks are for my family: Mama & Jan, Papa & Karin, and my three sisters Léonie, Tirza and Janine. Jan carefully read the introduction and suggested many improvements with regard to English usage. I am happy to have Tir and Nien for my 'paranimfen'.

Bas Luttik

Amsterdam, January 2002

1

Introduction

We shall conduct a systematic investigation of *choice quantification* in the context of *process algebra*. Choice quantification is used to describe the act of selecting an instantiation of a process with an arbitrary element from a data domain. This first chapter is meant to introduce the context in which the above mentioned subjects play a rôle. We first explain the basics of formal process specification, and why it is sometimes convenient to give a separate specification of some relevant data. Then, we shall describe a general method for assigning a mathematical meaning to formal specifications of processes, and we shall discuss the consequences for this method if some of the data is to be specified separately. Finally, we shall briefly mention the results about choice quantification that will be obtained in the remainder of this thesis.

1.1 Process specification

To start with, here is an informal description of a very simple process. At almost every street corner in downtown Amsterdam there is a car park ticket dispenser. This is a quite simple device that translates coins into parking time. When this thesis was written (on the eve of the introduction of the 'euro'), the machine accepted the following Dutch coins: 'kwartjes' (Dfl. 0.25), 'guldens' (Dfl. 1.00), 'rijksdaalders' (Dfl. 2.50) and 'vijfjes' (Dfl. 5.00).

A car owner wishing to avoid a wheel clamp will look for the nearest ticket dispenser —advertised by the capital P— immediately after parking his car. It presents him with the following options:

1. He can insert a coin.
2. He can press a green button to instruct the machine to produce a ticket. The dispenser requires the insertion of at least Dfl. 0.50 to print a ticket; otherwise, pressing the green button has no effect.
3. He can turn a red knob causing the machine to return all the coins that were inserted since the last time that either the button was pushed or the knob was turned.

In the city centre, the parking fee is approximately Dfl. 5.00 an hour. (To be entirely honest, the fee is Dfl. 5.75 per hour between 9am and 7pm, and Dfl. 3.25 between 7pm and 11pm, and outside these hours parking is for free. Further,

there is obviously a (physical) limit on the amount of money that the car owner can deposit in the dispenser. We ignore such details so as to guarantee that our example retains its promised simplicity. Also, we are not sure about the internal precision of our dispenser when it associates minutes with coins; somewhere in the process it presumably rounds off to the nearest minute. It is more convenient to work with a fee of Dfl. 5.00 per hour, so that the number of minutes associated with each coin is an integer; e.g., a 'kwartje' buys 3 minutes of parking time.)

1.1.1 Implicit data

Above, we have informally described the events that may occur in the process of obtaining a ticket from the ticket dispenser. We assign to each of these events a formal symbol from the following list

$$\mathrm{in_k}, \mathrm{in_g}, \mathrm{in_r}, \mathrm{in_v}, \mathrm{button}, \mathrm{print}, \mathrm{knob}, \mathrm{return}. \tag{1.1}$$

The symbols "$\mathrm{in_k}$", "$\mathrm{in_g}$", "$\mathrm{in_r}$" and "$\mathrm{in_v}$" respectively refer to the events of the car owner inserting a 'kwartje', a 'gulden', a 'rijksdaalder' or a 'vijfje' into the ticket dispenser. The symbol "button" refers to the event of him pressing the green button, and the symbol "knob" refers to the event of him turning the red knob. The symbol "print" refers to the event of the dispenser printing a ticket, and the symbol "return" refers to the event of it returning all the recently inserted coins.

Furthermore, let us attach the number of minutes for which the car owner has paid as a subscript to the name of the ticket dispenser. This gives us another list of formal symbols:

$$\mathrm{P}_0, \mathrm{P}_3, \ldots, \mathrm{P}_{3n}, \ldots \qquad (n \geq 0). \tag{1.2}$$

The number $3n$ ($n \geq 0$) may be thought of as the *state* the ticket dispenser got into when the car owner inserted coins to the equivalent of $3n$ minutes. The symbol P_{3n} refers to the behaviour of the ticket dispenser when it is in state $3n$.

Inserting a coin, pressing the button and turning the knob will generally have the effect of changing the state of the ticket dispenser; e.g., inserting a coin will increase the subscript by the number of minutes associated with that coin, and turning the knob while the dispenser is in some state $3n$ ($n \geq 1$) will make the dispenser return all the inserted coins and go back to state 0. We introduce the symbol "$\cdot$" to express that things happen consecutively. For instance, we write "$\mathrm{in_k} \cdot \mathrm{P}_3$" if we want to say that the insertion of a 'kwartje' makes the dispenser go into state 3, and we write "$\mathrm{knob} \cdot \mathrm{return} \cdot \mathrm{P}_0$" if we want to say that after turning the knob the dispenser returns the inserted coins and goes into state 0.

According to our informal descriptions, the car owner may activate a number of alternative events. To specify this, we introduce the symbol "+"; e.g., to express that the car owner may choose to insert a 'kwartje' to make the dispenser go into state 3, or to insert a 'gulden' to make the dispenser go into state 12, we write "$\mathrm{in_k} \cdot \mathrm{P}_3 + \mathrm{in_g} \cdot \mathrm{P}_{12}$". Incidentally, note that the car owner may choose to insert a coin, press the button or turn the knob irrespective of the actual state of the ticket dispenser.

We can now define the behaviour of our ticket dispenser by simultaneously specifying the behaviours that it may exhibit in each of its states:

$$\begin{aligned}
\mathrm{P}_0 &= \mathrm{in}_{\mathsf{k}} \cdot \mathrm{P}_3 + \mathrm{in}_{\mathsf{g}} \cdot \mathrm{P}_{12} + \mathrm{in}_{\mathsf{r}} \cdot \mathrm{P}_{30} + \mathrm{in}_{\mathsf{v}} \cdot \mathrm{P}_{60} + \mathrm{button} \cdot \mathrm{P}_0 + \mathrm{knob} \cdot \mathrm{P}_0;\\
\mathrm{P}_3 &= \mathrm{in}_{\mathsf{k}} \cdot \mathrm{P}_6 + \mathrm{in}_{\mathsf{g}} \cdot \mathrm{P}_{15} + \mathrm{in}_{\mathsf{r}} \cdot \mathrm{P}_{33} + \mathrm{in}_{\mathsf{v}} \cdot \mathrm{P}_{63}\\
&\qquad + \mathrm{button} \cdot \mathrm{P}_3 + \mathrm{knob} \cdot \mathrm{return} \cdot \mathrm{P}_0;
\end{aligned}$$

and for all $n \geq 2$:

$$\begin{aligned}
\mathrm{P}_{3n} &= \mathrm{in}_{\mathsf{k}} \cdot \mathrm{P}_{3n+3} + \mathrm{in}_{\mathsf{g}} \cdot \mathrm{P}_{3n+12} + \mathrm{in}_{\mathsf{r}} \cdot \mathrm{P}_{3n+30} + \mathrm{in}_{\mathsf{v}} \cdot \mathrm{P}_{3n+60}\\
&\qquad + \mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}_0 + \mathrm{knob} \cdot \mathrm{return} \cdot \mathrm{P}_0.
\end{aligned}$$

The equations above may serve as a formal specification of the behaviour of any car park ticket dispenser in downtown Amsterdam. Now, suppose that our car owner did not find a place to park his car in the city centre, and that he was forced to put it somewhere just outside the city centre. He is still in a part of Amsterdam where he has to pay, but parking time is twice as cheap: the fee is Dfl. 2.50 an hour. The car park ticket dispensers in this part of Amsterdam look very similar to those in the city centre; they carry the same initial (P) and appear to behave in the same fashion too. The difference only becomes apparent when one compares the amount of money inserted and the number of minutes allotted in the two regions.

What should we do to adapt the specification given above in such a way that it describes the behaviour of a ticket dispenser in this part of Amsterdam? We should double every number that appears as a subscript, thus obtaining a specification that assigns a behaviour to the symbols

$$\mathrm{P}_0, \mathrm{P}_6, \ldots, \mathrm{P}_{6n}, \ldots \qquad (n \geq 0).$$

The new specification accurately describes the behaviour of a ticket dispenser just outside the city centre. It is somewhat unfortunate, however, that the intuitively clear relationship between the new specification and the previous one is obscured by a computation. If we were to order ticket dispensers for all of Amsterdam, it would be more convenient if we could give the manufacturer just one specification of their behaviour, plus the going rates for the different regions.

1.1.2 Explicit data

We started out to say that a car park ticket dispenser is a machine that translates coins into minutes; coins and minutes are the types of *data* on which our ticket dispenser operates. That our specification involves data at all has thus far been implicit in our suggestive nomenclature for states and events. Let us now proceed and give explicit definitions of some of the data in our specifications. The accepted coins are the elements of a set C; in our example

$$\mathsf{C} = \{\mathsf{k}, \mathsf{g}, \mathsf{r}, \mathsf{v}\}.$$

When a car owner inserts a coin $c \in \mathsf{C}$ into the slot of a ticket dispenser, this has the effect of increasing the parking time bought by the number of minutes $\mathrm{T}(c)$ associated with c; e.g., in the case of a ticket dispenser in the city centre

$$\mathrm{T}(\mathsf{k}) = 3,\ \mathrm{T}(\mathsf{g}) = 12,\ \mathrm{T}(\mathsf{r}) = 30,\ \text{and } \mathrm{T}(\mathsf{v}) = 60;$$

and in the case of a ticket dispenser just outside the city centre

$$\mathrm{T}(\mathsf{k}) = 6,\ \mathrm{T}(\mathsf{g}) = 24,\ \mathrm{T}(\mathsf{r}) = 60,\ \text{and } \mathrm{T}(\mathsf{v}) = 120.$$

We are going to consider the set of coins C and the coins-to-minutes translation T as *parameters* of a general specification of the behaviour of car park ticket dispensers.

To emphasize that the data have now come to the fore, we stop pushing them away in subscripts: we write $\mathrm{P}(n)$ to refer to the ticket dispenser after the insertion of coins to the equivalent of n minutes; and we write $\mathrm{in}(\mathsf{k})$, $\mathrm{in}(\mathsf{g})$, $\mathrm{in}(\mathsf{r})$ and $\mathrm{in}(\mathsf{v})$ to refer to the events of inserting the respective coins. The car owner inserting a coin, with the dispenser in state n, could then be denoted by

$$\mathrm{in}(\mathsf{k}) \cdot \mathrm{P}(n + \mathrm{T}(\mathsf{k})) + \mathrm{in}(\mathsf{g}) \cdot \mathrm{P}(n + \mathrm{T}(\mathsf{g})) \\ + \mathrm{in}(\mathsf{r}) \cdot \mathrm{P}(n + \mathrm{T}(\mathsf{r})) + \mathrm{in}(\mathsf{v}) \cdot \mathrm{P}(n + \mathrm{T}(\mathsf{v})).$$

(Caution: the symbol "+" occurs in two different capacities: referring to a choice between events, and referring to addition of natural numbers.)

This notation abstracts from the particular association between coins and minutes, and hence is suitable for the specification of ticket dispensers inside and outside the city center. But it is quite long, and it contains some redundant information. That is, which coins the ticket dispensers accept is already clear upon presenting the parameter C; we can abstract from this information in the specification of their behaviours. We want to say that the car owner may insert *any* member c of the set of coins C, upon which the dispenser updates its state with the appropriate number of minutes $\mathrm{T}(c)$. We give the symbol "c" the status of a *variable* that ranges over C, introduce a new formal symbol "$\sum_c$", and specify the event by

$$\textstyle\sum_c \mathrm{in}(c) \cdot \mathrm{P}(n + \mathrm{T}(c)).$$

In contrast to the previous notation, the new notation has the additional advantage of being 'euro proof': to make the transition from the present Dutch coins to European currency, the only thing that has to be done is to adapt the parameter C, replacing the Dutch coins by euros, and to adapt the mapping T accordingly. Incidentally, the new notation also reflects in a more natural way the physical appearance of ticket dispensers in Amsterdam: they have only one slot, which takes all types of coins.

The effect of pressing the green button depends on the state n of the dispenser: if the car owner has paid for at least 2 times the amount of minutes associated with a 'kwartje', i.e., $n \geq 2 \times \mathrm{T}(\mathsf{k})$, then the dispenser produces a ticket; otherwise nothing

happens. To specify this in a concise way, we use the notation "$\lhd\, n \geq 2 \times \mathrm{T}(\mathsf{k}) \,\rhd$" (in general, read '$x \lhd b \rhd y$' as 'then x if b else y'); e.g., we write

$$\mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}(0) \lhd n \geq 2 \times \mathrm{T}(\mathsf{k}) \rhd \mathrm{button} \cdot \mathrm{P}(n)$$

to abbreviate 'if $n \geq 2 \times \mathrm{T}(\mathsf{k})$, then $\mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}(0)$ happens, and otherwise $\mathrm{button} \cdot \mathrm{P}(n)$ happens'.

We can now specify the behaviour of ticket dispensers in Amsterdam concisely by means of the following equation:

$$\begin{aligned} \mathrm{P}(3n) = {} & \textstyle\sum_c \mathrm{in}(c) \cdot \mathrm{P}(3n + \mathrm{T}(c)) \\ & + \mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}(0) \lhd 3n \geq 2 \times \mathrm{T}(\mathsf{k}) \rhd \mathrm{button} \cdot \mathrm{P}(3n) \\ & + \mathrm{knob} \cdot \mathrm{return} \cdot \mathrm{P}(0) \lhd 3n \geq 3 \rhd \mathrm{knob} \cdot \mathrm{P}(0). \end{aligned} \tag{1.3}$$

Note that, to make this behaviour specification completely euro proof, we should eliminate the explicit mention of k in $2 \times \mathrm{T}(\mathsf{k})$, e.g., by introducing a constant that represents the minimum number of minutes that can be bought.

Most of the symbols used in (1.3) explicitly refer to specific aspects of ticket dispensers, denoting specific events associated with such machines or naming specific behaviour that they may exhibit in a certain state. In contrast, the symbols "$+$", "$\cdot$", "$\sum\_$" and "$\lhd\ \_\ \rhd$" refer to mechanisms that are not specific to ticket dispensers. They have the kind of generality that one expects of the primitives of a general purpose specification formalism.

1.1.3 The process specification language μCRL

The previous two subsections serve to illustrate that there is at least a conceptual advantage in defining some relevant data separately when specifying a process. Many *process specification languages*, i.e., specification languages whose principal purpose is to specify the behaviour of systems, nowadays are accommodated with facilities to define data separately and with mechanisms to incorporate these in the actual behaviour specification. Examples of such process specification languages are, e.g., LOTOS (Bolognesi and Brinksma, 1987), PSF (Mauw and Veltink, 1990) and μCRL (Groote and Ponse, 1995). In this thesis, we shall elaborate on the theoretical foundations of the process specification language μCRL (micro Common Representation Language; for a survey, see Groote and Reniers (2001)).

In a μCRL specification, abstract data types are defined by means of a many-sorted algebraic specification (see, e.g., Bergstra *et al.*, 1989; Loeckx *et al.*, 1996). There is a facility to declare basic events that may take the specified data as parameters, and to aid the description of processes, μCRL includes the mechanisms symbolised by "$+$", "$\cdot$", "$\sum\_$" and "$\lhd\ \_\ \rhd$". To give some idea of what a μCRL specification looks like, we present in Table 1.1 a complete formal specification of a ticket dispenser in the centre of Amsterdam, in μCRL syntax. At this point, two further remarks about μCRL are in order.

Firstly, one should take our 'μCRL syntax' with a pinch of salt. The official syntax of μCRL was designed to be read by computers; to enhance readability for humans we deviate slightly from it. We use mathematical symbols (e.g., $\leq$, $+$)

sort B	**sort** M
func $\top, \bot :\rightarrow \mathsf{B}$	**func** $0, 3 :\rightarrow \mathsf{M}$
$\geq : \mathsf{M} \times \mathsf{M} \rightarrow \mathsf{B}$	$+ : \mathsf{M} \times \mathsf{M} \rightarrow \mathsf{M}$
var $x, y : \mathsf{M}$	**var** $x, y, z : \mathsf{M}$
rew $(x \geq 0) = \top$	**rew** $(x + y) + z = x + (y + z)$
$(0 \geq 3 + x) = \bot$	$x + y = y + x$
$(3 + x \geq 3 + y) = (x \geq y)$	$x + 0 = x$

sort C	**act** $\mathrm{button}, \mathrm{print}, \mathrm{knob}, \mathrm{return}$
func $\mathsf{k}, \mathsf{g}, \mathsf{r}, \mathsf{v} :\rightarrow \mathsf{C}$	$\mathrm{in} : \mathsf{C}$
$\mathrm{T} : \mathsf{C} \rightarrow \mathsf{M}$	
rew $\mathrm{T}(\mathsf{k}) = 3$	
$\mathrm{T}(\mathsf{g}) = 4 \times 3$	
$\mathrm{T}(\mathsf{r}) = 10 \times 3$	
$\mathrm{T}(\mathsf{v}) = 20 \times 3$	

proc $\mathrm{P}(m : \mathsf{M}) = \sum_{c:\mathsf{C}} \mathrm{in}(c) \cdot \mathrm{P}(m + \mathrm{T}(c))$
$+ \mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}(0) \lhd m \geq 2 \times \mathrm{T}(\mathsf{k}) \rhd \mathrm{button} \cdot \mathrm{P}(m)$
$+ \mathrm{knob} \cdot \mathrm{return} \cdot \mathrm{P}(0) \lhd m \geq 3 \rhd \mathrm{knob} \cdot \mathrm{P}(0)$

Table 1.1: A μCRL specification of a car park ticket dispenser. We use an abbreviation that ought to be spelled out: if t is a term of sort M and n is a natural number, then we write $n \times t$ to denote the term $(\cdots(t + t) + \cdots + t) + t$ with n occurrences of t.

as names of functions declared in **func** sections and write them infix, whereas the official μCRL syntax only allows strings of letters from the Latin alphabet as names for such functions and prescribes that they be written prefix. Also, we write $\sum_{c:\mathsf{C}} \_$ instead of $\mathtt{sum}(c : \mathsf{C}, \_)$ and $\_ \lhd \_ \rhd \_$ instead of $\_\mathtt{<|}\_\mathtt{|>}\_$.

Our specification in Table 1.1 is, of course, a rather simplistic example, and that it describes what we intended to describe is a fact that hardly needs additional justification. But for larger and more complex specifications, it is vital to have computer support, e.g., to simulate specifications and to verify that they have certain properties. For μCRL, such computer support is available (see Blom *et al.* (2001), or consult `http://www.cwi.nl/~mcrl`).

Secondly, μCRL has additional mechanisms to facilitate the specification of processes; e.g., it includes mechanisms to specify that a process consists of several components running in parallel, to specify that certain parallel components must

synchronise, and to specify that certain events should be considered unobservable. In this thesis these mechanisms will not play a rôle. Interestingly, one of the computerised tools for μCRL, the so-called *lineariser* (see Groote *et al.*, 2001), is able to translate many μCRL specifications to μCRL specifications without these additional mechanisms. The other tools operate on the output of this lineariser.

1.2 Process theory

We have introduced a collection of formal symbols to write down μCRL specifications. Our explanations of the meanings of these symbols are still informal, saying something to the effect that "+" indicates a choice between alternatives, that "·" indicates that events occur consecutively, that "$\lhd \_ \rhd$" indicates a choice that depends on a condition, and that "$\sum\_$" indicates a choice that depends on input. So far, we got away with such informalities, because we have been specifying a ticket dispenser and most people already have a pretty good idea of how such slot machines tend to behave. But it is, of course, an undesirable situation that the behaviour of the ticket dispenser explains the meaning of its μCRL specification. It should be the other way around: our μCRL specifications should explain the behaviours of the systems they are meant to describe.

In other words, we want to give μCRL specifications a meaning that is independent of the systems that they intend to describe, preferably as a mathematical abstraction of the concept of a process. By interpreting μCRL specifications as mathematical objects, μCRL becomes a mathematical language. A distinct advantage of this is that we can then *prove* by mathematical means that a system behaves (or does not behave) the way it should. Before explaining the approach taken in this thesis to turn μCRL into a mathematical language, we first discuss one of our methodological considerations in a more general context.

1.2.1 Process calculi

A mathematical theory about objects that are thought of as mathematical abstractions of processes, is frequently called a *model of concurrency*, since, intuitively, a process consists of a number of activities running in parallel. Such a model of concurrency together with a formal language to reason about its elements, is what we call a *process calculus*. The pioneers of the design of process calculi are Hoare (1985) and Milner (1980).

Hoare introduced CSP (Communicating Sequential Processes) to reason about a mathematical model in which a process is viewed as a set of *failures*. A failure consists of a sequence of events in which the process may engage, together with a set of events that it subsequently refuses to engage in. Typically, the formal symbols of CSP are interpreted as operations on the failures model. These operations are shown to satisfy a set of basic mathematical laws in the form of equations, which support the mathematical reasoning about them (see Brookes *et al.* (1984)).

Milner introduced CCS (Calculus of Communicating Systems; see also Milner (1989, 1999)) to reason about a mathematical model in which a process is viewed as a *labeled transition system* modulo *observation equivalence*. A labeled transition

system consists of *states*, and *transitions* between states labeled with names of events; a transition marks the occurrence of the associated event. With each such labeled transition system one can, intuitively, associate a notion of observable behaviour. To consider a labeled transition system modulo observation equivalence means to consider a set consisting of all labeled transition systems that represent the same observable behaviour.

Again, the formal symbols of CCS are interpreted as operations on the mathematical model for which CCS was introduced, i.e., on sets of labeled transition systems modulo observation equivalence. And again, to support the mathematical reasoning, these operations are shown to satisfy a set of basic mathematical laws in the form of equations. We quote Milner (1983):

> "These four operators [of CCS] obey (as we show) several algebraic identities. It is not too much to hope that a class of these identities may be isolated as axioms of an algebraic 'concurrency' theory, analogous (say) to rings or vector spaces. For the present, however, we concentrate on an interpretation of the calculus derived from an operational or dynamic understanding of each operator, whereupon the algebraic identities arise as theorems."

By using the terms 'rings' and 'vector spaces', Milner makes a connection with an established area of mathematics, that of *abstract algebra* (see, e.g., Hungerford, 1974). It comprehends the study of *algebras*, structures that consist of a set (*universe*) with a sequence of operations defined on it. The desideratum is to abstract from the nature of the elements of the universe, and to study the fundamental properties of the operations, conventionally expressed in the form of equations. Typically, one studies all algebras that satisfy a particular collection of equational axioms.

1.2.2 Process algebra

In the literature, there is not (yet) an established consensus about what is the appropriate mathematical abstraction of the notion of process, judging by the many different models of concurrency that are currently in use. However, the languages associated with these models (if any) often include mechanisms to express

1. that a process consists of a choice between a number of alternative behaviours (*alternative composition*);

2. that a process consists of a number of behaviours that are performed consecutively (*sequential composition*); and

3. that a process consists of a number of behaviours that are executed in parallel (*parallel composition*).

Bergstra and Klop (1984) propose to study these and other process theoretic mechanisms through the axiomatic method, instead of via a presupposed model of concurrency. They coined the term *process algebra* for their approach.

Wholly in the style of the contemporary textbooks on abstract algebra, Bergstra and Klop present their algebraic theory of processes in a modularised fashion. They begin with formulating the algebraic theory BPA (Basic Process Algebra) of alternative and sequential composition, both represented as binary operations. Then, they consider the algebras that satisfy the axioms of BPA and in which there is a neutral element for alternative composition that acts as a left zero for sequential composition. That element stands for *deadlock*, the process without any behaviour. The algebraic theory of alternative composition, sequential composition and deadlock is called BPA_δ (Basic Process Algebra with deadlock). Subsequently, they discuss extensions of the theories BPA and BPA_δ with a binary operation for *parallel composition*. First, they consider parallel composition without communication, obtaining the theories PA (Process Algebra) and PA_δ (Process Algebra with deadlock). Thereafter, they also consider a form of parallel composition in combination with mechanisms to express and to require synchronisation between parallel components; the resulting theory is called ACP (Algebra of Communicating Processes).

What makes the process theories of Bergstra and Klop truly algebraic is that they are axiomatic, and, moreover, not prescriptive with respect to the objects that are taken to represent processes (in the same way as group theory does not prescribe what the elements of a group should be). This has a didactical advantage; for instance, to understand what alternative composition is, one does not need to first digest, e.g., the mathematically quite involved definition of labeled transition system modulo observation equivalence. Furthermore, placing process theoretic mechanisms in a general algebraic context has the methodological advantage that they easily make contact with mechanisms studied elsewhere. For instance, it is at once clear that an algebra satisfying the axioms of BPA is a semilattice with respect to alternative composition, and that it therefore has a natural partial order associated with it, defined in terms of alternative composition. This partial order turns out to be a convenient tool in process algebra.

The algebraic theory BPA of alternative and sequential composition may be used to give a precise mathematical interpretation of our first formal specification of the ticket dispenser (the one with implicit data). The most important stipulation is that the symbols "$+$" and "$\cdot$" denote the binary operations of alternative and sequential composition from the theory BPA. To assign a mathematical object to our specification, we need to select

1. a particular algebra, say $\mathbf{P}$, that satisfies the axioms of BPA, and

2. interpretations of the symbols listed in (1.1) on p. 2 as elements of $\mathbf{P}$.

Henceforth we shall refer to the combination of 1 and 2 as a *model of* BPA *with actions*. Then, a *solution* of our specification in $\mathbf{P}$ is an assignment of elements of $\mathbf{P}$ to the symbols listed in (1.2) such that all the defining equations are true in $\mathbf{P}$.

To proceed a little more generally we define grammatical categories of

process expressions: (i) each of the symbols in the lists (1.1) and (1.2) are process expressions; (ii) if p and q are both process expressions, then so are $p + q$

and $p \cdot q$; and (iii) every process expression can be obtained by finitely many applications of (i) and (ii);

process equations: if P is a symbol from the list (1.2), and p is a process expression, then $P = p$ is a process equation that defines P; and

process specifications: a set of process equations such that each symbol from the list (1.2) is defined exactly once.

We see that the symbols in the list (1.1) and those in the list (1.2) have different grammatical functions in a process specification. To distinguish them, we agree to call the symbols in the first list *action symbols*, and those in the second list *process variables*.

We can now speak of the solution of an arbitrary process specification in a model **P** of BPA with actions. The point of the algebraic approach, however, is that we do not need to commit ourselves to a particular model of BPA with actions, before we can start doing calculations. For instance, we can already prove the equivalence of process specifications by applying the axioms of BPA (the first five axioms in Table 2.1 on p. 17 below) to the right-hand sides of their process equations, or by applying other rules that preserve the solutions of recursive specifications in *any* model of BPA with actions (see, e.g., Ponse and Usenko, 2001).

Now, let us consider our μCRL specification of the ticket dispenser to see whether it can also be given a precise mathematical interpretation by means of the theory BPA. At first sight, it does not fit our definition of a process specification, because of the occurrences of the symbols "$\sum_{\_}$" and "$\lhd\ \_\ \rhd$". However, recall that our initial motivation for introducing these extra symbols was to be able say things more succinctly, and not to be able to say new things. Our μCRL specification of the ticket dispenser was intended to specify in a better way what had already been specified by our first specification. The latter is, according to our definition above, a genuine process specification. We could perhaps give a mathematical interpretation to μCRL specifications by first translating them to process specifications.

In our example, the translation may be carried out in three straightforward steps:

1. Replace the expression $\sum_c \mathrm{in}(c) \cdot \mathrm{P}(n + \mathrm{T}(c))$ by the sum of all the instances of the expression $\mathrm{in}(c) \cdot \mathrm{P}(n + \mathrm{T}(c))$ with a coin for the variable c.

2. Collect in a set all the instantiations of the equation defining $\mathrm{P}(m : \mathsf{M})$ with a natural number of the form $3n$ for the variable m.

3. Eliminate all occurrences of "$\lhd\ \_\ \rhd$" from the equations in the set obtained in the second step. This can be done by replacing the occurrences of expressions '$\mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}(0) \lhd 3n \geq 2 \times \mathrm{T}(\mathsf{k}) \rhd \mathrm{button} \cdot \mathrm{P}(3n)$' by '$\mathrm{button} \cdot \mathrm{print} \cdot \mathrm{P}(0)$' if $3n \geq 2 \times \mathrm{T}(\mathsf{k})$ evaluates to true and by '$\mathrm{button} \cdot \mathrm{P}(3n)$' if it evaluates to false, and by treating the occurrences of '$\mathrm{knob} \cdot \mathrm{return} \cdot \mathrm{P}(0) \lhd 3n \geq 0 \rhd \mathrm{knob} \cdot \mathrm{P}(0)$' in a similar fashion.

Note that the first two steps of the translation eliminate all occurrences of variables, and that this guarantees that the evaluation of the middle component of an

occurrence of "$\lhd \_ \rhd$" in the third step can always be done. Further note that the set of equations generated in the second step is infinite; but this is not a problem since process specifications consisting of an infinite set of process equations are allowed according to our definition.

Nevertheless, the recipe does not work in general, and the culprit is in the first step. The variable associated with an occurrence of the symbol "$\sum\_$" may range over any of the specified data sorts, and consequently it may range over an infinite set (e.g., it could range over the sort M in our example specification). So the syntactic sum that should be associated with a $\mu\mathsf{CRL}$ expression starting with an occurrence of the symbol "$\sum\_$" may, in general, consist of infinitely many components. However, such an infinite sum is not a process expression according to our inductive definition. And with reason: the intended infinite alternative composition may fail to exist for some models of BPA with actions. That is, whether a model of BPA with actions is suitable for the interpretation of a certain $\mu\mathsf{CRL}$ specification, depends on whether it has the right sums.

1.2.3 Infinite sums

Let $\mathbf{P}$ be an algebra that satisfies the axioms of BPA_δ (in what follows it will be convenient to assume the presence of the neutral element δ for alternative composition). We have already indicated that $\mathbf{P}$, being a semilattice with respect to alternative composition, has a natural partial order associated with it. Conversely, the alternative composition of two elements of $\mathbf{P}$ may be defined as their least upper bound with respect to that partial order. Generalising this definition, the alternative composition of the empty set is the minimal element δ, and the alternative composition of a nonempty finite subset P' of the universe of $\mathbf{P}$ is its least upper bound, say $\sum \mathsf{P}'$. If P' is an infinite set, then it may not have a least upper bound $\sum \mathsf{P}'$ in the universe of $\mathbf{P}$.

In Chapter 2 of this thesis we shall develop a general theory about infinite sums in algebras that satisfy the axioms of BPA_δ. If P is the universe of such an algebra, then we define on it an operation

$$\sum : \mathcal{D} \to \mathsf{P}, \text{ with } \mathcal{D} \subseteq \{\mathsf{P}' \mid \mathsf{P}' \subseteq \mathsf{P}\}.$$

In the terminology of Rasiowa and Sikorski (1963), the operation $\sum$ is a "generalised operation". Assigning an element of P to some (but generally not to all) subsets of the universe P, it may be thought of as a partial operation with variable (and possibly infinite) arity. The element $\sum \mathsf{P}'$ assigned to a subset P' of P must satisfy a few requirements. To tie in with process algebraic traditions, we shall define those requirements by means of axioms in the form of equations. The theory that is obtained by adding these axioms to the axioms of BPA_δ we shall refer to as GBPA_δ (Generalised Basic Process Algebra with deadlock); the mechanism embodied by $\sum$ we call *generalised summation*.

To substantiate our definitions, we shall study certain natural extensions with infinite sums of the algebra of finitely branching transition trees of finite depth (for which the axioms of BPA_δ are known to be sound and complete). In the case of transition trees, the sum $\sum \mathsf{T}$ of a set T of transition trees is the transition tree

that we get by identifying the roots of the trees in T. Clearly, $\sum \mathsf{T}$ is a countably branching tree provided that the set T is countable and its elements are countably branching trees. Therefore, on the algebra of countably branching transition trees the operation $\sum$ assigns a sum to every countable set of transition trees. In the same way we can define for every infinite cardinal κ an algebra of transition trees with branching degree $< \kappa$ that is closed under sums of cardinality $< \kappa$. We shall prove that the axioms of GBPA_δ are sound and complete for each of these algebras (see Theorem 2.11).

1.3 Choice quantification

The symbol "$\sum\_$" of $\mu\mathsf{CRL}$ refers to a rather special kind of generalised summation. Let x be a variable that ranges over an arbitrary set D of data values (e.g., associated with a sort in the data part of a $\mu\mathsf{CRL}$ specification). If p is an expression that denotes a process after instantiating its data variables, then

$$\textstyle\sum_x p = \sum\{p[x := \mathsf{d}] \mid \mathsf{d} \in \mathsf{D}\}.$$

In words, $\sum_x p$ is the generalised sum of all the instantiations of p with an element of D for the variable x. Thus, $\sum_x$ refers to taking the generalised sum of a set that is obtained by quantification over D; henceforth, we speak of *choice quantification*, and call $\sum_x$ a *choice quantifier*.

From Chapter 3 onwards, we shall study choice quantification in the context of pCRL (pico Common Representation Language), which is $\mu\mathsf{CRL}$ without parallelism. With respect to its common definition, we shall make two further simplifications:

1. We restrict our attention to processes that are specified from an alphabet of actions parametrised with data by means of alternative and sequential composition, conditional composition and choice quantification; in particular, we do not consider recursion. Thus, we only consider processes whose behaviour is finite with respect to the number of actions that they can perform consecutively. This is certainly not a minor restriction. Clearly, $\mu\mathsf{CRL}$ without recursion would not have many applications as a process specification language. For all that, we do believe that it is sensible to try and understand the case without recursion first, since it usually greatly helps the understanding of the case with recursion as well.

2. We shall only consider choice quantification over a single data domain, which may be fitted with functions and relations. This is for the sake of clarity of presentation only; all of our results generalise in the obvious way to choice quantification over multiple domains and thus fit in with $\mu\mathsf{CRL}$.

In Chapter 3 we give a semantics to the language pCRL by explaining how its expressions denote elements of suitable models of GBPA_δ, where suitability depends on the availability of generalised sums. For every particular choice of a data domain $\mathbf{D}$ and an alphabet of parametrised actions $\mathcal{A}$ we shall define a suitable algebra of transition trees that is initial in the class of all suitable models

of GBPA_δ (Theorem 3.15). If the cardinality of D is infinite, then there exist pCRL expressions whose interpretation in this initial algebra is an infinitely branching transition tree.

For use in later chapters, we shall establish in Chapter 3 two results regarding particular syntactic forms of pCRL expressions. The first result states that every pCRL expression is semantically equivalent to a *tree form*, a pCRL expression whose syntactic structure, intuitively, reflects the structure of the transition tree that it denotes (Lemma 3.22). The second result makes use of a straightforward translation of the finite, sequential fragment of value-passing CCS (as discussed, e.g., by Hennessy and Lin (1996)) into pCRL expressions. Our interest is in the *input prefix* mechanism, the translation of which involves choice quantification: if the translation assigns the pCRL expression p' to an expression p of value-passing CCS (let us concisely denote this by $p \mapsto p'$), then

$$c?x.p \mapsto \sum_x \mathrm{c}(x) \cdot p'.$$

Note that the variable x of the choice quantifier $\sum_x$ occurs as a parameter of the action c that immediately follows it. If all choice quantifiers in an expression have this property, then we shall say that the expression has *explicit instantiation*. It turns out that the tree forms associated with (translations of) expressions of value-passing CCS have explicit instantiation (Lemma 3.26).

1.3.1 Expressiveness

In Chapter 4 we shall investigate the expressiveness of the mechanisms of pCRL by considering the complexity of pCRL equations. Not surprisingly, the complexity of a pCRL equation depends on the data that occur in it. The results that we shall obtain are therefore relative to the complexity of the incorporated data. Precisely, we prove that any pCRL equation can be effectively transformed into an equivalent first-order assertion about the data (Theorem 4.10), and that, conversely, any first-order assertion about the data gives rise to an equivalent pCRL equation (Theorem 4.17). Hence, pCRL is as expressive as first-order logic, with respect to the incorporated data.

In particular, we shall see that pCRL owes to a large extent its expressiveness to choice quantification. It accounts for the simulation of the universal as well as the existential quantifiers of first-order logic. It turns out that an equation of pCRL expressions with explicit instantiation has the content of a universal first-order assertion about the data that occurs in it (Corollary 4.23). Hence, the finite, sequential fragment of value-passing CCS is as expressive as the universal fragment of first-order logic, with respect to the incorporated data.

1.3.2 Deductive system

In Chapter 5 we shall present a deductive system for pCRL equations, so that when doing calculations with pCRL expressions we may proceed entirely syntactically. The desiderata for the design of our deductive system are

1. to separate reasoning about the data inside pCRL expressions from reasoning about behavioural aspects; and

2. to fit in as much as possible with standard equational reasoning.

A natural question to ask about a deductive system is whether it is complete, i.e., whether it allows a deduction for every pCRL equation that holds in every suitable model of GBPA_δ. The expressiveness results of Chapter 4 indicate that such a completeness result cannot be obtained, unless drastic restrictions on the incorporated data are imposed. We prove that our deductive system is complete provided that it may ask an oracle to provide deductions of valid first-order assertions about the incorporated data (Theorem 5.20).

1.3.3 Algebraic semantics

The framework developed in Chapters 2, 3 and 5 has a *syntactical* side and a *semantical* side (see Figure 1.1). On the syntactical side we find the formal system pCRL; it extends the formal system associated with BPA_δ with choice quantifiers and conditionals. On the semantical side we find the algebraic theory GBPA_δ; it extends the algebraic theory BPA_δ with generalised summation. The reason for extending BPA_δ differently on both sides is as follows. On the one hand, generalised summation is an infinitary operation, i.e., it may take infinitely many arguments. Since is a desirable property of a formal system that its expressions are finite, an infinitary operation is not a convenient construction to have in such a system. On the other hand, the choice quantifiers of pCRL are binders, relying on the syntactic nature of their arguments, while the desideratum of an algebraic theory is to abstract from the nature of the objects under consideration.

A pCRL expression p together with a valuation that assigns data values to data variables describes a process, an element of a generalised basic process algebra with deadlock. The pCRL expression p itself may thus be thought of as the description of function from the set data values into a universe of processes, i.e. as the description of a *parametrised process*. In Chapter 6 we shall propose an algebraic theory of parametrised processes. It unites the syntactical and the semantical sides of pCRL in a single purely algebraic theory of *basic process modules* (BPM). It is obtained from pCRL by abstracting from the syntactic aspects of choice quantification, and it is obtained from GBPA_δ by adding a notion of dimension. We shall prove that the (ground) equational theory of basic process modules is equivalent to that of pCRL (Theorem 6.37).

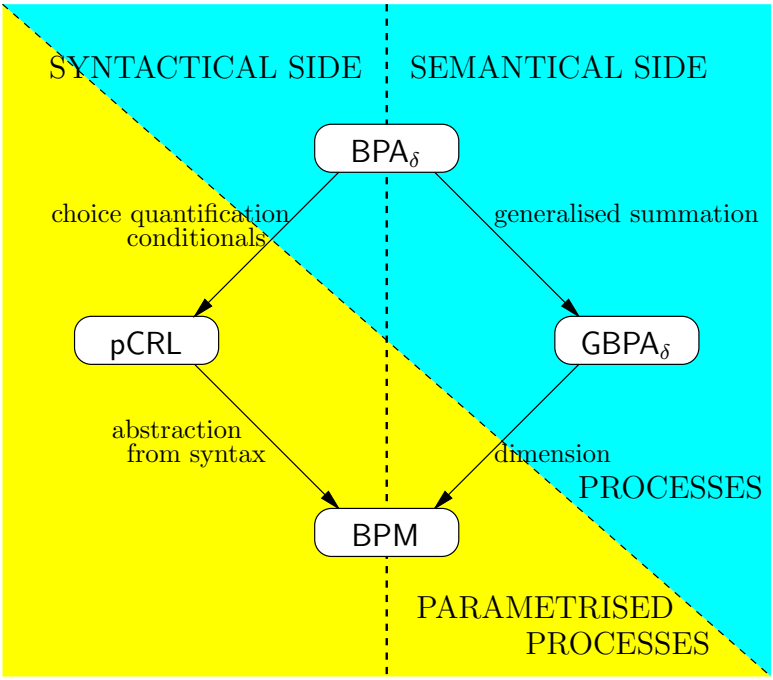


Figure 1.1: The process theories in this thesis.

2

Process algebras with infinite sums

A *basic process algebra with deadlock* is an algebra[1] $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta \rangle$ that satisfies for all $\mathsf{p}, \mathsf{q}, \mathsf{r} \in \mathsf{P}$ the equalities in Table 2.1. The class of all basic process algebras with deadlock is denoted by BPA_δ. The elements of a basic process algebra with deadlock we shall call *processes*. Intuitively, a process $\mathsf{p} \in \mathsf{P}$ is a collection of behaviours that we shall refer to as the *alternatives* in p.

(A1)	$\mathsf{p} + \mathsf{q}$	$= \mathsf{q} + \mathsf{p}$
(A2)	$\mathsf{p} + (\mathsf{q} + \mathsf{r})$	$= (\mathsf{p} + \mathsf{q}) + \mathsf{r}$
(A3)	$\mathsf{p} + \mathsf{p}$	$= \mathsf{p}$
(A4)	$(\mathsf{p} + \mathsf{q}) \cdot \mathsf{r}$	$= \mathsf{p} \cdot \mathsf{r} + \mathsf{q} \cdot \mathsf{r}$
(A5)	$(\mathsf{p} \cdot \mathsf{q}) \cdot \mathsf{r}$	$= \mathsf{p} \cdot (\mathsf{q} \cdot \mathsf{r})$
(A6)	$\mathsf{p} + \delta$	$= \mathsf{p}$
(A7)	$\delta \cdot \mathsf{p}$	$= \delta$

Table 2.1: The axioms of basic process algebras with deadlock.

The operation $+$ stands for *alternative composition* (or: *choice*); if p and q are processes, then $\mathsf{p} + \mathsf{q}$ is the process that executes either an alternative in p or an alternative in q. According to (A1)–(A3), the structure $\langle \mathsf{P}, + \rangle$ is a semilattice. According to (A6), this semilattice has a neutral element δ that we call *deadlock*; it is the process with no alternatives.

The operation $\cdot$ stands for *sequential composition*. If p and q are processes, then $\mathsf{p} \cdot \mathsf{q}$ is the process that starts with executing an alternative in p, and if this execution terminates, then it proceeds with executing an alternative in q. Sequential composition is associative by (A5), and it distributes from the right over alternative composition by (A4). The process δ is a left zero for sequential composition by (A7). Note that we do not require that sequential composition distributes from the left over alternative composition. The underlying idea is that the choices in a process are not resolved beforehand, but in the course of execution. We shall illustrate this by means of an example.

[1] We shall assume that the reader is familiar with the basic definitions of set theory (see, e.g., Halmos, 1974) and universal algebra (see, e.g., Burris and Sankappanavar, 1981; McKenzie *et al.*, 1987).

Example 2.1 Let us consider a simple protocol for the acknowledged transmission of a message from a sender S to a receiver R through an unreliable medium M (see Figure 2.1). The sender has a connection to the medium that we call c_1 and

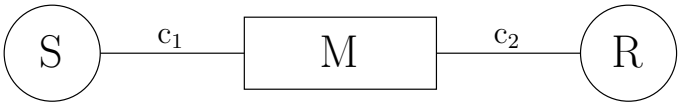


Figure 2.1: A simple protocol for the acknowledged transmission of messages.

the receiver has a connection to the medium that we call c_2. The sender sends a message m into the medium along c_1. In the medium the contents of m may get corrupted; we assume that the receiving party has the means to verify the validity of a message. After the receiver has received m or a corrupted version, it responds by sending an acknowledgment to the sender through the medium (to keep the example simple we assume that the medium does not corrupt acknowledgments): it sends a positive acknowledgment ($\mathsf{1}$) to the sender if it has received a valid message; otherwise it sends a negative acknowledgment ($\mathsf{0}$). The sending party may be modeled as the following process:

$$\mathrm{S} = \mathrm{s}_1(\mathsf{m}) \cdot (\mathrm{r}_1(\mathsf{0}) + \mathrm{r}_1(\mathsf{1})),$$

where $\mathrm{s}_1(\mathsf{m})$ denotes the action of sending m along c_1, $\mathrm{r}_1(\mathsf{0})$ denotes the action of receiving $\mathsf{0}$ along c_1 and $\mathrm{r}_1(\mathsf{1})$ denotes the action of receiving $\mathsf{1}$ along c_1. It is understood here that an action $\mathrm{r}_1(a)$ ($a \in \{\mathsf{0}, \mathsf{1}\}$) synchronises with an action $\mathrm{s}_1(a)$ from the medium. Thus, the choice between $\mathrm{r}_1(\mathsf{0})$ or $\mathrm{r}_1(\mathsf{1})$ is not made by the sender. It is determined by the medium, and it is not made before the action $\mathrm{s}_1(\mathsf{m})$ has occurred. So, we want that $\mathrm{s}_1(\mathsf{m}) \cdot (\mathrm{r}_1(\mathsf{0}) + \mathrm{r}_1(\mathsf{1})) \neq \mathrm{s}_1(\mathsf{m}) \cdot \mathrm{r}_1(\mathsf{0}) + \mathrm{s}_1(\mathsf{m}) \cdot \mathrm{r}_1(\mathsf{1})$.

2.1 Generalised basic process algebras with deadlock

Since $\langle \mathsf{P}, + \rangle$ is a semilattice, we may associate with every basic process algebra with deadlock a partial order $\leq$ defined for $\mathsf{p}, \mathsf{q} \in \mathsf{P}$ by

$$\mathsf{p} \leq \mathsf{q} \text{ if, and only if, } \mathsf{q} = \mathsf{q} + \mathsf{p}.$$

Deadlock is the least element with respect to this partial order and any two processes p and q have a least upper bound $\mathsf{p} + \mathsf{q}$. Thus, in a basic process algebra with deadlock any finite set $\{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ of processes has a least upper bound $\mathsf{p}_1 + \cdots + \mathsf{p}_n$.

In Example 2.1 we have modeled the action of receiving an acknowledgment as the alternative composition of the actions of receiving a negative acknowledgment and receiving a positive acknowledgment. Similarly, if $\mathsf{D} = \{\mathsf{d}_1, \ldots, \mathsf{d}_n\}$ is any arbitrary finite data domain, then we may model the receipt of an arbitrary element of D as the process $\mathrm{r}(\mathsf{d}_1) + \cdots + \mathrm{r}(\mathsf{d}_n)$, i.e., as the least upper bound of the set of actions that stand for the receipt of a particular element of D. Taking this a

(GA1)	$\mathsf{p} \leq \sum \mathsf{P}'$, for all $\mathsf{p} \in \mathsf{P}'$;
(GA2)	if $\mathsf{p} \leq \mathsf{q}$ for all $\mathsf{p} \in \mathsf{P}'$, then $\sum \mathsf{P}' \leq \mathsf{q}$; and
(GA3)	$\sum \mathsf{P}' \cdot \mathsf{q} = \sum \{\mathsf{p} \cdot \mathsf{q} \mid \mathsf{p} \in \mathsf{P}'\}$.

Table 2.2: The axioms for generalised summation.

little further, if D happens to be an infinite set, then the process that models the receipt of an arbitrary element from D would be the least upper bound of the set of processes that represent the receipt of any particular element of D. Thus, in order to be able to model the receipt of an arbitrary element from an infinite domain, we need to generalise the operation for alternative composition.

Rasiowa and Sikorski (1963) give a treatment of first-order logic from the point of view of the theory of abstract algebras. To deal with existential and universal quantifications, which coincide with certain infinite joins and meets in the Boolean algebra of first-order formulas, they propose to generalise the notion of operations in an algebra. Let A be a set; a *generalised operation* O on A is a partial mapping from the subsets of A to A. That is, $\mathrm{O} : \mathcal{D} \to \mathsf{A}$, where $\mathcal{D}$ is a set of subsets of A. The class $\mathcal{D}$ is called the *domain* of the generalised operation O and the sets in $\mathcal{D}$ are called the *admissible sets* of the operation O. Then Rasiowa and Sikorski proceed to define a *generalised (abstract) algebra* as a structure $\langle \mathsf{A}, \mathrm{o}_1, \ldots, \mathrm{o}_m, \mathrm{O}_1, \ldots, \mathrm{O}_n \rangle$, where A is a set, $\mathrm{o}_1, \ldots, \mathrm{o}_m$ is a sequence of finitary operations on A and $\mathrm{O}_1, \ldots, \mathrm{O}_n$ is a sequence of generalised operations on A. We shall adapt their definitions to our setting.

We shall be interested in process algebras in which certain infinite sets of processes have a least upper bound. Therefore, we equip our BPA_δ with a generalised operation

$$\sum : \mathcal{D} \to \mathsf{P},$$

where $\mathcal{D} \subseteq \{\mathsf{P}' \mid \mathsf{P}' \subseteq \mathsf{P}\}$ is a set of (finite or infinite) subsets of P.

Suppose that P' is admissible for $\sum$. If $\sum$ satisfies (GA1) of Table 2.2, then $\sum \mathsf{P}'$ is an upper bound of P' with respect to $\leq$. If $\sum$ also satisfies (GA2) of Table 2.2, then $\sum \mathsf{P}'$ is the least upper bound of P' with respect to $\leq$. If (GA1) and (GA2) hold for all admissible P', then we say that $\sum$ *generalises* $+$. We have the following lemma.

Lemma 2.2 If $\sum$ generalises $+$, then $\sum\{\mathsf{p}_1, \ldots, \mathsf{p}_n\} = \mathsf{p}_1 + \cdots + \mathsf{p}_n$ for all finite sets $\{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ that are admissible for $\sum$.

We see that there is only one way to define $\sum$ on a finite set of processes in such a way that it generalises $+$. This property extends to infinite sets, so, in general, if $\sum$ generalises $+$ in a basic process algebra with deadlock, then it is uniquely determined by its domain $\mathcal{D}$.

Note that if $\mathsf{P}' = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ and $\{\mathsf{p} \cdot \mathsf{q} \mid \mathsf{p} \in \mathsf{P}'\}$ are both admissible for $\sum$,

then by (A4) and Lemma 2.2

$$\sum \mathsf{P}' \cdot \mathsf{q} = (\mathsf{p}_1 + \cdots + \mathsf{p}_n) \cdot \mathsf{q} = \mathsf{p}_1 \cdot \mathsf{q} + \cdots + \mathsf{p}_n \cdot \mathsf{q} = \sum\{\mathsf{p} \cdot \mathsf{q} \mid \mathsf{p} \in \mathsf{P}'\}.$$

We want this equation to hold for infinite sums too, but this is not automatic.

Example 2.3 Let Ω be the first uncountable ordinal; then $\langle \Omega, \cup, \times, 0\rangle$ is a basic process algebra with deadlock. Indeed, set-theoretic union is commutative, associative and idempotent, and the binary operation $\times$ (ordinal multiplication) is associative and distributes from the right over $\cup$. Furthermore, the ordinal 0 is a neutral element for $\cup$ and a left zero for $\times$.

The set Ω is closed under countable unions (see, e.g., Halmos, 1974), so that we may define a generalised operation

$$\bigcup : \{\Gamma \subseteq \Omega \mid |\Gamma| \leq \aleph_0\} \to \Omega.$$

Clearly, $\bigcup$ generalises $\cup$, but $\times$ does not distribute from the right over $\cup$; e.g.,

$$\bigcup \omega \times 2 = \omega \times 2 \neq \omega = \bigcup\{n \times 2 \mid n \in \omega\}.$$

We shall consider extensions of basic process algebras with deadlock with an operation $\sum$ that generalises $+$ in such a way that $\cdot$ distributes from the right over $\sum$. Distributivity from the right of $\cdot$ over $\sum$ is formulated as (GA3) in Table 2.2, which is to be interpreted in the sense that if one side of the equality is defined, then so is the other.

Definition 2.4 A *generalised* basic process algebra with deadlock is a generalised algebra $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \sum\rangle$ such that

(i) $\langle \mathsf{P}, +, \cdot, \delta\rangle$ is a basic process algebra with deadlock;

(ii) $\mathsf{P}' \subseteq \mathsf{P}$ is admissible if, and only if, $\{\mathsf{p} \cdot \mathsf{q} \mid \mathsf{p} \in \mathsf{P}'\}$ is admissible for all $\mathsf{q} \in \mathsf{P}$;

(iii) (GA1)–(GA3) of Table 2.2 hold for all admissible $\mathsf{P}' \subseteq \mathsf{P}$ and for all $\mathsf{q} \in \mathsf{P}$.

We denote by GBPA_δ the class of generalised basic process algebras with deadlock.

Suppose that $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta\rangle$ is an arbitrary basic process algebra with deadlock. If we want to extend it with a generalised operation $\sum$ that satisfies the axioms in Table 2.2, then we have some freedom with respect to the specification of the admissible sets of $\mathcal{D}$ (in fact, as we have seen above, this is the only freedom we have). For instance, we may define $\sum$ as having no admissible sets at all (the *trivial generalisation* of $\mathbf{P}$), or as having as admissible sets precisely the finite subsets of P (the *finitary generalisation* of $\mathbf{P}$). But mostly, we shall be interested in the *maximal generalisation* of $\mathbf{P}$ in which the domain of $\sum$ is the largest set of subsets of P such that $\langle \mathsf{P}, +, \cdot, \delta, \sum\rangle$ is a generalised basic process algebra with deadlock.

Example 2.5 Suppose that the messages of Example 2.1 are drawn from a (possibly infinite) set M, and that the receiving party can determine whether received messages are valid. Then, the receiving party may be modeled as the following process:

$$\mathrm{R} = \sum(\{\mathrm{r}_2(\mathsf{m}) \cdot \mathrm{s}_2(1) \mid \mathsf{m} \in \mathsf{M} \;\&\; \mathsf{m} \text{ is valid}\} \cup \{\mathrm{r}_2(\mathsf{m}) \cdot \mathrm{s}_2(0) \mid \mathsf{m} \in \mathsf{M} \;\&\; \mathsf{m} \text{ is not valid}\}).$$

We have specified processes by explaining how they are obtained from certain simpler processes —we have called them actions— through applications of the fundamental operations of generalised basic process algebras with deadlock. In Example 2.1 we have specified the process S by explaining how it is obtained from the actions $\mathrm{s}_1(\mathsf{m})$, $\mathrm{r}_1(0)$ and $\mathrm{r}_1(1)$ by means of the operations for sequential and alternative composition. In Example 2.5 we have specified the process R by explaining how it is obtained from the actions $\mathrm{r}_2(\mathsf{m})$, $\mathrm{s}_2(0)$ and $\mathrm{s}_2(1)$ by means of the operations of sequential composition and generalised choice. Let us now generalise a few more standard definitions from abstract algebra.

A subset $\mathsf{Q} \subseteq \mathsf{P}$ is *closed* under the generalised operation $\sum$ if $\sum \mathsf{P}' \in \mathsf{Q}$ for every $\mathsf{P}' \subseteq \mathsf{Q}$ that is admissible for $\sum$ in $\mathbf{P}$. A generalised basic process algebra with deadlock $\mathbf{Q} = \langle \mathsf{Q}, +, \cdot, \delta, \sum \rangle$ is a *subalgebra* of $\mathbf{P}$ if:

(i) $\langle \mathsf{Q}, +, \cdot, \delta \rangle$ is a subalgebra of $\langle \mathsf{P}, +, \cdot, \delta \rangle$;

(ii) Q is closed under $\sum$; and

(iii) $\sum$ in $\mathbf{Q}$ is the restriction to Q of $\sum$ in $\mathbf{P}$.

A set $\mathsf{P}_0 \subseteq \mathsf{P}$ is a set of *generators* for $\mathbf{P}$ if the least subalgebra of $\mathbf{P}$ that contains P_0 is $\mathbf{P}$ itself. Let $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \sum \rangle$ be a generalised basic process algebra with deadlock, and let us fix a set $\mathsf{A} \subseteq \mathsf{P}$ of *actions*. The least subalgebra that contains A contains precisely those elements of $\mathbf{P}$ that can be obtained from the actions in A by means of applications of the fundamental operations of generalised basic process algebras with deadlock. Hence, if A is a set of generators for $\mathbf{P}$, then every process can be obtained from actions by means of applications of the fundamental operations of generalised basic process algebras with deadlock.

2.2 Transition trees

We shall now construct a collection of generalised basic process algebras with deadlock in which certain infinite alternative compositions exist. We start from an infinite cardinal κ and a non-void set $\mathcal{L}$ of urelements[2] that we shall call *labels* and we define the set $\mathsf{T}_\kappa(\mathcal{L})$ of *transition trees with branching degree* $< \kappa$ as the least set such that

(i) $\{\ell\} \in \mathsf{T}_\kappa(\mathcal{L})$ for all $\ell \in \mathcal{L}$;

[2]Urelements (see, e.g., Shoenfield, 1967) are elements that are not sets themselves and do not involve sets in their construction; we work with a set theory based on urelements to rule out confusion between labels and trees.

(ii) if $\ell \in \mathcal{L}$ and $\mathsf{t} \in \mathsf{T}_\kappa(\mathcal{L})$, then $\{\langle \ell, \mathsf{t}\rangle\} \in \mathsf{T}_\kappa(\mathcal{L})$; and

(iii) if $\mathsf{T}' \subseteq \mathsf{T}_\kappa(\mathcal{L})$ and $|\mathsf{T}'| < \kappa$, then $\bigcup \mathsf{T}' \in \mathsf{T}_\kappa(\mathcal{L})$.

The elements of a tree we shall call *branches*. Clearly, if b is a branch, then either $\mathsf{b} \in \mathcal{L}$ or there exists a label ℓ and a tree t such that $\mathsf{b} = \langle \ell, \mathsf{t}\rangle$. If $\mathsf{t} \in \mathsf{T}_\kappa(\mathcal{L})$, then t has less than κ branches. The elements of $\mathsf{T}_{\aleph_0}(\mathcal{L})$ we shall call *finitely branching* ($\aleph_0$ denotes the cardinality of ω); they may be pictured as in Figure 2.2.

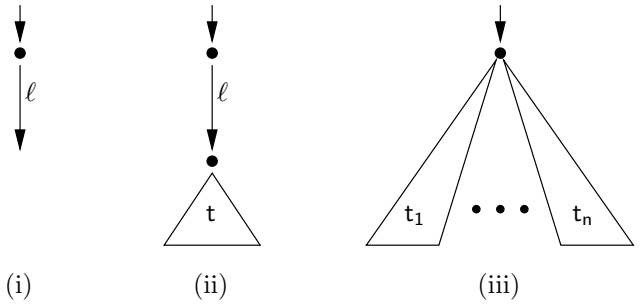


Figure 2.2: The finitely branching transition trees as constructed in (i)–(iii).

Henceforth we shall denote the empty transition tree $\emptyset$ with the symbol δ; if t and u are transition trees, then $\mathsf{t} + \mathsf{u}$ is their union; and we define $\mathsf{t} \cdot \mathsf{u}$ by:

$$\mathsf{t} \cdot \mathsf{u} = \textstyle\bigcup_{\mathsf{b} \in \mathsf{t}} (\mathsf{b} \odot \mathsf{u}),$$
where $\langle \ell, \mathsf{t}'\rangle \odot \mathsf{u} = \langle \ell, \mathsf{t}' \cdot \mathsf{u}\rangle$ and $\ell \odot \mathsf{u} = \langle \ell, \mathsf{u}\rangle$ $(\ell \in \mathcal{L},\ \mathsf{t}' \in \mathsf{T}_\kappa(\mathcal{L}))$.

Let us denote by $\mathcal{D}_\kappa$ the subsets of $\mathsf{T}_\kappa(\mathcal{L})$ with cardinality $< \kappa$, i.e., let

$$\mathcal{D}_\kappa = \{\mathsf{T}' \subseteq \mathsf{T}_\kappa(\mathcal{L}) \mid |\mathsf{T}'| < \kappa\};$$

we define on $\mathsf{T}_\kappa(\mathcal{L})$ a generalised operation

$$\textstyle\sum : \mathcal{D}_\kappa \to \mathsf{T}_\kappa(\mathcal{L}) \text{ such that } \mathsf{T}' \mapsto \bigcup \mathsf{T}'.$$

Proposition 2.6 The algebra $\mathbf{T}_\kappa(\mathcal{L}) = \langle \mathsf{T}_\kappa(\mathcal{L}), +, \cdot, \delta, \sum\rangle$ is a generalised basic process algebra with deadlock, for every infinite cardinal κ.

Proof. It is immediate that $\langle \mathsf{T}_\kappa(\mathcal{L}), +\rangle$ is a semilattice. The partial order associated with it is set inclusion, and clearly, with respect to set inclusion, $\emptyset = \delta$ is the least element in $\mathsf{T}_\kappa(\mathcal{L})$ and $\bigcup \mathsf{T}' = \sum \mathsf{T}'$ is the least upper bound of any admissible $\mathsf{T}' \subseteq \mathsf{T}_\kappa(\mathcal{L})$. Hence (A1)–(A3), (A6), (GA1) and (GA2) hold in $\mathbf{T}_\kappa(\mathcal{L})$. It is immediate from the definitions that (GA3) holds. Since $\sum \emptyset = \emptyset = \delta$, (A7) is a special case of (GA3). Since $\sum$ generalises $+$ and every two-element set of trees is admissible for $\sum$, it follows from Lemma 2.2 that (A4) is also a special case of

(GA3). So, it remains to show that (A5) holds, i.e., that $(\mathsf{t} \cdot \mathsf{u}) \cdot \mathsf{v} = \mathsf{t} \cdot (\mathsf{u} \cdot \mathsf{v})$ for all t, u and v; we proceed by induction on the rank of t:

$$\begin{aligned}
(\mathsf{t} \cdot \mathsf{u}) \cdot \mathsf{v} &= (\{\langle \ell, \mathsf{t}' \cdot \mathsf{u}\rangle \mid \langle \ell, \mathsf{t}'\rangle \in \mathsf{t}\} \cup \{\langle \ell, \mathsf{u}\rangle \mid \ell \in \mathsf{t}\}) \cdot \mathsf{v} \\
&= \{\langle \ell, (\mathsf{t}' \cdot \mathsf{u}) \cdot \mathsf{v}\rangle \mid \langle \ell, \mathsf{t}'\rangle \in \mathsf{t}\} \cup \{\langle \ell, \mathsf{u} \cdot \mathsf{v}\rangle \mid \ell \in \mathsf{t}\} \\
&= \{\langle \ell, \mathsf{t}' \cdot (\mathsf{u} \cdot \mathsf{v})\rangle \mid \langle \ell, \mathsf{t}'\rangle \in \mathsf{t}\} \cup \{\langle \ell, \mathsf{u} \cdot \mathsf{v}\rangle \mid \ell \in \mathsf{t}\} \qquad \text{by (IH)} \\
&= (\{\langle \ell, \mathsf{t}'\rangle \mid \langle \ell, \mathsf{t}'\rangle \in \mathsf{t}\} \cup \{\ell \mid \ell \in \mathsf{t}\}) \cdot (\mathsf{u} \cdot \mathsf{v}) \\
&= \mathsf{t} \cdot (\mathsf{u} \cdot \mathsf{v}).
\end{aligned}$$

□

If $\ell \in \mathcal{L}$, then we shall call the singleton $\{\ell\}$ a *tree action*. Clearly, there is a one-to-one correspondence between the actions and the labels, and between the sequential compositions of the form $\mathsf{a} \cdot \mathsf{t}$, where a is an action, and the branches of the form $\langle \ell, \mathsf{t}\rangle$. Hence, when we picture trees, it will not give rise to confusion if we label the edges with actions instead of with the corresponding labels. See Figure 2.3 for an example that proves that in $\mathbf{T}_\kappa(\mathcal{L})$ sequential composition does not distribute from the left over alternative composition, provided that there are at least two distinct actions (it is required that b and c are distinct).

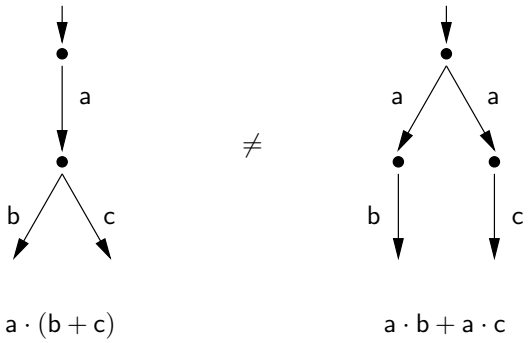


Figure 2.3: In $\mathbf{T}_\kappa(\mathcal{L})$, sequential composition does not distribute from the left over alternative composition.

In the following lemma we list a few elementary properties of tree actions.

Lemma 2.7 If a and b are tree actions, then

(i) $\mathsf{a} \not\leq \delta$, and $\mathsf{a} \cdot \mathsf{t} \not\leq \delta$ for all trees t;

(ii) for all trees t, u and v:

  (a) $\mathsf{a} \leq \mathsf{t} + \mathsf{u}$ if, and only if, $\mathsf{a} \leq \mathsf{t}$ or $\mathsf{a} \leq \mathsf{u}$, and

  (b) $\mathsf{a} \cdot \mathsf{t} \leq \mathsf{u} + \mathsf{v}$ if, and only if, $\mathsf{a} \cdot \mathsf{t} \leq \mathsf{u}$ or $\mathsf{a} \cdot \mathsf{t} \leq \mathsf{v}$;

(iii) for all admissible $\mathsf{T}' \subseteq \mathsf{T}_\kappa(\mathcal{L})$:

  (a) $\mathsf{a} \leq \sum \mathsf{T}'$ if, and only if, there exists $\mathsf{t}' \in \mathsf{T}'$ such that $\mathsf{a} \leq \mathsf{t}'$, and

(b) $\mathsf{a} \cdot \mathsf{t} \leq \sum \mathsf{T}'$ if, and only if, there exists $\mathsf{t}' \in \mathsf{T}'$ such that $\mathsf{a} \cdot \mathsf{t} \leq \mathsf{t}'$;

(iv) $\mathsf{a} \not\leq \mathsf{b} \cdot \mathsf{t}$ and $\mathsf{a} \cdot \mathsf{t} \not\leq \mathsf{b}$, for all trees t;

(v) $\mathsf{a} \leq \mathsf{b}$ if, and only if, $\mathsf{a} = \mathsf{b}$; and

(vi) $\mathsf{a} \cdot \mathsf{t} \leq \mathsf{b} \cdot \mathsf{u}$ if, and only if, $\mathsf{a} = \mathsf{b}$ and $\mathsf{t} = \mathsf{u}$, for all trees t and u.

2.3 Free GBPA_δ's

Let $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \sum \rangle$ be a generalised basic process algebra with deadlock. Sometimes, we want to use the processes of $\mathbf{P}$ to specify processes of another generalised basic process algebra with deadlock, say $\mathbf{Q} = \langle \mathsf{Q}, +, \cdot, \delta, \sum \rangle$. Then, we define

$$h : \mathsf{P} \to \mathsf{Q}$$

and we require that it preserves the fundamental operations of generalised basic process algebras with deadlock. Let $h(\mathsf{P}') = \{h(\mathsf{p}) \mid \mathsf{p} \in \mathsf{P}'\}$; if for all $\mathsf{P}' \subseteq \mathsf{P}$ admissible for $\sum$ in $\mathbf{P}$

(i) $h(\mathsf{P}')$ is admissible for $\sum$ in $\mathbf{Q}$ and

(ii) $h(\sum \mathsf{P}') = \sum h(\mathsf{P}')$,

then we say that h *preserves* $\sum$. A *homomorphism* of generalised basic process algebras with deadlock is a homomorphism of basic process algebras with deadlock that preserves $\sum$; if h is a homomorphism from $\mathbf{P}$ into $\mathbf{Q}$, then we shall write $h : \mathbf{P} \to \mathbf{Q}$. Suppose that we start from a designated set $\mathsf{A} \subseteq \mathsf{P}$ of actions and a mapping

$$f : \mathsf{A} \to \mathsf{Q}.$$

If A is a set of generators for $\mathbf{P}$ and f extends to a homomorphism, then this extension is unique. However, f does not necessarily extend to a homomorphism from $\mathbf{P}$ to $\mathbf{Q}$.

Example 2.8 Suppose that $\mathcal{L}$ is a set of labels; we denote by $\mathcal{L}^*$ the set of finite sequences of elements of $\mathcal{L}$. A *language* over $\mathcal{L}$ is any subset of $\mathcal{L}^*$; let L be the set of all languages over $\mathcal{L}$. We denote the empty language by δ; we define $X + Y = X \cup Y$ and $X \cdot Y = \{xy \mid x \in X \text{ and } y \in Y\}$ for all $X, Y \in \mathsf{L}$; and we define $\sum \mathsf{L}' = \bigcup \mathsf{L}'$ for all $\mathsf{L}' \subseteq \mathsf{L}$. The generalised algebra $\mathbf{L} = \langle \mathsf{L}, +, \cdot, \delta, \sum \rangle$ is a generalised basic process algebra with deadlock and it is generated by the set $\mathsf{L}_0 = \{\{\ell\} \mid \ell \in \mathcal{L}\}$. Moreover, in $\mathbf{L}$ sequential composition is left-distributive over alternative composition, so, in particular,

$$\{\ell_1\} \cdot (\{\ell_2\} + \{\ell_3\}) = \{\ell_1\} \cdot \{\ell_2\} + \{\ell_1\} \cdot \{\ell_3\}.$$

Consequently, if $\mathbf{Q} = \langle \mathsf{Q}, +, \cdot, \delta, \sum \rangle$ is a generalised basic process algebra with deadlock and every mapping $f : \mathsf{L}_0 \to \mathsf{Q}$ extends to a homomorphism $h : \mathbf{L} \to \mathbf{Q}$,

then

$$\begin{aligned} f(\{\ell_1\}) \cdot (f(\{\ell_2\}) + f(\{\ell_3\})) &= h(\{\ell_1\} \cdot (\{\ell_2\} + \{\ell_3\})) \\ &= h(\{\ell_1\} \cdot \{\ell_2\} + \{\ell_1\} \cdot \{\ell_3\}) \\ &= f(\{\ell_1\}) \cdot f(\{\ell_2\}) + f(\{\ell_1\}) \cdot f(\{\ell_3\}), \end{aligned}$$

which allows us to conclude that sequential composition distributes from the left over alternative composition in $\mathbf{Q}$. But then, since this is not so in $\mathbf{T}_\kappa(\mathcal{L})$ (see Figure 2.3), it follows that not every mapping $f : \mathsf{L}_0 \to \mathsf{T}_\kappa(\mathcal{L})$ extends to a homomorphism $h : \mathbf{L} \to \mathbf{T}_\kappa(\mathcal{L})$.

Suppose that $\mathcal{K}$ is any subclass of GBPA$_\delta$ and let P_0 be a set of generators for $\mathbf{P}$; then $\mathbf{P}$ is *free for* $\mathcal{K}$ *over* P_0 if every mapping $f : \mathsf{P}_0 \to \mathsf{Q}$ from P_0 into the universe Q of an element $\mathbf{Q}$ of $\mathcal{K}$ can be extended to a homomorphism $h : \mathbf{P} \to \mathbf{Q}$. We say that $\mathbf{P}$ is *free in* $\mathcal{K}$ *over* P_0 if $\mathbf{P} \in \mathcal{K}$ and $\mathbf{P}$ is free for $\mathcal{K}$ over P_0. If $\mathbf{P}$ is free in $\mathcal{K}$ over P_0, then P_0 is called a *free generating set for* $\mathbf{P}$, and $\mathbf{P}$ is said to be *freely generated by* P_0.

In abstract algebra, the elements of the free generating set for a free algebra in a particular class $\mathcal{K}$ of algebras of the same type satisfy, intuitively, no other conditions than the identities that hold for every element in every other algebra in $\mathcal{K}$ (e.g., Example 2.8 shows that the algebra $\mathbf{L}$ is not free in any class that also contains the algebra $\mathbf{T}_\kappa(\mathcal{L})$). For generalised algebras we get an extra requirement: every admissible set of the free generalised algebra must correspond to an admissible set of any other generalised algebra in the class.

Example 2.9 Let a be an action of $\mathbf{T}_{\aleph_1}(\mathcal{L})$ ($\aleph_1$ denotes the cardinality of Ω, the smallest uncountable ordinal number). We define a^n ($n \geq 1$) inductively as follows: $\mathsf{a}^1 = \mathsf{a}$ and $\mathsf{a}^{n+1} = \mathsf{a} \cdot \mathsf{a}^n$. Clearly, the set

$$\mathsf{T}' = \{\mathsf{a}^n \mid n \geq 1\}$$

is admissible for $\sum$ in $\mathbf{T}_{\aleph_1}(\mathcal{L})$. With Lemma 2.7 and induction on m and n it is easily verified that $\mathsf{a}^m = \mathsf{a}^n$ implies $m = n$, so $|\mathsf{T}'| = \aleph_0$. Consequently, T' is not admissible for $\sum$ in $\mathbf{T}_{\aleph_0}(\mathcal{L})$, so if f is the identity mapping on the actions of $\mathbf{T}_{\aleph_1}(\mathcal{L})$, f does not extend to a homomorphism from $\mathbf{T}_{\aleph_1}(\mathcal{L})$ to $\mathbf{T}_{\aleph_0}(\mathcal{L})$. Hence, the algebra $\mathbf{T}_{\aleph_1}(\mathcal{L})$ is not free in any class of algebras that also contains $\mathbf{T}_{\aleph_0}(\mathcal{L})$.

In abstract algebra, the most interesting classes of algebras are the varieties, the classes that consist precisely of all algebras that satisfy a particular set of identities. We see from Example 2.9 that a free algebra in the class of *all* generalised basic process algebras with deadlock should not have too many admissible sets; in fact, one can show that it has no admissible sets at all. Our interest is in the operation $\sum$, but in a free generalised basic process algebra with deadlock it is not defined. Hence, we shall mostly be interested in particular classes of generalised basic process algebras with deadlock that satisfy an extra requirement with respect to the admissible sets. For instance, the domain of the generalised operation $\sum$ of $\mathbf{T}_\kappa(\mathcal{L})$ consists precisely of the subsets of $\mathsf{T}_\kappa(\mathcal{L})$ that have cardinality less than κ.

Definition 2.10 A generalised basic process algebra with deadlock with universe P is κ-*complete* if every $\mathsf{P}' \subseteq \mathsf{P}$ such that $|\mathsf{P}'| < \kappa$ is admissible for $\sum$.

We shall now prove that $\mathbf{T}_\kappa(\mathcal{L})$ is a free κ-complete generalised basic process algebra with deadlock, freely generated by its actions.

Theorem 2.11 For every infinite cardinal κ, $\mathbf{T}_\kappa(\mathcal{L})$ is free in the class of κ-complete generalised basic process algebras with deadlock, with free generating set $\mathsf{T}_0 = \{\{\ell\} \mid \ell \in \mathcal{L}\}$.

Proof. Clearly, $\mathbf{T}_\kappa(\mathcal{L})$ is κ-complete, and it is generated by T_0. Let $\mathbf{P}$ be any κ-complete generalised basic process algebra, and suppose $f : \mathsf{T}_0 \to \mathsf{P}$. We define a mapping $h : \mathsf{T}_\kappa(\mathcal{L}) \to \mathsf{P}$ by induction on the rank of transition trees:

$$h(\mathsf{t}) = \textstyle\sum\{g(\mathsf{b}) \mid \mathsf{b} \in \mathsf{t}\}, \text{ where } g(\langle \ell, \mathsf{t}' \rangle) = f(\{\ell\}) \cdot h(\mathsf{t}') \text{ and } g(\ell) = f(\{\ell\}) \ .$$

Since t has less than κ branches, the set $\{g(\mathsf{b}) \mid \mathsf{b} \in \mathsf{t}\}$ is admissible for $\sum$ in $\mathbf{P}$.

Note that $h(\delta) = \sum \emptyset = \delta$. It is clear that h preserves $\sum$, whence, by Lemma 2.2, h also preserves $+$. To prove that $h(\mathsf{t} \cdot \mathsf{u}) = h(\mathsf{t}) \cdot h(\mathsf{u})$ we do induction on the rank of t:

$$\begin{aligned}
&h(\mathsf{t} \cdot \mathsf{u}) \\
&\quad = h(\{\langle \ell, \mathsf{t}' \cdot \mathsf{u} \rangle \mid \langle \ell, \mathsf{t}' \rangle \in \mathsf{t}\} \cup \{\langle \ell, \mathsf{u} \rangle \mid \ell \in \mathsf{t}\}) \\
&\quad = \textstyle\sum(\{f(\{\ell\}) \cdot h(\mathsf{t}' \cdot \mathsf{u}) \mid \langle \ell, \mathsf{t}' \rangle \in \mathsf{t}\} \cup \{f(\{\ell\}) \cdot h(\mathsf{u}) \mid \ell \in \mathsf{t}\}) \\
&\quad = \textstyle\sum(\{f(\{\ell\}) \cdot h(\mathsf{t}') \cdot h(\mathsf{u}) \mid \langle \ell, \mathsf{t}' \rangle \in \mathsf{t}\} \cup \{f(\{\ell\}) \cdot h(\mathsf{u}) \mid \ell \in \mathsf{t}\}) \quad \text{by (IH)} \\
&\quad = \textstyle\sum(\{f(\{\ell\}) \cdot h(\mathsf{t}') \mid \langle \ell, \mathsf{t}' \rangle \in \mathsf{t}\} \cup \{f(\{\ell\}) \mid \ell \in \mathsf{t}\}) \cdot h(\mathsf{u}) \\
&\quad = h(\mathsf{t}) \cdot h(\mathsf{u}).
\end{aligned}$$

Hence, h is the (unique) homomorphism that extends f. □

Bibliographic notes

Milner (1983) explains how his *value-passing* CCS (see Section 3.6), with input as a primitive construct, can be reduced to *pure* CCS, without the input construct, but with summations of the form $\sum_{i \in I} \mathsf{p}_i$, where I is a possibly infinite set. The input mechanism of Milner's value-passing CCS is a variable binding construct; his pure CCS, on the other hand, has no binders, and therefore it is more suitable for an algebraic treatment. It inspired Bergstra and Klop (1984) when they introduced their algebras of processes. They replaced Milner's infinitary operation $\sum$ with a binary operation $+$, noting that the "algebraic specification [of infinite sums] is much less obvious than that of finite sums". The definition of basic process algebras with deadlock, which we have extended with an infinitary operation $\sum$, is due to Bergstra and Klop (1984).

Baeten and Weijland (1990) provide an introduction to process algebra with an emphasis on the axiomatic approach. They present the axiom systems of Bergstra and Klop, and discuss for each of these axiom systems several models, i.e., concrete process algebras that satisfy the axioms. A nowadays standard technique

to obtain concrete process algebras is to associate a transition relation with a set of process terms by assigning a structural operational semantics (see Aceto *et al.*, 2001) to the operations, and to subsequently divide out a behavioural equivalence. Fokkink (2000), in his introduction to process algebra, puts more emphasis on that particular construction, using bisimulation as behavioural equivalence. He considers the axiom systems as tools to reason about concrete process algebras, rather than as the definition of a class of process algebras.

Incidentally, that labeled trees give rise to concrete process algebras is well-known (see, e.g., Milner, 1980; Baeten and Weijland, 1990). The particular definition of the algebras $\mathbf{T}_\kappa(\mathcal{L})$ and the proof that they are free κ-complete generalised basic process algebras with deadlock (Theorem 2.11) generalise a definition and a proof of Rodenburg (2000). (Rodenburg defines an algebra of finitely branching transition trees and proves that it is a free (basic) process algebra.) Note that Theorem 2.11 also generalises the completeness theorem for BPA_δ (see Baeten and Weijland (1990)). Namely, it is not hard to see that $\mathbf{T}_{\aleph_0}(\mathcal{L})$ is isomorphic to the algebra of finite acyclic process graphs modulo bisimulation. Since, by Lemma 2.2, the operation $\sum$ is a defined operator in $\mathbf{T}_{\aleph_0}(\mathcal{L})$, it must also be a free algebra in BPA_δ. Hence, it is isomorphic to the initial algebra of BPA_δ-terms with actions from T_0.

3

The syntax and semantics of pCRL

In the previous chapter we have acknowledged the fact that in some process algebras certain infinite sums exist, and that they play a role when we want to model input over some infinite domain. We have proposed generalised basic process algebras with deadlock to allow an explicit treatment of infinite sums. In this chapter, we put forward a formal framework to describe elements of generalised basic process algebras with deadlock. Our framework is called pCRL (pico Common Representation Language) as it consists of the core of the specification formalism μCRL. We defer the technicalities of pCRL to Section 3.2 and first give an informal introduction.

Let $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \sum \rangle$ be a generalised basic process algebra with deadlock. The development of our formal framework begins with the hypothesis that associated with $\mathbf{P}$ is a set A of *action symbols* (e.g., $\mathrm{s}_1(\mathsf{m})$, $\mathrm{r}_1(\mathsf{0})$) and a mapping

$$\mathsf{act} : \mathsf{A} \to \mathsf{P}$$

that interprets these action symbols as elements of $\mathbf{P}$. To describe other elements of $\mathbf{P}$ we may use the fundamental operations of basic process algebras with deadlock. For instance, given an interpretation of $\mathrm{s}_1(\mathsf{m})$, $\mathrm{r}_1(\mathsf{0})$ and $\mathrm{r}_1(\mathsf{1})$ as actions of $\mathbf{P}$, we may describe another element of $\mathbf{P}$ with the expression $\mathrm{s}_1(\mathsf{m}) \cdot (\mathrm{r}_1(\mathsf{0}) + \mathrm{r}_1(\mathsf{1}))$ (see Example 2.1). Similarly, if we already have an expression for each element of $\mathsf{P}' \subseteq \mathsf{P}$ and P' is a finite set, then we could describe the least upper bound of the elements in P' writing the symbol $\sum$ and listing the expressions for the elements of P' between brackets. For instance, we could describe the least upper bound of the set consisting of three actions denoted by $\mathrm{r}(\mathsf{m}_1)$, $\mathrm{r}(\mathsf{m}_2)$ and $\mathrm{r}(\mathsf{m}_3)$ with the expression $\sum\{\mathrm{r}(\mathsf{m}_1), \mathrm{r}(\mathsf{m}_2), \mathrm{r}(\mathsf{m}_3)\}$.

When P' is an infinite set, listing the expressions for the elements of P' is not an option. We need a method to denote the least upper bound of an infinite set P' with a finite expression. Recall our motivation for treating infinite sums as first-class citizens of our process algebras: the process that inputs an arbitrary element from a set D can be modeled as the least upper bound of the set of actions that model the receipt of a particular element of D, i.e., as the process $\sum\{\mathrm{r}(\mathsf{d}) \mid \mathsf{d} \in \mathsf{D}\}$. Note how we make use of the intuitive structure of the expression $\mathrm{r}(\mathsf{d})$ to explain which process we mean. The key step towards pCRL is to make this structure explicit: we presuppose a nonempty set $\mathcal{A}$ of *parametrised action symbols* with fixed arities, and we assume that the set of action symbols is of the form

$$\mathsf{A} = \{\mathrm{a}(\mathsf{d}_1, \ldots, \mathsf{d}_n) \mid \mathrm{a} \in \mathcal{A} \text{ of arity } n \text{ and } \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}\}. \tag{3.1}$$

Then, certain infinite sums are expressible using quantification over D. Let x be a variable that ranges over D; we denote the process $\sum\{\mathrm{r}(\mathsf{d}) \mid \mathsf{d} \in \mathsf{D}\}$ with the expression $\sum_x \mathrm{r}(x)$.

We further enhance the expressiveness of our language by allowing that D too has some structure. To describe processes that perform calculations on a received value, we equip D with operations that represent these calculations.

Example 3.1 Let N be the set of natural numbers and suppose that we want to describe the process that inputs a natural number and subsequently outputs its square. If $\mathrm{sqr} : \mathsf{N} \to \mathsf{N}$ is such that $\mathsf{n} \mapsto \mathsf{n}^2$, then

$$\textstyle\sum_x \mathrm{in}(x) \cdot \mathrm{out}(\mathrm{sqr}(x)) = \sum\{\mathrm{in}(\mathsf{n}) \cdot \mathrm{out}(\mathsf{n}^2) \mid \mathsf{n} \in \mathsf{N}\}.$$

To describe processes in which choices depend on a received value, we include a conditional in our language and we equip D with relations.

Example 3.2 Consider the receiving party R of the protocol described in the previous chapter (see Example 2.5): which acknowledgment is to be sent, depends on the contents of the received message. Let V be a unary relation on the set of messages M such that $\mathrm{V}(\mathsf{m})$ holds if, and only if, m is valid. Writing $\mathrm{r}_2(x) \cdot \mathrm{s}_2(\mathsf{1}) \lhd \mathrm{V}(x) \rhd \mathrm{r}_2(x) \cdot \mathrm{s}_2(\mathsf{0})$ for the set

$$\{\mathrm{r}_2(\mathsf{m}) \cdot \mathrm{s}_2(\mathsf{1}) \mid \mathsf{m} \in \mathsf{M} \;\&\; \mathrm{V}(\mathsf{m})\} \cup \{\mathrm{r}_2(\mathsf{m}) \cdot \mathrm{s}_2(\mathsf{0}) \mid \mathsf{m} \in \mathsf{M} \;\&\; \text{not } \mathrm{V}(\mathsf{m})\},$$

we may denote the receiving party R with the expression

$$\textstyle\sum_x (\mathrm{r}_2(x) \cdot \mathrm{s}_2(\mathsf{1}) \lhd \mathrm{V}(x) \rhd \mathrm{r}_2(x) \cdot \mathrm{s}_2(\mathsf{0})).$$

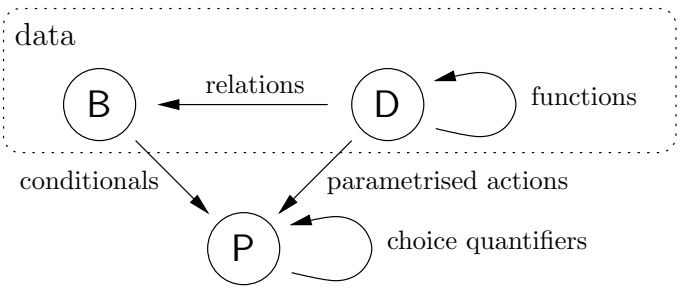


Figure 3.1: Introducing a formal framework to describe elements of generalised basic process algebras with deadlock.

We shall model relations as functions from D into a two-element set $\mathsf{B} = \{\top, \bot\}$ of *Booleans*. A domain together with functions and relations we shall refer to as *data*. Figure 3.1 outlines the framework that was informally introduced above. We now turn to the technicalities.

3.1 Data

We assume that data are given as a two-sorted algebra.

Definition 3.3 A *data algebra* $\mathbf{D}$ is a two-sorted algebra that consists of

(i) an algebra $\langle \mathsf{D}, \mathcal{F} \rangle$: D is a set and $\mathcal{F}$ is a set of operations on D, i.e., each $F \in \mathcal{F}$ is a mapping

$$F : \mathsf{D}^n \to \mathsf{D} \text{ for some } n \in \omega;$$

(ii) the two-element Boolean algebra $\langle \mathsf{B}, \vee, \wedge, \neg, \top, \bot \rangle$, i.e., $\mathsf{B} = \{\bot, \top\}$ $(\bot \neq \top)$, $\neg$ is a unary operation on B defined by $\neg \bot = \top$ and $\neg \top = \bot$, and $\wedge$ and $\vee$ are binary operations on B defined by the following tables:

$\wedge$	$\bot$	$\top$
$\bot$	$\bot$	$\bot$
$\top$	$\bot$	$\top$

$\vee$	$\bot$	$\top$
$\bot$	$\bot$	$\top$
$\top$	$\top$	$\top$

(iii) a set $\mathcal{R}$ of operations from D to B, i.e., each $R \in \mathcal{R}$ is a mapping

$$R : \mathsf{D}^n \to \mathsf{B} \text{ for some } n \in \omega.$$

The set D we call the *domain* of $\mathbf{D}$; the elements of $\mathcal{F}$ we call *functions* and the elements of $\mathcal{R}$ we call *relations*.

Remark 3.4 Note that, by definition, a data algebra does not have operations from B to D. So it may be thought of as a *model* in the sense of first-order model theory (see, e.g., Chang and Keisler, 1990).

Example 3.5 The set R of real numbers gives rise to a data algebra

$$\mathbf{R} = \langle \mathsf{R}, +, \cdot, -, 0, 1, \leq \rangle$$

with domain R, real addition $(+)$ and real multiplication $(\cdot)$ as binary functions, the real numbers 0 and 1 as nullary functions, and a binary relation $\leq$ defined by

$$(\mathsf{r}_1 \leq \mathsf{r}_2) = \top \text{ if, and only if, } \mathsf{r}_1 \text{ is at most } \mathsf{r}_2.$$

Note that in the above example we have implicitly introduced a *symbolism*. For instance, we wrote the symbol "+" to denote a certain binary function on the real numbers (instead of writing the intended set of ordered triples), and we wrote the symbol "$\mathsf{0}$" for a certain real number (instead of, e.g., the left side of a Dedekind cut with the rational number 0 as least upper bound). Henceforth, we shall assume that every data algebra $\mathbf{D}$ comes with a fixed set of symbols that denote the functions and relations of $\mathbf{D}$, one for every function and one for every relation of $\mathbf{D}$. A symbol that corresponds to an n-ary function of $\mathbf{D}$, i.e., an operation from D^n into D, we shall call a *function symbol of arity n*. A symbol

that corresponds to an n-ary relation of $\mathbf{D}$, i.e., an operation from D^n into B, we shall call a *relation symbol of arity* n. The collection of all function symbols and relation symbols, together with their arities, we call the *language* of $\mathbf{D}$. For the rest of this thesis we assume that the language associated with $\mathbf{D}$ is countable.

Let us fix for the remainder of this thesis a countably infinite set X of *(data) variables*. With respect to the language of a data algebra $\mathbf{D}$, we now define two sets of expressions. The set $\mathcal{D}$ of *data expressions* associated with $\mathbf{D}$ consists of all terms built from the variables in X and the function symbols in the language of $\mathbf{D}$; i.e., $\mathcal{D}$ is generated by

$$d ::= \; x \;\mid\; \mathrm{f}(d, \ldots, d), \tag{3.2}$$

where x is a variable, f is a function symbol of arity n and $d, \ldots, d$ is a sequence of length n. The set $\mathcal{B}$ of *Boolean expressions* associated with $\mathbf{D}$ is generated by

$$b ::= \; \mathrm{r}(d_1, \ldots, d_n) \;\mid\; \top \;\mid\; \bot \;\mid\; \neg\, b \;\mid\; b \vee b \;\mid\; b \wedge b, \tag{3.3}$$

where r is a relation symbol of arity n and $d_1, \ldots, d_n$ are data expressions. (On the very few occasions that we actually write data expressions —this will mainly be in examples— we adopt the standard notational conventions; e.g., whenever appropriate we use infix notation for binary function symbols or binary relation symbols and we leave out parentheses if this does not lead to confusion.)

A *valuation* is a mapping from the set of variables X into the domain D of a data algebra $\mathbf{D}$. Let us fix a valuation $\nu : X \to \mathsf{D}$; we denote by $\bar{\nu}$ its unique extension to a homomorphism from the two-sorted algebra of data and Boolean expressions into $\mathbf{D}$. That is, $\bar{\nu}$ associates with every data expression an element of D such that

$$\begin{aligned}
&\bar{\nu}(x) = \nu(x) \text{ and} \\
&\bar{\nu}(\mathrm{f}(d_1, \ldots, d_n)) = F(\bar{\nu}(d_1), \ldots, \bar{\nu}(d_n)),
\end{aligned}$$

where F is the n-ary function of $\mathbf{D}$ denoted by the function symbol f. Furthermore, $\bar{\nu}$ associates with every Boolean expression an element of B such that

$$\begin{aligned}
&\bar{\nu}(\top) = \top, \\
&\bar{\nu}(\bot) = \bot, \\
&\bar{\nu}(\neg\, b) = \neg\, \bar{\nu}(b), \\
&\bar{\nu}(b \wedge c) = \bar{\nu}(b) \wedge \bar{\nu}(c), \\
&\bar{\nu}(b \vee c) = \bar{\nu}(b) \vee \bar{\nu}(c) \text{ and} \\
&\bar{\nu}(\mathrm{r}(d_1, \ldots, d_n)) = R(\bar{\nu}(d_1), \ldots, \bar{\nu}(d_n)),
\end{aligned}$$

where R is the n-ary relation of $\mathbf{D}$ denoted by the relation symbol r.

Remark 3.6 Note that we are using "$\bot$", "$\top$", "$\neg$", "$\wedge$" and "$\vee$" both to refer to semantic objects (viz., to elements and operations of the Boolean algebra that is contained in a data algebra) and to syntactic objects (viz., to symbols that occur in Boolean expressions).

A *data equation* is a formula of the form $d \approx e$, where d and e are data expressions; if $\bar{\nu}(d) = \bar{\nu}(e)$, then we say that ν *satisfies* $d \approx e$ in $\mathbf{D}$ (notation: $\mathbf{D}, \nu \models d \approx e$); and if every valuation satisfies $d \approx e$, then we say that $d \approx e$ is *valid* in $\mathbf{D}$ (notation: $\mathbf{D} \models p \approx q$). Likewise, a *Boolean equation* is a formula of the form $b \approx c$, where b and c are Boolean expressions; if $\bar{\nu}(b) = \bar{\nu}(c)$, then we say that ν *satisfies* $b \approx c$ in $\mathbf{D}$ (notation: $\mathbf{D}, \nu \models b \approx c$); and if every valuation satisfies $b \approx c$, then we say that $b \approx c$ is *valid* in $\mathbf{D}$ (notation: $\mathbf{D} \models b \approx c$).

3.2 The language pCRL

Now, suppose that $\mathcal{A}$ is a nonempty countable set of *parametrised action symbols* with fixed arities. The set $\mathcal{P}$ of pCRL *expressions* is generated by the following grammar:

$$p ::= \mathrm{a}(d_1, \ldots, d_n) \mid \delta \mid p + p \mid p \cdot p \mid p \lhd b \rhd p \mid \sum_x p \tag{3.4}$$

where a is a parametrised action symbol of arity n, $d_1, \ldots, d_n$ are data expressions, x is a variable and b is a Boolean expression.

Most of the time we shall write pq instead of $p \cdot q$. We assign syntactic precedence to the constructs according to the following order:

$$+ < \textstyle\sum_x < \lhd\, b \,\rhd < \cdot\ ,$$

i.e., $+$ binds weakest and $\cdot$ binds strongest. The construct $\lhd\, b \,\rhd$ is called a *conditional*, and the Boolean expression b is sometimes called its *condition*. The construct $\sum_x$ we shall call a *choice quantifier*; it binds the variable x in its argument. An occurrence of a variable x is *free* in a pCRL expression if it is not in the scope of a $\sum_x$; otherwise it is *bound*. The set of variables with a free occurrence in p we denote by $\mathsf{FV}(p)$. A pCRL expression without free variables is *closed*.

We need to exercise some prudence when applying substitutions to pCRL expressions. Suppose that d is substituted for x in p; then only the free occurrences of x should be replaced by d, and an occurrence of a variable y in d should not become bound by this replacement. A substitution $\sigma : X \to \mathcal{D}$ is *correct* for p if, for all $x \in \mathsf{FV}(p)$, no free occurrence of a variable y in $\sigma(x)$ is in the scope of a $\sum_y$ when x is replaced by $\sigma(x)$ in p. A substitution σ is extended to a partial mapping $\bar{\sigma}$ from expressions to expressions: $\bar{\sigma}$ is defined only for expressions for which σ is correct, and it distributes over all the constructs of the language, except that

$$\bar{\sigma}(\textstyle\sum_x p) = \sum_x \bar{\sigma}'(p), \text{ where } \sigma'(y) = \begin{cases} y & \text{if } y = x; \text{ and} \\ \sigma(y) & \text{otherwise.} \end{cases}$$

Let $\vec{x} = x_1, \ldots, x_n$ be a sequence of variables, and let $\vec{d} = d_1, \ldots, d_n$ be a sequence of data expressions. If σ is a correct substitution for p that is the identity on all variables, except that $\sigma(x_i) = d_i$ for all $i = 1, \ldots, n$, then, we shall frequently write $p[\vec{x} := \vec{d}]$ to designate $\bar{\sigma}(p)$. Moreover, if p designates a pCRL expression, then by writing $p[\vec{x} := \vec{d}]$ we shall always mean the pCRL expression obtained from p in the manner just described; in particular, it will be tacitly assumed that the involved substitution is correct.

Suppose that p is a pCRL expression with a subexpression of the form $\sum_x p'$; then we may replace this subexpression by $\sum_y p'[x := y]$, where $y \notin \mathsf{FV}(p')$; p and q are *α-congruent* if q can be obtained from p by a series of replacements of this kind. Although a substitution σ may not be correct for p, there is always an element in $[p]_\alpha = \{q \mid q \text{ is } \alpha\text{-congruent with } p\}$ for which σ is correct. Moreover, if σ is correct for both p and q and $[p]_\alpha = [q]_\alpha$, then also $[\bar{\sigma}(p)]_\alpha = [\bar{\sigma}(q)]_\alpha$. Hence, there exists a unique total mapping on α-congruence classes such that $[p]_\alpha \mapsto [\bar{\sigma}(p)]_\alpha$; let us denote it by $\bar{\sigma}/\alpha$. In general, a partial mapping f on expressions induces a unique total mapping f/α on α-congruence classes of expressions such that $[p]_\alpha \mapsto [f(p)]_\alpha$, provided that

1. for every p there exists an α-congruent q for which f is defined; and
2. if f is defined for α-congruent p and q, then $f(p) = f(q)$.

In the remainder, we shall leave the proof that there exists a unique mapping f/α to the reader, and we shall adopt the following convention (similar to the 'variable convention' of the λ-calculus (Barendregt, 1984)).

Convention 3.7 We identify expressions and their respective congruence classes; i.e., we use p also to denote the set $[p]_\alpha$. Whenever we define a partial mapping f on expressions that gives rise to a unique total mapping f/α on α-congruence classes of expressions, we identify f and f/α; i.e., we use f also to denote f/α.

The syntax of pCRL suggests a correspondence with the operations of generalised basic process algebras with deadlock. When we use pCRL expressions to denote elements of a generalised basic process algebra with deadlock, then we want that $p + q$ denotes the alternative composition of the elements denoted by p and q, that $p \cdot q$ denotes their sequential composition, and that the pCRL expression δ refers to deadlock. If we want to make a similar remark about the correspondence between choice quantification and generalised summation, then we need to fix a domain of values for the variables.

Example 3.8 Suppose that variables range over the set R of real numbers. According to our remarks at the beginning of this chapter, with the expression $\sum_x \mathrm{in}(x)$ we mean the process that inputs an arbitrary real number. This is the infinite sum

$$\sum\{\mathrm{in}(\mathsf{r}) \mid \mathsf{r} \in \mathsf{R}\}$$

in a generalised basic process algebra with deadlock with for every real number r a process that is denoted with the action name $\mathrm{in}(\mathsf{r})$ and that models the action of inputting r. There may not be a pCRL expression to denote the process $\mathrm{in}(\mathsf{r})$; e.g., with respect to the language of $\mathbf{R}$ in Example 3.5, $\sqrt{2}$ is not a data expression, and hence $\mathrm{in}(\sqrt{2})$ is not a pCRL expression. Also note that the set of pCRL expressions associated with R is countable, while the pCRL expression $\sum_x \mathrm{in}(x)$ refers to the least upper bound of a continuum of alternatives (there is an action $\mathrm{in}(\mathsf{r})$ for every real number $\mathsf{r} \in \mathsf{R}$).

Let us now fix a data algebra $\mathbf{D}$, and let us assume that variables range over the domain D of $\mathbf{D}$. The above example illustrates that, in general, the expression $\sum_x p$ does not refer to a generalised sum of pCRL expressions. Intuitively, it refers to $\sum\{p[x := \mathsf{d}] \mid \mathsf{d} \in \mathsf{D}\}$, where $p[x := \mathsf{d}]$ is obtained by replacing the free occurrences of x in p by the element $\mathsf{d} \in \mathsf{D}$. To get a formalisation that reflects our intuition, we introduce expressions of the form $p[x := \mathsf{d}]$ as an auxiliary notion.

The set $\mathsf{Pol}_{\mathcal{D}}(\mathsf{D})$ of *data polynomials* is generated by

$$dpol ::= \; x \;\mid\; \mathsf{d} \;\mid\; \mathrm{f}(dpol, \ldots, dpol),$$

where x is a variable, d is an element of D, f is a function symbol of arity n and $dpol, \ldots, dpol$ is a sequence of length n (cf. (3.2)).

Example 3.9 With respect to the data algebra $\mathbf{R}$ of Example 3.5, a data expression $d(x_1, \ldots, x_n)$ is a polynomial in n indeterminates over R with natural coefficients, while a data polynomial $dpol(x_1, \ldots, x_n)$ is a polynomial in n indeterminates over R with real coefficients. Note that the set of data expressions associated with $\mathbf{R}$ is countable, whereas the set of data polynomials associated with $\mathbf{R}$ is uncountable.

The set $\mathsf{Pol}_{\mathcal{B}}(\mathsf{D})$ of *Boolean polynomials* is generated by the grammar in (3.3) by letting $d_1, \ldots, d_n$ range over data polynomials instead of over data expressions. The set $\mathsf{Pol}_{\mathcal{P}}(\mathsf{D})$ of pCRL *polynomials* is generated by the grammar in (3.4) by letting $d_1, \ldots, d_n$ range over data polynomials and b over Boolean polynomials. The set $\mathsf{Pol}_{\mathcal{P}}(\mathsf{D})$ is the universe of a generalised algebra similar to generalised basic process algebras with deadlock:

$$\mathbf{Pol}(\mathcal{A}, \mathbf{D}) = \langle \mathsf{Pol}_{\mathcal{P}}(\mathsf{D}), +, \cdot, \delta, \textstyle\sum \rangle;$$

a set $P \subseteq \mathsf{Pol}_{\mathcal{P}}(\mathsf{D})$ is admissible for $\sum$ in $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ if there exists a pCRL polynomial p and a free variable x such that

$$P = \{p[x := \mathsf{d}] \mid \mathsf{d} \in \mathsf{D}\}, \tag{3.5}$$

and we define

$$\textstyle\sum P = \sum_x p.$$

Remark 3.10 Examples 3.8 and 3.9 illustrate why we have taken pCRL expression as the fundamental notion in our language and treat polynomial as auxiliary: we wish to reason about the least upper bound of a continuum of alternatives (e.g., the pCRL expression $\sum_x \mathrm{in}(x)$ refers to the least upper bound of a continuum of pCRL polynomials) without reverting to an uncountable language. In this way, the integration operation of real time process algebra (Baeten and Bergstra, 1991), which is used to specify that an action occurs somewhere within a time interval, is a special form of choice quantification.

Groote and Ponse (1995) require in their original definition of μCRL that data algebras are minimal (i.e., every element is denoted by a data expression), and

they let variables range over data expressions. Thus, they escape the introduction of polynomials, but at the same time exclude uncountable domains as data. Consequently, the integration operation is not a special instance of their choice quantifier. In the timed version of μCRL of Groote *et al.* (2000) it is no longer required that the data algebra is minimal.

Remark 3.11 In μCRL, data is defined with a many-sorted algebraic specification, which must at least include the specification of a sort Bool (a Boolean algebra). Furthermore, μCRL has choice quantification over every sort (including the sort Bool). In this thesis, we shall only consider two-sorted data algebras and we assume that choice quantification is not over the Booleans. This restriction is only to simplify notation; it is not essential for our results.

3.3 The semantics of pCRL

We are now going to establish an interpretation of pCRL expressions as elements of a generalised basic process algebra with deadlock $\mathbf{P}$. A closed pCRL expression should denote a unique element of $\mathbf{P}$. In general, a pCRL expression may contain free variables, and then it should denote a unique element of $\mathbf{P}$ for every assignment of values to its free variables. We shall define the interpretation ι of pCRL expressions as elements of $\mathbf{P}$ as a family

$$\iota = \{\iota_\nu \mid \nu \text{ a valuation}\}$$

of mappings that interpret each pCRL expression p as an element $\iota_\nu(p)$ of $\mathbf{P}$. Clearly, the interpretation ι should reflect the relation that we have established between the syntax of pCRL and the operations of generalised basic process algebras with deadlock, so we require that each

$$\iota_\nu : \mathbf{Pol}(\mathcal{A}, \mathbf{D}) \to \mathbf{P}$$

is a homomorphism from the algebra of pCRL polynomials $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into $\mathbf{P}$; we call it the *interpretation homomorphism generated by* ν.

We began with the hypothesis that associated with every generalised basic process algebra with deadlock $\mathbf{P}$ is a set of action names A and a mapping $\mathsf{act} : \mathsf{A} \to \mathsf{P}$ that interprets action names as elements of $\mathbf{P}$. The action names in A, we have argued, should be thought of as having a particular structure (see (3.1)). We have, as we may now observe, assumed that A consists of a special kind of pCRL polynomial. The elements of A are of the form $\mathrm{a}(\mathsf{d}_1, \ldots, \mathsf{d}_n)$, with $\mathrm{a} \in \mathcal{A}$ and $\mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}$. Henceforth, we call such polynomials pCRL *actions*. The mapping

$$\mathsf{act} : \mathsf{A} \to \mathsf{P}$$

that interprets pCRL actions as elements of $\mathbf{P}$ we call the A-*interpretation* associated with $\mathbf{P}$. We require that each interpretation homomorphism ι_ν of pCRL polynomials into $\mathbf{P}$ extends the A-interpretation associated with $\mathbf{P}$.

In accordance with McKenzie *et al.* (1987), we denote by $\mathrm{Sg}(\mathsf{A})$ the subuniverse of $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ generated by A (i.e., $\mathrm{Sg}(\mathsf{A})$ is the least set that contains A and is

closed under the operations of generalised basic process algebras with deadlock); we define

$$\mathbf{Act}(\mathcal{A}, \mathbf{D}) = \langle \mathrm{Sg}(\mathsf{A}), +, \cdot, \delta, \textstyle\sum \rangle.$$

The A-interpretation associated with $\mathbf{P}$ does not necessarily extend to a homomorphism from $\mathbf{Act}(\mathcal{A}, \mathbf{D})$ to $\mathbf{P}$, since the image of a set of pCRL actions admissible in $\mathbf{Act}(\mathcal{A}, \mathbf{D})$ may not be admissible in $\mathbf{P}$. Let $\overline{\mathsf{act}} : \mathrm{Sg}(\mathsf{A}) \rightharpoonup \mathsf{P}$ be the maximal extension of act to a partial mapping from $\mathrm{Sg}(\mathsf{A})$ to P that respects the operations of generalised basic process algebras with deadlock. Since A generates $\mathrm{Sg}(\mathsf{A})$, $\overline{\mathsf{act}}$ is unique.

Definition 3.12 Let $\mathbf{P}$ be a generalised basic process algebra with deadlock with an associated A-interpretation act. We say that $\mathbf{P}$ is pCRL-*complete* with respect to act if the following closure condition holds for all pCRL polynomials $p(x)$ in one variable:

> if $\overline{\mathsf{act}}(p(\mathsf{d}))$ is defined for all $\mathsf{d} \in \mathsf{D}$, then the set $\{\overline{\mathsf{act}}(p(\mathsf{d})) \mid \mathsf{d} \in \mathsf{D}\}$ is admissible in $\mathbf{P}$.

Example 3.13 The algebra $\mathbf{T}_\kappa(\mathcal{L})$ of transition trees with branching degree $< \kappa$ is pCRL-complete under any interpretation of the pCRL actions, provided that the domain of $\mathbf{D}$ has cardinality $< \kappa$. For example, if $\mathbf{D}$ has a finite domain, then $\mathbf{T}_{\aleph_0}(\mathcal{L})$ is pCRL-complete; if $\mathbf{D}$ has a countably infinite domain, then $\mathbf{T}_{\aleph_1}(\mathcal{L})$ is pCRL-complete, but $\mathbf{T}_{\aleph_0}(\mathcal{L})$ is not.

Clearly, our requirement that ι_ν must be a homomorphism that extends act, can only be satisfied if $\mathbf{P}$ is pCRL-complete. On the other hand, if $\mathbf{P}$ is pCRL-complete, then act uniquely extends to a homomorphism

$$\overline{\mathsf{act}} : \mathbf{Act}(\mathcal{A}, \mathbf{D}) \to \mathbf{P}.$$

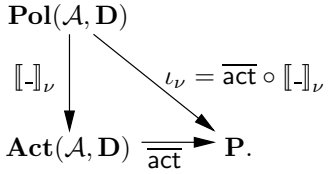
Now, to complete the definition of ι_ν, it suffices to explain how, given a valuation ν, arbitrary pCRL polynomials should be interpreted as elements of $\mathbf{Act}(\mathcal{A}, \mathbf{D})$. To this end, we associate with every valuation ν a particular homomorphism

$$[\![\_]\!]_\nu : \mathbf{Pol}(\mathcal{A}, \mathbf{D}) \to \mathbf{Act}(\mathcal{A}, \mathbf{D}).$$

Henceforth, let $\bar{\nu}$ denote the extension of ν to a homomorphism from the two-sorted algebra of data and Boolean polynomials into $\mathbf{D}$ (this is an extension of our earlier definition of $\bar{\nu}$, given on p. 32); we define $[\![\_]\!]_\nu$ as follows:

$$[\![\mathrm{a}(dpol_1, \ldots, dpol_n)]\!]_\nu = \mathrm{a}(\bar{\nu}(dpol_1), \ldots, \bar{\nu}(dpol_n));$$
$$[\![\delta]\!]_\nu = \delta;$$
$$[\![p + q]\!]_\nu = [\![p]\!]_\nu + [\![q]\!]_\nu;$$
$$[\![p \cdot q]\!]_\nu = [\![p]\!]_\nu \cdot [\![q]\!]_\nu;$$
$$[\![p \lhd bpol \rhd q]\!]_\nu = \begin{cases} [\![p]\!]_\nu & \text{if } \bar{\nu}(bpol) = \top \\ [\![q]\!]_\nu & \text{if } \bar{\nu}(bpol) = \bot; \text{ and} \end{cases}$$
$$[\![\textstyle\sum_x p]\!]_\nu = \textstyle\sum \{[\![p[x := \mathsf{d}]]\!]_\nu \mid \mathsf{d} \in \mathsf{D}\}.$$

The homomorphic image of $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ under $[\![\_]\!]_\nu$ is the subalgebra of $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ generated by A. We define the interpretation homomorphism ι_ν generated by the valuation ν as the composition of $\overline{\mathsf{act}}$ and $[\![\_]\!]_\nu$:



A pCRL *equation* is a formula of the form $p \approx q$, where p and q are pCRL expressions. If $\iota_\nu(p) = \iota_\nu(q)$, then we say that ν *satisfies* $p \approx q$ in $\mathbf{P}$ (notation: $\mathbf{P}, \nu \models p \approx q$). If every valuation satisfies $p \approx q$ in $\mathbf{P}$, then we say that $p \approx q$ is *valid* in $\mathbf{P}$, and we write $\mathbf{P} \models p \approx q$. A pCRL *summand inclusion* is a formal expression of the form $p \preccurlyeq q$, where p and q are pCRL expressions. If $\iota_\nu(p) \leq \iota_\nu(q)$, then we say that ν *satisfies* $p \preccurlyeq q$ in $\mathbf{P}$ (notation: $\mathbf{P}, \nu \models p \preccurlyeq q$). If every valuation satisfies $p \preccurlyeq q$ in $\mathbf{P}$, then we say that $p \preccurlyeq q$ is *valid* in $\mathbf{P}$, and we write $\mathbf{P} \models p \preccurlyeq q$. Note that it follows from the definition of $\leq$ on p. 18 that

$$\mathbf{P}, \nu \models p \preccurlyeq q \text{ if, and only if, } \mathbf{P}, \nu \models q \approx q + p.$$

3.4 pCRL trees

Consider the algebra $\mathbf{T}_\kappa(\mathcal{L})$ with an injective A-interpretation

$$\mathsf{act} : \mathsf{A} \to \mathsf{T}_0 = \{\{\ell\} \mid \ell \in \mathcal{L}\}$$

that associates with every pCRL action a unique tree action, and suppose that the domain of $\mathbf{D}$ has cardinality $< \kappa$. The homomorphism

$$\overline{\mathsf{act}} : \mathbf{Act}(\mathcal{A}, \mathbf{D}) \to \mathbf{T}_\kappa(\mathcal{L})$$

induced by this A-interpretation allows us to picture certain closed pCRL polynomials as transition trees with actions as labels.

Example 3.14 If we take as data the additive group of integers ordered by $\leq$, then the pCRL expression

$$\textstyle\sum_x \mathrm{r}(x)\mathrm{s}(x) \lhd 0 \leq x \rhd \mathrm{r}(x)\mathrm{s}(-x),$$

may be pictured as the tree in Figure 3.2.

Their interpretation as transition trees induces an equivalence on the pCRL polynomials. We apply a standard technique in universal algebra to construct from $\mathbf{Act}(\mathcal{A}, \mathbf{D})$ a generalised basic process algebra with deadlock, with as universe the set of pCRL polynomials modulo this equivalence. First, we need to generalise the notion of congruence. Suppose that ϑ is a congruence of an algebra $\langle \mathsf{P}, +, \cdot, \delta \rangle$ similar to basic process algebras with deadlock. As usual, with p/ϑ we shall denote

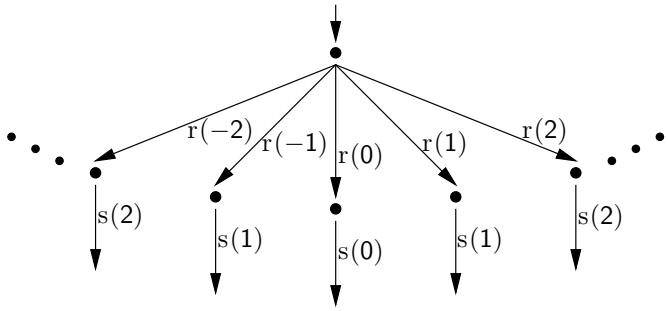


Figure 3.2: The transition tree associated with the expression of Example 3.14.

the congruence class with respect to ϑ that contains p, i.e., $\mathsf{p}/\vartheta = \{\mathsf{q} \mid \langle \mathsf{q}, \mathsf{p}\rangle \in \vartheta\}$, and if $\mathsf{P}' \subseteq \mathsf{P}$, then

$$\mathsf{P}'/\vartheta = \{\mathsf{p}/\vartheta \mid \mathsf{p} \in \mathsf{P}'\}.$$

The relation ϑ is a *congruence* of the algebra $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \sum\rangle$ similar to generalised basic process algebras with deadlock if it is a congruence of $\langle \mathsf{P}, +, \cdot, \delta\rangle$ and it satisfies the following substitution property with respect to $\sum$:

$$\text{if } \mathsf{P}', \mathsf{P}'' \subseteq \mathsf{P} \text{ are admissible for } \textstyle\sum \text{ and } \mathsf{P}'/\vartheta = \mathsf{P}''/\vartheta, \text{ then } \langle \textstyle\sum \mathsf{P}', \sum \mathsf{P}''\rangle \in \vartheta.$$

If ϑ is a congruence of $\mathbf{P}$, then we may define on P/ϑ the operations $+$, $\cdot$, and δ as usual, and we may also define a generalised operation $\sum$ by

$$\textstyle\sum(\mathsf{P}'/\vartheta) = (\sum \mathsf{P}')/\vartheta \quad (\mathsf{P}'/\vartheta \text{ is admissible if } \mathsf{P}' \text{ is admissible for } \sum \text{ in } \mathbf{P});$$

we get a *generalised quotient algebra* $\mathbf{P}/\vartheta = \langle \mathsf{P}/\vartheta, +, \cdot, \delta, \sum\rangle$.

Now, consider the homomorphism $\overline{\mathsf{act}} : \mathbf{Act}(\mathcal{A}, \mathbf{D}) \to \mathbf{T}_\kappa(\mathcal{L})$, induced by the bijection act. The *kernel* of this homomorphism is the relation

$$\vartheta = \{\langle p, q\rangle \subseteq \mathrm{Sg}(\mathsf{A}) \times \mathrm{Sg}(\mathsf{A}) \mid \overline{\mathsf{act}}(p) = \overline{\mathsf{act}}(q)\};$$

it is a congruence on $\mathbf{Act}(\mathcal{A}, \mathbf{D})$.[1] We denote the generalised quotient algebra by $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$, i.e.,

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}) = \mathbf{Act}(\mathcal{A}, \mathbf{D})/\vartheta = \langle \mathrm{Sg}(\mathsf{A})/\vartheta, +, \cdot, \delta, \textstyle\sum\rangle.$$

Clearly, $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ is a generalised basic process algebra with deadlock, and we associate with it an A-interpretation defined by

$$\mathrm{a}(\mathsf{d}_1, \ldots, \mathsf{d}_n) \mapsto \mathrm{a}(\mathsf{d}_1, \ldots, \mathsf{d}_n)/\vartheta.$$

With respect to this A-interpretation $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ is pCRL-complete. An element of $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ we call a pCRL *tree*.

[1] We use a generalised version of the Homomorphism Theorem (see (McKenzie *et al.*, 1987, p.28) or (Burris and Sankappanavar, 1981, p.46)); the generalisation is straightforward.

Recall that our definition of the interpretation homomorphisms ι_ν hinges on a presupposed interpretation act of pCRL actions as elements of $\mathbf{P}$. Moreover, $\mathbf{P}$ should be pCRL-complete with respect to act. The set of pCRL actions and the associated definition of pCRL-completeness are relative to a particular choice of $\mathcal{A}$ and $\mathbf{D}$. Henceforth, we shall denote by $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ the class of all suitable combinations of a generalised basic process algebra with deadlock $\mathbf{P}$ and an interpretation of the pCRL actions, given the specific instance of pCRL with $\mathcal{A}$ and $\mathbf{D}$. Formally, $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ consists of all pairs $\langle \mathbf{P}, \mathsf{act}\rangle$ of a generalised basic process algebra with deadlock $\mathbf{P}$ and an interpretation act of pCRL actions as elements of $\mathbf{P}$ such that $\mathbf{P}$ is pCRL complete. Par abus de language, if $\langle \mathbf{P}, \mathsf{act}\rangle \in \mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$, then we shall often just say that $\mathbf{P}$ is in $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$, leaving act implicit.

Let $\mathbf{P}$ and $\mathbf{Q}$ be generalised basic process algebras with deadlock, and with A-interpretations $\mathsf{act}_\mathbf{P}$ and $\mathsf{act}_\mathbf{Q}$, respectively; a homomorphism $h : \mathbf{P} \to \mathbf{Q}$ is said to *preserve* A if $\mathsf{act}_\mathbf{Q} = h \circ \mathsf{act}_\mathbf{P}$. A generalised basic process algebra with deadlock $\mathbf{P}$ together with an associated A-interpretation is *initial* for $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ if for every element $\mathbf{Q}$ of $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ there is a unique A-preserving homomorphism $h : \mathbf{P} \to \mathbf{Q}$.

Theorem 3.15 The algebra $\mathbf{T}_\mathbf{D}(\mathcal{A})$ is initial in $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$.

Proof. Consider a generalised basic process algebra with deadlock $\mathbf{T}_\kappa(\mathcal{L})$ with $|\mathsf{D}| < \kappa$ and a bijective A-interpretation $\mathsf{act} : \mathsf{A} \to \mathsf{T}_0$. Within $\mathbf{T}_\kappa(\mathcal{L})$ we find an isomorphic copy of $\mathbf{T}_\mathbf{D}(\mathcal{A})$; it is the subalgebra generated by T_0 of the algebra that is obtained by restricting the admissible sets of $\mathbf{T}_\kappa(\mathcal{L})$ to those denoted by a pCRL polynomial with one free variable (see (3.5)). So, that $\mathbf{T}_\mathbf{D}(\mathcal{A})$ is an element of $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ is immediate. That $\mathbf{T}_\mathbf{D}(\mathcal{A})$ is initial for $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ can be seen as follows.

There is a unique f from T_0 into the universe of a generalised basic process algebra with deadlock $\mathbf{P}$ with A-interpretation act' such that $\mathsf{act}' = f \circ \mathsf{act}$. By a straightforward adaptation of our proof of Theorem 2.11 we get that f extends to an A-preserving homomorphism h from the isomorphic copy of $\mathbf{T}_\mathbf{D}(\mathcal{A})$ in $\mathbf{T}_\kappa(\mathcal{L})$ to $\mathbf{P}$, provided that $\mathbf{P}$ is pCRL-complete with respect to act'. Clearly, h is unique. Hence $\mathbf{T}_\mathbf{D}(\mathcal{A})$ is initial for $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$. □

Let us write $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}), \nu \models p \approx q$ if $\mathbf{P}, \nu \models p \approx q$ for all $\mathbf{P}$ in $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$. Then, Theorem 3.15 has the following corollary.

Corollary 3.16 For all pCRL expressions p and q,

$$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}), \nu \models p \approx q \text{ if, and only if, } \mathbf{T}_\mathbf{D}(\mathcal{A}), \nu \models p \approx q.$$

Proof. The implication from left to right is immediate; we prove the implication from right to left. Let $\mathbf{P}$ be an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$. By Theorem 3.15 there exists a unique homomorphism $h : \mathbf{T}_\mathbf{D}(\mathcal{A}) \to \mathbf{P}$ that preserves A. So, if we denote by act the A-interpretation associated with $\mathbf{T}_\mathbf{D}(\mathcal{A})$, then $\mathsf{act}_\mathbf{P} = h \circ \mathsf{act}$ is the A-interpretation associated with $\mathbf{P}$. The mapping act extends uniquely to a homomorphism $\overline{\mathsf{act}} : \mathbf{Act}(\mathcal{A},\mathbf{D}) \to \mathbf{T}_\mathbf{D}(\mathcal{A})$, so

$$h \circ \overline{\mathsf{act}} : \mathbf{Act}(\mathcal{A},\mathbf{D}) \to \mathbf{P}$$

is a homomorphism. Clearly, the mapping $h \circ \overline{\mathsf{act}}$ extends $h \circ \mathsf{act}$, so $\overline{\mathsf{act}_{\mathbf{P}}} = h \circ \overline{\mathsf{act}}$. From $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q$ it follows that

$$\overline{\mathsf{act}_{\mathbf{P}}}(\llbracket p \rrbracket_\nu) = h(\overline{\mathsf{act}}(\llbracket p \rrbracket_\nu)) = h(\overline{\mathsf{act}}(\llbracket q \rrbracket_\nu)) = \overline{\mathsf{act}_{\mathbf{P}}}(\llbracket q \rrbracket_\nu),$$

and hence $\mathbf{P}, \nu \models p \approx q$. This concludes the proof of the implication from right to left and of the corollary. □

3.5 Tree forms

We shall now associate with every pCRL expression an equivalent pCRL expression in a certain special form, with a close resemblance to the transition tree it describes.

An *action expression* is a pCRL expression of the form $\mathrm{a}(d_1, \ldots, d_n)$, where a is an n-ary parametrised action symbol and $d_1, \ldots, d_n$ is a sequence of data expressions. By a *simple* pCRL expression we shall understand an expression of the form

$$\textstyle\sum_{\vec{x}} a \lhd b \rhd \delta \text{ or of the form } \sum_{\vec{x}} ap \lhd b \rhd \delta, \tag{3.6}$$

where $\sum_{\vec{x}}$ abbreviates the sequence $\sum_{x_1} \cdots \sum_{x_n}$ for a sequence $\vec{x} = x_1, \ldots, x_n$ of variables, a is an action expression, b is a Boolean expression and p is a pCRL expression. If in (3.6) the sequence $\vec{x}$ is empty, then the simple pCRL expression has no leading choice quantifiers. We call p the *continuation* of the simple pCRL expression $\sum_{\vec{x}} ap \lhd b \rhd \delta$.

Definition 3.17 The set $\mathcal{T}$ of *tree forms* is generated by

$$t ::= \ \delta \ \mid \ \textstyle\sum_{\vec{x}} a \lhd b \rhd \delta \ \mid \ \sum_{\vec{x}} at \lhd b \rhd \delta \ \mid \ t + t,$$

where a is an action expression, b is a Boolean expression, and $\vec{x}$ is a (possibly empty) sequence of variables.

Example 3.18 With the pCRL expression of Example 3.2 we may associate the tree form

$$\textstyle\sum_x \mathrm{r}_2(x)\mathrm{s}_2(1) \lhd \mathrm{V}(x) \rhd \delta + \sum_x \mathrm{r}_2(x)\mathrm{s}_2(0) \lhd \neg\, \mathrm{V}(x) \rhd \delta.$$

With the pCRL expression of Example 3.14 we may associate the tree form

$$\textstyle\sum_x \mathrm{r}(x)\mathrm{s}(x) \lhd 0 \leq x \rhd \delta + \sum_x \mathrm{r}(x)\mathrm{s}(-x) \lhd \neg(0 \leq x) \rhd \delta;$$

the first simple expression describes the right half of the transition tree in Figure 3.2 and that the second simple expression describes the left half.

Below, we shall define a function $\theta : \mathcal{P} \to \mathcal{T}$ that associates with every pCRL expression p an equivalent tree form $\theta(p)$. First, we give the definitions of three auxiliary functions:

The function $\theta_{\mathrm{seq}} : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$ is recursively defined by

$$\begin{aligned}
&\theta_{\mathrm{seq}}(\delta, t) = \delta;\\
&\theta_{\mathrm{seq}}(\textstyle\sum_{\vec{x}} a \lhd b \rhd \delta, t) = \textstyle\sum_{\vec{x}} a \cdot t \lhd b \rhd \delta && (\{\vec{x}\} \cap \mathsf{FV}(t) = \emptyset);\\
&\theta_{\mathrm{seq}}(\textstyle\sum_{\vec{x}} a \cdot t \lhd b \rhd \delta, u) = \textstyle\sum_{\vec{x}} a \cdot \theta_{\mathrm{seq}}(t, u) \lhd b \rhd \delta && (\{\vec{x}\} \cap \mathsf{FV}(u) = \emptyset);\\
&\theta_{\mathrm{seq}}(t + u, v) = \theta_{\mathrm{seq}}(t, v) + \theta_{\mathrm{seq}}(u, v).
\end{aligned}$$

Suppose t and u are tree forms; $\theta_{\mathrm{seq}}(t, u)$ is defined provided that the bound variables in t are distinct from the free variables in u. The function θ_{seq} induces a total function on α-congruence classes of tree forms which is by Convention 3.7 also denoted by θ_{seq}.

Lemma 3.19 $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models \theta_{\mathrm{seq}}(t, u) \approx t \cdot u$.

Proof. Without loss of generality, we may assume that the bound variables in t are distinct from the free variables in u. Our proof is by induction on the structure of t. Let ν be an arbitrary valuation, and let ι_ν be interpretation homomorphism generated by ν from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$; we show that $\iota_\nu(\theta_{\mathrm{seq}}(t, u)) = \iota_\nu(t \cdot u)$.
If $t = \delta$, then, with an application of (A7), $\iota_\nu(\theta_{\mathrm{seq}}(t, u)) = \delta = \delta \cdot \iota_\nu(u) = \iota_\nu(t \cdot u)$.
Suppose $t = \sum_{\vec{x}} a \lhd b \rhd \delta$, with $\vec{x} = x_1, \ldots, x_n$. By our assumption on the variables in t and u, $\{\vec{x}\} \cap \mathsf{FV}(u) = \emptyset$, so $u[\vec{x} := \vec{\mathsf{d}}] = u$ for all $\vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}$. Hence, by (GA3)

$$\begin{aligned}
&\iota_\nu(\theta_{\mathrm{seq}}(t, u))\\
&\quad = \textstyle\sum\{\iota_\nu(a \cdot u[\vec{x} := \vec{\mathsf{d}}]) \mid \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ s.t. } \bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) = \top\}\\
&\quad = \textstyle\sum\{\iota_\nu(a[\vec{x} := \vec{\mathsf{d}}]) \mid \ \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ s.t. } \bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) = \top\} \cdot \iota_\nu(u)\\
&\quad = \iota_\nu(t \cdot u).
\end{aligned}$$

Suppose $t = \sum_{\vec{x}} a \cdot t' \lhd b \rhd \delta$, with $\vec{x} = x_1, \ldots, x_n$. By the induction hypothesis we get that, for all $\vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}$,

$$\iota_\nu(\theta_{\mathrm{seq}}(t', u)[\vec{x} := \vec{\mathsf{d}}]) = \iota_\nu(t'[\vec{x} := \vec{\mathsf{d}}]) \cdot \iota_\nu(u[\vec{x} := \vec{\mathsf{d}}])$$

Hence, since our assumption on the variables in t and u implies $u[\vec{x} := \vec{\mathsf{d}}] = u$,

$$\iota_\nu(\theta_{\mathrm{seq}}(t', u)[\vec{x} := \vec{\mathsf{d}}]) = \iota_\nu(t'[\vec{x} := \vec{\mathsf{d}}]) \cdot \iota_\nu(u).$$

We now obtain by (A5) and (GA3) that

$$\begin{aligned}
&\iota_\nu(\theta_{\mathrm{seq}}(t, u))\\
&\quad = \textstyle\sum\{\iota_\nu(a[\vec{x} := \vec{\mathsf{d}}]) \cdot \iota_\nu(\theta_{\mathrm{seq}}(t', u)[\vec{x} := \vec{\mathsf{d}}]) \mid\\
&\qquad\qquad \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ s.t. } \bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) = \top\}\\
&\quad = \textstyle\sum\{\iota_\nu(a[\vec{x} := \vec{\mathsf{d}}]) \cdot \iota_\nu(t'[\vec{x} := \vec{\mathsf{d}}]) \mid\\
&\qquad\qquad \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ s.t. } \bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) = \top\} \cdot \iota_\nu(u)\\
&\quad = \iota_\nu(t \cdot u).
\end{aligned}$$

If $t = t' + t''$, then by the induction hypothesis and (A4)

$$\begin{aligned}\iota_\nu(\theta_{\mathrm{seq}}(t,u)) &= \iota_\nu(\theta_{\mathrm{seq}}(t',u)) + \iota_\nu(\theta_{\mathrm{seq}}(t'',u))\\ &= \iota_\nu(t'\cdot u) + \iota_\nu(t''\cdot u)\\ &= \iota_\nu(t'+t'')\cdot\iota_\nu(u)\\ &= \iota_\nu(t\cdot u).\end{aligned}$$

The proof of the lemma is complete. □

The function $\theta_{\mathrm{cnd}} : \mathcal{T}\times\mathcal{B}\to\mathcal{T}$ is recursively defined by

$$\begin{aligned}&\theta_{\mathrm{cnd}}(\delta,b) = \delta;\\ &\theta_{\mathrm{cnd}}(\textstyle\sum_{\vec{x}} a\lhd c\rhd\delta, b) = \sum_{\vec{x}} a\lhd b\wedge c\rhd\delta && (\{\vec{x}\}\cap\mathsf{FV}(b)=\emptyset);\\ &\theta_{\mathrm{cnd}}(\textstyle\sum_{\vec{x}} a\cdot t\lhd c\rhd\delta, b) = \sum_{\vec{x}} a\cdot t\lhd b\wedge c\rhd\delta && (\{\vec{x}\}\cap\mathsf{FV}(b)=\emptyset);\\ &\theta_{\mathrm{cnd}}(t+u,b) = \theta_{\mathrm{cnd}}(t,b)+\theta_{\mathrm{cnd}}(u,b).\end{aligned}$$

Suppose t is a tree form and b is a boolean expression; $\theta_{\mathrm{cnd}}(t,b)$ is defined provided that the bound variables in t are distinct from the (free) variables in b. The function θ_{cnd} induces a total function on α-congruence classes of tree forms which is by Convention 3.7 also denoted by θ_{cnd}.

Lemma 3.20 $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models \theta_{\mathrm{cnd}}(t,b) \approx t\lhd b\rhd\delta$.

Proof. Without loss of generality we may assume that the bound variables in t do not occur in b. Our proof is by induction on the structure of t. Let ν be an arbitrary valuation, and let ι_ν be the interpretation homomorphism generated by ν from $\mathbf{Pol}(\mathcal{A},\mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$; we show that $\iota_\nu(\theta_{\mathrm{cnd}}(t,b)) = \iota_\nu(t\lhd b\rhd\delta)$.
If $t=\delta$, then $\iota_\nu(\theta_{\mathrm{cnd}}(t,b)) = \delta = \iota_\nu(\delta\lhd b\rhd\delta) = \iota_\nu(t\lhd b\rhd\delta)$.
Suppose that $t = \sum_{\vec{x}} t^*\lhd c\rhd\delta$, with $\vec{x} = x_1,\ldots,x_n$. By our assumption on the variables in t and b, $\{\vec{x}\}\cap\mathsf{FV}(b)=\emptyset$, so $\bar\nu(b[\vec{x}:=\vec{\mathsf{d}}]) = \bar\nu(b)$ for all sequences $\vec{\mathsf{d}} = \mathsf{d}_1,\ldots,\mathsf{d}_n\in\mathsf{D}$; there are two cases:

1.If $\bar\nu(b[\vec{x}:=\vec{\mathsf{d}}]) = \bar\nu(b) = \top$, then

$$\begin{aligned}&\iota_\nu(\theta_{\mathrm{cnd}}(t,b))\\ &\quad= \sum\{\iota_\nu(t^*[\vec{x}:=\vec{\mathsf{d}}]) \mid\\ &\qquad\qquad \vec{\mathsf{d}} = \mathsf{d}_1,\ldots,\mathsf{d}_n \text{ s.t. } \bar\nu(b[\vec{x}:=\vec{\mathsf{d}}])\wedge\bar\nu(c[\vec{x}:=\vec{\mathsf{d}}]) = \top\}\\ &\quad= \sum\{\iota_\nu(t^*[\vec{x}:=\vec{\mathsf{d}}]) \mid \vec{\mathsf{d}} = \mathsf{d}_1,\ldots,\mathsf{d}_n \text{ s.t. } \bar\nu(c[\vec{x}:=\vec{\mathsf{d}}]) = \top\}\\ &\quad= \iota_\nu(t)\end{aligned}$$

and $\iota_\nu(t\lhd b\rhd\delta) = \iota_\nu(t)$, so $\iota_\nu(\theta_{\mathrm{cnd}}(t,b)) = \iota_\nu(t\lhd b\rhd\delta)$.

2.If $\bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) = \bar{\nu}(b) = \bot$, then, with applications of (A6) and (GA2),

$$\begin{aligned}
&\iota_\nu(\theta_{\mathrm{cnd}}(t, b)) \\
&\quad = \textstyle\sum\{\iota_\nu(t^*[\vec{x} := \vec{\mathsf{d}}]) \mid \\
&\qquad\qquad \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \text{ s.t. } \bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) \wedge \bar{\nu}(c[\vec{x} := \vec{\mathsf{d}}]) = \top\} \\
&\quad = \textstyle\sum \emptyset = \delta
\end{aligned}$$

and also $\iota_\nu(t \lhd b \rhd \delta) = \delta$, so $\iota_\nu(\theta_{\mathrm{cnd}}(t, b)) = \iota_\nu(t \lhd b \rhd \delta)$.
If $t = t' + t''$, then by the induction hypothesis

$$\begin{aligned}
\iota_\nu(\theta_{\mathrm{cnd}}(t, u)) &= \iota_\nu(\theta_{\mathrm{cnd}}(t', u)) + \iota_\nu(\theta_{\mathrm{cnd}}(t'', u)) \\
&= \iota_\nu(t' \lhd b \rhd \delta) + \iota_\nu(t'' \lhd b \rhd \delta).
\end{aligned}$$

So, if $\bar{\nu}(b) = \top$, then $\iota_\nu(\theta_{\mathrm{cnd}}(t, u)) = \iota_\nu(t' + t'') = \iota_\nu(t \lhd b \rhd \delta)$; and if $\bar{\nu}(b) = \bot$, then, with an application of (A3), $\iota_\nu(\theta_{\mathrm{cnd}}(t, u)) = \delta + \delta = \delta = \iota_\nu(t \lhd b \rhd \delta)$. □

The function $\theta_{\mathrm{sum}} : X \times \mathcal{T} \to \mathcal{T}$ is recursively defined by

$$\begin{aligned}
&\theta_{\mathrm{sum}}(x, \delta) = \delta; \\
&\theta_{\mathrm{sum}}(x, \textstyle\sum_{\vec{x}} a \lhd b \rhd \delta) = \sum_{x,\vec{x}} a \lhd b \rhd \delta; \\
&\theta_{\mathrm{sum}}(x, \textstyle\sum_{\vec{x}} a \cdot t \lhd b \rhd \delta) = \sum_{x,\vec{x}} a \cdot t \lhd b \rhd \delta; \text{ and} \\
&\theta_{\mathrm{sum}}(x, t + u) = \theta_{\mathrm{sum}}(x, t) + \theta_{\mathrm{sum}}(x, u).
\end{aligned}$$

Lemma 3.21 $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models \theta_{\mathrm{sum}}(x, t) \approx \sum_x t$.

Proof. Our proof is by induction on the structure of t.
Let ν be an arbitrary valuation, and let ι_ν be the interpretation homomorphism generated by ν from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$; we show that $\iota_\nu(\theta_{\mathrm{sum}}(x, t)) = \iota_\nu(\sum_x t)$.
If $t = \delta$, then, since $\delta[x := \mathsf{d}] = \delta$ and by (GA1) and (GA2)

$$\iota_\nu(\theta_{\mathrm{sum}}(x, t)) = \iota_\nu(\delta) = \delta = \textstyle\sum\{\iota_\nu(\delta[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} = \iota_\nu(\sum_x t).$$

If t is a simple expression, then $\theta_{\mathrm{sum}}(x, t) = \sum_x t$ by definition.
If $t = t' + t''$, then by the induction hypothesis

$$\begin{aligned}
\iota_\nu(\theta_{\mathrm{sum}}(x, t)) &= \iota_\nu(\textstyle\sum_x t') + \iota_\nu(\sum_x t'') \\
&= \textstyle\sum\{\iota_\nu(t'[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} + \sum\{\iota_\nu(t''[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\},
\end{aligned}$$

and

$$\iota_\nu(\textstyle\sum_x t) = \sum\{\iota_\nu(t'[x := \mathsf{d}]) + \iota_\nu(t''[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\}.$$

On the one hand, we get by (GA1) that

$$\iota_\nu(t'[x := \mathsf{d}]), \iota_\nu(t''[x := \mathsf{d}]) \leq \iota_\nu(\textstyle\sum_x t),$$

so by (GA2)

$$\iota_\nu(\textstyle\sum_x t') \leq \iota_\nu(\sum_x t) \text{ and } \iota_\nu(\sum_x t'') \leq \iota_\nu(\sum_x t);$$

hence $\iota_\nu(\theta_{\mathrm{sum}}(x,t)) \leq \iota_\nu(\sum_x t)$.
On the other hand, we get by (GA1) that

$$\iota_\nu(t'[x := \mathsf{d}]) \leq \iota_\nu(\textstyle\sum_x t') \leq \iota_\nu(\theta_{\mathrm{sum}}(x,t)),$$

and similarly,

$$\iota_\nu(t''[x := \mathsf{d}]) \leq \iota_\nu(\theta_{\mathrm{sum}}(x,t)),$$

so that

$$\iota_\nu(t'[x := \mathsf{d}]) + \iota_\nu(t''[x := \mathsf{d}]) \leq \iota_\nu(\theta_{\mathrm{sum}}(x,t));$$

hence, by (GA2), $\iota_\nu(\sum_x t) \leq \iota_\nu(\theta_{\mathrm{sum}}(x,t))$. □

Now, we define θ as follows:

$$\begin{aligned}
&\theta(\delta) = \delta;\\
&\theta(a) = a \lhd \top \rhd \delta;\\
&\theta(p + q) = \theta(p) + \theta(q);\\
&\theta(p \cdot q) = \theta_{\mathrm{seq}}(\theta(p), \theta(q));\\
&\theta(p \lhd b \rhd q) = \theta_{\mathrm{cnd}}(\theta(p), b) + \theta_{\mathrm{cnd}}(\theta(q), \neg b);\\
&\theta(\textstyle\sum_x p) = \theta_{\mathrm{sum}}(x, \theta(p)).
\end{aligned}$$

Lemma 3.22 (Tree forms) The function $\theta : \mathcal{P} \to \mathcal{T}$ associates with every pCRL expression p a tree form $\theta(p)$ such that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models \theta(p) \approx p$.

Proof. Clearly, $\theta(p)$ is a tree form for every pCRL expression p. To prove that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models p \approx \theta(p)$, we fix an arbitrary valuation ν and an interpretation homomorphism ι_ν generated by ν from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into some element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$, and we show that $\iota_\nu(p) = \iota_\nu(\theta(p))$ by structural induction.
If $p = \delta$, then $\iota_\nu(\theta(p)) = \iota_\nu(p)$ by definition.
If p is an action expression, then, since $\bar{\nu}(\top) = \top$,

$$\iota_\nu(\theta(p)) = \iota_\nu(p \lhd \top \rhd \delta) = \iota_\nu(p).$$

If $p = p' + p''$, then by the induction hypothesis

$$\iota_\nu(\theta(p)) = \iota_\nu(\theta(p')) + \iota_\nu(\theta(p'')) = \iota_\nu(p') + \iota_\nu(p'') = \iota_\nu(p).$$

If $p = p' \cdot p''$, then

$$\begin{aligned}
\iota_\nu(\theta(p)) &= \iota_\nu(\theta(p')) \cdot \iota_\nu(\theta(p'')) && \text{by Lemma 3.19}\\
&= \iota_\nu(p') \cdot \iota_\nu(p'') = \iota_\nu(p) && \text{by (IH).}
\end{aligned}$$

If $p = p' \lhd b \rhd p''$, then

$$\begin{aligned}\iota_\nu(\theta(p)) &= \iota_\nu(\theta(p') \lhd b \rhd \delta) + \iota_\nu(\theta(p'') \lhd \neg\, b \rhd \delta) && \text{by Lemma 3.20}\\ &= \iota_\nu(p' \lhd b \rhd \delta) + \iota_\nu(p'' \lhd \neg\, b \rhd \delta) && \text{by (IH).}\end{aligned}$$

We now distinguish cases: if $\bar{\nu}(b) = \top$, then $\iota_\nu(\theta(p)) = \iota_\nu(p') + \delta = \iota_\nu(p)$ by (A6); otherwise, if $\bar{\nu}(b) = \bot$, then by (A1) and (A6) $\iota_\nu(\theta(p)) = \delta + \iota_\nu(p'') = \iota_\nu(p)$. If $p = \sum_x p'$, then by Lemma 3.21 $\iota_\nu(\theta(p)) = \iota_\nu(\sum_x \theta(p'))$, and from the induction hypothesis we get $\iota_\nu(\theta(p')[x := \mathsf{d}]) = \iota_\nu(p'[x := \mathsf{d}])$ for all $\mathsf{d} \in \mathsf{D}$; hence,

$$\begin{aligned}\iota_\nu(\theta(p)) &= \textstyle\sum\{\iota_\nu(\theta(p')[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\}\\ &= \textstyle\sum\{\iota_\nu(p'[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} = \iota_\nu(\sum_x p').\end{aligned}$$

This completes the proof of the lemma. □

For technical purposes it is convenient to impose some extra restrictions on how a tree form is written down. Let t be a tree form; t is *ordered* if

$$t = t_1 + \cdots + t_m + t_{m+1} + \cdots + t_n, \tag{3.7}$$

where t_i is a simple tree form with an ordered continuation for all $1 \leq i \leq m$ and t_i is a simple tree form without continuation for all $m < i \leq n$. By convention, if $m = 0$ then $t = t_{m+1} + \cdots + t_n$; if $m = n$, then $t = t_1 + \cdots + t_m$; and if $m = n = 0$, then $t = \delta$. We denote the set of ordered tree forms by $\mathcal{T}_o$.

Modulo the commutativity and the associativity of $+$ and using that δ is a neutral element for $+$, any tree form can be written as an ordered tree form. Hence, θ is easily modified so that it yields only ordered tree forms; let θ_o be the recursive function that results from this modification.

Corollary 3.23 The recursive function $\theta_o : \mathcal{P} \to \mathcal{T}_o$ associates with every pCRL expression p an ordered tree form $\theta_o(p)$ such that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}), \nu \models p \approx \theta_o(p)$.

3.6 Value-passing CCS

Instead of introducing choice quantifiers and using them to model input, one may choose to add the input mechanism directly, as a special kind of action. This latter approach is taken in, e.g., *value-passing* CCS (Milner, 1989). To enable a comparison of both approaches later on, we now define a simple language, which roughly corresponds to the finite, sequential fragment of value-passing CCS. We give a translation of the expressions of this language to pCRL expressions. And we show that the tree forms associated with these pCRL expressions have a special form.

Suppose that $\mathrm{a} \in \mathcal{A}$ is an n-ary parametrised action symbol. An *input prefix* is an expression of the form

$$\mathrm{a}?x_1, \ldots, x_n, \text{ where } x_1, \ldots, x_n \text{ is a sequence of variables.}$$

An *output prefix* is an expression of the form

$$\mathrm{a}!d_1, \ldots, d_n, \text{ where } d_1, \ldots, d_n \text{ is a sequence of data expressions.}$$

The set $\mathcal{IO}$ of *input/output expressions* is defined by

$$io ::= \text{ nil } \mid \text{ a}?x_1, \ldots, x_n.io \mid \text{ a}!d_1, \ldots, d_n.io \mid io + io \mid b \to io,$$

where a is an n-ary parametrised action symbol, $x_1, \ldots, x_n$ is a sequence of variables, $d_1, \ldots, d_n$ is a sequence of data expressions and b is a Boolean expression. As usual we shall abbreviate $\text{a}?x_1, \ldots, x_n.\text{nil}$ by $\text{a}?x_1, \ldots, x_n$ and $\text{a}!d_1, \ldots, d_n.\text{nil}$ by $\text{a}!d_1, \ldots, d_n$.

Example 3.24 We consider again the simple protocol that we described in Chapter 2. The sending party (see Example 2.1) may be denoted by the input/output expression

$$\text{S} = \text{c}_1!\mathsf{m}.\text{c}_1?x.$$

The receiving party (see Examples 2.5 and 3.2, and also Example 3.18) may be denoted by the input/output expression

$$\text{R} = \text{c}_2?x.(\text{V}(\mathsf{m}) \to \text{c}_2!1 + \neg\, \text{V}(\mathsf{m}) \to \text{c}_2!0).$$

To provide them with a semantics, we inductively associate a pCRL expression with every input/output expression:

nil $\mapsto \delta$;
if $io \mapsto p$, then $\text{a}?x_1, \ldots, x_n.io \mapsto \sum_{x_1,\ldots,x_n} \text{a}(\vec{x})p$;
if $io \mapsto p$, then $\text{a}!d_1, \ldots, d_n.io \mapsto \text{a}(d_1, \ldots, d_n)p$;
if $io_1 \mapsto p_1$ and $io_2 \mapsto p_2$, then $(io_1 + io_2) \mapsto p_1 + p_2$; and
if $io \mapsto p$, then $(b \to io) \mapsto p \lhd b \rhd \delta$.

We shall generally not make the distinction between input/output expressions and the pCRL expressions associated with them; in particular, we shall often call a pCRL expression p an input/output expression if there is one associated with it.

If p is an input/output expression, then in $\theta_o(p)$, the ordered tree form associated with p, the construct $\sum_x$ only occurs in a special way.

Definition 3.25 Let $t = t_1 + \cdots + t_m + t_{m+1} + \cdots + t_n$ be an ordered tree form with

$$t_i = \begin{cases} \sum_{\vec{x}_i} a_i t_i' \lhd b_i \rhd \delta & 1 \le i \le m; \\ \sum_{\vec{x}_i} a_i \lhd b_i \rhd \delta & m < i \le n. \end{cases}$$

We say t has *explicit instantiation* if its continuations t_i' $(1 \le i \le m)$ have explicit instantiation, and for all $1 \le i \le n$ such that $|\vec{x}_i| > 0$:

$$a_i = \text{a}_i(\vec{x}_i) \text{ for some parametrised action symbol } \text{a}_i \text{ of arity } |\vec{x}_i|$$

($|\vec{x}_i|$ denotes the length of the sequence $\vec{x}_i$).

Lemma 3.26 If p is an input/output expression, then the ordered tree form $\theta_o(p)$ associated with p has explicit instantiation.

Proof. The proof is by induction on the structure of input/output expressions; we treat two of the five cases.

1. Suppose p is associated with $\mathrm{a}?x_1, \ldots, x_n.io$, i.e., suppose that p' is the pCRL expression associated with io and let

   $$p = \sum_{x_1,\ldots,x_n} \mathrm{a}(x_1, \ldots, x_n)p'.$$

   By the induction hypothesis, the ordered tree form $\theta_o(p')$ associated with p' has explicit instantiation; hence

   $$\theta_o(p) = \sum_{x_1,\ldots,x_n} \mathrm{a}(x_1, \ldots, x_n)\theta_o(p') \lhd \top \rhd \delta$$

   has explicit instantiation.

2. Suppose p is associated with $b \to io$ and let p' be the pCRL expression associated with io; then

   $$\theta_o(p) = \theta_{\mathrm{cnd}}(\theta_o(p'), b) + \theta_{\mathrm{cnd}}(\theta_o(\delta), \neg\, b) = \theta_{\mathrm{cnd}}(\theta_o(p'), b) + \delta.$$

   From the induction hypothesis we get that the tree form $\theta_o(p')$ has explicit instantiation. Moreover, it is easily shown by induction on the structure of tree forms that then also $\theta_{\mathrm{cnd}}(\theta_o(p'), b)$ has explicit instantiation. It follows that $\theta_o(p)$ has explicit instantiation. □

Thus, value-passing CCS gives rise to a proper subfragment of pCRL. In the next chapter, where we study the complexity of choice quantification, we shall see that this subfragment is essentially less complex than full pCRL, due to the restricted form of choice quantification.

Bibliographic notes

After Milner's proposal to provide value-passing CCS with a semantics via a translation into pure CCS (Milner, 1983), research was focused for a while on the pure variant. The 1990's showed a renewed interest in the input mechanism with a series of papers on value-passing CCS started by Hennessy (1991), and with the introduction of the π-calculus by Milner *et al.* (1992).

In retrospect, the transition from pure CCS to value-passing CCS consists of distinguishing input and output actions, and giving input actions binding parameters. It is essential for this transition that actions are prefixes. In languages with an associative binary operation for sequential composition a scoping ambiguity has to be solved; e.g., since

$$\mathrm{a}?x \cdot (p \cdot q) \approx (\mathrm{a}?x \cdot p) \cdot q,$$

$\mathrm{a}?x$ cannot bind x in q. Baeten and Bergstra (1994) propose to circumvent the scoping ambiguity by adding prefixes as primitive constructs to ACP.

In the process specification languages PSF (Mauw and Veltink, 1990) and μCRL (Groote and Ponse, 1995), which are also based on ACP, the scoping ambiguity is solved by means of choice quantifiers; e.g., in the expression $\sum_x \mathrm{a}(x)p$ the choice quantifier establishes a link between the variable x in $\mathrm{a}(x)$ and possible occurrences of x in p. Thus, the binding aspect of the input mechanism is detached from the action of receiving input. This accounts for greater expressiveness compared to when the input mechanism is included as a prefix. For instance, in μCRL we can specify

- *restricted input*: if x ranges over natural numbers and the predicate $\mathrm{even}(x)$ holds if, and only if, x is even, then the expression

  $$\textstyle\sum_x \mathrm{in}(x) \cdot p \lhd \mathrm{even}(x) \rhd \delta$$

  specifies the process that inputs an even natural number n and proceeds as the process $p[x := \mathsf{n}]$; and

- *nondeterministic output*: if $\mathsf{N}' \subseteq \mathsf{N}$ is a finite subset of the set N of natural numbers, then the recursion equation

  $$\mathrm{X}(\mathsf{N}') = \textstyle\sum_x \mathrm{out}(x) \cdot \mathrm{X}(\mathsf{N}' - \{x\}) \lhd x \in \mathsf{N}' \rhd \delta$$

  specifies the process that outputs the elements of N' in random order.

Both features have proved to be useful for the specification and verification of protocols (see, e.g., Shankland and Van der Zwaag, 1998), which is the main application area of μCRL. Note that the displayed occurrences of choice quantifiers are compatible with the requirement of explicit instantiation (Definition 3.25).

We see a similar phenomenon in the *fusion calculus* of Parrow and Victor (1998), which is a generalisation of the π-calculus. Also there, input actions have no binding effect themselves; the binding effect is achieved by means of *scope operators*. Furthermore, there is a special kind of actions, called *fusion actions*, which keep track of certain identifications of names. Fusion actions and scope operators together are used to express the passing of names between components. In addition, *delayed* input, which cannot be specified directly in the π-calculus, has a straightforward specification in the fusion calculus.

4

A correspondence between pCRL and first-order logic

The language pCRL is parametrised with a data algebra $\mathbf{D}$. As explained in Section 3.4, its expressions correspond with certain infinitely branching trees. Which trees correspond with pCRL expressions depends in part on $\mathbf{D}$. For instance, for the infinitely branching tree pictured in Figure 3.2 on p. 39 we need that the domain of $\mathbf{D}$ consists of integers, and that $\mathbf{D}$ has a relation $\leq$ or a function $|\_|$ that computes the absolute value.

It is to be expected that the validity in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ of pCRL equations also depends in some way on $\mathbf{D}$. For instance, if d and e are closed data expressions and a is a unary parametrised action, then

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \mathrm{a}(d) \approx \mathrm{a}(e) \text{ if, and only if, } \mathbf{D} \models d \approx e.$$

Also, if b is a closed Boolean expression and p and q are closed pCRL expressions, then

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \begin{cases} p \lhd b \rhd q \approx p & \text{if } \mathbf{D} \models b \approx \top; \text{ and} \\ p \lhd b \rhd q \approx q & \text{if } \mathbf{D} \models b \approx \bot. \end{cases}$$

And even if the validity of data equations and Boolean equation in $\mathbf{D}$ is decidable, the validity of a pCRL equation in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ may still be undecidable.

Example 4.1 Suppose that we take as data the natural numbers with Kleene's T-predicate: if z is the encoding (i.e., Gödel number) of Turing machine Z, then

$$T(z, x, y) = \top \text{ if, and only if, } y \text{ encodes a computation}^1 \text{ of } Z \text{ on } x.^2$$

Kleene's T-predicate is known to be primitive recursive. Now, consider the pCRL expression

$$p(z, x) = \textstyle\sum_y c \lhd T(z, x, y) \rhd \delta, \text{ where } c \text{ is any closed action expression.}$$

[1] A computation is a sequence of pairs consisting of a state and a string that represents the contents of the tape, such that the last state in the sequence is a final state.

[2] In the recursion theory literature (e.g., Davis, 1982; Rogers, Jr., 1992) one finds the predicates $T_n(z, x_1, \ldots, x_n, y)$, where Z takes the sequence $x_1, \ldots, x_n$ as input; we shall only use T_1 and drop the subscript.

If Z has a successful computation on input x, then $\mathbf{T_D}(\mathcal{A}) \models p(z,x) \approx c$; otherwise $\mathbf{T_D}(\mathcal{A}) \models p(z,x) \approx \delta$. So $p(z,x) \approx c$ holds in $\mathbf{T_D}(\mathcal{A})$ if, and only if, the first-order formula

$$(\exists y)\, T(z,x,y)$$

holds in $\mathbf{D}$. This formula defines an undecidable relation on the natural numbers —it corresponds to the *halting problem* (Turing, 1936)— so validity in $\mathbf{T_D}(\mathcal{A})$ is undecidable.

Although existential quantifiers are not part of our definition of Boolean expressions, they pop up when we consider validity in $\mathbf{T_D}(\mathcal{A})$. Example 4.1 shows that the validity in $\mathbf{T_D}(\mathcal{A})$ of a pCRL equation may be undecidable if there exist undecidable first-order assertions about the data. We shall see below that it is necessary and sufficient for the decidability of validity in $\mathbf{T_D}(\mathcal{A})$ that *all* first-order assertions about the data are decidable.

The set Φ of *first-order formulas* is generated by

$$\varphi ::= \ \mathrm{r}(d_1, \ldots, d_n) \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ (\exists x)\varphi,$$

where $d_1, \ldots, d_n$ are data expressions, r is a relation symbol of arity n, and x is a variable. The construct $(\exists x)$ binds the variable x in its argument; we adopt Convention 3.7 also for first-order formulas. For a given valuation $\nu : X \to \mathsf{D}$ we define the *satisfaction relation* $\mathbf{D}, \nu \models \varphi$ inductively as follows:

1. $\mathbf{D}, \nu \models \mathrm{r}(d_1, \ldots, d_n)$ if, and only if, $R(\bar{\nu}(d_1), \ldots, \bar{\nu}(d_n)) = \top$, where R is the n-ary relation of $\mathbf{D}$ corresponding to the relation symbol r;
2. $\mathbf{D}, \nu \models \neg\varphi$ if, and only if, $\mathbf{D}, \nu \not\models \varphi$;
3. $\mathbf{D}, \nu \models \varphi \vee \psi$ if, and only if, $\mathbf{D}, \nu \models \varphi$ or $\mathbf{D}, \nu \models \psi$; and
4. $\mathbf{D}, \nu \models (\exists x)\varphi$ if, and only if, there exists $\mathsf{d} \in \mathsf{D}$ such that $\mathbf{D}, \nu[x := \mathsf{d}] \models \varphi$, where $\nu[x := \mathsf{d}]$ is the valuation such that

$$\nu[x := \mathsf{d}](y) = \begin{cases} \mathsf{d} & \text{if } y = x; \text{ and} \\ \nu(y) & \text{otherwise.} \end{cases}$$

If $\mathbf{D}, \nu \models \varphi$ for all valuations ν, then we write $\mathbf{D} \models \varphi$. The *first-order theory* of $\mathbf{D}$ is the set of all formulas φ such that $\mathbf{D} \models \varphi$.

We also define the pCRL *theory* of $\mathbf{D}$, as the set of all pCRL equations $p \approx q$ such that $\mathbf{T_D}(\mathcal{A}) \models p \approx q$. We shall reveal the following intimate relationship between the pCRL theory of $\mathbf{D}$ and the first-order theory of $\mathbf{D}$:

> The pCRL theory of $\mathbf{D}$ and the first-order theory of $\mathbf{D}$ are recursively isomorphic.

That is, there exists a recursive bijection between both theories (see Rogers, Jr., 1992). To prove this, it is by a theorem of Myhill (1955) enough to show that the pCRL theory of $\mathbf{D}$ and the first-order theory of $\mathbf{D}$ have the same degree of unsolvability with respect to one-one reducibility (Rogers, Jr., 1992). That is, it suffices to define two one-one recursive functions:

1. a one-one recursive function $\phi : \mathcal{P} \times \mathcal{P} \to \Phi$ such that for every valuation ν

   $$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q \text{ if, and only if, } \mathbf{D}, \nu \models \phi(p, q); \text{ and}$$

2. a one-one recursive function $\eta : \Phi \to \mathcal{P} \times \mathcal{P}$ such that for every valuation ν

   $$\mathbf{D}, \nu \models \varphi \text{ if, and only if, } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q, \text{ where } \eta(\varphi) = \langle p, q \rangle.$$

The function ϕ will be defined in Section 4.2 (see Theorem 4.10). The function η will be defined in Section 4.3 (see Theorem 4.17). First, however, it is convenient to devote a preliminary section on discussing the precise connection between the Boolean expressions used as conditions in pCRL expressions, and certain first-order formulas.

4.1 Boolean expressions and open first-order formulas

Following, e.g., Shoenfield (1967) and Chang and Keisler (1990), we call a first-order formula is *open* if it contains no quantifiers. Syntactically, every open first-order formula is also a Boolean expression, and the following proposition provides the semantical justification for this ambiguity.

Proposition 4.2 If φ is an open first-order formulas, then

$$\mathbf{D}, \nu \models \varphi \text{ if, and only if, } \mathbf{D}, \nu \models \varphi \approx \top$$

for every valuation ν.

Proof. We proceed by induction on the structure of φ.
If $\varphi = \mathrm{r}(d_1, \ldots, d_n)$ and r denotes the n-ary relation R of $\mathbf{D}$, then

$$\mathbf{D}, \nu \models \varphi \Leftrightarrow R(\bar{\nu}(d_1), \ldots, \bar{\nu}(d_n)) = \top \Leftrightarrow \mathbf{D}, \nu \models \varphi \approx \top$$

If $\varphi = \neg\, \psi$, then, according to the definition of $\bar{\nu}$ on p. 32,

$$\bar{\nu}(\varphi) = \top \text{ if, and only if, } \bar{\nu}(\psi) \neq \top;$$

hence, with an application of the induction hypothesis,

$$\mathbf{D}, \nu \models \varphi \Leftrightarrow \mathbf{D}, \nu \not\models \psi \Leftrightarrow \mathbf{D}, \nu \not\models \psi \approx \top \Leftrightarrow \mathbf{D}, \nu \models \varphi \approx \top.$$

If $\varphi = \psi \vee \chi$, then, according to the definition of $\bar{\nu}$ on p. 32,

$$\bar{\nu}(\varphi) = \top \text{ if, and only if, } \bar{\nu}(\psi) = \top \text{ or } \bar{\nu}(\chi) = \top;$$

hence, with an application of the induction hypothesis,

$$\begin{aligned} \mathbf{D}, \nu \models \varphi &\Leftrightarrow \mathbf{D}, \nu \models \psi \text{ or } \mathbf{D}, \nu \models \chi \\ &\Leftrightarrow \mathbf{D}, \nu \models \psi \approx \top \text{ or } \mathbf{D}, \nu \models \chi \approx \top \Leftrightarrow \mathbf{D}, \nu \models \varphi \approx \top. \end{aligned}$$

The proof is complete. □

Our definition of first-order formula deviates slightly from that of Shoenfield (1967); Shoenfield presupposes a binary relation symbol with a fixed interpretation as equality. The reason for our deviation is that, for the rest of this chapter, it is convenient to have that every open first-order formula is automatically a Boolean expression, whence may be used as a condition in a pCRL expression. If we now add equality as a special requirement on data algebras, then, of course, this property is maintained.

Definition 4.3 We say that a data algebra $\mathbf{D}$ has *equality* if, among the relations of $\mathbf{D}$, there is a binary relation denoted by the relation symbol eq such that for every valuation ν:

$$\mathbf{D}, \nu \models \begin{cases} \mathrm{eq}(x, y) \approx \top & \text{if } \nu(x) = \nu(y); \text{ and} \\ \mathrm{eq}(x, y) \approx \bot & \text{if } \nu(x) \neq \nu(y). \end{cases}$$

Note that, syntactically, Boolean expressions are open first-order formula, unless they contain occurrences of the symbols $\top$, $\bot$ or $\wedge$. But it is well-known that $\wedge$ is definable with $\neg$ and $\vee$, and with equality as a binary relation in $\mathbf{D}$, $\top$ and $\bot$ turn out to be definable as well. We get that every Boolean expression is semantically equivalent to a first-order formula.

Proposition 4.4 If $\mathbf{D}$ has equality, then for every Boolean expression b there exists an open first-order formula φ such that $\mathbf{D} \models b \approx \varphi$.

Proof. We make three observations.
Firstly, according to Definition 4.3, for every variable $x \in X$

$$\mathbf{D} \models \mathrm{eq}(x, x) \approx \top, \tag{4.1}$$

so if $b = \top$, then we can select $x \in X$ and put $\varphi = \mathrm{eq}(x, x)$.
Secondly, since $\neg \top = \bot$ by definition,

$$\mathbf{D} \models \neg \top \approx \bot, \tag{4.2}$$

so if $b = \bot$, then we can put $\varphi = \neg\, \mathrm{eq}(x, x)$.
Thirdly, suppose that $b = \psi \wedge \chi$ and ψ and χ are open first-order formulas. Then, since $\mathsf{b} \wedge \mathsf{c} = \neg(\neg\, \mathsf{b} \vee \neg\, \mathsf{c})$ for all $\mathsf{b}, \mathsf{c} \in \mathsf{B}$,

$$\mathbf{D} \models \psi \wedge \chi \approx \neg(\neg\, \psi \vee \neg\, \chi), \tag{4.3}$$

so we can put $\varphi = \neg(\neg\, \psi \vee \neg\, \chi)$.
With these observations the proposition follows by structural induction on b. □

For the most part, we shall be working with Boolean expressions modulo semantic equivalence, and with a data algebra that has equality. Then, according to the above proposition, every Boolean expression may be conceived as an open first-order formula: by (4.1)–(4.3) we may interpret occurrences of $\top$, $\bot$ and $\varphi \wedge \psi$ as abbreviations of $\mathrm{eq}(x, x)$, $\neg \top$, and $\neg(\neg\, \varphi \vee \neg\, \psi)$, respectively.

We introduce a few more standard abbreviations: $\varphi \to \psi$ abbreviates $\neg\, \varphi \vee \psi$; $\varphi \leftrightarrow \psi$ abbreviates $(\varphi \to \psi) \wedge (\psi \to \varphi)$; and $(\forall x)\varphi$ abbreviates $\neg(\exists x)\, \neg\, \varphi$. Furthermore, if $m \geq 1$ and $n \geq 0$, then we define the formula $\bigvee_{m \leq i \leq n} \varphi_i$ inductively as follows:

1. if $n = 0$, then $\bigvee_{m \leq i \leq n} \varphi_i = \bot$; and
2. if $n \geq m$, then $\bigvee_{m \leq i \leq n} \varphi_i = \bigvee_{m \leq i \leq n-1} \varphi_i \vee \varphi_n$.

4.2 The definition of ϕ

We start with an analysis of when a valuation ν satisfies $t \preccurlyeq u$ in $\mathbf{T_D}(\mathcal{A})$, where t and u are ordered tree forms. Our analysis will lead to the definition of a recursive function $\phi_{\preccurlyeq} : \mathcal{T}_o \times \mathcal{T}_o \to \Phi$ such that for all ordered tree forms t and u

$$\mathbf{D}, \nu \models \phi_{\preccurlyeq}(t, u) \text{ if, and only if, } \mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u.$$

We shall then obtain ϕ from $\phi_{\preccurlyeq}$ and the function θ that assigns to every pCRL expression an equivalent ordered tree form.

First, we distinguish cases according to the form of t:
Suppose that $t = \delta$. Since δ is the least element with respect to $\leq$ in every generalised basic process algebra with deadlock,

$$\mathbf{T_D}(\mathcal{A}), \nu \models \delta \preccurlyeq u. \tag{4.4}$$

Suppose that $t = t' + t''$, then, since an alternative composition is the least upper bound of its components in every generalised basic process algebra with deadlock,

$$\mathbf{T_D}(\mathcal{A}), \nu \models t' + t'' \preccurlyeq u \text{ if, and only if, } \mathbf{T_D}(\mathcal{A}), \nu \models t' \preccurlyeq u,\ t'' \preccurlyeq u. \tag{4.5}$$

Suppose t is a simple tree form, say $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$. We need some notation: if $\vec{x} = x_1, \ldots, x_n$ is a sequence of variables, and $\vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n$ is a sequence of elements of D, then with $[\vec{x} := \vec{\mathsf{d}}]$ we shall mean the sequence

$$[x_n := \mathsf{d}_n] \cdots [x_1 := \mathsf{d}_1].$$

(The inversion is for convenience of notation; e.g., we have, for a sequence of variables $\vec{x} = x_1, \ldots, x_n$, that $\iota_\nu(\sum_{\vec{x}} p) = \sum\{\iota_\nu(p[\vec{x} := \vec{\mathsf{d}}]) \mid \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}\}$, also if some variable occurs more than once in $\vec{x}$.)

Lemma 4.5 Suppose that $\vec{x} = x_1, \ldots, x_n$ is a sequence of variables such that $\{\vec{x}\} \cap \mathsf{FV}(u) = \emptyset$; then

$$\begin{aligned}&\mathbf{T_D}(\mathcal{A}), \nu \models \textstyle\sum_{\vec{x}} t^* \lhd b \rhd \delta \preccurlyeq u \text{ if, and only if,}\\ &\quad \text{for all sequences } \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}\\ &\qquad \mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top \text{ implies } \mathbf{T_D}(\mathcal{A}), \nu[\vec{x} := \vec{\mathsf{d}}] \models t^* \preccurlyeq u.\end{aligned} \tag{4.6}$$

Proof. Let $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$, and let ι_ν be the interpretation homomorphism from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into $\mathbf{T_D}(\mathcal{A})$ generated by ν; then

$$\iota_\nu(t) = \sum\{\iota_\nu(t^*[\vec{x} := \vec{\mathsf{d}}]) \mid \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ s.t. } \bar{\nu}(b[\vec{x} := \vec{\mathsf{d}}]) = \top\ \}.$$

If $\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u$, then, by (GA1),

$$\iota_\nu(t^*[\vec{x} := \vec{\mathsf{d}}]) \leq \iota_\nu(u) \text{ for all sequences } \vec{\mathsf{d}} \text{ such that } \mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top.$$

Since $x_i \notin \mathsf{FV}(u)$ for all $1 \leq i \leq n$,

$$\iota_\nu(u) = \iota_\nu(u[\vec{x} := \vec{\mathsf{d}}]).$$

Hence $\mathbf{T_D}(\mathcal{A}), \nu[\vec{x} := \vec{\mathsf{d}}] \models t^* \preccurlyeq u$.

Conversely, suppose

$$\mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top \text{ implies } \mathbf{T_D}(\mathcal{A}), \nu[\vec{x} := \vec{\mathsf{d}}] \models t^* \preccurlyeq u \text{ for all } \vec{\mathsf{d}}.$$

By (GA2),

$$\mathbf{T_D}(\mathcal{A}), \nu[\vec{x} := \vec{\mathsf{d}}] \models t \preccurlyeq u.$$

Hence, since $\iota_\nu(u) = \iota_\nu(u[\vec{x} := \vec{\mathsf{d}}])$, $\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u$. □

So, if $t = t_1 + \cdots + t_n$ and t_i is simple for all $1 \leq i \leq n$, then, by (4.4) and (4.5), whether a statement of the form $\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u$ is true is determined by whether statements of the form $\mathbf{T_D}(\mathcal{A}), \nu \models t_i \preccurlyeq u$ are true. Furthermore, if $t_i = \sum_{\vec{x}} t_i^* \lhd b \rhd \delta$, then, by (4.6), whether the statement $\mathbf{T_D}(\mathcal{A}), \nu \models t_i \preccurlyeq u$ is true is determined by whether a statement of the form $\mathbf{T_D}(\mathcal{A}), \nu \models t_i^* \preccurlyeq u$ is true. Note that if t_i is simple, then t_i^* is either an action expression or a sequential composition that starts with an action expression.

Let us fix an action expression a and a tree form t', and suppose that $t^* = a$ or $t^* = at'$. We shall now analyse when ν satisfies $t^* \preccurlyeq u$ in $\mathbf{T_D}(\mathcal{A})$; again we distinguish cases, this time according to the form of u:
Suppose that $u = \delta$; then, by Lemma 2.7(i),

$$\text{if } t^* = a \text{ or } t^* = at', \text{ then } \mathbf{T_D}(\mathcal{A}), \nu \not\models t^* \preccurlyeq \delta. \tag{4.7}$$

Suppose that $u = u' + u''$; then, by Lemma 2.7(ii),

$$\begin{aligned} &\text{if } t^* = a \text{ or } t^* = at', \text{ then} \\ &\quad \mathbf{T_D}(\mathcal{A}), \nu \models t^* \preccurlyeq u' + u'' \text{ if, and only if,} \\ &\qquad\qquad \mathbf{T_D}(\mathcal{A}), \nu \models t^* \preccurlyeq u' \text{ or } \mathbf{T_D}(\mathcal{A}), \nu \models t^* \preccurlyeq u'. \end{aligned} \tag{4.8}$$

For the case that u is a simple expression, we first prove a lemma.

Lemma 4.6 Suppose $t^* = a$ or $t^* = at'$, and let $\vec{x} = x_1, \ldots, x_n$ be a sequence of variables such that $\{\vec{x}\} \cap \mathsf{FV}(t^*) = \emptyset$; then

$$\begin{aligned} &\mathbf{T_D}(\mathcal{A}), \nu \models t^* \preccurlyeq \textstyle\sum_{\vec{x}} u^* \lhd b \rhd \delta \text{ if, and only if,} \\ &\quad \text{there is a sequence } \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ such that} \\ &\qquad \mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top \text{ and } \mathbf{T_D}(\mathcal{A}), \nu[\vec{x} := \vec{\mathsf{d}}] \models t^* \preccurlyeq u^*. \end{aligned} \tag{4.9}$$

Proof. Let $u = \sum_{\vec{x}} u^* \lhd b \rhd \delta$, and let ι_ν be the interpretation homomorphism from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into $\mathbf{T_D}(\mathcal{A})$ generated by ν; then

$$\iota_\nu(u) = \sum\{\iota_\nu(u^*[\vec{x} := \vec{\mathsf{d}}]) \mid \vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D} \text{ s.t. } \mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top \}.$$

Since $\iota_\nu(t^*) = \iota_\nu(a)$ or $\iota_\nu(t^*) = \iota_\nu(a) \cdot \iota_\nu(t')$ and $\iota_\nu(a)$ is a tree action, we find by Lemma 2.7(iii) that $\iota_\nu(t^*) \leq \iota_\nu(u)$ if, and only if, there exists $\vec{\mathsf{d}} = \mathsf{d}_1, \ldots, \mathsf{d}_n \in \mathsf{D}$ such that $\mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top$ and $\iota_\nu(t^*) \leq \iota_\nu(u^*[\vec{x} := \vec{\mathsf{d}}])$; the lemma follows. □

Now, suppose that u is a simple expression, say $u = \sum_{\vec{x}} u^* \lhd b \rhd \delta$ with $u^* = a'$ or $u^* = a'u'$; we conclude our analysis by distinguishing cases according to the forms of t^* and u^*:
if $t^* = a$ and $u^* = a'$, then, by Lemma 2.7(v),

$$\mathbf{T_D}(\mathcal{A}), \nu \models t^* \preccurlyeq u^* \text{ if, and only if, } \mathbf{T_D}(\mathcal{A}), \nu \models t^* \approx u^*; \tag{4.10}$$

if $t^* = at'$ and $u^* = a'u'$, then, by Lemma 2.7(vi),

$$\mathbf{T_D}(\mathcal{A}), \nu \models t^* \preccurlyeq u^* \text{ if, and only if, } \mathbf{T_D}(\mathcal{A}), \nu \models a \approx a',\ t' \approx u'; \tag{4.11}$$

if $t^* = at'$ and $u^* = a'$, or $t^* = a$ and $u^* = a'u'$, then, by Lemma 2.7(iv),

$$\mathbf{T_D}(\mathcal{A}), \nu \not\models t^* \preccurlyeq u^*. \tag{4.12}$$

Our analysis shows that a statement $\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u$ is equivalent to a first-order combination of statements of the form

1. $\mathbf{D}, \nu \models b \approx \top$, with b a Boolean expression;
2. $\mathbf{T_D}(\mathcal{A}), \nu \models a \approx a'$, where a and a' are action expressions; and
3. $\mathbf{T_D}(\mathcal{A}), \nu \models t' \preccurlyeq u'$ and $\mathbf{T_D}(\mathcal{A}), \nu \models u' \preccurlyeq t'$, where t' and u' are continuations of simple expressions in t and u, respectively.

It is straightforward to associate an appropriate first-order formula with a statement of the first form: conceive b as an open first-order formula (cf. Proposition 4.4 and the remarks directly following its proof).

Definition 4.7 Suppose that $\mathbf{D}$ has equality; we associate with every two action expressions $a = \mathrm{a}(d_1, \ldots, d_m)$ and $a' = \mathrm{a}'(e_1, \ldots, e_n)$ a Boolean expression $\mathrm{eq}(a, a')$ as follows:

$$\mathrm{eq}(a, a') = \begin{cases} \mathrm{eq}(d_1, e_1) \wedge \cdots \wedge \mathrm{eq}(d_m, e_n) & \text{if } \mathrm{a} = \mathrm{a}' \text{ and } m = n; \text{ and} \\ \bot & \text{otherwise} \end{cases}$$

If we take $\mathrm{eq}(a, a')$ as a first-order formula, then we have the following lemma.

Lemma 4.8 If $\mathbf{D}$ has equality, then

$$\mathbf{T_D}(\mathcal{A}), \nu \models a \approx a' \text{ if, and only if, } \mathbf{D}, \nu \models \mathrm{eq}(a, a').$$

Proof. We have

$$\begin{aligned}
&\mathbf{T_D}(\mathcal{A}), \nu \models a \approx a' \\
&\quad \Leftrightarrow \mathrm{a}(\bar{\nu}(d_1), \ldots, \bar{\nu}(d_m)) = \mathrm{a}'(\bar{\nu}(e_1), \ldots, \bar{\nu}(e_n)) \\
&\quad \Leftrightarrow \mathrm{a} = \mathrm{a}',\ m = n \text{ and } \bar{\nu}(d_i) = \bar{\nu}(e_i) \text{ for all } 1 \leq i \leq n \\
&\quad \Leftrightarrow \mathrm{a} = \mathrm{a}',\ m = n \text{ and } \mathbf{D}, \nu \models \mathrm{eq}(d_i, e_i) \text{ for all } 1 \leq i \leq n \\
&\quad \Leftrightarrow \mathrm{a} = \mathrm{a}',\ m = n \text{ and } \mathbf{D}, \nu \models \mathrm{eq}(d_1, e_1) \wedge \cdots \wedge \mathrm{eq}(d_n, e_n) \\
&\quad \Leftrightarrow \mathbf{D}, \nu \models \mathrm{eq}(a, a'),
\end{aligned}$$

by which the lemma is proved. □

Thus, we associate with a statement of the second form the first-order formula $\mathrm{eq}(a, a')$. With statements of the third form we are going to deal recursively. First, we associate with every tree form t a natural number $|t|$:

$$\begin{aligned}
&|\delta| = 0; && |\textstyle\sum_{\vec{x}} a \lhd b \rhd \delta| = 1; \\
&|t' + t''| = |t'| + |t''|; && |\textstyle\sum_{\vec{x}} at' \lhd b \rhd \delta| = |t'| + 1.
\end{aligned}$$

If t' is the continuation of a simple expression in t, then $|t'| < |t|$. Consequently, if t' and u' are continuations of simple expressions in t and u, respectively, then

$$|t'| + |u'| < |t| + |u|.$$

Hence, by induction on $|t| + |u|$ it follows that the expression $\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u$ is equivalent to a first-order combination of expressions of the first two forms. The recursive algorithm in Table 4.1 reflects our analysis, except that it applies (4.6) and (4.9) without verifying the provisos of Lemmas 4.5 and 4.6. Let us say that the algorithm in Table 4.1 is *correct* for t and u if for every variable x

(i) if $\sum_x$ occurs in t, then x does not occur at all in u; and

(ii) if $\sum_x$ occurs in u, then x does not occur at all in t.

Proposition 4.9 Suppose that $\mathbf{D}$ has equality. If the algorithm in Table 4.1 is correct for t and u, then it associates with t and u a first-order formula $\phi_{\preccurlyeq}(t, u)$ such that

$$\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u \text{ if, and only if, } \mathbf{D}, \nu \models \phi_{\preccurlyeq}(t, u).$$

Proof. The proof is by induction on $|t| + |u|$. If $|t| + |u| = 0$, then $t = \delta$, so $\mathbf{T_D}(\mathcal{A}), \nu \models t \preccurlyeq u$ by (4.4) and $\mathbf{D}, \nu \models \phi_{\preccurlyeq}(t, u)$ since $\phi_{\preccurlyeq}(t, u) = \top$. Suppose that $|t| + |u| > 0$; we proceed by distinguishing cases according to the form of t. We shall only treat the cases that involve an application of the induction hypothesis.
First, suppose that $t = \sum_{\vec{x}} at' \lhd b \rhd \delta$. Since the algorithm is correct for t and u, $\{\vec{x}_i\} \cap \mathsf{FV}(at') = \emptyset$ for all $1 \leq i \leq m$, so by (4.9) and (4.11)

$$\begin{aligned}
&\mathbf{T_D}(\mathcal{A}), \nu \models at' \preccurlyeq u_i \text{ if, and only if, there exists a sequence } \vec{\mathsf{d}} \text{ such that} \\
&\qquad \mathbf{D}, \nu[\vec{x}_i := \vec{\mathsf{d}}] \models b_i \approx \top, \text{ and } \mathbf{T_D}(\mathcal{A}), \nu[\vec{x}_i := \vec{\mathsf{d}}] \models a \approx a_i,\ t' \approx u_i'.
\end{aligned}$$

compute $\phi_{\preccurlyeq}(t, u)$**:**

let $u = u_1 + \cdots + u_m + u_{m+1} + \cdots + u_n$**,**

where $u_i = \begin{cases} \sum_{\vec{x}_i} a_i \cdot u_i' \lhd b_i \rhd \delta & 1 \leq i \leq m; \\ \sum_{\vec{x}_i} a_i \lhd b_i \rhd \delta & m < i \leq n. \end{cases}$

case

$t = \delta$**:**
return $\top$**.**

$t = \sum_{\vec{x}} a \lhd b \rhd \delta$**:**
return

$$(\forall \vec{x}) \left(b \to \bigvee_{m<i\leq n} (\exists \vec{x}_i) \left(b_i \wedge \mathrm{eq}(a, a_i) \right) \right).$$

$t = \sum_{\vec{x}} a \cdot t' \lhd b \rhd \delta$**:**
compute $\phi_{\preccurlyeq}(t', u_i')$ **for all** $1 \leq i \leq m$**;**
compute $\phi_{\preccurlyeq}(u_i', t')$ **for all** $1 \leq i \leq m$**;**
return

$$(\forall \vec{x}) \left(b \to \bigvee_{1\leq i\leq m} (\exists \vec{x}_i) \left(b_i \wedge \mathrm{eq}(a, a_i) \wedge \phi_{\preccurlyeq}(t', u_i') \wedge \phi_{\preccurlyeq}(u_i', t') \right) \right).$$

$t = t' + t''$**:**
compute $\phi_{\preccurlyeq}(t', u)$**;**
compute $\phi_{\preccurlyeq}(t'', u)$**;**
return $\phi_{\preccurlyeq}(t', u) \wedge \phi_{\preccurlyeq}(t'', u)$**.**

end.

Table 4.1: The algorithm that computes $\phi_{\preccurlyeq}$.

By Lemma 4.8

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x}_i := \vec{\mathsf{d}}] \models a \approx a_i \text{ if, and only if, } \mathbf{D}, \nu[\vec{x}_i := \vec{\mathsf{d}}] \models \mathrm{eq}(a, a_i),$$

so with two applications of the induction hypothesis, using that $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q$ if, and only if, $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \preccurlyeq q$ and $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models q \preccurlyeq p$, we get

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x}_i := \vec{\mathsf{d}}] \models t' \approx u_i' \text{ if, and only if,}$$
$$\mathbf{D}, \nu[\vec{x}_i := \vec{\mathsf{d}}] \models \phi_{\preccurlyeq}(t', u_i') \wedge \phi_{\preccurlyeq}(u_i', t').$$

Hence

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models at' \preccurlyeq u_i \text{ if, and only if,}$$
$$\mathbf{D}, \nu \models (\exists \vec{x}_i)\,(b_i \wedge \mathrm{eq}(a, a_i) \wedge \phi_{\preccurlyeq}(t', u_i') \wedge \phi_{\preccurlyeq}(u_i', t'))\,.$$

Consequently, by (4.8) and (4.12)

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models at' \preccurlyeq u \text{ if, and only if,}$$
$$\mathbf{D}, \nu \models \bigvee_{1 \le i \le m} (\exists \vec{x}_i)\,(b_i \wedge \mathrm{eq}(a, a_i) \wedge \phi_{\preccurlyeq}(t', u_i') \wedge \phi_{\preccurlyeq}(u_i', t'))\,.$$

Also since the algorithm is correct for t and u, $\{\vec{x}\} \cap \mathsf{FV}(u) = \emptyset$, so by (4.6)

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models \textstyle\sum_{\vec{x}} at' \lhd b \rhd \delta \preccurlyeq u \text{ if, and only if,}$$
$$\mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models \bigvee_{m < i \le n} (\exists \vec{x}_i)\,(b_i \wedge \mathrm{eq}(a, a_i) \wedge \phi_{\preccurlyeq}(t', u_i') \wedge \phi_{\preccurlyeq}(u_i', t'))$$
$$\text{for all sequences } \vec{\mathsf{d}} \text{ such that } \mathbf{D}, \nu[\vec{x} := \vec{\mathsf{d}}] \models b \approx \top.$$

Hence $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t \preccurlyeq u$ if, and only if, $\mathbf{D}, \nu \models \phi_{\preccurlyeq}(t, u)$.
Next, suppose that $t = t' + t''$; by the induction hypothesis

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t' \preccurlyeq u \text{ if, and only if, } \mathbf{D}, \nu \models \phi_{\preccurlyeq}(t', u),$$

and

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t'' \preccurlyeq u \text{ if, and only if, } \mathbf{D}, \nu \models \phi_{\preccurlyeq}(t'', u),$$

so by (4.5), $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t \preccurlyeq u$ if, and only if, $\mathbf{D}, \nu \models \phi_{\preccurlyeq}(t, u)$. □

The algorithm in Table 4.1 yields a partial recursive function $\phi_{\preccurlyeq} : \mathcal{T}_o \times \mathcal{T}_o \to \Phi$ that is defined on t and u if the algorithm is correct for t and u. It induces a total function on α-congruence classes of tree forms which is by Convention 3.7 also denoted by $\phi_{\preccurlyeq}$; we have that

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t \preccurlyeq u \text{ if, and only if, } \mathbf{D}, \nu \models \phi_{\preccurlyeq}(t, u).$$

Since $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q$ if, and only if, $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \preccurlyeq q$ and $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models q \preccurlyeq p$, and by Corollary 3.23, we get that

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q \text{ if, and only if, } \mathbf{D}, \nu \models \phi_{\preccurlyeq}(\theta_o(p), \theta_o(q)) \wedge \phi_{\preccurlyeq}(\theta_o(q), \theta_o(p)).$$

Thus, we have a candidate for ϕ, except that it is not one-one. (If t is an ordered tree form, then $\theta_o(t+\delta) = t$, so $\phi_{\preccurlyeq}(\theta_o(t), \theta_o(q)) = \phi_{\preccurlyeq}(\theta_o(t+\delta), \theta_o(q))$ for all q.) We obtain a one-one function as follows. Let $\ulcorner \_ \urcorner : \mathcal{P} \to (\omega - \{0\})$ be any recursive injection of $\mathcal{P}$ into the set of positive natural numbers (any recursive coding of strings over the set of symbols used to write pCRL expressions will do; it is well-known that such codings exist for finite strings over a countable alphabet). For $n \geq 1$ we define $(\bot)^n$ by $(\bot)^1 = \bot$ and $(\bot)^{n+1} = (\bot)^n \vee \bot$; note that $\mathbf{D}, \nu \models \varphi \vee (\bot)^n$ if, and only if, $\mathbf{D}, \nu \models \varphi$, for all formulas φ. Now, let $\phi : \mathcal{P} \times \mathcal{P} \to \Phi$ be such that for all p and q

$$\langle p, q \rangle \mapsto (\phi_{\preccurlyeq}(\theta_o(p), \theta_o(q)) \wedge \phi_{\preccurlyeq}(\theta_o(q), \theta_o(p))) \vee (\bot)^{\ulcorner p \urcorner} \vee (\bot)^{\ulcorner q \urcorner}$$

Then, ϕ is the one-one recursive function we needed to define; we have proved

Theorem 4.10 Suppose that $\mathbf{D}$ has equality. Then there exists a one-one recursive function $\phi : \mathcal{P} \times \mathcal{P} \to \Phi$ such that for all pCRL expressions p and q

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q \text{ if, and only if, } \mathbf{D}, \nu \models \phi(p, q).$$

4.3 The definition of η

We shall now associate with every first-order formula φ a pair of pCRL expressions $\eta(\varphi) = \langle p, q \rangle$ such that $\mathbf{D}, \nu \models \varphi$ if, and only if $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q$. Recall that an open first-order formula may be viewed as a Boolean expression (cf. Section 4.1).

Lemma 4.11 If φ is an open first-order formula and c is a closed action expression, then

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models c \lhd \varphi \rhd \delta \approx c \text{ if, and only if, } \mathbf{D}, \nu \models \varphi.$$

Proof. By Proposition 4.2, $\mathbf{D}, \nu \models \varphi$ if, and only if, $\mathbf{D}, \nu \models \varphi \approx \top$. If $\mathbf{D}, \nu \models \varphi$, then $\iota_\nu(c \lhd \varphi \rhd \delta) = \iota_\nu(c)$; otherwise $\iota_\nu(c \lhd \varphi \rhd \delta) = \delta$. Since $\iota_\nu(c) \neq \delta$ the lemma follows. □

A formula φ is in *prenex form* if it has the form

$$(\mathcal{Q}x_1) \ldots (\mathcal{Q}x_n)\psi$$

where each $(\mathcal{Q}x_i)$ is either $(\exists x_i)$ or $(\forall x_i)$, the variables $x_1, \ldots, x_n$ are all distinct, and ψ is open. We call $(\mathcal{Q}x_1) \ldots (\mathcal{Q}x_n)$ the *prefix* of φ and ψ the *matrix*.

Lemma 4.12 There exists a recursive function $\pi : \Phi \to \Phi$ that associates with every first-order formula φ a prenex form $\pi(\varphi)$ such that

$$\mathbf{D}, \nu \models \pi(\varphi) \text{ if, and only if, } \mathbf{D}, \nu \models \varphi.$$

Proof. See Shoenfield (1967) or Rogers, Jr. (1992). □

Lemma 4.11 shows how an open first-order formula can be expressed as a pCRL equation. We shall prove now that universal and existential quantifiers can be expressed as transformations on pairs of pCRL expressions. Then, we shall conclude that every prenex form is expressible as a pCRL equation, and we shall define the function η using π (with a similar trick as in the definition of ϕ to ensure that η is one-one).

Since universal quantification generalises conjunction, it is instructive to see how conjunction is expressible.

Example 4.13 Suppose that t_1, t_2, u_1 and u_2 are trees. We wish to construct trees t and u such that $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ and $\mathsf{t}_2 = \mathsf{u}_2$. Let a_1 and a_2 be distinct tree actions; we define $\mathsf{t} = \mathsf{a}_1 \cdot \mathsf{t}_1 + \mathsf{a}_2 \cdot \mathsf{t}_2$ and $\mathsf{u} = \mathsf{a}_1 \cdot \mathsf{u}_1 + \mathsf{a}_2 \cdot \mathsf{u}_2$ (see Figure 4.1).

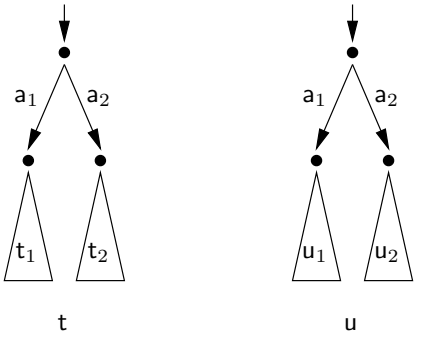


Figure 4.1: $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ and $\mathsf{t}_2 = \mathsf{u}_2$.

By Lemma 2.7(vi) $\mathsf{a}_1 \cdot \mathsf{t}_1 = \mathsf{a}_1 \cdot \mathsf{u}_1$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$, and also, since $\mathsf{a}_1 \neq \mathsf{a}_2$, $\mathsf{a}_1 \cdot \mathsf{t}_1 \neq \mathsf{a}_2 \cdot \mathsf{u}_2$. Hence by Lemma 2.7(ii) $\mathsf{a}_1 \cdot \mathsf{t}_1 \leq \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$. Similarly it follows that $\mathsf{a}_2 \cdot \mathsf{t}_2 \leq \mathsf{u}$ if, and only if, $\mathsf{t}_2 = \mathsf{u}_2$, so $\mathsf{t} \leq \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ and $\mathsf{t}_2 = \mathsf{u}_2$. By a symmetric argument it also follows that $\mathsf{u} \leq \mathsf{t}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ and $\mathsf{t}_2 = \mathsf{u}_2$; we get $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ and $\mathsf{t}_2 = \mathsf{u}_2$.

Let a be a unary parametrised action symbol; we define

$$(\forall x)_1\langle p, q\rangle = \textstyle\sum_x \mathrm{a}(x)p; \text{ and}$$
$$(\forall x)_2\langle p, q\rangle = \textstyle\sum_x \mathrm{a}(x)q.$$

Intuitively, $\mathrm{a}(x)$ pairs a particular instance of p with the *same* instance of q: if $\mathsf{d}_1, \mathsf{d}_2 \in \mathsf{D}$ are distinct, then it is possible that $\iota_\nu(p[x := \mathsf{d}_1]) = \iota_\nu(q[x := \mathsf{d}_2])$ for some valuation ν, while $\iota_\nu(\mathrm{a}(\mathsf{d}_1)) \neq \iota_\nu(\mathrm{a}(\mathsf{d}_2))$ implies that

$$\iota_\nu(\mathrm{a}(\mathsf{d}_1)) \cdot \iota_\nu(p[x := \mathsf{d}_1]) \neq \iota_\nu(\mathrm{a}(\mathsf{d}_2)) \cdot \iota_\nu(q[x := \mathsf{d}_2]).$$

Compare this to the use of a_1 and a_2 in Figure 4.1: it follows from $\mathsf{a}_1 \neq \mathsf{a}_2$ that $\mathsf{a}_1 \cdot \mathsf{t}_1 \neq \mathsf{a}_2 \cdot \mathsf{u}_2$.

Lemma 4.14 ($\forall$-introduction) If p and q are pCRL expressions, then

$$\mathbf{T_D}(\mathcal{A}), \nu \models (\forall x)_1\langle p, q\rangle \approx (\forall x)_2\langle p, q\rangle \text{ if, and only if,}$$
$$\mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models p \approx q \text{ for all } \mathsf{d} \in \mathsf{D}.$$

Proof.

($\Rightarrow$) If $\mathbf{T_D}(\mathcal{A}), \nu \models (\forall x)_1\langle p, q\rangle \approx (\forall x)_2\langle p, q\rangle$, then

$$\sum\{\iota_\nu(\mathrm{a}(\mathsf{d}_1)) \cdot \iota_\nu(p[x := \mathsf{d}_1]) \mid \mathsf{d}_1 \in \mathsf{D}\} = \sum\{\iota_\nu(\mathrm{a}(\mathsf{d}_2)) \cdot \iota_\nu(q[x := \mathsf{d}_2]) \mid \mathsf{d}_2 \in \mathsf{D}\},$$

so by, Lemma 2.7(iii,vi), for every $\mathsf{d}_1 \in \mathsf{D}$ there exists $\mathsf{d}_2 \in \mathsf{D}$ such that $\mathrm{a}(\mathsf{d}_1) = \mathrm{a}(\mathsf{d}_2)$ and $\iota_\nu(p[x := \mathsf{d}_1]) = \iota_\nu(q[x := \mathsf{d}_2])$. Since $\mathrm{a}(\mathsf{d}_1) = \mathrm{a}(\mathsf{d}_2)$ implies $\mathsf{d}_1 = \mathsf{d}_2$, it follows that

$$\iota_\nu(p[x := \mathsf{d}]) = \iota_\nu(q[x := \mathsf{d}]) \text{ for all } \mathsf{d} \in \mathsf{D};$$

hence $\mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models p \approx q$.

($\Leftarrow$) If $\mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models p \approx q$ for all $\mathsf{d} \in \mathsf{D}$, then

$$\iota_\nu(\mathrm{a}(\mathsf{d})) \cdot \iota_\nu(p[x := \mathsf{d}]) = \iota_\nu(\mathrm{a}(\mathsf{d})) \cdot \iota_\nu(q[x := \mathsf{d}]),$$

so $\mathbf{T_D}(\mathcal{A}), \nu \models (\forall x)_1\langle p, q\rangle \approx (\forall x)_2\langle p, q\rangle$. □

Existential quantification generalises disjunction; the following example explains how disjunction is expressible.

Example 4.15 Suppose that t_1, t_2, u_1 and u_2 are trees. We wish to construct trees t and u such that $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ or $\mathsf{t}_2 = \mathsf{u}_2$. Let a_1, a_2 and c be distinct tree actions; we define $\mathsf{t} = \mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{t}_1 + \mathsf{a}_2 \cdot \mathsf{u}_2) + \mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{u}_1 + \mathsf{a}_2 \cdot \mathsf{t}_2)$ and $\mathsf{u} = \mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{t}_1 + \mathsf{a}_2 \cdot \mathsf{u}_2) + \mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{u}_1 + \mathsf{a}_2 \cdot \mathsf{t}_2) + \mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{t}_1 + \mathsf{a}_2 \cdot \mathsf{t}_2)$ (see Figure 4.2).

Clearly, $\mathsf{t} \leq \mathsf{u}$ and $\mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{t}_1 + \mathsf{a}_2 \cdot \mathsf{u}_2) + \mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{u}_1 + \mathsf{a}_2 \cdot \mathsf{t}_2) \leq \mathsf{t}$; so $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{c} \cdot (\mathsf{a}_1 \cdot \mathsf{t}_1 + \mathsf{a}_2 \cdot \mathsf{t}_2) \leq \mathsf{t}$. Hence, by Lemma 2.7(ii,vi), $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ or $\mathsf{t}_2 = \mathsf{u}_2$.

Let c be a closed action expression and let a be a unary parametrised action symbol; we define

$$(\exists x)_1\langle p, q\rangle = \textstyle\sum_x \mathrm{c}\,(\sum_x \mathrm{a}(x)p + \mathrm{a}(x)q)\,; \text{ and}$$
$$(\exists x)_2\langle p, q\rangle = (\exists x)_1\langle p, q\rangle + \mathrm{c}\,(\textstyle\sum_x \mathrm{a}(x)p)\,.$$

Note that in the definition of $(\exists x)_1\langle p, q\rangle$ the first (i.e., left-most) occurrence of $\sum_x$ binds the variable x in $\mathrm{a}(x)q$, while the second occurrence binds the variable x in $\mathrm{a}(x)p$. Intuitively, by executing c an instance $\mathrm{a}(\mathsf{d}) \cdot q[x := \mathsf{d}]$ of $\mathrm{a}(x)q$ is fixed, but from the execution of c it cannot be seen which particular element of D is selected. Compare this to the function of the tree action c in Figure 4.2: by executing c a choice is made between $\mathsf{a}_i \cdot \mathsf{t}_i$ and $\mathsf{a}_i \cdot \mathsf{u}_i$ for $i = 1, 2$.

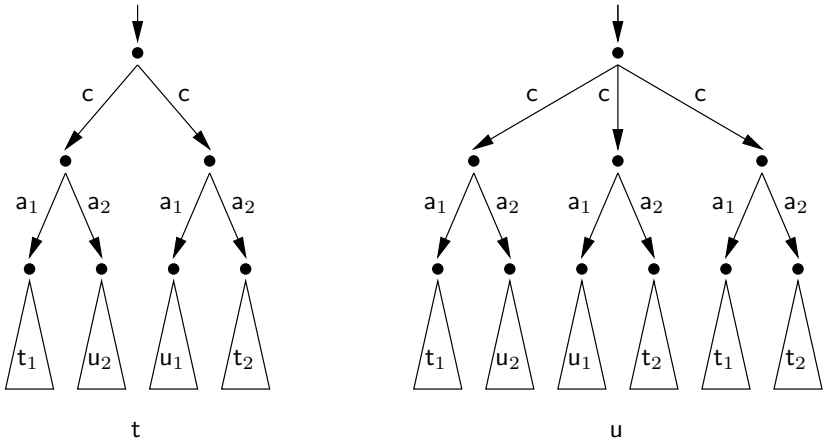


Figure 4.2: $\mathsf{t} = \mathsf{u}$ if, and only if, $\mathsf{t}_1 = \mathsf{u}_1$ or $\mathsf{t}_2 = \mathsf{u}_2$.

Lemma 4.16 (∃-introduction) If p and q are pCRL expressions, then

$$\mathbf{T_D}(\mathcal{A}), \nu \models (\exists x)_1\langle p, q\rangle \approx (\exists x)_2\langle p, q\rangle \text{ if, and only if,}$$
$$\text{there exists } \mathsf{d} \in \mathsf{D} \text{ such that } \mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models p \approx q.$$

Proof. Note that

$$\begin{aligned}
&\mathbf{T_D}(\mathcal{A}), \nu \models (\exists x)_1\langle p, q\rangle \approx (\exists x)_2\langle p, q\rangle \\
&\quad \Leftrightarrow \mathbf{T_D}(\mathcal{A}), \nu \models c(\textstyle\sum_x \mathrm{a}(x)p) \preccurlyeq \sum_x c(\sum_x \mathrm{a}(x)p + \mathrm{a}(x)q) \\
&\quad \Leftrightarrow \text{there exists } \mathsf{d} \in \mathsf{D} \text{ such that} \\
&\qquad\quad \mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models \textstyle\sum_x \mathrm{a}(x)p \approx \sum_x \mathrm{a}(x)p + \mathrm{a}(x)q \\
&\quad \Leftrightarrow \text{there exists } \mathsf{d} \in \mathsf{D} \text{ such that } \mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models \mathrm{a}(x)q \preccurlyeq \textstyle\sum_x \mathrm{a}(x)p
\end{aligned}$$

and, since $\mathrm{a}(\mathsf{d}_1) = \mathrm{a}(\mathsf{d}_2)$ if, and only if, $\mathsf{d}_1 = \mathsf{d}_2$,

$$\Leftrightarrow \text{there exists } \mathsf{d} \in \mathsf{D} \text{ such that } \mathbf{T_D}(\mathcal{A}), \nu[x := \mathsf{d}] \models p \approx q. \qquad \square$$

Theorem 4.17 There exists a one-one recursive function $\eta : \Phi \to \mathcal{P} \times \mathcal{P}$ such that for every first-order formula φ

$$\mathbf{D}, \nu \models \varphi \text{ if, and only if, } \mathbf{T_D}(\mathcal{A}), \nu \models p \approx q, \text{ where } \eta(\varphi) = \langle p, q\rangle$$

(provided there are at least a closed action expression and a parametrised action symbol with arity > 0).

Proof. Let φ be a prenex form; we define pCRL expressions $P(\varphi)$ and $Q(\varphi)$ as follows:

1. if the prefix of φ is empty, i.e., φ is an open formula, then $P(\varphi) = c \lhd \varphi \rhd \delta$ and $Q(\varphi) = c$, where c is a closed action expression;

2. if the prefix of φ begins with a universal quantifier, say $\varphi = (\forall x)\psi$, then

$$P(\varphi) = (\forall x)_1\langle P(\psi), Q(\psi)\rangle \text{ and } Q(\varphi) = (\forall x)_2\langle P(\psi), Q(\psi)\rangle; \text{ and}$$

3. if the prefix of φ begins with an existential quantifier, say $\varphi = (\exists x)\psi$, then

$$P(\varphi) = (\exists x)_1\langle P(\psi), Q(\psi)\rangle \text{ and } Q(\varphi) = (\exists x)_2\langle P(\psi), Q(\psi)\rangle.$$

By Lemmas 4.11, 4.14 and 4.16 and an easy induction on the length of the prefix of φ it follows that

$$\mathbf{D}, \nu \models \varphi \text{ if, and only if, } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models P(\varphi) \approx Q(\varphi).$$

To ensure that η is one-one, we use a recursive injection $\ulcorner \_ \urcorner : \Phi \to (\omega - \{0\})$ of Φ into the set of positive natural numbers; we define the function $\eta : \Phi \to \mathcal{P} \times \mathcal{P}$ by

$$\varphi \mapsto \langle P(\pi(\varphi)) + (\delta)^{\ulcorner\varphi\urcorner}, Q(\pi(\varphi))\rangle,$$
$$\text{where } (\delta)^1 = \delta \text{ and } (\delta)^{n+1} = \delta \cdot \delta^n \text{ for } n \geq 1.$$

Clearly, η satisfies the requirements of the theorem, so the proof is complete. □

By Theorem 4.10 the pCRL theory of $\mathbf{D}$ is one-one reducible to the first-order theory of $\mathbf{D}$, and Theorem 4.17 proves the converse. Hence, the pCRL theory and the first-order theory of $\mathbf{D}$ have the same degree of unsolvability with respect to one-one reducibility. By a theorem of Myhill (see Rogers, Jr., 1992) we get the following corollary.

Corollary 4.18 If $\mathbf{D}$ has equality, then the pCRL theory of $\mathbf{D}$ and the first-order theory of $\mathbf{D}$ are recursively isomorphic (provided there are at least a closed action expression and a parametrised action symbol with arity > 0).

4.4 A universal fragment

The choice quantifier is a powerful construct: it may be used to simulate both the universal and the existential quantifier of first-order logic. Indeed, the algorithm of Table 4.1 yields an open formula when applied to tree forms t and u without choice quantifiers, and with any open formula Lemma 4.11 associates a pCRL expression without choice quantifiers. The main application of choice quantifiers is to model input. We shall now investigate how much of the expressiveness of choice quantifiers persists if we *only* use it to model input.

In Section 3.6 we have introduced a fragment of value-passing CCS. We have associated with every process expression of that language a pCRL expression. Thus, value-passing CCS gives rise to a fragment of pCRL; a pCRL expression that is associated with some process expression of value-passing CCS we have called an

input/output expression. The *input/output theory* of $\mathbf{D}$ consists of all pCRL equations $p \approx q$, with p and q input/output expressions, such that $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models p \approx q$. We shall see below that the input/output theory of $\mathbf{D}$ is essentially less complex than the full pCRL theory of $\mathbf{D}$: it is recursively isomorphic to the universal fragment of the first-order theory of $\mathbf{D}$. We easily get a variant of Lemma 4.11.

Lemma 4.19 Suppose that φ is an open first-order formula, and let c be a closed output action. Then $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models (\varphi \to c) \approx c$ if, and only if, $\mathbf{D}, \nu \models \varphi$.

If p and q are input/output expressions, then $(\forall x)_1\langle p, q\rangle$ and $(\forall x)_2\langle p, q\rangle$ are also input/output expressions:

$$(\forall x)_1\langle p, q\rangle = \mathrm{a}?x.p; \text{ and}$$
$$(\forall x)_2\langle p, q\rangle = \mathrm{a}?x.q.$$

Hence, we have the following lemma.

Lemma 4.20 (∀-introduction) If p and q are input/output expressions, then

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models (\forall x)_1\langle p, q\rangle \approx (\forall x)_2\langle p, q\rangle \text{ if, and only if,}$$
$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[x := \mathsf{d}] \models p \approx q \text{ for all } \mathsf{d} \in \mathsf{D}.$$

A first-order formula is *universal* if it is in prenex form and all quantifiers in its prefix are universal; we denote by $\Phi_{\mathcal{U}}$ the set of universal formulas. From Lemmas 4.19 and 4.20 we straightforwardly get a variant of Theorem 4.17.

Theorem 4.21 There exists a one-one recursive function $\eta_{io} : \Phi_{\mathcal{U}} \to \mathcal{IO} \times \mathcal{IO}$ such that for every universal first-order formula φ

$$\mathbf{D}, \nu \models \varphi \text{ if, and only if, } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q, \text{ where } \eta_{io}(\varphi) = \langle p, q\rangle$$

(provided there is a closed output action and a parametrised action symbol with arity > 0).

The transformation $\langle(\exists x)_1, (\exists x)_2\rangle$ defined in Section 4.3 uses a distinct feature of the choice quantifier that is not expressible by means of an input prefix: the variable x, bound by the left-most choice quantifier in

$$\textstyle\sum_x c(\sum_x \mathrm{a}(x)p + \mathrm{a}(x)q)$$

does not occur in the action expression c that immediately follows it. Recall that, intuitively, by executing c an instance $\mathrm{a}(\mathsf{d}) \cdot q[x := \mathsf{d}]$ of $\mathrm{a}(x)q$ is fixed, but from the execution of c it cannot be seen which particular element of D is selected.

From Lemma 3.26 on p. 48 we get that if p is an input/output expression, then the ordered tree form $\theta_o(p)$ associated to p has explicit instantiation. We shall now prove that all existential quantifiers can be eliminated from the formula $\phi_{\preccurlyeq}(t, u)$ if t and u are ordered tree forms with explicit instantiation.

Theorem 4.22 Suppose that **D** has equality, and let t and u be ordered tree forms with explicit instantiation. Then there exists a universal first-order formula φ such that $\mathbf{D} \models \phi_{\preccurlyeq}(t,u) \leftrightarrow \varphi$.

Proof. We shall apply a few elementary results of first-order logic that are proved, e.g., by Shoenfield (1967); in particular we need the following results on quantifiers:

$$((\forall x)\varphi \wedge \psi) \leftrightarrow (\forall x)(\varphi \wedge \psi), \text{ provided that } x \notin \mathsf{FV}(\psi); \tag{4.13}$$
$$((\forall x)\varphi \vee \psi) \leftrightarrow (\forall x)(\varphi \vee \psi), \text{ provided that } x \notin \mathsf{FV}(\psi); \tag{4.14}$$
$$(\varphi \rightarrow (\forall x)\psi) \leftrightarrow (\forall x)(\varphi \rightarrow \psi), \text{ provided that } x \notin \mathsf{FV}(\varphi); \tag{4.15}$$
$$(\exists x)(\mathrm{eq}(x,d) \wedge \varphi) \leftrightarrow \varphi[x := d]. \tag{4.16}$$

The proof is by induction on $|t|+|u|$; we shall only do the induction step. Suppose $|t|+|u|>0$; we distinguish cases according to the form of t.
If $t=\delta$, then $\phi_{\preccurlyeq}(t,u)=\top$, which is a universal formula.
If $t=t'+t''$, then by the induction hypothesis $\phi_{\preccurlyeq}(t',u)$ and $\phi_{\preccurlyeq}(t'',u)$ are equivalent to universal first-order formulas, say $(\forall x_1)\ldots(\forall x_k)\varphi'$ and $(\forall y_1)\ldots(\forall y_l)\varphi''$. Without loss of generality we may assume that $x_i \neq y_j$, $x_i \notin \mathsf{FV}(\varphi'')$ and $y_j \notin \mathsf{FV}(\varphi')$, for all $1 \leq i \leq k$ and $1 \leq j \leq l$. Hence by (4.13)

$$\begin{aligned}\phi_{\preccurlyeq}(t,u) &= \phi_{\preccurlyeq}(t',u) \wedge \phi_{\preccurlyeq}(t'',u)\\ &\leftrightarrow (\forall x_1)\ldots(\forall x_k)\varphi' \wedge (\forall y_1)\ldots(\forall y_l)\varphi''\\ &\leftrightarrow (\forall x_1)\ldots(\forall x_k)(\forall y_1)\ldots(\forall y_l)(\varphi' \wedge \varphi'').\end{aligned}$$

In the two cases that remain t is a simple expression; we shall only treat the case that t has a continuation. Suppose $t = \sum_{\vec{x}} at' \lhd b \rhd \delta$ and let $u = u_1 + \cdots + u_m + u_{m+1} + \cdots + u_n$ with

$$u_i = \begin{cases} \sum_{\vec{x}_i} a_i \cdot u_i' \lhd b_i \rhd \delta & 1 \leq i \leq m;\\ \sum_{\vec{x}_i} a_i \lhd b_i \rhd \delta & m < i \leq n.\end{cases}$$

Then

$$\phi_{\preccurlyeq}(t,u) = (\forall \vec{x})\left(b \rightarrow \bigvee_{1\leq i\leq m} (\exists \vec{x}_i)\,(b_i \wedge \mathrm{eq}(a,a_i) \wedge \phi_{\preccurlyeq}(t',u_i') \wedge \phi_{\preccurlyeq}(u_i',t'))\right).$$

Now consider the subformula

$$(\exists \vec{x}_i)\,(b_i \wedge \mathrm{eq}(a,a_i) \wedge \phi_{\preccurlyeq}(t',u_i') \wedge \phi_{\preccurlyeq}(u_i',t'))\,.$$

By (4.14) and (4.15) it suffices to prove that it is equivalent to a universal formula. By the induction hypothesis $\phi_{\preccurlyeq}(t',u_i')$ and $\phi_{\preccurlyeq}(u_i',t')$ are equivalent to universal formulas, say $(\forall x_1)\ldots(\forall x_k)\varphi$ and $(\forall y_1)\ldots(\forall y_l)\psi$. If $|\vec{x}_i|=0$, then the theorem follows immediately from (4.13), and if $\mathrm{eq}(a,a_i)=\bot$, then the theorem follows since $(\exists x)\bot \leftrightarrow \bot$. Otherwise a and a_i are instances of the same parametrised

action symbol and, since u has explicit instantiation, $a_i = \mathrm{a}(\vec{x}_i)$. Let $a = \mathrm{a}(\vec{d})$, where $\vec{d}$ is a sequence of data expressions with $|\vec{x}_i| = |\vec{d}|$. Then,

$$\mathrm{eq}(a, a_i) = \mathrm{eq}(x_{i1}, d_1) \wedge \cdots \wedge \mathrm{eq}(x_{ik}, d_k),$$

whence by (4.16)

$$(\exists \vec{x}_i)\,(b_i \wedge \mathrm{eq}(a, a_i) \wedge \varphi \wedge \psi) \leftrightarrow (b_i[\vec{x}_i := \vec{d}] \wedge \varphi[\vec{x}_i := \vec{d}] \wedge \psi[\vec{x}_i := \vec{d}]).$$

From this the theorem follows, since by (4.13) the right-hand side is equivalent to a universal formula. □

Hence, the universal fragment of the first-order theory of **D** is one-one reducible to the input-output theory of **D**, and from Lemma 3.26 and Theorem 4.22 we get the converse. Hence, the input/output theory of **D** and the universal fragment of the first-order theory of **D** have the same degree of unsolvability with respect to one-one reducibility. Consequently, by a theorem of Myhill (see Rogers, Jr., 1992) we get the following

Corollary 4.23 If **D** has equality, then the input/output theory of **D** and the universal fragment of the first-order theory of **D** are recursively isomorphic (provided there exist a closed output action and a parametrised action symbol with arity > 0).

Bibliographic notes

Ponse (1996) investigated the complexity of another fragment of μCRL. He considers data algebras with recursive functions and relations, and, with respect to our fragment, he omits the choice quantifiers and includes data-parametric recursion. For (pairs of) specifications in this fragment he classifies a number of properties in the Arithmetical Hierarchy. In particular, he shows that, restricting to computable data, equivalence between two recursive specifications in his fragment is complete in Π^0_1. So, approximately, the contribution of data-parametric recursion to μCRL corresponds to the contribution of universal quantifiers to first-order logic.

Hennessy and Lin (1995) have already proved part of Corollary 4.23 for value-passing CCS, giving an algorithm that associates to each pair of finite value-passing processes a universal formula that holds if, and only if, the processes are bisimilar. Theorem 4.21 extends their result with the converse, that the universal quantifiers introduced by their algorithm cannot be eliminated.

There is a vast literature exploring the connection between process theory and modal logic (see Bradfield and Stirling (2001) and Stirling (2001) for recent accounts). The connection proceeds via labeled transition systems: a process can be viewed as a labeled transition system modulo bisimulation, a modal formula can be viewed as the specifation of a property of a state in a labeled transition system. Incidentally, a labeled transition system may be conceived as a first-order model, interpreting the transition relation as a family of binary relations indexed by the labels. This point of view gives rise to a correspondence between process theory

and first-order logic quite different from the one considered in this chapter. In this context, Hollenberg (1998) studies which operations on labeled transition systems are *first-order definable*, i.e., definable through a set of first-order formulas.

5

A deductive system for pCRL

Let us call a pCRL equation $p \approx q$ *valid* if $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models p \approx q$; e.g., the equations $\theta(p) \approx p$, where θ is the function of Section 3.5 which associates a tree form with every pCRL expression, are valid (cf. Lemma 3.22). To prove that a pCRL equation is valid, can be quite a laborious enterprise. The general technique is to presuppose an arbitrary valuation ν and an arbitrary element $\mathbf{P}$ of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$, and prove by means of the axioms of GBPA_δ's that $\iota_\nu(p) = \iota_\nu(q)$, where ι_ν is the interpretation homomorphism from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into $\mathbf{P}$ generated by ν. We illustrate this in the following example.

Example 5.1 Suppose that p denotes the pCRL expression

$$\textstyle\sum_x \mathrm{r}(x)\mathrm{s}(x) \lhd 0 \leq x \rhd \mathrm{r}(x)\mathrm{s}(-x)$$

from Example 3.14, and let q be the tree form

$$\textstyle\sum_x \mathrm{r}(x)\mathrm{s}(x) \lhd 0 \leq x \rhd \delta + \sum_x \mathrm{r}(x)\mathrm{s}(-x) \lhd \neg(0 \leq x) \rhd \delta$$

that we associated with it in Example 3.18. We want to prove that $p \approx q$ is valid. To this end, we fix an arbitrary valuation ν and an arbitrary element $\mathbf{P}$ of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$; then,

$$\begin{aligned}
&\iota_\nu(p) = \textstyle\sum(\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \mid \mathsf{n} \geq 0\} \cup \{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \mid \mathsf{n} < 0\}), \text{ and}\\
&\iota_\nu(q) = \textstyle\sum\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \mid \mathsf{n} \geq 0\} + \sum\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \mid \mathsf{n} < 0\}.
\end{aligned}$$

By (GA1),

$$\begin{aligned}
&\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \leq \iota_\nu(p) \text{ for all } \mathsf{n} \geq 0, \text{ and}\\
&\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \leq \iota_\nu(p) \text{ for all } \mathsf{n} < 0;
\end{aligned}$$

so by (GA2),

$$\begin{aligned}
&\textstyle\sum\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \mid \mathsf{n} \geq 0\} \leq \iota_\nu(p), \text{ and}\\
&\textstyle\sum\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \mid \mathsf{n} < 0\} \leq \iota_\nu(p);
\end{aligned}$$

hence, by (A2), $\iota_\nu(q) \leq \iota_\nu(p)$.

On the other hand, by (GA1),

$$\begin{aligned}
&\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \leq \textstyle\sum\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \mid \mathsf{n} \geq 0\} \text{ for all } \mathsf{n} \geq 0, \text{ and}\\
&\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \leq \textstyle\sum\{\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \mid \mathsf{n} < 0\} \text{ for all } \mathsf{n} < 0;
\end{aligned}$$

so, by (A1) and (A2),

$$\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(\mathsf{n})) \leq \iota_\nu(q) \text{ for all } \mathsf{n} \geq 0, \text{ and}$$
$$\iota_\nu(\mathrm{r}(\mathsf{n})\mathrm{s}(-\mathsf{n})) \leq \iota_\nu(q) \text{ for all } \mathsf{n} < 0;$$

hence, by (GA2), $\iota_\nu(p) \leq \iota_\nu(q)$.

From $\iota_\nu(q) \leq \iota_\nu(p)$ and $\iota_\nu(p) \leq \iota_\nu(q)$, we conclude, by (A1), that $\iota_\nu(p) = \iota_\nu(q)$. Hence, $p \approx q$ is valid.

It is easily verified that the specific form of the subexpressions $\mathrm{r}(x)\mathrm{s}(x)$ and $\mathrm{r}(x)\mathrm{s}(-x)$ is not relevant for the calculations in the above example. We have actually proved for all pCRL expressions p and q the validity of

$$\textstyle\sum_x p \lhd b \rhd q \approx \sum_x p \lhd b \rhd \delta + \sum_x q \lhd \neg\, b \rhd \delta. \tag{5.1}$$

This equation, in turn, is a consequence of two more general equations, which are also valid:

$$p \lhd b \rhd q \approx p \lhd b \rhd \delta + q \lhd \neg\, b \rhd \delta; \text{ and} \tag{5.2}$$
$$\textstyle\sum_x (p + q) \approx \sum_x p + \sum_x q. \tag{5.3}$$

To get (5.1), we first apply (5.2) to the subexpression $p \lhd b \rhd q$ of the left-hand side and replace it by $p \lhd b \rhd \delta + q \lhd \neg\, b \rhd \delta$ to obtain $\sum_x (p \lhd b \rhd \delta + q \lhd \neg\, b \rhd \delta)$; to justify this replacement, we postulate that for pCRL expressions p and q

$$p \approx q \text{ implies } \textstyle\sum_x p \approx \sum_x q. \tag{5.4}$$

Subsequently, we apply (5.3) to the expression $\sum_x (p \lhd b \rhd \delta + q \lhd \neg\, b \rhd \delta)$ to get the right-hand side of (5.1). If for all pCRL expressions p, q and r

$$p \approx q \text{ and } q \approx r \text{ implies } p \approx r, \tag{5.5}$$

then we may conclude (5.1).

Starting from (5.2) and (5.3), we have deduced (5.1) using (5.4) and (5.5). Of course, to conclude from this deduction that the pCRL expressions of Example 5.1 are equivalent under any interpretation in any element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$, we still need to apply the technique of Example 5.1 to verify that (5.2) and (5.3) are indeed valid, and that the validity of the antecedents of the implications (5.4) and (5.5) implies the validity of the conclusions. However, a verification of (5.2)–(5.5) has a much wider applicability than the verification in Example 5.1. It follows that not only (5.1) is valid for all pCRL expressions p and q, but also any other equation that can be deduced with (5.2)–(5.5).

In Chapter 3, we have introduced the language pCRL to describe elements of generalised basic process algebras with deadlock. In this chapter, we associate with it a *deductive system* to facilitate formal proofs of valid equations. We shall designate particular valid equations as *axioms* (e.g., (5.3)), and certain valid implications between equations as *inference rules* (e.g., (5.4) and (5.5)). Our inference rules correspond to the elementary steps of equational reasoning; e.g., (5.5) says

that $\approx$ is a transitive relation on pCRL expressions. Each of our axioms expresses a property of a construct or a combination of constructs of pCRL that we consider basic; e.g., (5.3) expresses that choice quantifiers distribute over alternative compositions. From the axioms we require a certain degree of generality. For instance, we shall see that (5.2) can be deduced from more general valid equations, and we shall take this as an argument not to include it as an axiom.

Intuitively, the proof that an equation $p \approx q$ is valid consists of three parts: a part that establishes some necessary properties of $\mathbf{D}$; a part that describes how these properties prove the validity of pCRL equations; and a part that involves reasoning about the constructs of pCRL, independent of any data occurring in p and q. The design of our deductive system reflects this trichotomy. Reasoning about data is delegated to a subsidiary deductive system $\mathcal{S}$ for data equations and for Boolean equations. Two simple laws explain how the provable equations of $\mathcal{S}$ give rise to valid pCRL equations: if $\mathcal{S}$ proves the data equation $d \approx e$ and a pCRL expression q is obtained from another pCRL expression p by replacing an occurrence of d by e, then $p \approx q$ is valid; and if $\mathcal{S}$ proves the Boolean equation $b \approx c$, then $p \lhd b \rhd q \approx p \lhd c \rhd q$ is valid.

The remaining axioms and rules of our deductive system are independent of specific properties of the data; they express certain fundamental properties of the constructs of pCRL. A deduction within our deductive system may thus be thought of as the explanation of a valid equation in terms of these fundamental properties, with an occasional reference to the subsidiary deductive system $\mathcal{S}$ for the explanation of a property of the data. Then, the question naturally arises whether *every* valid pCRL equation may be explained in this way, i.e., whether our deductive system is *complete*. Clearly, this depends to a large extent on the deductive power of the subsidiary system $\mathcal{S}$ for the data. We shall investigate completeness under the assumption that $\mathbf{D}$ is fixed and that $\mathcal{S}$ is powerful enough to infer any property of $\mathbf{D}$ that may be needed to establish the validity of a pCRL equation. Such a powerful enough $\mathcal{S}$ acts as an *oracle* for $\mathbf{D}$, and the result that we shall prove may be called *relative completeness*: our deductive system is complete if it may consult an oracle that answers any question about the data.

5.1 The deductive system

In the process specification language μCRL, on which the language pCRL is based, abstract data types are defined by means of many-sorted algebraic specifications. We wish to stay close to this, so we take many-sorted equational logic as a basis for our subsidiary deductive system for the data.

Definition 5.2 A *data specification* $\mathcal{S}$ is a two-sorted equational specification

$$\mathcal{S} = \langle \Delta \cup \Sigma_{\mathsf{B}}, \mathcal{E}_{\mathsf{D}}, \mathcal{E}_{\mathsf{B}} \rangle$$

with sorts D and B that consists of

(i) the language Δ of a data algebra, extended with the signature of Boolean

(BA1)	$a \vee (b \vee c)$	$\approx (a \vee b) \vee c$
(BA2)	$b \vee c$	$\approx c \vee b$
(BA3)	$b \vee (b \wedge c)$	$\approx b$
(BA4)	$a \vee (b \wedge c)$	$\approx (a \vee b) \wedge (a \vee c)$
(BA5)	$b \vee \neg b$	$\approx \top$
(BA1′)	$a \wedge (b \wedge c)$	$\approx (a \wedge b) \wedge c$
(BA2′)	$b \wedge c$	$\approx c \wedge b$
(BA3′)	$b \wedge (b \vee c)$	$\approx b$
(BA4′)	$a \wedge (b \vee c)$	$\approx (a \wedge b) \vee (a \wedge c)$
(BA5′)	$b \wedge \neg b$	$\approx \bot$

Table 5.1: The Boolean axioms that are included in every data specification.

algebras

$$\Sigma_{\mathsf{B}} = \{\vee, \wedge\colon \mathsf{B} \times \mathsf{B} \to \mathsf{B},\ \neg : \mathsf{B} \to \mathsf{B},\ \top, \bot :\to \mathsf{B}\};$$

(ii) a sequence $\mathcal{E}_{\mathsf{D}} = (d_1 \approx e_1), (d_2 \approx e_2), \ldots$ of data equations; and

(iii) a sequence $\mathcal{E}_{\mathsf{B}} = (b_1 \approx c_1), (b_2 \approx c_2), \ldots$ of Boolean equations such that $\mathcal{E}_{\mathsf{B}}$ at least contains all the instances of the axiom schemata in Table 5.1 with Boolean expressions for the meta variables a, b and c.

We shall call Δ the *language* of $\mathcal{S}$, the equations in $\mathcal{E}_{\mathsf{D}}$ are called the *data axioms* of $\mathcal{S}$ and the equations in $\mathcal{E}_{\mathsf{B}}$ are called the *Boolean axioms* of $\mathcal{S}$.

We write $\mathcal{S} \vdash d \approx e$ if the data equation $d \approx e$ can be deduced from the equations in $\mathcal{E}_{\mathsf{D}}$ and $\mathcal{E}_{\mathsf{B}}$ by means of many-sorted equational logic (cf., e.g., Goguen and Meseguer, 1985); and similarly, we write $\mathcal{S} \vdash b \approx c$ if the Boolean equation $b \approx c$ can be deduced from the equations in $\mathcal{E}_{\mathsf{D}}$ and $\mathcal{E}_{\mathsf{B}}$. It is easily verified from Definition 3.3 on p. 31 that the Boolean axioms generated by the schemata in Table 5.1 hold in every data algebra $\mathbf{D}$. Whenever $\mathcal{S}$ is a two-sorted equational specification with language Δ, $\mathbf{D}$ is a data algebra with the same language Δ, and *all* the axioms of $\mathcal{S}$ hold in $\mathbf{D}$, then $\mathbf{D}$ is called a *model* of $\mathcal{S}$. If $\mathbf{D}$ is a model of $\mathcal{S}$, then it follows from the soundness of many-sorted equational logic that

1. $\mathcal{S} \vdash d \approx e$ implies $\mathbf{D} \models d \approx e$ for all data expressions d and e; and
2. $\mathcal{S} \vdash b \approx c$ implies $\mathbf{D} \models b \approx c$ for all Boolean expressions b and c.

If also the converses of 1 and 2 hold, then we say that $\mathcal{S}$ is a *complete* (equational) specification of $\mathbf{D}$.

We define a *deductive system* $\Pi(\mathcal{A}, \mathcal{S})$, parametrised with a set of parametrised action symbols $\mathcal{A}$ and a data specification $\mathcal{S}$, as follows: the *axioms* of $\Pi(\mathcal{A}, \mathcal{S})$ are the instances of the axiom schemata listed in Table 5.2 with pCRL expressions for

$\mathcal{S}$-independent axiom schemata:

$$
\begin{array}{lll}
\text{(A1)} & p + q & \approx q + p \\
\text{(A2)} & p + (q + r) & \approx (p + q) + r \\
\text{(A3)} & p + p & \approx p \\
\text{(A4)} & (p + q) \cdot r & \approx p \cdot r + q \cdot r \\
\text{(A5)} & (p \cdot q) \cdot r & \approx p \cdot (q \cdot r) \\
\text{(A6)} & p + \delta & \approx p \\
\text{(A7)} & \delta \cdot p & \approx \delta
\end{array}
$$

$$
\begin{array}{lll}
\text{(C1)} & p \lhd \top \rhd q & \approx p \\
\text{(C2)} & p \lhd b \rhd q & \approx q \lhd \neg b \rhd p \\
\text{(C3)} & (p \lhd b \rhd q) \lhd c \rhd q & \approx p \lhd b \wedge c \rhd q \\
\text{(C4)} & (p + q) \lhd b \rhd (r + s) & \approx p \lhd b \rhd r + q \lhd b \rhd s \\
\text{(C5)} & p \lhd b \vee c \rhd \delta & \approx p \lhd b \rhd \delta + p \lhd c \rhd \delta \\
\text{(C6)} & (p \lhd b \rhd q) \cdot (r \lhd b \rhd s) & \approx p \cdot r \lhd b \rhd q \cdot s
\end{array}
$$

$$
\begin{array}{lllll}
\text{(CQ1)} & \sum_x p & \approx p & & \text{if } x \notin \mathsf{FV}(p) \\
\text{(CQ2)} & \sum_x p & \approx \sum_y p[x := y] & & \text{if } y \notin \mathsf{FV}(p) \\
\text{(CQ3)} & \sum_x p & \approx \sum_x p + p[x := d] & & \\
\text{(CQ4)} & \sum_x (p + q) & \approx \sum_x p + \sum_x q & & \\
\text{(CQ5)} & (\sum_x p) \cdot q & \approx \sum_x p \cdot q & & \text{if } x \notin \mathsf{FV}(q) \\
\text{(CQ6)} & \sum_x p \lhd b \rhd \sum_x q & \approx \sum_x (p \lhd b \rhd q) & & \text{if } x \text{ does not occur in } b
\end{array}
$$

$\mathcal{S}$-induced axiom schemata:

$$
\begin{array}{lll}
\text{(DATA)} & p[x := d] \approx p[x := e] & \text{if } \mathcal{S} \vdash d \approx e \\
\text{(BOOL)} & p \lhd b \rhd q \approx p \lhd c \rhd q & \text{if } \mathcal{S} \vdash b \approx c
\end{array}
$$

Inference rule schemata:

$$
\text{(REFL)}\ \frac{}{p \approx p} \qquad \text{(SYM)}\ \frac{p \approx q}{q \approx p} \qquad \text{(TRANS)}\ \frac{p \approx q,\ q \approx r}{p \approx r}
$$

$$
\text{(CONG}_{(+)}\text{)}\ \frac{p_1 \approx q_1,\ p_2 \approx q_2}{p_1 + p_2 \approx q_1 + q_2} \qquad \text{(CONG}_{(\cdot)}\text{)}\ \frac{p_1 \approx q_1,\ p_2 \approx q_2}{p_1 \cdot p_2 \approx q_1 \cdot q_2}
$$

$$
\text{(CONG}_{(\lhd b \rhd)}\text{)}\ \frac{p_1 \approx q_1,\ p_2 \approx q_2}{p_1 \lhd b \rhd p_2 \approx q_1 \lhd b \rhd q_2} \qquad \text{(CONG}_{(\sum_x)}\text{)}\ \frac{p \approx q}{\sum_x p \approx \sum_x q}
$$

Table 5.2: The deductive system for pCRL with respect to a data specification $\mathcal{S}$; p, q and r range over pCRL expressions; x is a data variable, d and e range over data expressions and b and c range over Boolean expressions.

the meta variables p, q, etc.; the *inference rules* are the instances of the inference rule schemata listed in Table 5.2. Formally, a *deduction* (within $\Pi(\mathcal{A},\mathcal{S})$) is a finite sequence of pCRL equations

$$(p_1 \approx q_1), (p_2 \approx q_2), \ldots, (p_n \approx q_n)$$

such each $p_i \approx q_i$ is either an axiom of $\Pi(\mathcal{A},\mathcal{S})$ or the conclusion of an inference rule of which the premisses occur earlier in the sequence. We call the last equation $p_n \approx q_n$ of a deduction its *conclusion*, and we write $\Pi(\mathcal{A},\mathcal{S}) \vdash p \approx q$ if the axioms and the inference rules in Table 5.2 permit a deduction that has the equation $p \approx q$ as conclusion; we shall then also say that p and q are *provably equivalent*. If it holds that p and q are provably equivalent only if $p \approx q$ is valid, then $\Pi(\mathcal{A},\mathcal{S})$ is called *sound* with respect to $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$. Before we prove that our deductive system is sound with respect to $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$ whenever $\mathbf{D}$ is a model of $\mathcal{S}$, we make a few comments about the axioms and the inference rules, and we give a few deductions to illustrate their use.

The inference rules are the well-known laws of equational reasoning, adapted to our setting. Namely, (Refl)–(Trans) ensure that provable equivalence is indeed an equivalence relation on pCRL expressions, and ($\text{Cong}_{(+)}$)–($\text{Cong}_{(\sum_x)}$) allow the inference of $p \approx q$ if q can be obtained from p by replacing a subexpression of p by a provably equivalent expression.

Remark 5.3 Our presentation is nonstandard in that it omits the so-called "substitution rules." Note, however, that our definition of pCRL expression does not involve a notion of "variable" for which one might want to substitute another pCRL expression. It is folklore that, in general, if one is only interested in "ground" terms, i.e., in terms without this kind of variables, then it is possible to do without substitution rules by taking all instances with ground terms of the axioms.

By (A1)–(A7), the set of pCRL expressions modulo provable equivalence is a BPA_δ (cf. Table 2.1 on p. 17); we write $\Pi(\mathcal{A},\mathcal{S}) \vdash p \preccurlyeq q$ for $\Pi(\mathcal{A},\mathcal{S}) \vdash q \approx q + p$.

Lemma 5.4 $\Pi(\mathcal{A},\mathcal{S}) \vdash p \approx q$ if, and only if, $\Pi(\mathcal{A},\mathcal{S}) \vdash p \preccurlyeq q,\ q \preccurlyeq p$.

Proof. To prove the implication from left to right, consider a deduction

$$(p_1 \approx q_1), \ldots, (p_n \approx q_n) = (p \approx q) \tag{5.6}$$

that justifies writing $\Pi(\mathcal{A},\mathcal{S}) \vdash p \approx q$. We append to it the sequence

$$(p \approx p), (p + p \approx p + q), (p + p \approx p), (p \approx p + p), (p \approx p + q). \tag{5.7}$$

Since $p \approx p$ by (Refl), $p+p \approx p+q$ can be inferred with ($\text{Cong}_{(+)}$). Furthermore, by (A3), $p + p \approx p$, so, by (Sym), $p \approx p + p$, and, subsequently, $p \approx p + q$ can be inferred with (Trans). Hence, the concatenation of (5.6) and (5.7) is a deduction that justifies $\Pi(\mathcal{A},\mathcal{S}) \vdash q \preccurlyeq p$. If we interchange p and q in the sequence (5.7) and moreover prefix it with the equation $q \approx p$, which can be inferred from $p \approx q$ by (Sym), we get a deduction that justifies $\Pi(\mathcal{A},\mathcal{S}) \vdash p \preccurlyeq q$. We conclude that $\Pi(\mathcal{A},\mathcal{S}) \vdash p \approx q$ implies $\Pi(\mathcal{A},\mathcal{S}) \vdash p \preccurlyeq q$ and $\Pi(\mathcal{A},\mathcal{S}) \vdash q \preccurlyeq p$.

To prove the implication from right to left, suppose there are deductions with conclusions $p \approx p + q$ and $q \approx q + p$. We concatenate these deductions and subsequently append the sequence

$$(p + q \approx q + p), (p \approx q + p), (q + p \approx q), (p \approx q).$$

From $p \approx p + q$ and the equation $p + q \approx q + p$, which is by (A1), we infer $p \approx q + p$ by (TRANS). From $q \approx q + p$ we get $q + p \approx q$ by (SYM). Hence, by (TRANS), $p \approx q$. We conclude that $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \preccurlyeq q$ and $\Pi(\mathcal{A}, \mathcal{S}) \vdash q \preccurlyeq p$ implies $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \approx q$. □

We have given a very detailed proof of Lemma 5.4 to demonstrate a precise application of our deductive system. Henceforth, we shall sacrifice some detail for the sake of succinctness, leaving out all references to applications of inference rules; e.g., to prove the implication from right to left of Lemma 5.4 we confine ourselves to assuming (*) $p \approx p + q$ and (†) $q \approx q + p$, and giving the following derivation:

$$\begin{aligned} p &\approx p + q && \text{by (*)} \\ &\approx q + p && \text{by (A1)} \\ &\approx q && \text{by (†).} \end{aligned}$$

Note that we have proved that $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \preccurlyeq q,\ q \preccurlyeq p$ implies $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \approx q$ without using any special properties of the deductions of $p \preccurlyeq q$ and $q \preccurlyeq p$. That is, the implication may be considered as a derived inference rule, sanctioned by the axioms and inference rules of $\Pi(\mathcal{A}, \mathcal{S})$. Later, we shall add axioms to $\Pi(\mathcal{A}, \mathcal{S})$, and then it is convenient to know that we may still deduce $p \approx q$ from $p \preccurlyeq q$ and $q \preccurlyeq p$. We write

$$\Pi(\mathcal{A}, \mathcal{S}), p_1 \approx q_1, \ldots, p_n \approx q_n \vdash p \approx q$$

if $p \approx q$ may be deduced in the deductive system that consists of the axioms and inference rules of $\Pi(\mathcal{A}, \mathcal{S})$ with $p_1 \approx q_1, \ldots, p_n \approx q_n$ added as axioms.

(C1)–(C5) are adapted from a paper by Manes (1985) about the equational theory of abelian monoids extended with an if-then-else construct (every generalised basic process algebra with deadlock is an abelian monoid with $+$ as binary operation and δ as neutral element). These axioms express fundamental relations between the conditional on the one hand, and the Boolean operations $\top$, $\neg$, $\wedge$ and $\vee$, and the operations $+$ and δ on the other hand. We have added (C6), which defines the interaction between conditionals and sequential composition. We shall now derive (5.2), which we used in the deduction following Example 5.1, with these axioms; the deduction is due to Manes.

Lemma 5.5 $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \lhd b \rhd q \approx p \lhd b \rhd \delta + q \lhd \neg b \rhd \delta$.

Proof. The proof is by the following deduction:

$$\begin{aligned} p \lhd b \rhd q &\approx (p + \delta) \lhd b \rhd (\delta + q) && \text{by (A1), (A6)} \\ &\approx p \lhd b \rhd \delta + \delta \lhd b \rhd q && \text{by (C4)} \\ &\approx p \lhd b \rhd \delta + q \lhd \neg b \rhd \delta && \text{by (C2).} \end{aligned}$$

□

Conditionals of the form $p \lhd b \rhd \delta$ correspond to Dijkstra's "guarded commands" (see Dijkstra, 1976); in such a conditional the Boolean expression b may be viewed as a *guard* for p. Guards distribute over alternative and sequential compositions.

Lemma 5.6 (i) $\Pi(\mathcal{A}, \mathcal{S}) \vdash (p + q) \lhd b \rhd \delta \approx (p \lhd b \rhd \delta) + (q \lhd b \rhd \delta)$; and

(ii) $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \cdot q \lhd b \rhd \delta \approx (p \lhd b \rhd \delta) \cdot (q \lhd b \rhd \delta)$.

Proof. By (A6), $(p + q) \lhd b \rhd \delta \approx (p + q) \lhd b \rhd (\delta + \delta)$, so (i) follows by (C4). The proof of (ii) is similar, using (A7) and (C6). □

(CQ1)–(CQ6) reflect a few properties of quantification in general, and of choice quantification in particular. If x does not occur free in p, then the quantifier $\sum_x$ has no effect on p; this is expressed by (CQ1). By (CQ2) we may rename the bound variable x in the expression $\sum_x p$ to y if y does not already occur free in p. Data expressions refer to elements of the domain over which $\sum_x$ quantifies, so (CQ3) reflects the intuition that any instance $p[x := d]$ of p is a summand of $\sum_x p$ (cf. (GA1) in Table 2.2 on p. 19). According to (CQ4), a choice quantifier distributes over alternative composition. According to (CQ5) sequential composition distributes from the right over choice quantification, provided $\sum_x$ has no effect on the second argument of the sequential composition (cf. (GA3) in Table 2.2). According to (CQ6), a choice quantifier $\sum_x$ distributes over conditionals if x does not occur free in the condition.

Lemma 5.7 $\Pi(\mathcal{A}, \mathcal{S}), p \preccurlyeq q \vdash \sum_x p \preccurlyeq \sum_x q$.

Proof. If (*) $p \approx p + q$, then we have the deduction

$$\begin{aligned} \textstyle\sum_x p &\approx \textstyle\sum_x (p + q) && \text{by (*)} \\ &\approx \textstyle\sum_x p + \sum_x q && \text{by (CQ4);} \end{aligned}$$

this proves the lemma. □

(DATA) and (BOOL) import into our deductive system for pCRL all the data equations and all the Boolean equations that can be deduced from the subsidiary data specification $\mathcal{S}$. If q can be obtained from p by replacing an occurrence in p of the data expression d by another data expression e such that $\mathcal{S} \vdash d \approx e$, then, according to (DATA), $p \approx q$ is an axiom. According to (BOOL), conditionals are equivalent if $\mathcal{S}$ proves the conditions equivalent. The proof of the following lemma shows an application of (BOOL).

Lemma 5.8 $\Pi(\mathcal{A}, \mathcal{S}) \vdash p \lhd b \rhd p \approx p$.

Proof. Since $\mathcal{S} \vdash b \vee \neg b \approx \top$ (it is an instance of BA5 of Table 5.1), we have the following deduction:

$$\begin{aligned} p \lhd b \rhd p &\approx p \lhd b \rhd \delta + p \lhd \neg b \rhd \delta && \text{by Lem. 5.5} \\ &\approx p \lhd b \vee \neg b \rhd \delta && \text{by (C5)} \\ &\approx p \lhd \top \rhd \delta && \text{by (BOOL)} \\ &\approx p && \text{by (C1);} \end{aligned}$$

this proves the lemma. □

We may use the above lemma to prove that sequential compositions distribute from the right over guarded commands.

Lemma 5.9 $\Pi(\mathcal{A},\mathcal{S}) \vdash p \cdot q \lhd b \rhd \delta \approx (p \lhd b \rhd \delta) \cdot q$.

Proof. Since $\mathcal{S} \vdash b \vee \neg b \approx \top$ (it is an instance of BA5 of Table 5.1), we have the following deduction:

$$\begin{aligned} p \cdot q \lhd b \rhd \delta &\approx p \cdot q \lhd b \rhd \delta \cdot q && \text{by (A7)} \\ &\approx (p \lhd b \rhd \delta) \cdot (q \lhd b \rhd q) && \text{by (C6)} \\ &\approx (p \lhd b \rhd \delta) \cdot q && \text{by Lem. 5.8;} \end{aligned}$$

this proves the lemma. □

We shall now show that if pCRL expressions p and q are provably equivalent, then $p \approx q$ is valid. We need the following lemma.

Lemma 5.10 The relation

$$\vartheta = \{\langle p, q\rangle \in \mathsf{Pol}_{\mathcal{P}}(\mathsf{D}) \times \mathsf{Pol}_{\mathcal{P}}(\mathsf{D}) \mid \mathsf{GBPA}_{\delta}(\mathcal{A},\mathbf{D}) \models p \approx q\}$$

respects the constructs of pCRL.

Proof. Since the interpretation mappings ι_ν are homomorphisms from $\mathbf{Pol}(\mathcal{A},\mathbf{D})$ into elements of $\mathsf{GBPA}_{\delta}(\mathcal{A},\mathbf{D})$, it is routine to show that ϑ is a congruence on $\mathbf{Pol}(\mathcal{A},\mathbf{D})$. So, it remains to prove that ϑ preserves choice quantifiers and conditionals. Suppose $p\ \vartheta\ q$. Then $p[x := \mathsf{d}]\ \vartheta\ q[x := \mathsf{d}]$ for all $\mathsf{d} \in \mathsf{D}$, so

$$\textstyle\sum_x p = \sum\{p[x := \mathsf{d}] \mid \mathsf{d} \in \mathsf{D}\}\ \vartheta\ \sum\{q[x := \mathsf{d}] \mid \mathsf{d} \in \mathsf{D}\} = \sum_x q.$$

Hence, ϑ preserves choice quantifiers.

To prove that ϑ preserves conditionals, suppose that $p_1\ \vartheta\ q_1$ and $p_2\ \vartheta\ q_2$. Let ν be a valuation and let ι_ν be the interpretation homomorphism from $\mathbf{Pol}(\mathcal{A},\mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_{\delta}(\mathcal{A},\mathbf{D})$; it suffices to show that

$$\iota_\nu(p_1 \lhd \mathit{bpol} \rhd p_2) = \iota_\nu(q_1 \lhd \mathit{bpol} \rhd q_2).$$

We distinguish cases:
If $\bar{\nu}(\mathit{bpol}) = \top$, then $\iota_\nu(p_1 \lhd \mathit{bpol} \rhd p_2) = \iota_\nu(p_1) = \iota_\nu(q_1) = \iota_\nu(q_1 \lhd \mathit{bpol} \rhd q_2)$.
If $\bar{\nu}(\mathit{bpol}) = \bot$, then $\iota_\nu(p_1 \lhd \mathit{bpol} \rhd p_2) = \iota_\nu(p_2) = \iota_\nu(q_2) = \iota_\nu(q_1 \lhd \mathit{bpol} \rhd q_2)$.
Hence, ϑ respects conditionals. □

We are now in a position to prove the soundness of our deductive system.

Theorem 5.11 Let $\mathcal{S}$ be a data specification, and let $\mathbf{D}$ be a model of $\mathcal{S}$. Then, for all pCRL expressions p and q, $\Pi(\mathcal{A},\mathcal{S}) \vdash p \approx q$ implies $\mathsf{GBPA}_{\delta}(\mathcal{A},\mathbf{D}) \models p \approx q$.

Proof. Consider the relation

$$\vartheta = \{\langle p, q\rangle \in \mathsf{Pol}_{\mathcal{P}}(\mathsf{D}) \times \mathsf{Pol}_{\mathcal{P}}(\mathsf{D}) \mid \mathsf{GBPA}_{\delta}(\mathcal{A},\mathbf{D}) \models p \approx q\}.$$

By Lemma 5.10 it preserves the constructs of pCRL, so it is closed under the rules in Table 5.2. Therefore, it remains to prove that $\langle p, q\rangle \in \vartheta$ if $p \approx q$ is an instance of an axiom in Table 5.2. For this we fix an arbitrary valuation ν and the interpretation homomorphism ι_ν associated with ν from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$, and we prove that $\iota_\nu(p) = \iota_\nu(q)$.

Since every element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$ is a generalised basic process algebra with deadlock, it is clear that $\iota_\nu(p) = \iota_\nu(q)$ if $p \approx q$ is an instance of one of (A1)–(A7).

To show that $\iota_\nu(p) = \iota_\nu(q)$ if $p \approx q$ is an instance of one of (C1)–(C6), one distinguishes cases according to whether the conditions evaluate to $\top$ or to $\bot$ under ν. We consider, by way of example, the instances of (C5) and (C6):

(C5) If $\bar\nu(b) = \top$ and $\bar\nu(c) = \top$, then $\bar\nu(b \vee c) = \top$; hence, since $+$ is idempotent in every generalised basic process algebra with deadlock,

$$\begin{aligned}\iota_\nu(p \lhd b \vee c \rhd \delta) &= \iota_\nu(p)\\ &= \iota_\nu(p) + \iota_\nu(p)\\ &= \iota_\nu(p \lhd b \rhd \delta + p \lhd c \rhd \delta).\end{aligned}$$

If $\bar\nu(b) = \top$ and $\bar\nu(c) = \bot$, then $\bar\nu(b \vee c) = \top$; hence, since $\iota_\nu(\delta) = \delta$ is a neutral element for $+$ in every generalised basic process algebra with deadlock,

$$\begin{aligned}\iota_\nu(p \lhd b \vee c \rhd \delta) &= \iota_\nu(p)\\ &= \iota_\nu(p) + \iota_\nu(\delta)\\ &= \iota_\nu(p \lhd b \rhd \delta + p \lhd c \rhd \delta).\end{aligned}$$

The case where $\bar\nu(b) = \bot$ and $\bar\nu(c) = \top$ is symmetric to the previous case.

If $\bar\nu(b) = \bot$ and $\bar\nu(c) = \bot$, then $\bar\nu(b \vee c) = \bot$; hence, since $+$ is idempotent in every generalised basic process algebra with deadlock,

$$\iota_\nu(p \lhd b \vee c \rhd \delta) = \iota_\nu(\delta) = \iota_\nu(\delta) + \iota_\nu(\delta) = \iota_\nu(p \lhd b \rhd \delta + p \lhd c \rhd \delta).$$

(C6) If $\bar\nu(b) = \top$, then

$$\begin{aligned}\iota_\nu((p \lhd b \rhd q) \cdot (r \lhd b \rhd s)) &= \iota_\nu(p) \cdot \iota_\nu(r)\\ &= \iota_\nu(p \cdot r)\\ &= \iota_\nu(p \cdot r \lhd b \rhd q \cdot s).\end{aligned}$$

If $\bar\nu(b) = \bot$, then

$$\begin{aligned}\iota_\nu((p \lhd b \rhd q) \cdot (r \lhd b \rhd s)) &= \iota_\nu(q) \cdot \iota_\nu(s)\\ &= \iota_\nu(q \cdot s)\\ &= \iota_\nu(p \cdot r \lhd b \rhd q \cdot s).\end{aligned}$$

To show that $\iota_\nu(p) = \iota_\nu(q)$ if $p \approx q$ is an instance of one of (CQ1)–(CQ6), one employs the axioms (GA1)–(GA3) of generalised basic process algebras with deadlock. Again by way of example, we consider the instances of (CQ3) and (CQ5):

(CQ3) We get by induction on the structure of pCRL polynomials that

$$(*)\ \iota_\nu(p[x := \bar\nu(d)]) = \iota_\nu(p[x := d]).$$

It enables us to make the following derivation:

$$\begin{aligned}\iota_\nu(\textstyle\sum_x p) &= \textstyle\sum\{\iota_\nu(p[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\}\\ &= \textstyle\sum\{\iota_\nu(p[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} + \iota_\nu(p[x := \bar\nu(d)]) \quad \text{by (GA1)}\end{aligned}$$

($\bar\nu(d)$ is, after all, an element of D)

$$\begin{aligned}&= \textstyle\sum\{\iota_\nu(p[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} + \iota_\nu(p[x := d]) \qquad \text{by (*)}\\ &= \iota_\nu(\textstyle\sum_x p + p[x := d]).\end{aligned}$$

(CQ5) If $x \notin \mathsf{FV}(q)$, then (*) $\iota_\nu(q[x := \mathsf{d}]) = \iota_\nu(q)$ for all $\mathsf{d} \in \mathsf{D}$, so

$$\begin{aligned}\iota_\nu((\textstyle\sum_x p) \cdot q) &= \textstyle\sum\{\iota_\nu(p[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} \cdot \iota_\nu(q)\\ &= \textstyle\sum\{\iota_\nu(p[x := \mathsf{d}]) \cdot \iota_\nu(q) \mid \mathsf{d} \in \mathsf{D}\} \qquad \text{by (GA3)}\\ &= \textstyle\sum\{\iota_\nu(p[x := \mathsf{d}]) \cdot \iota_\nu(q[x := \mathsf{d}]) \mid \mathsf{d} \in \mathsf{D}\} \quad \text{by (*)}\\ &= \iota_\nu(\textstyle\sum_x p \cdot q).\end{aligned}$$

For (DATA), suppose that d and e are data expressions such that $\mathcal{S} \vdash d \approx e$; then, since $\mathbf{D}$ is a model of $\mathcal{S}$, $\bar\nu(d) = \bar\nu(e)$, and hence

$$\iota_\nu(p[x := d]) = \iota_\nu(p[x := \bar\nu(d)]) = \iota_\nu(p[x := \bar\nu(e)] = \iota_\nu(p[x := e]).$$

For (BOOL), suppose that b and c are Boolean expressions such that $\mathcal{S} \vdash b \approx c$; we distinguish cases: if $\bar\nu(b) = \top$, then, since $\mathbf{D}$ is a model of $\mathcal{S}$, also $\bar\nu(c) = \top$, so

$$\iota_\nu(p \lhd b \rhd q) = \iota_\nu(p) = \iota_\nu(p \lhd c \rhd q);$$

otherwise $\bar\nu(b) = \bot$, whence also $\bar\nu(c) = \bot$, so

$$\iota_\nu(p \lhd b \rhd q) = \iota_\nu(q) = \iota_\nu(p \lhd c \rhd q).$$

The proof is of the theorem is now complete. □

5.2 Tree forms revisited

By Theorem 5.11, the deductive system $\Pi(\mathcal{A}, \mathcal{S})$ may be used to circumvent the technique of Example 5.1, and to give a formal proof of the validity of a pCRL equation. We shall now illustrate this by giving a formal proof of the result, obtained in Section 3.5, that each pCRL expression is equivalent to the tree form associated to it by the function θ. First, we present three lemmas in which we derive the correctness of the auxiliary functions θ_{seq}, θ_{cnd} and θ_{sum}.

Lemma 5.12 $\Pi(\mathcal{A}, \mathcal{S}) \vdash \theta_{\mathrm{seq}}(t, u) \approx t \cdot u$ for all tree forms t and u.

Proof. By (CQ2) we may assume, without loss of generality, that the bound variables in t are distinct from the free variables in u[1]; we show by induction on t that $\theta_{\mathrm{seq}}(t,u) \approx t \cdot u$ is derivable.
If $t = \delta$, then $\theta_{\mathrm{seq}}(t,u) = \delta \approx t \cdot u$ by (A7).
If $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$, with $t^* = a$ or $t^* = at'$ for some action expression a and some tree form t', then, by the induction hypothesis and (A5), $a \cdot \theta_{\mathrm{seq}}(t',u) \approx (at') \cdot u$; hence

$$\begin{aligned}
\theta_{\mathrm{seq}}(t,u) &\approx \textstyle\sum_{\vec{x}} t^* u \lhd b \rhd \delta \\
&\approx \textstyle\sum_{\vec{x}} (t^* \lhd b \rhd \delta) u && \text{by Lem. 5.9} \\
&\approx t \cdot u && \text{by (CQ5).}
\end{aligned}$$

If $t = t' + t''$, then $\theta_{\mathrm{seq}}(t,u) \approx t \cdot u$ by the induction hypothesis and (A4). □

Lemma 5.13 $\Pi(\mathcal{A},\mathcal{S}) \vdash \theta_{\mathrm{cnd}}(t,b) \approx t \lhd b \rhd \delta$ for every tree form t and every Boolean expression b.

Proof. By (CQ2) we may assume, without loss of generality, that the bound variables in t are distinct from the variables in b; we show by induction on t that $\theta_{\mathrm{cnd}}(t,b) \approx t \lhd b \rhd \delta$ is derivable.
If $t = \delta$, then $\theta_{\mathrm{cnd}}(t,b) = \delta \approx \delta \lhd b \rhd \delta$ by Lemma 5.8.
If $t = \sum_{\vec{x}} t^* \lhd c \rhd \delta$, then

$$\begin{aligned}
\theta_{\mathrm{cnd}}(t,b) &\approx \textstyle\sum_{\vec{x}} (t^* \lhd c \rhd \delta) \lhd b \rhd \delta && \text{by (C3)} \\
&\approx t \lhd b \rhd \textstyle\sum_{\vec{x}} \delta && \text{by (CQ6)} \\
&\approx t \lhd b \rhd \delta && \text{by (CQ1).}
\end{aligned}$$

If $t = t' + t''$, then

$$\begin{aligned}
\theta_{\mathrm{cnd}}(t,b) &\approx t' \lhd b \rhd \delta + t'' \lhd b \rhd \delta && \text{by (IH)} \\
&\approx (t' + t'') \lhd b \rhd (\delta + \delta) && \text{by (C4)} \\
&\approx t \lhd b \rhd \delta && \text{by (A6).}
\end{aligned}$$

This completes the proof of the lemma. □

Lemma 5.14 $\Pi(\mathcal{A},\mathcal{S}) \vdash \theta_{\mathrm{sum}}(x,t) \approx \sum_x t$ for every tree form t and variable x.

Proof. We show by induction on t that $\theta_{\mathrm{sum}}(x,t) \approx \sum_x t$ is derivable.
If $t = \delta$, then $\theta_{\mathrm{sum}}(x,t) = \delta \approx \sum_x t$ by (CQ1).
If t is a simple tree form, then $\theta_{\mathrm{sum}}(x,t) = \sum_x t$ by definition.
If $t = t' + t''$, then $\theta_{\mathrm{sum}}(x,t) \approx \sum_x t$ by the induction hypothesis and (CQ4). □

[1]Strictly speaking, the renamings that are needed to achieve this ought to be incorporated in the definition of θ_{seq}, and (CQ2) justifies such a modification.

Now, we can also derive the correctness of θ.

Lemma 5.15 (Tree forms) $\Pi(\mathcal{A},\mathcal{S}) \vdash \theta(p) \approx p$ for every pCRL expression p.

Proof. We prove the lemma by structural induction on p.
If $p = \delta$, then $\theta(p) = p$ by definition.
If p is an action expression, then $\theta(p) = p \lhd \top \rhd \delta \approx p$ by (C1).
If $p = p' + p''$, then $\theta(p) = \theta(p') + \theta(p'') \approx p$ by the induction hypothesis.
If $p = p' \cdot p''$, then $\theta(p) \approx p$ by Lemma 5.12 and the induction hypothesis.
If $p = p' \lhd b \rhd p''$, then

$$\begin{aligned}
\theta(p) &\approx \theta(p') \lhd b \rhd \delta + \theta(p'') \lhd \neg b \rhd \delta && \text{by Lem. 5.13}\\
&\approx p' \lhd b \rhd \delta + p'' \lhd \neg b \rhd \delta && \text{by (IH)}\\
&\approx p' \lhd b \rhd p'' && \text{by Lem. 5.5.}
\end{aligned}$$

If $p = \sum_x p'$, then $\theta(p) \approx p$ by Lemma 5.14 and the induction hypothesis. □

5.3 Relative completeness

The main purpose of our deductive system $\Pi(\mathcal{A},\mathcal{S})$ is to formalise those parts of validity proofs that are independent of specific properties of $\mathbf{D}$. Reasoning about $\mathbf{D}$ is entirely delegated to the subsidiary system $\mathcal{S}$, and incorporated in a straightforward manner via (DATA) and (BOOL). The previous section shows a typical application of $\Pi(\mathcal{A},\mathcal{S})$: the lion's share of the deduction consists of applications of $\mathcal{S}$-independent axioms; in the deduction of Lemma 5.9, which is used in Lemma 5.12, a property of $\mathbf{D}$ is needed, namely that $b \vee \neg b \approx \top$ for every Boolean expression b, and it is incorporated via (BOOL). Mindful of its particular purpose, we shall now analyse the adequacy of our deductive system.

5.3.1 Data requirements

First, we formulate three requirements on $\mathcal{S}$ and $\mathbf{D}$, which together ensure that $\mathcal{S}$ is sufficiently powerful to infer all the properties of $\mathbf{D}$ needed to establish that a pCRL equation is valid.

Clearly, if a is a unary parametrised action symbol and d and e are data expressions such that $\mathbf{D} \models d \approx e$, then $\mathrm{a}(d) \approx \mathrm{a}(e)$ is valid. Similarly, if a is an action expression and b and c are Boolean expressions such that $\mathbf{D} \models b \approx c$, then $a \lhd b \rhd \delta \approx a \lhd c \rhd \delta$ is valid. This brings us to the first requirement:

(I) $\mathcal{S}$ must be a *complete* specification of $\mathbf{D}$.

Note that if $\mathcal{S}$ and $\mathcal{S}'$ are both complete specifications of $\mathbf{D}$, then the deductive systems $\Pi(\mathcal{A},\mathcal{S})$ and $\Pi(\mathcal{A},\mathcal{S}')$ are equally powerful; i.e., they prove exactly the same equations. To abstract from the particular choice of axioms for $\mathbf{D}$, we shall henceforth write $\Pi(\mathcal{A},\mathbf{D})$ to refer to some instance $\Pi(\mathcal{A},\mathcal{S})$ of our deductive system with a complete specification $\mathcal{S}$ of $\mathbf{D}$; in deductions of $\Pi(\mathcal{A},\mathbf{D})$, we shall write $\mathbf{D} \models d \approx e$ and $\mathbf{D} \models b \approx c$ instead of $\mathcal{S} \vdash d \approx e$ and $\mathcal{S} \vdash b \approx c$, respectively.

In Chapter 4 we established a correspondence between the pCRL theory of $\mathbf{D}$ (i.e., the set of valid pCRL equations) and the first-order theory of $\mathbf{D}$. The other two data requirements serve to upgrade the set of Boolean expressions so that it has full first-order expressiveness.

Example 5.16 We take as data the natural numbers with a binary operation $\langle \mathsf{m}, \mathsf{n} \rangle \mapsto \mathsf{m}^{\mathsf{n}}$ for exponentiation, a binary operation $\langle \mathsf{m}, \mathsf{n} \rangle \mapsto \mathsf{m} \bmod \mathsf{n}$, which yields the remainder of m on division by n, and a unary relation P such that $\mathrm{P}(\mathsf{n}) = \top$ if, and only if, n is prime. Then

$$\mathrm{a}(x^y \bmod y) \lhd \mathrm{P}(y) \rhd \delta \approx \mathrm{a}(x \bmod y) \lhd \mathrm{P}(y) \rhd \delta$$

is valid by Fermat's Little Theorem: if p is prime, then $\mathsf{n}^{\mathsf{p}} \equiv \mathsf{n} \pmod{\mathsf{p}}$. If we assume that there is also a binary relation $\mathrm{eq}(\mathsf{m}, \mathsf{n})$ such that $\mathrm{eq}(\mathsf{m}, \mathsf{n}) = \top$ if, and only if, $\mathsf{m} = \mathsf{n}$, then we may express Fermat's Little Theorem by the Boolean expression

$$\neg\, \mathrm{P}(y) \vee \mathrm{eq}(x^y \bmod y, x \bmod y).$$

Our second requirement is:

(II) $\mathbf{D}$ must have *equality* (cf. Definition 4.3, p. 54).

Example 5.17 In the data algebra $\mathbf{R}$ of Example 3.5 an equality relation eq is definable as $\mathrm{eq}(\mathsf{r}_1, \mathsf{r}_2) = \mathsf{r}_1 \leq \mathsf{r}_2 \wedge \mathsf{r}_2 \leq \mathsf{r}_1$.

In Chapter 4 we proved that, for every first-order assertion φ about $\mathbf{D}$, there is a pCRL equation $p \approx q$ such that $\mathbf{D} \models \varphi$ if, and only if, $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models p \approx q$ (cf. Theorem 4.17). That is, for every true first-order assertion about $\mathbf{D}$ there exists a valid pCRL equation that essentially depends on it. In the light of our stance that reasoning about $\mathbf{D}$ should be entirely delegated to the subsidiary system $\mathcal{S}$, every such first-order assertion about $\mathbf{D}$ should be expressible within $\mathcal{S}$. We now give a criterion that ensures that every first-order assertion about $\mathbf{D}$ is equivalent to a Boolean expression.

Definition 5.18 A data algebra $\mathbf{D}$ has *quantifier elimination* if there exists for every first-order formula φ an open formula ψ such that $\mathsf{FV}(\varphi) = \mathsf{FV}(\psi)$ and

$$\mathbf{D} \models \varphi \leftrightarrow \psi.$$

Recall that every open first-order formula is a Boolean expression. If $\mathbf{D}$ has quantifier elimination, then, in view of Proposition 4.2, there exists a mapping

$$\beta : \Phi \to \mathcal{B}$$

that associates with every first-order formula φ a Boolean expression $\beta(\varphi)$ such that

$$\mathbf{D}, \nu \models \varphi \text{ if, and only if, } \mathbf{D}, \nu \models \beta(\varphi) \approx \top.$$

Our third requirement reads:

(Eq)	$\mathrm{a}(d_1,\ldots,d_n) \lhd \mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n) \rhd \delta$ $\approx \mathrm{a}(e_1,\ldots,e_n) \lhd \mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n) \rhd \delta$
(Qe)	$\sum_x p \lhd b \rhd \delta \approx p \lhd \beta((\exists x)b) \rhd \delta$ if $x \notin \mathsf{FV}(p)$

Table 5.3: Extra axioms for a data algebra with equality and quantifier elimination

(III) **D** must have *quantifier elimination*.

Example 5.19 By a classical result of Tarski (1951) the algebra **R** of Example 3.5 has quantifier elimination.

5.3.2 Completeness

Requirements (II) and (III) demand a certain additional expressiveness of the Booleans. To make use of this additional expressiveness in deductions, we need to add axioms for conditionals according to the schemata in Table 5.3.

We denote by $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$ the deductive system that consists of $\Pi(\mathcal{A},\mathbf{D})$ with the axioms of Table 5.3 added. It is complete in the following sense.

Theorem 5.20 If **D** has equality and quantifier elimination, then

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx q \text{ if, and only if, } \mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models p \approx q$$

for all pCRL expressions p and q.

Let us first prove the implication from left to right:

Lemma 5.21 If **D** has equality and quantifier elimination, then

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx q \text{ implies } \mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models p \approx q$$

for all pCRL expressions p and q.

Proof. In view of Theorem 5.11 it suffices to verify the validity of the axioms induced by (Eq) and (Qe). Let us fix again an arbitrary valuation ν and the interpretation homomorphism ι_ν associated with ν from $\mathbf{Pol}(\mathcal{A},\mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$. We prove the soundness of (Eq) by a case distinction on $\bar\nu(\mathrm{eq}(d_1,e_1)) \wedge \cdots \wedge \bar\nu(\mathrm{eq}(d_n,e_n))$:
If $\bar\nu(\mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n)) = \top$, then $\bar\nu(d_i) = \bar\nu(e_i)$ for all $1 \le i \le n$, so

$$\begin{aligned}\iota_\nu(\mathrm{a}(d_1,\ldots,d_n) &= \mathrm{a}(\bar\nu(d_1),\ldots,\bar\nu(d_n))\\ &= \mathrm{a}(\bar\nu(e_1),\ldots,\bar\nu(e_n))\\ &= \iota_\nu(\mathrm{a}(e_1,\ldots,e_n)),\end{aligned}$$

and hence

$$\iota_\nu(\mathrm{a}(d_1,\ldots,d_n) \lhd \mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n) \rhd \delta) = \iota_\nu(\mathrm{a}(e_1,\ldots,e_n) \lhd \mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n) \rhd \delta);$$

On the other hand, if $\bar{\nu}(\mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n)) = \bot$, then

$$\iota_\nu(\mathrm{a}(d_1,\ldots,d_n) \lhd \mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n) \rhd \delta) = \iota_\nu(\delta) = \iota_\nu(\mathrm{a}(e_1,\ldots,e_n) \lhd \mathrm{eq}(d_1,e_1) \wedge \cdots \wedge \mathrm{eq}(d_n,e_n) \rhd \delta).$$

For (QE) note that if $x \notin \mathsf{FV}(p)$, then $\iota_\nu(p[x := \mathsf{d}]) = \iota_\nu(p)$ for all $\mathsf{d} \in \mathsf{D}$, and hence

$$\iota_\nu(\textstyle\sum_x p \lhd b \rhd \delta) = \sum\{\iota_\nu(p) \mid \mathsf{d} \in \mathsf{D} \text{ such that } \bar{\nu}(b[x := \mathsf{d}]) = \top\}.$$

If $\bar{\nu}(\beta((\exists x)b)) = \top$, then the set on the right-hand side is $\{\iota_\nu(p)\}$, so that

$$\iota_\nu(\textstyle\sum_x p \lhd b \rhd \delta) = \iota_\nu(p) = \iota_\nu(p \lhd \beta((\exists x)b) \rhd \delta);$$

otherwise, the set on the right-hand side is empty, so that

$$\iota_\nu(\textstyle\sum_x p \lhd b \rhd \delta) = \iota_\nu(\delta) = \iota_\nu(p \lhd \beta((\exists x)b) \rhd \delta).$$

So $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$ is sound with respect to $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$. □

The other implication of Theorem 5.20 we prove by induction on the depth of sequential nesting in p and q; we inductively define a mapping $\#$ from pCRL expressions to natural numbers by

$\#(\delta) = 0$;
$\#(\mathrm{a}(d_1,\ldots,d_n)) = 1$ if a is a parametrised action symbol of arity n;
$\#(p \cdot q) = \#(p) + \#(q)$;
$\#(p + q) = \#(p \lhd b \rhd q) = \max\{\#(p),\#(q)\}$; and
$\#(\sum_x p) = \#(p)$.

It is easily deduced from the definition of θ on p. 45 that $\#(p) \geq \#(\theta(p))$ for every pCRL expression p. Moreover, sequential nesting depth is preserved under applications of (A1), (A2) and (A6) (i.e., if there is a deduction with $p \approx q$ as conclusion that merely consists of applications of these axioms, then $\#(p) = \#(q)$). Hence, and by Lemma 5.15, we may throughout the proof always replace a pCRL expression by a provably equivalent ordered tree form (cf. (3.7) on p. 46).

Lemma 5.22 For every pCRL expression p there exists an ordered tree form t such that $\Pi(\mathcal{A},\mathbf{D}) \vdash p \approx t$ and $\#(p) \geq \#(t)$.

In particular, it suffices to prove the implication from right to left of Theorem 5.20 only for ordered tree forms. Furthermore, by (A1) and since $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models p \approx q$

implies $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models p \preccurlyeq q,\ q \preccurlyeq p$, it is enough to show that for all ordered tree forms t and u

$$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t \preccurlyeq u \text{ implies } \Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t \preccurlyeq u. \tag{5.8}$$

If $t = \delta$, then (5.8) is immediate by (A6).
If $t = t_1 + \cdots + t_m$ for some $m > 0$, then $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t_i \preccurlyeq u$ for all $1 \le i \le m$. If we would know in addition that $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t_i \preccurlyeq u$ for all $1 \le i \le m$, then we may conclude (5.8) by (A2).
Therefore, we shall now concentrate on proving (5.8) in the case that t is a simple tree form; suppose

$$t = \textstyle\sum_{\vec{x}} t^* \lhd b \rhd \delta \text{ and } u = u_1 + \cdots + u_n \ (u_i \text{ simple for all } 1 \le i \le n).$$

A crucial step in our proof consists of splitting t into n simple tree forms $t_1, \ldots, t_n$ such that $\Pi(\mathcal{A},\mathbf{D}) \vdash t \approx t_1 + \cdots + t_n$ and $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t_i \preccurlyeq u_i$ for all $1 \le i \le n$. This is illustrated in the following example.

Example 5.23 We take as data the algebra $\mathbf{R}$ from Example 3.5 and consider the following pCRL expressions:

$$\begin{aligned} p &= \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x^2) \lhd -2 \le x \le 1 \rhd \delta,^2 \text{ and} \\ q &= q_1 + q_2, \text{ where } q_1 = \textstyle\sum_{x,y} \mathrm{in}(x)\mathrm{out}(y) \lhd y \le -2x \rhd \delta \text{ and} \\ &\qquad\qquad q_2 = \textstyle\sum_{x,y} \mathrm{in}(x)\mathrm{out}(y) \lhd y \le x \rhd \delta. \end{aligned}$$

Note that $x^2 \le -2x$ if, and only if, $-2 \le x \le 0$, and $x^2 \le x$ if, and only if, $0 \le x \le 1$ (see Figure 5.1 on the next page). We now split p into two pCRL expressions

$$\begin{aligned} p_1 &= \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x^2) \lhd -2 \le x \le 0 \rhd \delta, \text{ and} \\ p_2 &= \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x^2) \lhd 0 \le x \le 1 \rhd \delta. \end{aligned}$$

Then, since $\bar{\nu}(-2 \le x \le 0) = \top$ implies $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}), \nu \models \mathrm{in}(x)\mathrm{out}(x^2) \preccurlyeq q_1$,

$$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models p_1 \preccurlyeq q_1$$

and, since $\bar{\nu}(0 \le x \le 1) = \top$ implies $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}), \nu \models \mathrm{in}(x)\mathrm{out}(x^2) \preccurlyeq q_2$,

$$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models p_2 \preccurlyeq q_2.$$

Since $\mathbf{R} \models (-2 \le x \le 1) \approx (-2 \le x \le 0 \vee 0 \le x \le 1)$, $\Pi(\mathcal{A},\mathbf{D})$ has the following deduction:

$$\begin{aligned} p &\approx \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x^2) \lhd -2 \le x \le 0 \vee 0 \le x \le 1 \rhd \delta && \text{by (Bool)} \\ &\approx \textstyle\sum_x \left(\begin{array}{c} \mathrm{in}(x)\mathrm{out}(x^2) \lhd -2 \le x \le 0 \rhd \delta \\ + \\ \mathrm{in}(x)\mathrm{out}(x^2) \lhd 0 \le x \le 1 \rhd \delta \end{array} \right) && \text{by (C5)} \\ &\approx p_1 + p_2 && \text{by (CQ4).} \end{aligned}$$

[2] $r \le s \le t$ abbreviates the Boolean expression $r \le s \wedge s \le t$, and x^2 abbreviates $x \cdot x$.

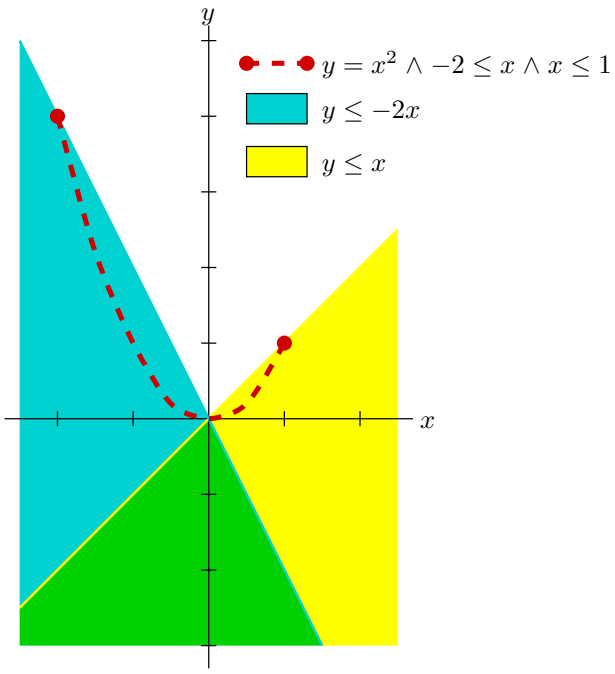


Figure 5.1: Graphical rendering of the correlations between input (x) and output (y) as defined by the pCRL expressions p, q_1 and q_2 of Example 5.23.

The pCRL expression p of the preceding example has been split by means of the Boolean expressions $-2 \leq x \leq 0$ and $0 \leq x \leq 1$. These Boolean expressions characterise precisely the sets of real numbers r such that $\mathrm{in}(\mathsf{r})\mathrm{out}(\mathsf{r}^2) \leq q_1$, and such that $\mathrm{in}(\mathsf{r})\mathrm{out}(\mathsf{r}^2) \leq q_2$, respectively. In general, to split a simple tree form $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$ into n simple tree forms $t_1, \ldots, t_n$ given the hypothesis that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t \preccurlyeq u_1 + \cdots + u_n$, we shall use Boolean expressions $b_1, \ldots, b_n$ which respectively characterise the valuations ν such that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}), \nu \models t^* \preccurlyeq u_i$.

Definition 5.24 Let p and q be pCRL expressions; a *p-simulation condition* for q is a Boolean expression b such that, for every valuation ν,

$$\mathbf{D}, \nu \models b \approx \top \text{ if, and only if, } \mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}), \nu \models p \preccurlyeq q.$$

Example 5.25 In the setting of Example 5.23, the Boolean expression

$$-2 \leq x \leq 0$$

is an $\mathrm{in}(\mathsf{r})\mathrm{out}(\mathsf{r}^2)$-simulation condition for q_1, and the Boolean expression

$$0 \leq x \leq 1$$

is an $\mathrm{in}(\mathsf{r})\mathrm{out}(\mathsf{r}^2)$-simulation condition for q_2.

That there exist t^*-simulation conditions $b_1, \ldots, b_n$ for $u_1, \ldots, u_n$, respectively, follows from Theorem 4.10 on p. 61 in combination with our assumption that $\mathbf{D}$ has equality and quantifier elimination.[3]

Theorem 5.26 If $\mathbf{D}$ has equality and quantifier elimination, then, for any two pCRL expressions p and q, there exists a p-simulation condition for q.

Proof. Since $\mathbf{D}$ has equality there exists, by Theorem 4.10, a first-order formula φ such that $\mathbf{D}, \nu \models \varphi$ if, and only if, $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}), \nu \models p \preccurlyeq q$, and since $\mathbf{D}$ has quantifier elimination there exists a mapping β from first-order formulas to Boolean expressions such that $\mathbf{D}, \nu \models \beta(\varphi) \approx \top$ if, and only if, $\mathbf{D}, \nu \models \varphi$; this proves the theorem. □

By the theorem above, each of the properties (4.4)–(4.12) that we have derived in Section 4.2 may be reformulated as properties of simulation conditions: (4.4) says that $\top$ is a δ-simulation condition for u; (4.5) says that if b' is a t'-simulation condition for u and b'' is a t''-simulation condition for u, then $b' \wedge b''$ is a $t' + t''$-simulation condition for u; etc. Henceforth, we shall frequently validate properties of simulation conditions by referring to (4.4)–(4.12). For the remainder, it is convenient to introduce an abbreviation: for Boolean expressions b and c we write $\mathbf{D} \models b \preccurlyeq c$ if

$$\mathbf{D}, \nu \models b \approx \top \text{ implies } \mathbf{D}, \nu \models c \approx \top, \text{ for every valuation } \nu.$$

Then we have the following lemma.

Lemma 5.27 For all Boolean expressions b and c,

$$\mathbf{D} \models b \preccurlyeq c \text{ if, and only if, } \mathbf{D} \models c \approx c \vee b \text{ if, and only if, } \mathbf{D} \models b \approx b \wedge c. \quad (5.9)$$

Proof. The second "if, and only if," is a well-known property of lattices (see, e.g., McKenzie *et al.*, 1987), and every Boolean algebra is a lattice. We prove the first "if, and only if,".

"if" Suppose $\mathbf{D} \models b \preccurlyeq c$, and let ν be an arbitrary valuation. Then, if $\bar{\nu}(c) = \top$, also $\bar{\nu}(c) \vee \bar{\nu}(b) = \top$, and conversely, if $\bar{\nu}(c) = \bot$, then also $\bar{\nu}(b) = \bot$, since $\bar{\nu}(b) = \top$ implies $\bar{\nu}(c) = \top$, so $\bar{\nu}(c) \vee \bar{\nu}(b) = \bot$.

"only if" Suppose $\mathbf{D} \models c \approx c \vee b$, and let ν be a valuation such that $\bar{\nu}(b) = \top$; then also $\bar{\nu}(c) = \top$ since $\bar{\nu}(c) = \bar{\nu}(c) \vee \bar{\nu}(b)$. □

[3] Strictly speaking, the results of Chapter 4 are about the algebra $\mathbf{T}_\mathbf{D}(\mathcal{A})$, and we ought to apply Corollary 3.16 to translate them into results about $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$; we leave such applications of Corollary 3.16 implicit.

The following lemma is a straightforward consequence of Lemma 4.5.

Lemma 5.28 Suppose that $p = \sum_{\vec{x}} p^* \lhd b \rhd \delta$ and q are pCRL expressions, and suppose that $\{\vec{x}\} \cap \mathsf{FV}(q) = \emptyset$. If b^* is a p^*-simulation condition for q, then

$$\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models p \preccurlyeq q \text{ if, and only if, } \mathbf{D} \models b \preccurlyeq b^*.$$

Lemma 5.29 (Split Lemma) Suppose $\mathbf{D}$ has equality and quantifier elimination.
If t and $u_1, \ldots, u_n$ are simple tree forms such that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t \preccurlyeq u_1 + \cdots + u_n$, then there exist simple tree forms $t_1, \ldots, t_n$ such that $\#(t) \geq \#(t_i)$,

$$\Pi(\mathcal{A}, \mathbf{D}) \vdash t \approx t_1 + \cdots + t_n, \text{ and } \mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t_i \preccurlyeq u_i \quad (1 \leq i \leq n).$$

Proof. Suppose $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$, with $t^* = a$ or $t^* = at'$.
If $n = 0$, then $u_1 + \cdots + u_n = \delta$ by convention, so it is enough to show that $\Pi(\mathcal{A}, \mathbf{D}) \vdash t \approx \delta$. Since $\bot$ is a t^*-simulation condition for δ (cf. (4.7) on p. 56), we get by Lemma 5.28 that $\mathbf{D} \models b \approx \bot$. This justifies the following deduction:

$$\begin{array}{ll} t \approx \sum_{\vec{x}} t^* \lhd \bot \rhd \delta & \text{by (Bool)} \\ \quad \approx \sum_{\vec{x}} \delta \lhd \top \rhd t^* & \text{by (C2)} \\ \quad \approx \sum_{\vec{x}} \delta & \text{by (C1)} \\ \quad \approx \delta & \text{by (CQ1).} \end{array}$$

For the remainder of the proof we assume $n > 0$ and $\{\vec{x}\} \cap \mathsf{FV}(u_i) = \emptyset$ for all $1 \leq i \leq n$. Let $b_1, \ldots, b_n$ be t^*-simulation conditions for $u_1, \ldots, u_n$, respectively; we define

$$t_i = \textstyle\sum_{\vec{x}} t^* \lhd b \wedge b_i \rhd \delta \qquad (1 \leq i \leq n).$$

Clearly, $\#(t) \geq \#(t_i)$. Since $\mathbf{D} \models b \wedge b_i \approx (b \wedge b_i) \wedge b_i$, we may conclude by (5.9) that $\mathbf{D} \models (b \wedge b_i) \preccurlyeq b_i$, so, by Lemma 5.28, $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t_i \preccurlyeq u_i$ for all $1 \leq i \leq n$. It remains to show that $\Pi(\mathcal{A}, \mathbf{D}) \vdash t \approx t_1 + \cdots + t_n$. Since $b_1 \vee \cdots \vee b_n$ is a t^*-simulation condition for $u_1 + \cdots + u_n$ (cf. (4.8) on p. 56) and $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t \preccurlyeq u_1 + \cdots + u_n$, we obtain by Lemma 5.28 and (5.9) that

$$\mathbf{D} \models b \approx b \wedge (b_1 \vee \cdots \vee b_n) \approx (b \wedge b_1) \vee \cdots \vee (b \wedge b_n).$$

Consequently, we have the following derivation:

$$\begin{array}{ll} t \approx \sum_{\vec{x}} t^* \lhd (b \wedge b_1) \vee \cdots \vee (b \wedge b_n) \rhd \delta & \text{by (Bool)} \\ \quad \approx \sum_{\vec{x}} (t^* \lhd b \wedge b_1 \rhd \delta + \cdots + t^* \lhd b \wedge b_n \rhd \delta) & \text{by (C4)} \\ \quad \approx t_1 + \cdots + t_n & \text{by (CQ4).} \end{array}$$

This completes the proof of the lemma. □

By means of the Split Lemma we have reduced the proof obligation for (5.8) to the case where both t and u are simple expressions. As an illustration, let us first finish the proof that we began in Example 5.23.

Example 5.30 Let p, p_1, p_2, q, q_1 and q_2 be as in Example 5.23.
We want to prove $\Pi(\mathcal{A}, \mathbf{D}) \vdash p \preccurlyeq q$; for this it is, by (A1) and (A2), enough to show $\Pi(\mathcal{A}, \mathbf{D}) \vdash p_i \preccurlyeq q_i$ for $i = 1, 2$. For $i = 1$ we use that $\mathbf{D} \models -2 \leq x \leq 0 \approx x^2 \leq -2x$ (see Figure 5.1), and we derive

$$\begin{aligned} p_1 &\approx \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x^2) \lhd x^2 \leq -2x \rhd \delta && \text{by (Bool)} \\ &\preccurlyeq \textstyle\sum_{x,y} \mathrm{in}(x)\mathrm{out}(y) \lhd y \leq -2x \rhd \delta = q_1 && \text{by (CQ3), Lemma 5.7.} \end{aligned}$$

For $i = 2$ a similar deduction can be given, using that $\mathbf{D} \models 0 \leq x \leq 1 \approx x^2 \leq x$.

Actually, that $\mathbf{D} \models -2 \leq x \leq 0 \approx x^2 \leq -2x$ is not essential for the first step of the above deduction. According to the following lemma, it would have been enough that $\mathbf{D} \models -2 \leq x \leq 0 \preccurlyeq x^2 \leq -2x$.

Lemma 5.31 If $\mathbf{D} \models b \preccurlyeq c$, then $\Pi(\mathcal{A}, \mathbf{D}) \vdash p \lhd b \rhd \delta \preccurlyeq p \lhd c \rhd \delta$.

Proof. If $\mathbf{D} \models b \preccurlyeq c$, then, by (5.9), $\mathbf{D} \models c \approx c \vee b$, so that we may derive

$$\begin{aligned} p \lhd c \rhd \delta &\approx p \lhd c \vee b \rhd \delta && \text{by (Bool)} \\ &\approx p \lhd c \rhd \delta + p \lhd b \rhd \delta && \text{by (C5);} \end{aligned}$$

this proves the lemma. □

For the application of (CQ3) in the second step of the deduction in Example 5.30 it is crucial that the pCRL expression $\mathrm{in}(x)\mathrm{out}(x^2) \lhd x^2 \leq -2x \rhd \delta$ is obtained from the pCRL expression $\mathrm{in}(x)\mathrm{out}(y) \lhd y \leq -2x \rhd \delta$ by substitution of a data expression (x^2) for a variable (y). In general, there may not be an appropriate data expression to facilitate an application of (CQ3).

Example 5.32 We take, again, as data the algebra $\mathbf{R}$ of Example 3.5, and consider the pCRL expressions

$$\begin{aligned} p &= \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x) \lhd 0 \leq x \rhd \delta, \text{ and} \\ q &= \textstyle\sum_{x,y} \mathrm{in}(x)\mathrm{out}(y^2) \lhd y^2 \leq x \rhd \delta, \end{aligned}$$

That $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{R}) \models p \preccurlyeq q$ essentially follows from the fact that $\mathsf{r} = (\sqrt{\mathsf{r}})^2$ for every nonnegative real number r. To give a formal deduction of $p \preccurlyeq q$ similar to the one in Example 5.30, we would need a data expression $\sqrt{x}$ that satisfies

$$\bar{\nu}(\sqrt{x}) = \sqrt{\nu(x)} \text{ for every valuation } \nu \text{ such that } 0 \leq \nu(x). \tag{5.10}$$

Given the language of $\mathbf{R}$, it is clear that if $\nu : X \to \mathsf{R}$ is a valuation that assigns an integer to each variable, then $\bar{\nu}(d)$ is also an integer. Hence, since, e.g., $\sqrt{2}$ is not an integer, a data expression d that satisfies (5.10) does not exist.

Nevertheless, the "result of substituting $\sqrt{x}$ for y" in a pCRL expression r is, in a semantical sense, expressible; we define

$$r\{\!|y := \sqrt{x}|\!\} = \textstyle\sum_y r \lhd \mathrm{eq}(y^2, x) \rhd \delta.$$

Note that, since $\mathbf{D} \models \mathrm{eq}(y^2, x) \preccurlyeq (y^2 \leq x)$, by (C3), (5.9) and (Bool)

$$(\mathrm{in}(x)\mathrm{out}(y^2) \lhd y^2 \leq x \rhd \delta)\{\!|y := \sqrt{x}|\!\} \approx \textstyle\sum_y \mathrm{in}(x)\mathrm{out}(y^2) \lhd \mathrm{eq}(y^2, x) \rhd \delta. \quad (5.11)$$

Hence, since $\mathbf{D} \models (0 \leq x) \approx \beta((\exists y)\mathrm{eq}(y^2, x))$, we get

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx \textstyle\sum_x (\mathrm{in}(x)\mathrm{out}(y^2) \lhd y^2 \leq x \rhd \delta)\{\!|y := \sqrt{x}|\!\}$$

from the deduction

$$\begin{aligned}
p &\approx \textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x) \lhd \beta((\exists y)\mathrm{eq}(y^2, x)) \rhd \delta && \text{by (Bool)}\\
&\approx \textstyle\sum_{x,y} \mathrm{in}(x)\mathrm{out}(x) \lhd \mathrm{eq}(y^2, x) \rhd \delta && \text{by (Qe)}\\
&\approx \textstyle\sum_{x,y} \mathrm{in}(x)\mathrm{out}(y^2) \lhd \mathrm{eq}(y^2, x) \rhd \delta && \text{by Lem. 5.6(ii) and (Eq)}\\
&\approx \textstyle\sum_x (\mathrm{in}(x)\mathrm{out}(y^2) \lhd y^2 \leq x \rhd \delta)\{\!|y := \sqrt{x}|\!\} && \text{by (5.11).}
\end{aligned}$$

Since $\mathbf{D} \models \mathrm{eq}(y^2, x) \preccurlyeq (y^2 \leq x)$, we get, by Lemma 5.31, that

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash \textstyle\sum_x (\mathrm{in}(x)\mathrm{out}(y^2) \lhd y^2 \leq x \rhd \delta)\{\!|y := \sqrt{x}|\!\} \preccurlyeq q,$$

so it follows that $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \preccurlyeq q$.

Note that, in the above example, $\mathrm{eq}(y^2, x)$ is an $\mathrm{in}(x)\mathrm{out}(x)$-simulation condition for $\mathrm{in}(x)\mathrm{out}(y^2)$, and $\beta((\exists y)\mathrm{eq}(y^2, x))$ is an $\mathrm{in}(x)\mathrm{out}(x)$-simulation condition for q. Intuitively, we use these simulation conditions to prove p equivalent to a semantical substitution instance of $\sum_x \mathrm{in}(x)\mathrm{out}(y^2) \lhd y^2 \leq x \rhd \delta$. We shall now generalise this technique so that it proves (5.8) for all simple expressions t and u. Suppose that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t \preccurlyeq u$ with

$$t = \textstyle\sum_{\vec{x}} t^* \lhd b \rhd \delta \text{ and } u = \sum_{\vec{y}} u^* \lhd c \rhd \delta.$$

We proceed in two steps: we shall first prove that $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t \preccurlyeq u$ under the hypothesis that $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta$, where b^* is a t^*-simulation condition for u^* (Lemma 5.34), and subsequently we shall prove the hypothesis (Lemma 5.39). We need a lemma about conditionals.

Lemma 5.33 If $\mathbf{D} \models b \preccurlyeq c$, then

$$\Pi(\mathcal{A}, \mathbf{D}), p \lhd c \rhd \delta \approx q \lhd c \rhd \delta \vdash p \lhd b \rhd \delta \approx q \lhd b \rhd \delta.$$

Proof. If $\mathbf{D} \models b \preccurlyeq c$, then, by (5.9), $\mathbf{D} \models b \approx b \wedge c$, so if

$$(*)\ p \lhd c \rhd \delta \approx q \lhd c \rhd \delta,$$

then we have the following derivation

$$\begin{aligned}
p \lhd b \rhd \delta &\approx (p \lhd c \rhd \delta) \lhd b \rhd \delta && \text{by (Bool), (C3)}\\
&\approx (q \lhd c \rhd \delta) \lhd b \rhd \delta && \text{by (*)}\\
&\approx q \lhd b \rhd \delta && \text{by (C3), (Bool);}
\end{aligned}$$

this proves the lemma. □

Lemma 5.34 Suppose that $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$ and $u = \sum_{\vec{y}} u^* \lhd c \rhd \delta$ are simple tree forms such that $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t \preccurlyeq u$, and let b^* be a t^*-simulation condition for u^*; then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}, t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta \vdash t \preccurlyeq u.$$

Proof. There is no loss of generality in assuming that

$$\{\vec{x}\} \cap (\mathsf{FV}(u) \cup \{\vec{y}\}) = \emptyset \text{ and } \{\vec{y}\} \cap (\mathsf{FV}(t) \cup \{\vec{x}\}) = \emptyset$$

(otherwise we first do some renamings of variables by means of (CQ2)).

By Lemma 4.6 on p. 56, $\beta((\exists \vec{y})(b^* \wedge c))$ is a t^*-simulation condition for u, so, by Lemma 5.28, $\mathbf{D} \models b \preccurlyeq \beta((\exists \vec{y})(b^* \wedge c))$. Hence, if (*) $t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta$, then

$$\begin{aligned}
t &\preccurlyeq \textstyle\sum_{\vec{x}} t^* \lhd \beta((\exists \vec{y})(b^* \wedge c)) \rhd \delta && \text{by Lem. 5.31, Lem. 5.7} \\
&\approx \textstyle\sum_{\vec{x}} \sum_{\vec{y}} t^* \lhd b^* \wedge c \rhd \delta && \text{by (QE)} \\
&\approx \textstyle\sum_{\vec{x}} \sum_{\vec{y}} u^* \lhd b^* \wedge c \rhd \delta && \text{by (*), Lem. 5.33} \\
&\preccurlyeq \textstyle\sum_{\vec{x}} \sum_{\vec{y}} u^* \lhd c \rhd \delta && \text{by Lem. 5.31, Lem. 5.7} \\
&\approx \textstyle\sum_{\vec{y}} u^* \lhd c \rhd \delta && \text{by (CQ1);}
\end{aligned}$$

this proves the lemma. □

It remains to prove that $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta$ if b^* is a t^*-simulation condition for u^*. Let us first deal with the case that t^* and u^* are both action expressions. Recall that with every two action expressions a and a' we have associated a Boolean expression $\mathrm{eq}(a, a')$ (see Definition 4.7 on p. 57); it is an a-simulation condition for a'.

Lemma 5.35 Let a and a' be action expressions; then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash a \lhd \mathrm{eq}(a, a') \rhd \delta \approx a' \lhd \mathrm{eq}(a, a') \rhd \delta.$$

Proof. Let $a = \mathrm{a}(d_1, \ldots, d_m)$ and $a' = \mathrm{a}'(e_1, \ldots, e_n)$; there are two cases: If $\mathrm{a} = \mathrm{a}'$ and $m = n$, then $\mathrm{eq}(a, a') = \mathrm{eq}(d_1, e_1) \wedge \cdots \wedge \mathrm{eq}(d_n, e_m)$; hence,

$$\begin{aligned}
&a \lhd \mathrm{eq}(a, a') \rhd \delta \\
&\quad = \mathrm{a}(d_1, \ldots, d_m) \lhd \mathrm{eq}(d_1, e_1) \wedge \cdots \wedge \mathrm{eq}(d_n, e_m) \rhd \delta \\
&\quad \approx \mathrm{a}'(e_1, \ldots, e_m) \lhd \mathrm{eq}(d_1, e_1) \wedge \cdots \wedge \mathrm{eq}(d_n, e_m) \rhd \delta \qquad \text{by (EQ)} \\
&\quad = a' \lhd \mathrm{eq}(a, a') \rhd \delta.
\end{aligned}$$

Otherwise $\mathrm{eq}(a, a') = \bot$, so that we get

$$a \lhd \mathrm{eq}(a, a') \rhd \delta \approx \delta \approx a' \lhd \mathrm{eq}(a, a') \rhd \delta$$

by (BOOL), (C1) and (C2). □

Next, suppose that t^* and u^* are both sequential compositions, say $t^* = at'$ and $u^* = a'u'$; we want to prove that

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta.$$

By Lemma 5.6(ii), we may distribute the simulation condition b^* over the sequential compositions, and since $\mathbf{D} \models b^* \preccurlyeq \mathrm{eq}(a, a')$, we get by Lemma 5.35 and Lemma 5.33 that

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash a \lhd b^* \rhd \delta \approx a' \lhd b^* \rhd \delta.$$

It remains to prove that

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t' \lhd b^* \rhd \delta \approx u' \lhd b^* \rhd \delta;$$

we shall see that this can be established by means of the induction hypothesis. However, before we may apply the induction hypothesis, we need to establish that

$$\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models t' \lhd b^* \rhd \delta \approx u' \lhd b^* \rhd \delta.$$

The following definition is helpful in this respect.

Definition 5.36 Let p and q be pCRL expressions; a *bisimulation condition* for p and q is a Boolean expression b such that, for every valuation ν,

$$\mathbf{D}, \nu \models b \approx \top \text{ if, and only if, } \mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}), \nu \models p \approx q.$$

Clearly, if b is a p-simulation condition for q and c is a q-simulation condition for p, then $b \wedge c$ is a bisimulation condition for p and q, so we get as an immediate corollary to Theorem 5.26 that there exists a bisimulation condition for every two pCRL expressions.

Corollary 5.37 If $\mathbf{D}$ has equality and quantifier elimination, then, for any two pCRL expressions p and q, there exists a bisimulation condition for p and q.

Lemma 5.38 If b is a bisimulation condition for p and q, then

$$\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models p \lhd b \rhd \delta \approx q \lhd b \rhd \delta.$$

Proof. Let ν be a valuation and let ι_ν be the interpretation homomorphism associated with ν from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into an arbitrary element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$.
If $\bar{\nu}(b) = \top$, then, since b is a bisimulation condition for p and q, $\iota_\nu(p) = \iota_\nu(q)$; hence $\iota_\nu(p \lhd b \rhd \delta) = \iota_\nu(p) = \iota_\nu(q) = \iota_\nu(q \lhd b \rhd \delta)$.
On the other hand, if $\bar{\nu}(b) = \bot$, then $\iota_\nu(p \lhd b \rhd \delta) = \iota_\nu(\delta) = \iota_\nu(q \lhd b \rhd \delta)$. □

So, if b' is a bisimulation condition for t' and u', then, by the preceding lemma, we may apply the induction hypothesis to obtain

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t' \lhd b' \rhd \delta \approx u' \lhd b' \rhd \delta.$$

In the next lemma we show how $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta$ then follows.

Lemma 5.39 Let b^* be a t^*-simulation condition for u^*.

(i) If t^* and u^* are action expressions, then

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta.$$

(ii) If $t^* = at'$ and $u^* = a'u'$ and b' is a bisimulation condition for t' and u', then

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}, t' \lhd b' \rhd \delta \approx u' \lhd b' \rhd \delta \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta.$$

(iii) If $t^* = at'$ and u^* is an action expression, or t^* is an action expression and $u^* = a'u'$, then

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta.$$

Proof. If t^* and u^* are action expressions, then b^* is a bisimulation condition for t^* and u^* (cf. (4.10) on p. 57), so, by Lemma 4.8 on p. 57, $\mathbf{D} \models b^* \approx \mathrm{eq}(t^*, u^*)$. Hence, we conclude (i) by Lemma 5.35 and (Bool).

If $t^* = at'$ and $u^* = a'u'$ and b' is a bisimulation condition for t' and u', then $\mathbf{D} \models b^* \preccurlyeq \mathrm{eq}(a, a') \wedge b'$ (cf. (4.11) on p. 57), whence $\mathbf{D} \models b^* \preccurlyeq \mathrm{eq}(a, a'),\ b^* \preccurlyeq b'$. From $\mathbf{D} \models b^* \preccurlyeq \mathrm{eq}(a, a')$ we conclude by Lemma 5.35 and Lemma 5.33 that

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash a \lhd b^* \rhd \delta \approx a' \lhd b^* \rhd \delta; \tag{5.12}$$

from $\mathbf{D} \models b^* \preccurlyeq b'$ we conclude by Lemma 5.33 that

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}, t' \lhd b' \rhd \delta \approx u' \lhd b' \rhd \delta \vdash t' \lhd b^* \rhd \delta \approx u' \lhd b^* \rhd \delta. \tag{5.13}$$

So, if (*) $t' \lhd b' \rhd \delta \approx u' \lhd b' \rhd \delta$, then we can make the following derivation:

$$\begin{aligned} t^* \lhd b^* \rhd \delta &\approx (a \lhd b^* \rhd \delta)(t' \lhd b^* \rhd \delta) && \text{by Lem. 5.6(ii)} \\ &\approx (a' \lhd b^* \rhd \delta)(u' \lhd b^* \rhd \delta) && \text{by (5.12), (5.13)} \\ &\approx u^* \lhd b^* \rhd \delta && \text{by Lem. 5.6(ii);} \end{aligned}$$

this proves (ii).

If $t^* = at'$ and u^* is an action expression, or t^* is an action expression and $u^* = a'u'$, then $\mathbf{D} \models b^* \approx \bot$ (cf. (4.12) on p. 57); hence $t^* \lhd b^* \rhd \delta \approx \delta \approx u^* \lhd b^* \rhd \delta$ by (Bool), (C1) and (C2); this proves (iii). □

We have now established all the necessary facts that are needed to prove Theorem 5.20; let us now put everything together.

Proof of Theorem 5.20. The implication from left to right is by Lemma 5.21; for the implication from right to left we prove that

$$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t \preccurlyeq u \text{ implies } \Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t \preccurlyeq u \tag{5.14}$$

for all ordered tree forms t and u (this is enough by Lemma 5.15); we proceed by induction on $\#(t)+\#(u)$.

First we consider the case that t and u are both simple tree forms; suppose that $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$ and $u = \sum_{\vec{y}} u^* \lhd c \rhd \delta$. By Theorem 5.26 there exists a t^*-simulation condition b^* for u^*, and for the implication (5.14) it suffices to prove

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t^* \lhd b^* \rhd \delta \approx u^* \lhd b^* \rhd \delta; \tag{5.15}$$

for then, (5.14) follows by Lemma 5.34. To see that (5.15) holds, we distinguish cases according to the syntactic forms that t^* and u^* may take:

1. If t^* and u^* are both action expressions, then we get (5.15) by Lemma 5.39(i).
2. Suppose $t^* = at'$ and $u^* = au'$. By Corollary 5.37 there exists a bisimulation condition b' for t' and u'; by Lemma 5.38

   $$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t' \lhd b' \rhd \delta \approx u' \lhd b' \rhd \delta.$$

   Let t'' and u'' be ordered tree forms, provably equivalent to $t' \lhd b' \rhd \delta$ and $u' \lhd b' \rhd \delta$, respectively; by Lemma 5.22 we may assume $\#(t'') < \#(t^*)$ and $\#(u'') < \#(u^*)$, so we obtain by the induction hypothesis that

   $$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t'' \approx u'',$$

   and hence

   $$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t' \lhd b' \rhd \delta \approx u' \lhd b' \rhd \delta.$$

   We now get (5.15) with an application of Lemma 5.39(ii).
3. If $t^* = at'$ and u^* is an action expression, or t^* is an action expression and $u^* = a'u'$, then we apply Lemma 5.39(iii) to get (5.15).

Hence, (5.14) holds for simple tree forms. To prove that (5.14) holds for *all* tree forms, we proceed by distinguishing cases according to the syntactic form of t:

1. If $t = \delta$, then (5.14) is immediate by (A6).
2. If t is a simple tree form, then it remains to consider the case that u is not simple, so suppose that $u = u_1 + \cdots + u_n$ for some $n \neq 1$, and suppose that $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t \preccurlyeq u$. We apply the Split Lemma (Lemma 5.29) to split t into n simple expressions $t_1,\ldots,t_n$ such that $\#(t) \geq \#(t_i)$,

   $$\Pi(\mathcal{A},\mathbf{D}) \vdash t \approx t_1 + \cdots + t_n, \text{ and } \mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t_i \preccurlyeq u_i \quad (1 \leq i \leq n).$$

It follows that $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t_i \preccurlyeq u_i$ for all $1 \leq i \leq n$ (this is the special case which we have dealt with), so by (A2) $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t \preccurlyeq u$. Hence, we may conclude that (5.14) holds if t is a simple expression.

3. If $t = t_1 + \cdots + t_m$ for some $m \geq 2$ and with each t_i $(1 \leq i \leq m)$ a simple tree form, then $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t_i \preccurlyeq u$ for all $1 \leq i \leq m$. It now follows from the previous case (2) that $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t_i \preccurlyeq u$ for all $1 \leq i \leq m$, and hence, by (A2), $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t \preccurlyeq u$.

This completes the proof of Theorem 5.20. □

Remark 5.40 With the inclusion of (Eq) and (Qe) (see Table 5.3), the axioms (Data) and (Cq3) have become redundant. This follows from Theorem 5.20 and the observation that the proof of this theorem does not involve applications of these axioms. Note that (Data) and (Cq3) are the only axioms of our deductive system in which the general notion of 'substituting an arbitrary *data expression* d for x in p' is used; (Cq2) only involves a simpler variant, that of 'substituting a *variable* y for x in p. Thus, by deleting (Data) and (Cq3) from $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$ we get a deductive system for pCRL that is conceptually simpler. We shall return to this issue in Chapter 6.

5.3.3 Skolem expressions

Recall the deduction of $p_1 \preccurlyeq q_1$ in Example 5.30; it does not involve (Qe). Instead, it uses (Cq3), and the fact that $\mathrm{in}(x)\mathrm{out}(x^2) \lhd x^2 \leq -2x \rhd \delta$ is obtained by substituting x^2 for y in $\mathrm{in}(x)\mathrm{out}(y) \lhd y \leq -2x \rhd \delta$. We have remarked that such a deduction cannot be generalised to prove that

$$\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D}) \models t \preccurlyeq u \text{ implies } \Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash t \preccurlyeq u$$

for all simple tree forms $t = \sum_{\vec{x}} t^* \lhd b \rhd \delta$ and $u = \sum_{\vec{y}} u^* \lhd c \rhd \delta$. For, $t^* \lhd b \rhd \delta$ may not be a substitution instance of $u^* \lhd c \rhd \delta$, because the appropriate data expressions to substitute for the $\vec{y}$ are not available (see Example 5.32).

Our stance in this chapter has been that questions about data that arise when proving that a pCRL equation is valid in $\mathsf{GBPA}_\delta(\mathcal{A},\mathbf{D})$, should be delegated to the subsidiary deductive system for the data. Perhaps we should expect that it also provides the data expressions needed in the circumstances described above? We shall now prove that if we do, then (Qe) becomes redundant. We strengthen our requirement (III) that $\mathbf{D}$ has quantifier elimination, adapting a definition from Chang and Keisler (1990).

Definition 5.41 Suppose that $\mathbf{D}$ is a data algebra and consider a first-order formula φ with $\mathsf{FV}(\varphi) - \{x\} = \{x_1,\ldots,x_n\}$. A data expression $d = d(x_1,\ldots,x_n)$ (the variables with an occurrence in d must be among the $x_1,\ldots,x_n$) we call a *Skolem expression* [4] for $(\exists x)\varphi$ if

$$\mathbf{D} \models (\exists x)\varphi \to \varphi[x := d].$$

[4] Chang and Keisler (1990) call it a "Skolem function".

We say that $\mathbf{D}$ has *Skolem expressions* if it has a Skolem expression for every first-order formula. If $\mathbf{D}$ is a data algebra with Skolem expressions, then $\mathrm{Sk}(x, \varphi)$ denotes a Skolem expression for $(\exists x)\varphi$.

Example 5.42 We expand the data algebra $\mathbf{R}$ of Example 3.5 with a function $\mathrm{sqrt} : \mathsf{R} \to \mathsf{R}$ such that

$$\mathrm{sqrt}(\mathsf{r}) = \begin{cases} \sqrt{\mathsf{r}} & \text{if } \mathsf{r} \geq 0; \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The data expression $\mathrm{sqrt}(x)$ is a Skolem expression for the formula $(\exists y)\mathrm{eq}(y^2, x)$ (cf. Example 5.32); for, $(\mathrm{sqrt}(\mathsf{r}))^2 = \mathsf{r}$ if, and only if, $\mathsf{r} \geq 0$.

If $\mathbf{D}$ has Skolem expressions, then we may define a mapping β from first-order formulas to Boolean expressions by

$$\begin{aligned} &\beta(\mathrm{r}(d_1, \ldots, d_n)) = \mathrm{r}(d_1, \ldots, d_n); \\ &\beta(\neg\varphi) = \neg\beta(\varphi); \\ &\beta(\varphi \vee \psi) = \beta(\varphi) \vee \beta(\psi); \text{ and} \\ &\beta((\exists x)\varphi) = \beta(\varphi)[x := \mathrm{Sk}(x, \varphi)]. \end{aligned}$$

If φ is a first-order formula, then

$$\mathbf{D} \models (\exists x)\varphi \leftrightarrow \varphi[x := \mathrm{Sk}(x, \varphi)].$$

(The implication from left to right is valid since $\mathrm{Sk}(x, \varphi)$ is a Skolem expression for $(\exists x)\varphi$; the other implication is trivial.) Hence, φ and $\beta(\varphi)$ are equivalent, and moreover, β yields a formula from which all quantifiers have been eliminated, i.e., $\beta(\varphi)$ is an open first-order formula. So, we get the following proposition.

Proposition 5.43 If $\mathbf{D}$ has Skolem expressions, then it has quantifier elimination.

We now show that (QE) can be deduced within $\Pi(\mathcal{A}, \mathbf{D})$ if $\mathbf{D}$ has Skolem expressions.

Lemma 5.44 If $\mathbf{D}$ has Skolem expressions, then, for every Boolean expression b,

$$\Pi(\mathcal{A}, \mathbf{D}) \vdash \textstyle\sum_x p \lhd b \rhd \delta \approx p \lhd \beta((\exists x)b) \rhd \delta, \text{ provided that } x \notin \mathsf{FV}(p).$$

Proof.

($\preccurlyeq$) Since $\mathbf{D} \models \beta((\exists x)b) \approx \beta((\exists x)b) \vee b$, we have the following derivation:

$$\begin{aligned} p \lhd \beta((\exists x)b) \rhd \delta &\approx \textstyle\sum_x p \lhd \beta((\exists x)b) \rhd \delta && \text{by (CQ1)} \\ &\approx \textstyle\sum_x p \lhd \beta((\exists x)b) \vee b \rhd \delta && \text{by (BOOL)} \\ &\approx \textstyle\sum_x (p \lhd \beta((\exists x)b) \rhd \delta + p \lhd b \rhd \delta) && \text{by (C5)} \\ &\approx \textstyle\sum_x p \lhd \beta((\exists x)b) \rhd \delta + \sum_x p \lhd b \rhd \delta && \text{by (CQ4)} \\ &\approx p \lhd \beta((\exists x)b) \rhd \delta + \textstyle\sum_x p \lhd b \rhd \delta && \text{by (CQ1).} \end{aligned}$$

Consequently, $\Pi(\mathcal{A}, \mathbf{D}) \vdash \sum_x p \lhd b \rhd \delta \preccurlyeq p \lhd \beta((\exists x)b) \rhd \delta$.

($\succcurlyeq$) Clearly, $\delta[x := d] = \delta$, and since $x \notin \mathsf{FV}(p)$, $p = p[x := d]$; so

$$\begin{aligned}\textstyle\sum_x p \lhd b \rhd \delta &\approx \textstyle\sum_x p \lhd b \rhd \delta + p \lhd b[x := \mathrm{Sk}(x, b)] \rhd \delta \\ &= \textstyle\sum_x p \lhd b \rhd \delta + p \lhd \beta((\exists x)b) \rhd \delta\end{aligned}$$

is an instance of CQ3.

Hence, $\Pi(\mathcal{A}, \mathbf{D}) \vdash p \lhd \beta((\exists x)b) \rhd \delta \preccurlyeq \sum_x p \lhd b \rhd \delta$.

It follows that $\Pi(\mathcal{A}, \mathbf{D}) \vdash \sum_x p \lhd b \rhd \delta \approx p \lhd \beta((\exists x)b) \rhd \delta$. □

Let us denote by $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}$ the deductive system that consists of $\Pi(\mathcal{A}, \mathbf{D})$ together with (EQ) from Table 5.3. By Proposition 5.43 and Lemma 5.44 we get the following corollary to Theorem 5.20.

Corollary 5.45 If $\mathbf{D}$ has equality and Skolem expressions, then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}} \vdash p \approx q \text{ if, and only if, } \mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D}) \models p \approx q$$

for all pCRL expressions p and q.

Bibliographic notes

Ponse (1991) proves relative completeness of a proof system for deriving partial correctness assertions about processes.

Hennessy (1991) advocates the idea of designing a deductive system for value-passing processes in which reasoning about data is factored out as much as possible. His deductive system is for a version of value-passing CCS, and he proves a relative completeness result with respect to a model based on Acceptance Trees (Hennessy, 1985). This work was continued by Hennessy and Lin (1996), who present a series of deductive systems that are proved relatively complete for finite processes modulo a series of symbolic bisimulation equivalences (Hennessy and Lin, 1995), and subsequently extended to settings with recursion, by Hennessy and Lin (1997) and Rathke (1997).

These deductive systems are designed to infer sequents of the form

$$b \rhd p \approx q,$$

meaning that the equation $p \approx q$ holds for every valuation ν such that $\bar{\nu}(b) = \top$. Thus, in our terminology, the sequents correspond to pCRL equations of the form

$$p \lhd b \rhd \delta \approx q \lhd b \rhd \delta.$$

We have already seen in Section 4.4 that value-passing CCS, having input prefixing as a primitive instead of choice quantification, is strictly less expressive than pCRL. Every equation of expressions of value-passing CCS corresponds to a *universal* first-order formula, and every universal formula is logically equivalent to a Boolean expression (with free variables). This explains why Hennessy and Lin (1996) only need our first two requirements (that the presupposed subsidiary deductive system

allows the inference of all valid data equations and all valid Boolean equations, and that $\mathbf{D}$ has equality) is needed to obtain a relative completeness result.

The technique of splitting expressions by means of conditions (cf. our Split Lemma on p. 90) seems to be standard in settings with operations that involve choice quantification over a certain domain (e.g., input prefixing, choice quantification, integration), but it may appear in different guises. Our form of splitting is with respect to some other pCRL expression. Hennessy and Lin's proof shows a similar kind of splitting. Also Parrow and Sangiorgi (1995) use this kind of splitting in completeness proofs for their axiomatisations of early bisimulation and early congruence in a π-calculus-like setting. Fokkink and Klusener (1995), on the other hand, associate with every process expression of a variant of real time ACP with prefix integration a unique partition of intervals of real numbers, and splitting is with respect to this partition. They use splitting to reduce each process expression to a normal form. Since these normal forms are unique, it follows that their set of axioms is complete.

Groote and Ponse (1994) have proposed a formal framework for μCRL in which reasoning about data and reasoning about processes is fully integrated. In their framework, a property of a μCRL specification is expressed, roughly, as a Boolean combination of data equations, Boolean equations or μCRL equations. The basis of their deductive system is a hybrid between (classical) natural deduction (but without the rules for existential and universal quantifiers), and equational logic. Axioms are added in a modular fashion. For instance, there is a module called pCRL, which contains (A1)–(A7), (CQ1)–(CQ5) (see Table 5.2), and

$$p \approx q \to \sum_x p \approx \sum_x q.$$

Note that this implication corresponds to our $(\text{CONG}_{(\sum_x)})$.

To facilitate reasoning about Boolean expressions and data expressions with (free) variables, Groote and Ponse include induction schemata based on a presupposed set of constructors. In particular, for Boolean expressions, they include the law of the *excluded middle* $\neg(b \approx \top) \to (b \approx \bot)$. Thus, the soundness of their deductive system hinges on the assumptions that the data algebra is minimal, and that there are no more than two Booleans. In contrast, our deductive system is also usable if the data algebra is not minimal (cf. Remark 3.10 on p. 35). If the data algebra happens to be minimal, then, since Boolean expressions and data expressions may contain variables, our requirement that $\mathcal{S}$ is a complete specification of $\mathbf{D}$ implies that $\mathcal{S}$ is inductively complete (i.e., it admits the inference of all valid equations that can be proved by means of structural induction).

Groote and Ponse only demonstrate the soundness of their deductive system; they do not address the issue of completeness. In fact, completeness would take a different form in their setting, since their deductive system allows the inference of Boolean combinations of equations (in particular, negations of equations). So, to get a complete system, it would be necessary to also add axioms such as, e.g., $\neg(a \approx a'p)$ for all action expressions a and a', and for every μCRL expression p. Then, to prove the result, one would still need similar requirements about the data as we presented at the beginning of Section 5.3, except, perhaps, that the

first requirement could be relaxed to *ground completeness*[5].

The completeness result presented in this chapter may be reused to obtain completeness results in related settings. For instance, Groote and Luttik (1998b) consider pCRL expressions modulo branching bisimulation. They add the standard laws for branching bisimulation (Van Glabbeek and Weijland, 1996) to the deductive system $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$ and then show that every pCRL expression is provably equal to a *compact* expression. Since compact pCRL expressions p and q are branching bisimilar if, and only if, $p \approx q$ is valid, it follows from the results in this chapter that the resulting deductive system is relatively complete. By means of a similar technique, Luttik (1999a) has obtained relatively complete deductive systems for pCRL expressions modulo weak-, delay-, and η-bisimulation. Likewise, Van der Zwaag (2000) and Groote *et al.* (2000) have proved that with respect to their deductive systems for timed versions of pCRL and μCRL, respectively, each expression is provably equal to a so-called well-timed deadlock-saturated expression, and that two such expressions p and q are timed bisimilar if, and only if, $p \approx q$ is valid in $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$. Thus, also in their settings, relative completeness follows from our result.

[5] A data specification is ground complete if all valid data equations without variables and all valid Boolean equations without variables can be deduced.

6

Algebraic pCRL

We now have a formal system to reason about elements of generalised basic process algebras with deadlock. Taking a sequence $\mathcal{A}$ of parametrised action symbols and a data algebra $\mathbf{D}$ with equality and quantifier elimination as parameters, it has two ingredients:

1. a set of meaningful expressions (here: the set of pCRL expressions), and
2. a deductive system (here: the deductive system $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$).

The axioms of the deductive system are all in the form of equations, and its inference rules closely resemble the conventional rules of equational logic. The temptation to qualify our formal system with the adjective 'algebraic' is therefore hard to resist. However, from an algebraic point of view, it is not wholly satisfactory. What is unsatisfactory about it, is best illustrated in comparison with a related deductive system that we do consider satisfactory. We enter a minor digression and review the situation in the theory of basic process algebras with deadlock.

It is not unusual to present the theory of basic process algebras with deadlock in the following manner. One starts with the declaration of an alphabet A of constant symbols that serves as a parameter of a formal system. The set of meaningful expressions of this formal system consists of the terms that can be built from the elements of A, another constant symbol δ, and binary function symbols $+$ and $\cdot$; let us, for the moment, use the symbol $\mathcal{P}$ to denote this set. The deductive system associated with this formal system has as axioms the equations generated by the schemata (A1)–(A7) in Table 5.2 on p. 75, and as inference rules the first five rules listed in Table 5.2 (i.e., (Refl)–(Cong$_{(\cdot)}$)), with the meta variables ranging over $\mathcal{P}$. We write $\mathsf{BPA}_\delta(\mathsf{A}) \vdash p \approx q$ if the deductive system permits a deduction that has the equation $p \approx q$ as conclusion.

What is algebraic about this formal system? For one thing, the set $\mathcal{P}$ is in a natural way the universe of an algebra

$$\mathbf{Pe} = \langle \mathcal{P}, +, \cdot, \delta \rangle,$$

As an immediate consequence of the inference rules of the deductive system associated with $\mathsf{BPA}_\delta(\mathsf{A})$, the relation

$$\vartheta = \{\langle p, q \rangle \in \mathcal{P} \times \mathcal{P} \mid \mathsf{BPA}_\delta(\mathsf{A}) \vdash p \approx q\}$$

is a congruence on **Pe**.

But there is a more profound reason why we may rightfully call it algebraic. Note that each of the schemata (A1)–(A7) associates with **Pe** a binary relation:

$$\begin{aligned}(\mathrm{A1})_{\mathbf{Pe}} &= \{\langle p+q, q+p\rangle \mid p,q \in \mathcal{P}\};\\ (\mathrm{A2})_{\mathbf{Pe}} &= \{\langle p+(q+r),(p+q)+r\rangle \mid p,q,r \in \mathcal{P}\};\\ &\vdots\\ (\mathrm{A7})_{\mathbf{Pe}} &= \{\langle \delta\cdot p, \delta\rangle \mid p \in \mathcal{P}\}.\end{aligned}$$

In these definitions, the occurrences of $+$, $\cdot$ and δ may be understood as referring to the algebraic structure of **Pe**, instead of to the syntactic structure of $\mathcal{P}$. This makes them essentially independent of the syntactic structure of the elements of $\mathcal{P}$; the only thing that matters is that $\mathcal{P}$ is the universe of an algebra **Pe** with two binary operations $+$ and $\cdot$, and with a distinguished element δ (e.g., **Pe** could just as well be the set of pCRL expressions with $+$, $\cdot$ and δ defined as before, or it could be the set of natural numbers with addition, multiplication, and the distinguished natural number 0).

That the schemata (A1)–(A7) make sense independent of the (syntactic) nature of the elements of $\mathcal{P}$, that is what makes this formal system genuinely algebraic. We define that an arbitrary algebraic structure $\mathbf{A} = \langle \mathsf{A}, +, \cdot, \delta\rangle$ *satisfies* (A1)–(A7) if each of the relations $(\mathrm{A1})_{\mathbf{A}}$–$(\mathrm{A7})_{\mathbf{A}}$ is included in the identity relation on A. The congruence ϑ induced on the algebra **Pe** includes each of the relations $(\mathrm{A1})_{\mathbf{Pe}}$–$(\mathrm{A7})_{\mathbf{Pe}}$, and from this it is easily concluded that the quotient algebra $\mathbf{Pe}/\vartheta$ satisfies (A1)–(A7). Incidentally, among the algebraic structures that satisfy (A1)–(A7), $\mathbf{Pe}/\vartheta$ is a special one, namely a free one with as free generators the congruence classes that contain an element of A.

So much for our digression, let us return to our earlier convention that $\mathcal{P}$ denotes the set of pCRL expressions associated with $\mathcal{A}$ and **D**. It seems quite natural to conceive $\mathcal{P}$ as an algebraic structure

$$\mathbf{Pe} = \langle \mathcal{P}, +, \cdot, \delta, \textstyle\sum_x, \lhd\, b \,\rhd\rangle_{x\in X, b\in\mathcal{B}}.$$

What we mean, is that **Pe** is an algebraic structure with binary operations $+$ and $\cdot$, and a distinguished element δ, and that it is further equipped with a sequence of unary operations $\sum_x$ indexed by a set of variables X, and with a sequence of binary operations $\lhd\, b \,\rhd$ indexed by a set of Boolean expressions $\mathcal{B}$. Additional justification for this point of view is provided by the fact that the relation

$$\vartheta = \{\langle p, q\rangle \in \mathcal{P}\times\mathcal{P} \mid \Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx q\}$$

is a congruence on **Pe**, due to the inference rules of $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$.

The proper question to ask next, is whether the axiom schemata of our deductive system make sense for arbitrary algebraic structures with two plus a $\mathcal{B}$-indexed sequence of binary operations, an X-indexed sequence of unary operations, and a distinguished element. From our earlier remarks it is clear that the schemata (A1)–(A7) make sense, and about the schemata (C1)–(C6), (Bool) and (Cq4)

similar remarks can be made. The remaining schemata deserve a more careful examination.

Consider the axiom schema (CQ1). It associates with **Pe** the binary relation

$$(\text{CQ1})_{\mathbf{Pe}} = \{\langle \textstyle\sum_x p, p\rangle \mid p \in \mathcal{P} \text{ and } x \in X \text{ such that } x \notin \mathsf{FV}(p)\}.$$

The difficulty is manifest: the definition of the relation $(\text{CQ1})_{\mathbf{Pe}}$ involves the predicate $x \notin \mathsf{FV}(\_)$, and thus it refers to a syntactic property of pCRL expressions. Similar difficulties arise when we consider the schemata (CQ2), (CQ5) and (QE). The schema (CQ3) reveals another kind of difficulty, which it shares with (CQ2) and (DATA). The binary relation

$$(\text{CQ3})_{\mathbf{Pe}} = \{\langle \textstyle\sum_x p, \sum_x p + p[x := d]\rangle \mid p \in \mathcal{P},\ x \in X \text{ and } d \in \mathcal{D}\}$$

depends on a particular syntactic accordance between p and $p[x := d]$. To complete our inventarisation of difficulties: the schema (CQ6) refers to a syntactic property of a condition, and the schema (EQ) refers to a syntactic relation between action expressions and a condition.

Having identified the algebraically unsatisfactory axiom schemata, we may wonder whether they tell us anything at all about the quotient algebra $\mathbf{Pe}/\vartheta$. One source of algebraic dissatisfaction was a proviso with the predicate $x \notin \mathsf{FV}(\_)$. Note that $x \notin \mathsf{FV}(\sum_x p)$ for all pCRL expressions p, so if we replace p by $\sum_x p$ in (CQ1), then we may safely omit the proviso; the quotient algebra $\mathbf{Pe}/\vartheta$ satisfies the schema

$$(\text{CQ1})' \quad \textstyle\sum_x \sum_x p \approx \sum_x p.$$

Similarly, (CQ5) may be transformed into

$$(\text{CQ5})' \quad (\textstyle\sum_x p) \cdot (\sum_x q) \approx \sum_x (p \cdot \sum_x q).$$

Note that, in view of (CQ1), replacing "q such that $x \notin \mathsf{FV}(q)$" by $\sum_x q$ in (CQ5) does not really result in a weaker schema. In the same manner, (QE) may be brought into an algebraically more pleasant (but equivalent) form.

We have found the algebraic counterpart of the statement "$x \notin \mathsf{FV}(p)$"; it corresponds to saying that "p is a pCRL expression that satisfies $\sum_x p \approx p$". In the present chapter we shall deal with the other algebraically unsatisfactory aspects as well, with the goal of finding a complete, and purely algebraic characterisation of the quotient algebra $\mathbf{Pe}/\vartheta$. At this point we lack the language to explain the algebraic counterparts of the other difficulties mentioned above. Nevertheless, there is an underlying idea that deserves mention ahead of things: if x does not occur in d, then

$$\Pi(\mathcal{A}, \mathbf{D})^{\text{eq}}_{\exists} \vdash p[x := d] \approx \textstyle\sum_x p \lhd \text{eq}(x, d) \rhd \delta \quad \text{(see Corollary 6.11 below).}$$

This will be used to eliminate the notion of substitution from the schemata concerned.

Before we move on to lay out the algebraic framework in which our project is to be carried out, it is appropriate to make two further remarks regarding

the conditional. The first remark concerns its arity. We have announced that the conditional gives rise to a sequence of binary operations indexed by Boolean expressions. But actually, as a brief glance on earlier chapters will readily reveal, we have a strong preference for conditionals in the form of guarded commands (i.e., with δ in the position of its right argument). To make our preference official, the algebras to be defined will be equipped with a sequence of unary guarded commands, instead of a sequence of binary conditional compositions.

The other remark is about the index set. To make our treatment still more independent of syntax, and thus more algebraic in spirit, we use as indices the elements of an algebra, rather than the expressions of a language. Naturally, this algebra, say $\mathbf{B}$, should be a Boolean algebra. Furthermore, this is a good moment to take advantage of our experience. We have argued that, in view of the results of Chapter 4, there are good reasons to require full first-order expressiveness of the Booleans (see the beginning of Section 5.3). Therefore, we shall require in addition that $\mathbf{B}$ is equipped with a sequence of unary operations $(\exists x)$ (one for every $x \in X$) and that it contains a sequence of distinguished elements $\mathrm{eq}(x, y)$ (one for every two $x, y \in X$), so that $\mathbf{B}$ is a *cylindric algebra*. Cylindric algebras have been introduced by Tarski and others, and they stand to first-order predicate logic with equality in the same relation as Boolean algebras stand to propositional logic.

6.1 ω-dimensional basic process modules

Consider the generalised basic process algebra with deadlock $\mathbf{T_D}(\mathcal{A})$ introduced in Section 3.4. We have shown that it is a pCRL-complete generalised basic process algebra with deadlock (i.e., an element of $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$), and hence suitable as a semantics of our formal system. As a matter of fact, it is an initial element of the class $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$ (cf. Theorem 3.15 on p. 40). In Section 3.3 we have explained how a pCRL expression p together with a valuation ν denotes a unique pCRL tree $\iota_\nu(p)$, via the interpretation homomorphism $\iota_\nu : \mathbf{Pol}(\mathcal{A}, \mathbf{D}) \to \mathbf{T_D}(\mathcal{A})$.

Let us now fix, for the remainder of this chapter, a particular enumeration (without repetitions) of the variables

$$X = x_0, x_1, \ldots, x_k, \ldots \qquad (k < \omega).$$

It gives rise to a one-to-one correspondence between valuations and elements of the cartesian power D^ω (precisely: the valuation ν corresponds to the element $\nu(x_0), \nu(x_1), \ldots, \nu(x_k), \ldots$ $(k < \omega)$ of D^ω). It requires only this minor shift of perspective to regard a pCRL expression p as a finite specification of the function from D^ω into $\mathbf{T_D}(\mathcal{A})$ that associates with every valuation ν the pCRL tree $\iota_\nu(p)$.

Since $\mathbf{T_D}(\mathcal{A})$ is initial in $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$, and since $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$ is sound and complete with respect to $\mathsf{GBPA}_\delta(\mathcal{A}, \mathbf{D})$, two pCRL expressions are provably equivalent if, and only if, they specify the same function. This gives us a mathematically attractive alternative for the elements of $\mathbf{Pe}/\vartheta$; we may view them as functions from D^ω into $\mathbf{T_D}(\mathcal{A})$. It is also convenient to be a little more general. Let $\mathbf{A} = \langle \mathsf{A}, +, \cdot, \delta, \sum \rangle$ be the maximal generalisation of an arbitrary basic process

algebra with deadlock $\langle \mathsf{A}, +, \cdot, \delta \rangle$. We consider

$$\mathcal{F} = (\mathsf{D}^\omega \to \mathsf{A}),$$

the set of all functions from D^ω into A. An element $\mathsf{d}_0, \mathsf{d}_1, \ldots, \mathsf{d}_k, \ldots$ $(k < \omega)$ of D^ω we call a *point*, and we denote it by $\{\mathsf{d}_k\}$; for each $i < \omega$, d_i is the *ith coordinate* of $\{\mathsf{d}_k\}$. An element of $\mathcal{F}$ assigns a process from $\mathbf{A}$ to each point, and may thus be thought of as an ω-dimensional Cartesian space with in each point a process from $\mathbf{A}$. We denote by δ the space that has δ in each point, and we define pointwise binary operations $+$ and $\cdot$ on $\mathcal{F}$. That is, for all $F, G \in \mathcal{F}$ and for all $\{\mathsf{d}_k\} \in \mathsf{D}^\omega$,

$$\begin{aligned}
&(F + G)(\{\mathsf{d}_k\}) = F(\{\mathsf{d}_k\}) + G(\{\mathsf{d}_k\}),\\
&(F \cdot G)(\{\mathsf{d}_k\}) = F(\{\mathsf{d}_k\}) \cdot G(\{\mathsf{d}_k\}) \text{ and}\\
&\delta(\{\mathsf{d}_k\}) = \delta.
\end{aligned}$$

We could now proceed to define on $\mathcal{F}$ a generalised operation $\sum$, also pointwise, but we prefer an alternative that makes use of the extra structure of $\mathcal{F}$. We write $\{\mathsf{d}_k\} \sim_i \{\mathsf{e}_k\}$ if the points $\{\mathsf{d}_k\}$ and $\{\mathsf{e}_k\}$ agree on each coordinate except possibly the ith, i.e.,

$$\{\mathsf{d}_k\} \sim_i \{\mathsf{e}_k\} \quad \text{if, and only if,} \quad \mathsf{d}_k = \mathsf{e}_k \text{ for all } k \in \omega - \{i\}.$$

The set of all points $\{\mathsf{e}_k\}$ such that $\{\mathsf{d}_k\} \sim_i \{\mathsf{e}_k\}$ we shall henceforth refer to as the *line* through $\{\mathsf{d}_k\}$ parallel to the ith coordinate axis. We associate with every $i < \omega$ a partial unary operation $\mathbf{s}_i : \mathcal{F} \to \mathcal{F}$ such that

$$(\mathbf{s}_i F)(\{\mathsf{d}_k\}) = \textstyle\sum\{F(\{\mathsf{e}_k\}) \mid \{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}\};$$

$\mathbf{s}_i$ is defined on F provided that, for all $\{\mathsf{d}_k\} \in \mathsf{D}^\omega$, the set $\{F(\{\mathsf{e}_k\}) \mid \{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}\}$ is admissible for $\sum$ in $\mathbf{A}$. That is, the operation $\mathbf{s}_i$ replaces the element in every point $\{\mathsf{d}_k\}$ by the generalised sum of all the elements on the line through $\{\mathsf{d}_k\}$ parallel to the ith coordinate axis; we call $\mathbf{s}_i$ the *projective summation* along i. Note that if $F = \mathbf{s}_i G$ for some $G \in \mathcal{F}$, then the elements of A on the line through a point $\{\mathsf{d}_k\} \in \mathsf{D}^\omega$ parallel to the ith coordinate axis are all the same; formally:

$$\{\mathsf{d}_k\} \sim_i \{\mathsf{e}_k\} \text{ implies } F(\{\mathsf{d}_k\}) = F(\{\mathsf{e}_k\}), \text{ for all } \{\mathsf{d}_k\}, \{\mathsf{e}_k\} \in \mathsf{D}^\omega. \tag{6.1}$$

If $F \in \mathcal{F}$ satisfies (6.1) we call it *uniform* along i.

The operations $\mathbf{s}_i$ $(i < \omega)$ satisfy the equalities in Table 6.1 with p and q ranging over $\mathcal{F}$. We shall not detail a proof of this; what we shall do is give some intuitions. According to (Cs1), projective summations commute; the composition $\mathbf{s}_i \mathbf{s}_j$ is the transformation that replaces the element in every point $\{\mathsf{d}_k\} \in \mathsf{D}^\omega$ by the generalised sum of the elements at the points $\{\mathsf{e}_k\}$ such that $\{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}$ or $\{\mathsf{e}_k\} \sim_j \{\mathsf{d}_k\}$. If $F \in \mathcal{F}$ is uniform along i, then, since $\sum\{\mathsf{p}\} = \mathsf{p}$ in every GBPA_δ, applying $\mathbf{s}_i$ to it has no effect; this explains (Cs2) and (Cs6), since $\mathbf{s}_i F$ and $\mathbf{s}_i \delta$ are both uniform along i. According to (Cs3), $F \leq \mathbf{s}_i F$ with respect to the partial order $\leq$ induced on $\mathcal{F}$ by $+$; this is an immediate consequence of

(Cs1)	$\mathbf{s}_i\mathbf{s}_j\mathsf{p}$	$= \mathbf{s}_j\mathbf{s}_i\mathsf{p}$
(Cs2)	$\mathbf{s}_i\mathbf{s}_i\mathsf{p}$	$= \mathbf{s}_i\mathsf{p}$
(Cs3)	$x + \mathbf{s}_i\mathsf{p}$	$= \mathbf{s}_i\mathsf{p}$
(Cs4)	$\mathbf{s}_i(\mathsf{p} + \mathsf{q})$	$= \mathbf{s}_i\mathsf{p} + \mathbf{s}_i\mathsf{q}$
(Cs5)	$\mathbf{s}_i(\mathsf{p} \cdot \mathbf{s}_i\mathsf{q})$	$= \mathbf{s}_i\mathsf{p} \cdot \mathbf{s}_i\mathsf{q}$
(Cs6)	$\mathbf{s}_i\delta$	$= \delta$

Table 6.1: The axioms for the projective summations in an ω-dimensional basic process algebra with deadlock ($i, j < \omega$).

(Ga1) in Table 2.2. By (Cs4), projective summations distribute over alternative compositions, an immediate consequence of the fact that

$$\sum\{\mathsf{p}' + \mathsf{p}'' \mid \mathsf{p}' \in \mathsf{P}',\ \mathsf{p}'' \in \mathsf{P}''\} = \sum \mathsf{P}' + \sum \mathsf{P}''$$

in every generalised basic process algebra with deadlock. To understand why (Cs5) is valid, first note that, by definition,

$$(\mathbf{s}_i(F \cdot \mathbf{s}_i G))(\{\mathsf{d}_k\}) = \sum\{F(\{\mathsf{e}_k\}) \cdot (\mathbf{s}_i G)(\{\mathsf{e}_k\}) \mid \{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}\}.$$

Since $(\mathbf{s}_i G)$ is uniform along i, $(\mathbf{s}_i G)(\{\mathsf{e}_k\}) = (\mathbf{s}_i G)(\{\mathsf{d}_k\})$ for all $\{\mathsf{e}_k\}$ such that $\{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}$, so

$$(\mathbf{s}_i(F \cdot \mathbf{s}_i G))(\{\mathsf{d}_k\}) = \sum\{F(\{\mathsf{e}_k\}) \cdot (\mathbf{s}_i G)(\{\mathsf{d}_k\}) \mid \{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}\}.$$

With an application of (Ga3) we may pull out $(\mathbf{s}_i G)(\{\mathsf{d}_k\})$ in the right-hand side to obtain $(\mathbf{s}_i F \cdot \mathbf{s}_i G)(\{\mathsf{d}_k\})$.

Definition 6.1 An *ω-dimensional* basic process algebra with deadlock is an algebra $\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \mathbf{s}_i \rangle_{i<\omega}$ that consists of a basic process algebra with deadlock $\langle \mathsf{P}, +, \cdot, \delta \rangle$ (see Table 2.1 on p. 17) together with a sequence

$$\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_k, \ldots \qquad (k < \omega)$$

of unary operations that satisfy the equalities in Table 6.1 for all $\mathsf{p}, \mathsf{q} \in \mathsf{P}$. For $i < \omega$, the operation $\mathbf{s}_i$ is called the *ith projective summation*.

We shall see later that the projective summations just introduced correspond to the unary operations that choice quantifiers induce on the set of pCRL expressions modulo provable equivalence. We now wish to define on $\mathcal{F}$ unary operations that correspond to guarded commands. We have already announced that we would like to incorporate them as a sequence of unary operations indexed by the elements of a cylindric algebra. Let us first recapitulate a few basic facts and intuitions from the theory of cylindric algebras (see the books of Henkin *et al.* (1971, 1985) for a thorough treatment).

(BA1)	$\mathsf{a} \vee (\mathsf{b} \vee \mathsf{c})$	$= (\mathsf{a} \vee \mathsf{b}) \vee \mathsf{c}$
(BA2)	$\mathsf{b} \vee \mathsf{c}$	$= \mathsf{c} \vee \mathsf{b}$
(BA3)	$\mathsf{b} \vee (\mathsf{b} \wedge \mathsf{c})$	$= \mathsf{b}$
(BA4)	$\mathsf{a} \vee (\mathsf{b} \wedge \mathsf{c})$	$= (\mathsf{a} \vee \mathsf{b}) \wedge (\mathsf{a} \vee \mathsf{c})$
(BA5)	$\mathsf{b} \vee \neg\,\mathsf{b}$	$= \top$
(BA1′)	$\mathsf{a} \wedge (\mathsf{b} \wedge \mathsf{c})$	$= (\mathsf{a} \wedge \mathsf{b}) \wedge \mathsf{c}$
(BA2′)	$\mathsf{b} \wedge \mathsf{c}$	$= \mathsf{c} \wedge \mathsf{b}$
(BA3′)	$\mathsf{b} \wedge (\mathsf{b} \vee \mathsf{c})$	$= \mathsf{b}$
(BA4′)	$\mathsf{a} \wedge (\mathsf{b} \vee \mathsf{c})$	$= (\mathsf{a} \wedge \mathsf{b}) \vee (\mathsf{a} \wedge \mathsf{c})$
(BA5′)	$\mathsf{b} \wedge \neg\,\mathsf{b}$	$= \bot$

Table 6.2: The axioms of Boolean algebras.

The idea of conceiving pCRL expressions modulo provable equivalence as functions from D^ω into $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ has an analogy for Boolean expressions modulo equivalence in $\mathbf{D}$. Accordingly, a Boolean expression b may thought of as a specification of a function from D^ω into the two-element Boolean algebra. We consider a Boolean expression b as a specification of the set of valuations under which b evaluates to $\top$ (this amounts to the same thing), and proceed to consider the powerset of D^ω.

Definition 6.2 A Boolean algebra is an algebra $\langle \mathsf{B}, \vee, \wedge, \neg, \top, \bot \rangle$ that satisfies for all $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathsf{B}$ the equalities in Table 6.2.

It is well-known that the powerset of any set is the universe of a Boolean algebra with $\vee$, $\wedge$, $\neg$ as (set-theoretic) union, intersection and complementation, respectively, and with the entire set as the distinguished element $\top$ and $\emptyset$ as the distinguished element $\bot$ (see, e.g., Koppelberg, 1989). We consider the powerset of D^ω as such a Boolean algebra, and additionally, we correlate with every $i < \omega$ a unary operation $\mathbf{c}_i$ on the powerset of D^ω such that for all $U \subseteq \mathsf{D}^\omega$

$$\mathbf{c}_i U = \{\{\mathsf{d}_k\} \in \mathsf{D}^\omega \mid \text{there exists } \{\mathsf{e}_k\} \in U \text{ such that } \{\mathsf{d}_k\} \sim_i \{\mathsf{e}_k\}\}.$$

A set $U \subseteq \mathsf{D}^\omega$ such that $\{\mathsf{d}_k\} \in U$ implies $\{\mathsf{e}_k\} \in U$ for all $\{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}$, is called a *cylinder* parallel to the ith axis or, for brevity, an *i-cylinder*. The operation $\mathbf{c}_i$ is called the ith *cylindrification*; when applied to the set $U \subseteq \mathsf{D}^\omega$, it yields the i-cylinder $\mathbf{c}_i U$ swept out by all translations of U parallel to the ith coordinate axis. Furthermore, we treat, for every $i, j < \omega$, the set

$$\mathbf{d}_{ij} = \{\{\mathsf{d}_k\} \in \mathsf{D}^\omega \mid \mathsf{d}_i = \mathsf{d}_j\}$$

as a distinguished element. The set $\mathbf{d}_{ij}$ is called a *diagonal element*; it consists of all points whose ith coordinates are equal to their jth coordinates. The cylindrifications and diagonal elements satisfy the equalities in Table 6.3, and this makes the powerset of D^ω into an ω-dimensional cylindric algebra.

(CA1)	$\mathbf{c}_i\bot$	$= \bot$
(CA2)	$\mathsf{b} \vee \mathbf{c}_i\mathsf{b}$	$= \mathbf{c}_i\mathsf{b}$
(CA3)	$\mathbf{c}_i(\mathsf{b} \wedge \mathbf{c}_i\mathsf{b}')$	$= \mathbf{c}_i\mathsf{b} \wedge \mathbf{c}_i\mathsf{b}'$
(CA4)	$\mathbf{c}_i\mathbf{c}_j\mathsf{b}$	$= \mathbf{c}_j\mathbf{c}_i\mathsf{b}$
(CA5)	$\mathbf{d}_{ii}$	$= \top$
(CA6)	if $i \neq j, k$, then $\mathbf{d}_{jk}$	$= \mathbf{c}_i(\mathbf{d}_{ji} \wedge \mathbf{d}_{ik})$
(CA7)	if $i \neq j$, then $\mathbf{c}_i(\mathbf{d}_{ij} \wedge \mathsf{b}) \wedge \mathbf{c}_i(\mathbf{d}_{ij} \wedge \neg\mathsf{b})$	$= \bot$

Table 6.3: The axioms for cylindrifications and diagonal elements in an ω-dimensional cylindric algebra ($i, j, k < \omega$).

Definition 6.3 An *ω-dimensional cylindric algebra* is an algebra

$$\mathbf{C} = \langle \mathsf{C}, \vee, \wedge, \neg, \top, \bot, \mathbf{c}_i, \mathbf{d}_{ij} \rangle_{i,j<\omega}$$

that consists of a Boolean algebra $\langle \mathsf{C}, \vee, \wedge, \neg, \top, \bot \rangle$ (see Table 6.2), with unary operations $\mathbf{c}_i : \mathsf{C} \to \mathsf{C}$ ($i < \omega$) and distinguished elements $\mathbf{d}_{ij} \in \mathsf{C}$ ($i, j < \omega$) that satisfy the axioms in Table 6.3 for all $\mathsf{b}, \mathsf{b}' \in \mathsf{C}$. The operations $\mathbf{c}_i$ are called *cylindrifications*, and the elements $\mathbf{d}_{ij}$ are called *diagonal elements*.

The theory of cylindric algebras has been designed for the purpose of algebraising first-order predicate logic with equality. To illustrate the correspondence between the ith cylindrification and the existential quantifier ($\exists x_i$), let U be the set of all valuations under which the Boolean expression b evaluates to $\top$. If $\{\mathsf{d}_k\} \in \mathbf{c}_i U$, then there exists a valuation $\{\mathsf{e}_k\} \in U$ such that $\{\mathsf{d}_k\} \sim_i \{\mathsf{e}_k\}$; the formula $(\exists x_i)b$ evaluates to $\top$ under any such valuation $\{\mathsf{d}_k\}$. Conversely, if $(\exists x_i)b$ evaluates to $\top$ under the valuation $\{\mathsf{d}_k\}$, then there exists a valuation $\{\mathsf{e}_k\} \sim_i \{\mathsf{d}_k\}$ such that b evaluates to $\top$ under $\{\mathsf{e}_k\}$, so $\{\mathsf{d}_k\} \in \mathbf{c}_i U$ by definition. There is a similar correspondence between the diagonal element $\mathbf{d}_{ij}$ and the Boolean expression $\mathrm{eq}(x_i, x_j)$; $\mathbf{d}_{ij}$ consists precisely of those valuations under which $\mathrm{eq}(x_i, x_j)$ evaluates to $\top$.

We now combine the theory of cylindric algebras with that of ω-dimensional basic process algebras with deadlock. If p is a pCRL expression, describing a function F from D^ω into $\mathbf{T}_\mathbf{D}(\mathcal{A})$, and b is a Boolean expression, describing a subset U of D^ω, then the pCRL expression $p \lhd b \rhd \delta$ describes the function G from D^ω into $\mathbf{T}_\mathbf{D}(\mathcal{A})$ such that $G(\{\mathsf{d}_k\}) = F(\{\mathsf{d}_k\})$ if $\{\mathsf{d}_k\} \in U$, and $G(\{\mathsf{d}_k\}) = \delta$ otherwise. We associate with every $U \subseteq \mathsf{D}^\omega$ a transformation $U{:}\!\to$ on $\mathcal{F}$ that replaces the element in a point outside U by δ and leaves the points in U unchanged, i.e.,

$$(U{:}\!\to F)(\{\mathsf{d}_k\}) = \begin{cases} F(\{\mathsf{d}_k\}) & \text{if } \{\mathsf{d}_k\} \in U; \text{ and} \\ \delta & \text{otherwise.} \end{cases}$$

We contend that the transformations $U{:}\!\to$ satisfy the equalities in Table 6.4 with p and q ranging over $\mathcal{F}$. The verifications are fairly straightforward; we shall only discuss (GC9)–(GC11).

To demonstrate the validity of (GC9), let $U \subseteq \mathsf{D}^\omega$ and $F \in \mathcal{F}$, and consider a line ℓ parallel to the ith coordinate axis. Note that it suffices to show that the

(GC1)	$\top{:}\to \mathsf{p}$	$= \mathsf{p}$	(GC5)	$\mathsf{b}{:}\to \delta$	$= \delta$
(GC2)	$\bot{:}\to \mathsf{p}$	$= \delta$	(GC6)	$\mathsf{b}{:}\to(\mathsf{p}+\mathsf{q})$	$= \mathsf{b}{:}\to \mathsf{p} + \mathsf{b}{:}\to \mathsf{q}$
(GC3)	$\mathsf{b} \vee \mathsf{c}{:}\to \mathsf{p}$	$= \mathsf{b}{:}\to \mathsf{p} + \mathsf{c}{:}\to \mathsf{p}$	(GC7)	$\mathsf{b}{:}\to(\mathsf{p}\cdot\mathsf{q})$	$= (\mathsf{b}{:}\to \mathsf{p})\cdot \mathsf{q}$
(GC4)	$\mathsf{b}{:}\to(\mathsf{c}{:}\to \mathsf{p})$	$= \mathsf{b} \wedge \mathsf{c}{:}\to \mathsf{p}$	(GC8)	$(\mathsf{b}{:}\to \mathsf{p})\cdot \mathsf{q}$	$= (\mathsf{b}{:}\to \mathsf{p})\cdot(\mathsf{b}{:}\to \mathsf{q})$

(GC9)	$\mathbf{s}_i(\mathsf{b}{:}\to \mathbf{s}_i\mathsf{p}) = \mathbf{c}_i\mathsf{b}{:}\to \mathbf{s}_i\mathsf{p}$
(GC10)	$\mathbf{s}_i(\mathbf{c}_i\mathsf{b}{:}\to \mathsf{p}) = \mathbf{c}_i\mathsf{b}{:}\to \mathbf{s}_i\mathsf{p}$
(GC11)	if $i \neq j$, then $\mathbf{d}_{ij}{:}\to \mathbf{s}_i(\mathbf{d}_{ij}{:}\to \mathsf{p}) = \mathbf{d}_{ij}{:}\to \mathsf{p}$

Table 6.4: The axioms for the guarded command in an ω-dimensional basic process module over $\mathbf{C}$ ($i, j < \omega$ and $\mathsf{b}, \mathsf{c} \in \mathbf{C}$).

sets $(\mathbf{s}_i(U{:}\to \mathbf{s}_iF))(\ell)$ and $(\mathbf{c}_iU{:}\to \mathbf{s}_iF)(\ell)$ are singletons and equal. Since $\mathbf{s}_iF$ is uniform along i, $(\mathbf{s}_iF)(\ell)$ is a singleton, say $(\mathbf{s}_iF)(\ell) = \{\mathsf{a}\}$ with a an element of $\mathbf{A}$. Now there are two cases. If U contains a point on ℓ, then $\mathbf{c}_iU$ contains every point on ℓ, so $(\mathbf{c}_iU{:}\to \mathbf{s}_iF)(\ell) = \{\mathsf{a}\}$. To see that also $(\mathbf{s}_i(U{:}\to \mathbf{s}_iF))(\ell) = \{\mathsf{a}\}$, observe that an application of $U{:}\to$ has the effect of replacing perhaps some, but certainly not all elements on ℓ by δ, i.e., $(U{:}\to \mathbf{s}_iF)(\ell) = \{\delta, \mathsf{a}\}$, and that a subsequent application of $\mathbf{s}_i$ replaces every element on ℓ by $\sum\{\delta, \mathsf{a}\}$. Since $\sum\{\delta, \mathsf{a}\} = \mathsf{a}$ in $\mathbf{A}$, it follows that $(\mathbf{s}_i(U{:}\to \mathbf{s}_iF))(\ell) = \{\mathsf{a}\}$. In the other case, U does not contain a point on ℓ, so that $\mathbf{c}_iU$ does not contain a point on ℓ either, whence $(\mathbf{c}_iU{:}\to \mathbf{s}_iF)(\ell) = \{\delta\}$. To see that also $(\mathbf{s}_i(U{:}\to \mathbf{s}_iF))(\ell) = \{\delta\}$, note that an application of $U{:}\to$ has the effect of replacing all elements on ℓ by δ, i.e., $(U{:}\to \mathbf{s}_iF)(\ell) = \{\delta\}$; since $\sum\{\delta\} = \delta$ in $\mathbf{A}$, it follows that $(\mathbf{s}_i(U{:}\to \mathbf{s}_iF))(\ell) = \{\delta\}$.

Next, we want to demonstrate the validity of (GC10). To this end, we assume that $U \subseteq \mathsf{D}^\omega$ and $F \in \mathcal{F}$, and we consider a line ℓ parallel to the ith coordinate axis. Note that $\mathbf{c}_iU$, being an i-cylinder, either contains every point on ℓ or no point on ℓ at all. In the first case, $(\mathbf{c}_iU{:}\to F)(\ell) = F(\ell)$, whence

$$(\mathbf{s}_i(\mathbf{c}_iU{:}\to F))(\ell) = \{\textstyle\sum F(\ell)\}$$

and also

$$(\mathbf{c}_iU{:}\to \mathbf{s}_iF)(\ell) = (\mathbf{s}_iF)(\ell) = \{\textstyle\sum F(\ell)\}.$$

In the second case, $(\mathbf{c}_iU{:}\to \mathbf{s}_iF)(\ell) = \{\delta\}$, and also $(\mathbf{c}_iU{:}\to F)(\ell) = \{\delta\}$, whence, since $\sum\{\delta\} = \delta$ in $\mathbf{A}$, $(\mathbf{s}_i(\mathbf{c}_iU{:}\to F))(\ell) = \{\delta\}$.

We assume $i \neq j$, and prove the equality of (GC11). Consider a line ℓ parallel to the ith coordinate axis. The effect of $\mathbf{d}_{ij}{:}\to$ is that the element in every point on ℓ is replaced by δ, except the element in the single point $\{\mathsf{d}_k\}$ on ℓ with the jth coordinate equal to the ith coordinate (note that for $\{\mathsf{d}_k\}$ to be unique it is imperative that $i \neq j$), i.e., for all $F \in \mathcal{F}$, $(\mathbf{d}_{ij}{:}\to F)(\{\mathsf{d}_k\}) = F(\{\mathsf{d}_k\})$ and $(\mathbf{d}_{ij}{:}\to F)(\{\mathsf{e}_k\}) = \delta$ if $\{\mathsf{e}_k\}$ is another point on ℓ, distinct from $\{\mathsf{d}_k\}$. The effect of $\mathbf{s}_i$ is that each element on ℓ is replaced by the generalised sum of the set of all the elements on ℓ. Hence, since $\sum\{\mathsf{a}\} = \mathsf{a}$ for every a in $\mathbf{A}$, the composite transformation $\mathbf{s}_i\,\mathbf{d}_{ij}{:}\to$ replaces every element on ℓ by the element in $\{\mathsf{d}_k\}$. What matters

is that the element in $\{\mathsf{d}_k\}$ is invariant under the transformations $\mathbf{s}_i\,\mathbf{d}_{ij}{:}\!\rightarrow$, i.e., that $(\mathbf{s}_i(\mathbf{d}_{ij}{:}\!\rightarrow F))(\{\mathsf{d}_k\}) = F(\{\mathsf{d}_k\})$. So, $(\mathbf{d}_{ij}{:}\!\rightarrow \mathbf{s}_i(\mathbf{d}_{ij}{:}\!\rightarrow F))(\{\mathsf{d}_k\}) = F(\{\mathsf{d}_k\})$, whereas $(\mathbf{d}_{ij}{:}\!\rightarrow \mathbf{s}_i(\mathbf{d}_{ij}{:}\!\rightarrow F))(\{\mathsf{e}_k\}) = \delta$ if $\{\mathsf{e}_k\}$ is another point on ℓ, distinct from $\{\mathsf{d}_k\}$. We conclude that $\mathbf{d}_{ij}{:}\!\rightarrow \mathbf{s}_i(\mathbf{d}_{ij}{:}\!\rightarrow F) = \mathbf{d}_{ij}{:}\!\rightarrow F$ for all $F \in \mathcal{F}$.

Definition 6.4 Let $\mathbf{C}$ be an ω-dimensional cylindric algeba with universe C. An *ω-dimensional basic process module over* $\mathbf{C}$ is an algebra

$$\mathbf{P} = \langle \mathsf{P}, +, \cdot, \delta, \mathbf{s}_i, \mathsf{b}{:}\!\rightarrow \rangle_{i<\omega,\ \mathsf{b}\in\mathbf{C}}$$

consisting of an ω-dimensional basic process algebra with deadlock $\langle \mathsf{P}, +, \cdot, \delta, \mathbf{s}_i\rangle_{i<\omega}$ equipped with unary operations $\mathsf{b}{:}\!\rightarrow$ ($\mathsf{b} \in \mathbf{C}$) that satisfy the equalities in Table 6.4 for all $\mathsf{p}, \mathsf{q} \in \mathsf{P}$. The operations $\mathsf{b}{:}\!\rightarrow$ are called *guarded commands* and b is the *guard* of $\mathsf{b}{:}\!\rightarrow$. The class of all ω-dimensional basic process modules over $\mathbf{C}$ we denote by $\mathbf{C}$-BPM_ω.

Recall that the unary operations $\mathbf{s}_i$ on $\mathcal{F}$ are partial; their definition depends on the presence of generalised sums in $\mathbf{A}$. As a consequence, to get a decent algebraic structure, we need to take a subset of $\mathcal{F}$ that is closed under the operations of cylindric basic process algebras, in particular under $\mathbf{s}_i$ for every $i < \omega$. Let $\mathcal{F}^*$ be any subset of $\mathcal{F}$ that contains δ, and that is closed under $+$, $\cdot$, $\mathbf{s}_i$ ($i < \omega$) and $U{:}\!\rightarrow$ ($U \subseteq \mathsf{D}^\omega$); then

$$\mathbf{F}^* = \langle \mathcal{F}^*, +, \cdot, \delta, \mathbf{s}_i, U{:}\!\rightarrow \rangle_{i<\omega,\ U\subseteq\mathsf{D}^\omega}$$

is an ω-dimensional basic process module over the cylindric algebra of subsets of D^ω.

Remark 6.5 The definitions above can be generalised by replacing 'ω' everywhere by an arbitrary ordinal (Henkin *et al.* do this for cylindric algebras in their books). Then, e.g., the theory of basic process algebras with deadlock coincides with the theory of 0-dimensional basic process algebras with deadlock. We do not need such generality here. For the sake of conciseness, we shall often suppress the adjective 'ω-dimensional', adopting the convention that 'cylindric algebra' always means 'ω-dimensional cylindric algebra', and that 'basic process module' always means 'ω-dimensional basic process module'. In contrast, 'basic process algebra with deadlock' will retain its old meaning; if we mean 'ω-dimensional basic process algebra with deadlock', then we shall always explicitly say so.

6.2 Comparing formal systems

If we have an algebraic framework that subsumes a certain formal system, then it frequently provides a convenient context for discussing the correspondence between this formal system and other formal systems. In this section we shall encounter two examples of this.

Our first example concerns the correspondence between the set Φ of first-order formulas and the set $\mathcal{B}$ of Boolean expressions associated with a data algebra $\mathbf{D}$;

we shall discuss it in the framework of cylindric algebras. Consider the set Φ of first-order formulas associated with $\mathbf{D}$ as an algebraic structure

$$\mathbf{Fm} = \langle \Phi, \vee, \wedge, \neg, \top, \bot, \mathbf{c}_i, \mathbf{d}_{ij} \rangle_{i,j<\omega}$$

with $\mathbf{c}_i : \Phi \to \Phi$ defined by $\mathbf{c}_i\varphi = (\exists x_i)\varphi$, and $\mathbf{d}_{ij} = \mathrm{eq}(x_i, x_j) \in \Phi$, for all $i, j < \omega$ (we assume that $\mathbf{D}$ has equality). The relation

$$\leftrightarrow_{\mathbf{D}} = \{\langle \varphi, \psi \rangle \in \Phi \times \Phi \mid \mathbf{D} \models \varphi \leftrightarrow \psi\}$$

is a congruence on $\mathbf{Fm}$, and the quotient algebra satisfies the axioms of cylindric algebras (as listed in Tables 6.2 and 6.3).

Theorem 6.6 Suppose that $\mathbf{D}$ has equality. The algebra $\mathbf{Fm}/{\leftrightarrow_{\mathbf{D}}}$ is a cylindric algebra; it is called the *cylindric algebra of formulas* associated with $\mathbf{D}$.

Proof. See Henkin *et al.* (1985) for details. □

Next, consider the set $\mathcal{B}$ of Boolean expressions, it is in a natural way the universe of an algebraic structure

$$\mathbf{Be} = \langle \mathcal{B}, \vee, \wedge, \neg, \top, \bot \rangle$$

similar to Boolean algebras, and the relation

$$\approx_{\mathbf{D}} = \{\langle b, c \rangle \in \mathcal{B} \times \mathcal{B} \mid \mathbf{D} \models b \approx c\}$$

is a congruence on $\mathbf{Be}$. Clearly, the quotient $\mathbf{Be}/{\approx_{\mathbf{D}}}$ is a Boolean algebra.

Recall that every open first-order formula is a Boolean expression; according to the following proposition, the relations $\leftrightarrow_{\mathbf{D}}$ and $\approx_{\mathbf{D}}$ coincide on the set of open first-order formulas.

Proposition 6.7 If φ and ψ are open first-order formulas, then

$$\mathbf{D} \models \varphi \leftrightarrow \psi \text{ if, and only if, } \mathbf{D} \models \varphi \approx \psi.$$

Proof. By Proposition 4.2 on p. 53 and (4.3) on p. 54, it suffices to prove that

$$\mathbf{D}, \nu \models \varphi \approx \psi \text{ if, and only if, } \mathbf{D}, \nu \models (\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi) \approx \top$$

for every valuation ν.

To establish the implication from left to right, we assume $\bar{\nu}(\varphi) = \bar{\nu}(\psi)$ and derive $\bar{\nu}((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)) = \top$. Note that $\neg \mathsf{b} \vee \mathsf{b} = \top$ for all $\mathsf{b} \in \mathsf{B}$, so that we may conclude from $\bar{\nu}(\varphi) = \bar{\nu}(\psi)$ that $\neg\,\bar{\nu}(\varphi) \vee \bar{\nu}(\psi) = \top$ and $\neg\,\bar{\nu}(\psi) \vee \bar{\nu}(\varphi) = \top$. Then, using the definitions on p. 32, we derive

$$\begin{aligned}&\bar{\nu}((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi))\\ &\quad = (\neg\,\bar{\nu}(\varphi) \vee \bar{\nu}(\psi)) \wedge (\neg\,\bar{\nu}(\psi) \vee \bar{\nu}(\varphi)) = \top \wedge \top = \top.\end{aligned}$$

To establish the implication from right to left, we assume $\bar{\nu}(\varphi) \neq \bar{\nu}(\psi)$ and derive $\bar{\nu}((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)) = \bot$. Note that, since B has two elements and by the definition of $\neg$, $\bar{\nu}(\varphi) \neq \bar{\nu}(\psi)$ implies

$$\neg\,\bar{\nu}(\varphi) = \bar{\nu}(\psi) \text{ and } \neg\,\bar{\nu}(\psi) = \bar{\nu}(\varphi).$$

Hence, since $\mathsf{b} \vee \mathsf{b} = \mathsf{b}$ for all $\mathsf{b} \in \mathsf{B}$,

$$\neg\,\bar{\nu}(\varphi) \vee \bar{\nu}(\psi) = \bar{\nu}(\psi) \text{ and } \neg\,\bar{\nu}(\psi) \vee \bar{\nu}(\varphi) = \bar{\nu}(\varphi).$$

Furthermore, since $\neg\,\mathsf{b} \wedge \mathsf{b} = \bot$ for all $\mathsf{b} \in \mathsf{B}$,

$$\bar{\nu}(\psi) \wedge \bar{\nu}(\varphi) = \bot.$$

So, we get

$$\begin{aligned}&\bar{\nu}((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi))\\ &\quad = (\neg\,\bar{\nu}(\varphi) \vee \bar{\nu}(\psi)) \wedge (\neg\,\bar{\nu}(\psi) \vee \bar{\nu}(\varphi)) = \bar{\nu}(\psi) \wedge \bar{\nu}(\varphi) = \bot.\end{aligned}$$

The proof of the proposition is complete. □

Let us write $[b]$ for the equivalence class of Boolean expressions in $\mathbf{Be}/{\approx_{\mathbf{D}}}$ that contains the Boolean expression b, and $[\varphi]$ for the equivalence class of first-order formulas in $\mathbf{Fm}/{\leftrightarrow_{\mathbf{D}}}$ that contains φ. If $\mathbf{D}$ has equality, then, by Proposition 4.4 on p. 54, every element $[b]$ of $\mathbf{Be}/{\approx_{\mathbf{D}}}$ contains an open first-order formula. If $\mathbf{D}$ has quantifier elimination, then every element $[\varphi]$ of $\mathbf{Fm}/{\leftrightarrow_{\mathbf{D}}}$ also contains an open first-order formula. Hence, we may conclude from the above proposition, that if $\mathbf{D}$ has equality and quantifier elimination, then $\mathbf{Be}/{\approx_{\mathbf{D}}}$ and $\mathbf{Fm}/{\leftrightarrow_{\mathbf{D}}}$ are isomorphic as Boolean algebras. The isomorphism is given by the association

$$[\beta(\varphi)] \mapsto [\varphi], \tag{6.2}$$

where β is the mapping that associates with every first-order formula an equivalent Boolean expression (cf. the definition of β on p. 84).

Via the isomorphism, $\mathbf{Be}/{\approx_{\mathbf{D}}}$ inherits from $\mathbf{Fm}/{\leftrightarrow_{\mathbf{D}}}$ cylindrifications and diagonal elements such that for all $i, j < \omega$

$$\begin{aligned}&\mathbf{c}_i[b] = [\beta((\exists x_i)b)]; \text{ and}\\ &\mathbf{d}_{ij} = [\mathrm{eq}(x_i, x_j)].\end{aligned} \tag{6.3}$$

Henceforth, if $\mathbf{D}$ has equality and quantifier elimination, then we shall always assume that $\mathbf{Be}/{\approx_{\mathbf{D}}}$ has unary operations $\mathbf{c}_i$ and distinguished elements $\mathbf{d}_{ij}$ satisfying the requirements in (6.3). Furthermore, for the sake of brevity, we shall write $\mathbf{B}$ instead of $\mathbf{Be}/{\approx_{\mathbf{D}}}$. We summarise the above in the following theorem.

Theorem 6.8 If $\mathbf{D}$ has equality and quantifier elimination, then

$$\mathbf{B} = \mathbf{Be}/{\approx_{\mathbf{D}}}$$

is a cylindric algebra with cylindrifications and diagonal elements defined as in (6.3).

This concludes our discussion of the correspondence between the first-order formulas and the Boolean expressions associated with $\mathbf{D}$. For our second example we return to the formal system of pCRL expressions modulo provable equivalence in the deductive system $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$. We shall first explain how it is a basic process module over $\mathbf{B}$. Then, we shall modify it to the effect that the notion of 'substituting an arbitrary data expression d for a variable x in a pCRL expression p' (which presently, e.g., occurs in the axiom schemes (CQ3) and (DATA)) is eliminated from the formalisation. It is replaced by the conceptually simpler notion of 'substituting another variable y for x in p'. The resulting formal system also gives rise to a basic process module over $\mathbf{B}$, which turns out to be isomorphic to the basic process module of pCRL expressions modulo provable equivalence.

The set $\mathcal{P}$ of pCRL expressions associated with $\mathcal{A}$ and $\mathbf{D}$, is the universe of an algebraic structure

$$\mathbf{Pe} = \langle \mathcal{P}, +, \cdot, \delta, \mathbf{s}_i \rangle_{i<\omega},$$

with the obvious definitions for $+$, $\cdot$ and δ, and with the unary operations $\mathbf{s}_i$ $(i<\omega)$ defined by

$$\mathbf{s}_i p = \textstyle\sum_{x_i} p.$$

The deductive system $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$ induces, because of its inference rules (see Table 5.2 on p. 75), a congruence

$$\vartheta = \{\langle p, q\rangle \in \mathcal{P}\times\mathcal{P} \mid \Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx q\}$$

on the algebra $\mathbf{Pe}$. We write $[p]$ to denote the equivalence class of pCRL expressions modulo provable equivalence that contains p, i.e.,

$$[p] = \{q \in \mathcal{P} \mid \Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx q\}.$$

Owing to (BOOL) and $(\mathrm{CONG}_{(\lhd b \rhd)})$ in Table 5.2, we can expand the quotient algebra $\mathbf{Pe}/\vartheta$ with unary operations $[b]{:}\rightarrow$ $([b]\in\mathbf{B})$ defined by

$$[b]{:}\rightarrow[p] = [p \lhd b \rhd \delta].$$

Thus, we obtain an algebra similar to basic process modules, the *algebra of* pCRL *expressions modulo provable equivalence* associated with $\mathcal{A}$ and $\mathbf{D}$:

$$\mathsf{pCRL}(\mathcal{A},\mathbf{D}) = \langle \mathcal{P}/\vartheta, +, \cdot, \delta, \mathbf{s}_i, [b]{:}\rightarrow \rangle_{i<\omega,\ [b]\in\mathbf{B}}.$$

Our next task is to prove that $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ is a basic process module over $\mathbf{B}$. There are essentially two methods at our disposal. The first method consists of formalising the correspondence between the equivalence classes of pCRL expressions modulo provable equivalence and certain functions from D^ω into $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$. In the previous section we have established that any set of functions

$$\mathcal{F}^* \subseteq (\mathsf{D}^\omega \to \mathbf{T}_{\mathbf{D}}(\mathcal{A}))$$

closed under the operations of basic process modules (with as guards the subsets of D^ω) constitutes a basic process module over the cylindric algebra of subsets of D^ω. It suffices to show that $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ can be embedded into such a basic process module. This can be done with an application of the completeness theorem of the previous chapter (Theorem 5.20 on p. 85). Instead, we use the second method. It consists of deriving the validity in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ of the axioms of basic process modules directly, as propositions about the deductive system $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$.

Theorem 6.9 The algebra $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is an ω-dimensional basic process module over $\mathbf{B}$.

Proof. That $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ satisfies the axioms of basic process algebras with deadlock (Table 2.1 on p. 17) is clear by the schemes (A1)–(A7) in Table 5.2.

We verify that the unary operations $\mathbf{s}_i$ ($i < \omega$) in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ satisfy the axioms of projective summations (see Table 6.1). For (Cs1), we need to show that, for all pCRL expressions p, $\mathbf{s}_i\mathbf{s}_j[p] = \mathbf{s}_j\mathbf{s}_i[p]$ in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. Since $x, y \notin \mathsf{FV}(\sum_{x,y} p)$, we have the following deduction within $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$:

$$\begin{aligned} \textstyle\sum_{y,x} p &\preccurlyeq \textstyle\sum_{y,x}\sum_{x,y} p && \text{by (Cq3) and Lem. 5.7 on p. 78} \\ &\approx \textstyle\sum_{x,y} p && \text{by (Cq1).} \end{aligned}$$

By a symmetric argument we may also deduce

$$\textstyle\sum_{x,y} p \preccurlyeq \sum_{y,x} p.$$

Hence $\sum_{x_i,x_j} p$ and $\sum_{x_j,x_i} p$ are provably equivalent, so

$$\mathbf{s}_i\mathbf{s}_j[p] = [\textstyle\sum_{x_i,x_j} p] = [\sum_{x_j,x_i} p] = \mathbf{s}_j\mathbf{s}_i[p].$$

For (Cs2) and (Cs5) the crucial observation is that $x_i \notin \mathsf{FV}(\sum_{x_i} p)$; because of this, their validity in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is immediate by (Cq1) and (Cq5), respectively. Similarly, since $x_i \notin \mathsf{FV}(\delta)$, the validity of (Cs6) in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ follows from (Cq1). Since $p[x := x] = p$, the validity of (Cs3) is by (Cq3). The validity of (Cs4) is immediate by (Cq4).

It remains to verify that $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ also satisfies (Gc1)–(Gc11). That (Gc1) holds in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is immediate by (C1), that (Gc2) holds follows from (C2) and (C1), and (Gc3) and (Gc4) correspond with (C5) and (C3). We obtain the validity of (Gc5) in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ by Lemma 5.8 on p. 78, (Gc6) by (i) of Lemma 5.6, (Gc7) by Lemma 5.9, and (Gc8) by Lemma 5.6(ii) and Lemma 5.9. Since $x_i \notin \mathsf{FV}(\sum_{x_i} p)$, it follows from (Qe) that (Gc9) holds in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. We get from (Cq1) that $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash \sum_{x_i} \delta \approx \delta$; hence, since x_i does not occur in $\beta((\exists x_i)b)$ (cf. Definition 5.18), we obtain (Gc10) by (Cq6). For the verification of (Gc11) we need the following lemma and its corollary.

Lemma 6.10 If d is a data expression and x is a variable, then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \lhd \mathrm{eq}(x, d) \rhd \delta \approx p[x := d] \lhd \mathrm{eq}(x, d) \rhd \delta \tag{6.4}$$

for every pCRL expression p.

Proof. The proof is by structural induction on p.
Suppose that p is an action expression, say $p = \mathrm{a}(d_0, \ldots, d_{n-1})$ with $\mathrm{a} \in \mathcal{A}$ of arity n and $\vec{d} = d_0, \ldots, d_{n-1}$ a sequence of data expressions. Let $e_0, \ldots, e_{n-1}$ be such that $e_i = d_i[x := d]$ $(1 \leq i \leq n)$, and let us denote with $\mathrm{eq}(\vec{d}, \vec{e})$ the boolean expression $\mathrm{eq}(d_0, e_0) \wedge \cdots \wedge \mathrm{eq}(d_{n-1}, e_{n-1})$. Then, since $\mathbf{D} \models \mathrm{eq}(x, d) \preccurlyeq \mathrm{eq}(\vec{d}, \vec{e})$, (6.4) follows from (Eq) with an application of Lemma 5.33 on p. 92.
If $p = \delta$, then (6.4) is trivial, and if $p = p_1 + p_2$ or $p = p_1 \cdot p_2$, then (6.4) is by Lemma 5.6 on p. 78 and the induction hypothesis.
If $p = p_1 \lhd b \rhd p_2$, then (6.4) is proved by means of Lemmas 5.5 and 5.6(i) on p. 77, (C3), Lemma 5.33 on p. 92 and the induction hypothesis.
If $p = \sum_y p'$, we may assume by (CQ2) that $y \neq x$ and that y has no occurrence in d; then, (6.4) follows by (CQ1), (CQ6) and the induction hypothesis.
The proof of the lemma is now complete. □

Corollary 6.11 If the variable x does not occur in the data expression d, then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p[x := d] \approx \textstyle\sum_x p \lhd \mathrm{eq}(x, d) \rhd \delta$$

for every pCRL expression p.

Proof. Note that $x \notin \mathsf{FV}(p[x := d])$ and $\mathbf{D} \models \beta((\exists x)\mathrm{eq}(x, d)) \approx \top$. Hence

$$\begin{aligned} p[x := d] &\approx p[x := d] \lhd \beta((\exists x)\mathrm{eq}(x, d)) \rhd \delta && \text{by (C1)} \\ &\approx \textstyle\sum_x p[x := d] \lhd \mathrm{eq}(x, d) \rhd \delta && \text{by (QE)} \\ &\approx \textstyle\sum_x p \lhd \mathrm{eq}(x, d) \rhd \delta && \text{by Lem. 6.10} \end{aligned}$$

by which the corollary is proved. □

With the above lemma and its corollary we can complete the verification that $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is a basic process module: since $i \neq j$ implies $x_i \neq x_j$, we have the following deduction

$$\begin{aligned} &(\textstyle\sum_{x_i} p \lhd \mathrm{eq}(x_i, x_j) \rhd \delta) \lhd \mathrm{eq}(x_i, x_j) \rhd \delta \\ &\quad \approx p[x_i := x_j] \lhd \mathrm{eq}(x_i, x_j) \rhd \delta && \text{by Cor. 6.11} \\ &\quad \approx p \lhd \mathrm{eq}(x_i, x_j) \rhd \delta && \text{by Lem. 6.10.} \end{aligned}$$

Hence, also (Gc11) holds in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$, so the proof of the theorem is complete. □

Corollary 6.11 shows that if d is a data expression, and x is a variable that does not occur in d, then the result of substituting d for x in the pCRL expression p can be obtained from p by applications of the constructs of pCRL. We shall now use this observation to realise a conceptual simplification. We design a formal system in which data expressions play a less prominent role. We shall then prove that this formal system gives rise to a basic process module that is isomorphic to $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$, thereby demonstrating that the new formal system has the same expressive and demonstrative power as the original one.

Definition 6.12 An action expression a is *flat* if $a = \mathrm{a}(y_0, \ldots, y_{n-1})$, where $\mathrm{a} \in \mathcal{A}$ is of arity n and $y_0, \ldots, y_{n-1}$ is a sequence of variables. A pCRL expression p is *flat* if all occurrences of the action expressions in p are flat; we denote by $\mathcal{P}_{\mathsf{flat}}$ the set of flat pCRL expressions.

Example 6.13 With $\mathbf{R}$ as data algebra (see Example 3.5), the pCRL expression

$$\textstyle\sum_x \mathrm{in}(x)\mathrm{out}(x^2)$$

is not flat. However, by Corollary 6.11 it is provably equivalent to the pCRL expression

$$\textstyle\sum_x \mathrm{in}(x)(\sum_y \mathrm{out}(y) \lhd \mathrm{eq}(y, x^2) \rhd \delta)$$

and this one is flat.

We are going to define a mapping F that associates with every pCRL expression a provably equivalent flat pCRL expression. For this it is convenient to have a short notation for expressions that will be used to simulate substitution: if d is a data expression, and x is a variable that does not occur in d, let

$$p\{\!\{x := d\}\!\} = \textstyle\sum_x p \lhd \mathrm{eq}(x, d) \rhd \delta. \tag{6.5}$$

Let $\mathit{F} : \mathcal{P} \to \mathcal{P}_{\mathsf{flat}}$ be a mapping from pCRL expressions to flat pCRL expressions that satisfies the following conditions:

1. if $p = \delta$, or p is a flat action expression, then $\mathit{F}(p) = p$;
2. if p is a nonflat action expression, say $p = \mathrm{a}(d_0, \ldots, d_{n-1})$ with $\mathrm{a} \in \mathcal{A}$ of arity n and $d_0, \ldots, d_{n-1}$ a sequence of data expressions of which at least one is not a variable, then

   $$\mathit{F}(p) = \mathrm{a}(y_0, \ldots, y_{n-1})\{\!\{y_0 := d_0\}\!\} \cdots \{\!\{y_{n-1} := d_{n-1}\}\!\},$$

   where $y_0, \ldots, y_{n-1}$ is a sequence of distinct variables that do not occur in any of the d_i $(0 \leq i < n)$;
3. F distributes over the other constructs of pCRL, i.e.,

   $$\begin{aligned}
   &\mathit{F}(p + q) = \mathit{F}(p) + \mathit{F}(q);\\
   &\mathit{F}(p \cdot q) = \mathit{F}(p) \cdot \mathit{F}(q);\\
   &\mathit{F}(p \lhd b \rhd q) = \mathit{F}(p) \lhd b \rhd \mathit{F}(q); \text{and}\\
   &\mathit{F}(\textstyle\sum_x p) = \sum_x \mathit{F}(p).
   \end{aligned}$$

Lemma 6.14 If $\mathbf{D}$ has equality and quantifier elimination, then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx \mathit{F}(p)$$

for every pCRL expression p.

(Eq)′	$\mathrm{a}(y_0,\ldots,y_{n-1}) \lhd \mathrm{eq}(y_0,z_0) \land \cdots \land \mathrm{eq}(y_{n-1},z_{n-1}) \rhd \delta$ $\approx \mathrm{a}(z_0,\ldots,z_{n-1}) \lhd \mathrm{eq}(y_0,z_0) \land \cdots \land \mathrm{eq}(y_{n-1},z_{n-1}) \rhd \delta$

Table 6.5: Those instances of (Eq) that preserve flatness; $y_0,\ldots,y_{n-1}$ and $z_0,\ldots,z_{n-1}$ range over X (repetitions are allowed).

Proof. The proof is by structural induction on p, and clearly the only nontrivial case is when p is a nonflat action expression. So, suppose $p = \mathrm{a}(d_0,\ldots,d_{n-1})$ and

$$\mathit{F}(p) = \mathrm{a}(y_0,\ldots,y_{n-1})\{\!\{y_0 := d_0\}\!\}\cdots\{\!\{y_{n-1} := d_{n-1}\}\!\}.$$

By Corollary 6.11

$$\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists} \vdash \mathit{F}(p) \approx \mathrm{a}(y_0,\ldots,y_{n-1})[y_0 := d_0]\cdots[y_{n-1} := d_{n-1}],$$

and since the variables in the sequence $y_0,\ldots,y_{n-1}$ are all distinct and without occurrence in the d_i $(0 \le i < n)$, $\mathrm{a}(y_0,\ldots,y_{n-1})[y_0 := d_0]\cdots[y_{n-1} := d_{n-1}] = p$. □

Thus, there is no loss in expressivity if we confine ourselves to flat pCRL expressions. To prove the validity of an equation of flat pCRL expressions, we could, of course, use the deductive system $\Pi(\mathcal{A},\mathbf{D})^{\mathrm{eq}}_{\exists}$ and only allow deductions that exclusively involve flat pCRL expressions. The question is whether we then still have a relatively complete system. Note that this is not clear beforehand, since applications of (Cq3), (Data) and (Eq) do not always preserve flatness. On the other hand, recall that our completeness proof of the previous chapter did not involve applications of (Data) and (Cq3) (see Remark 5.40), so if we just leave them out we still have a relatively complete deductive system. Furthermore, instead of (Eq) we could include the variant shown in Table 6.5, both sides of which are flat. We denote the resulting deductive system for flat pCRL expressions by $\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$. That is, with the understanding that the meta variables range over flat pCRL expressions, $\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ consists of

1. the axioms and the inference rules in Table 5.2 on p. 75 except (Data) and (Cq3);
2. (Qe) from Table 5.3 on p. 85; and
3. (Eq)′ from Table 6.5.

We shall establish below that $\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ has a deduction for every valid equation of flat pCRL expressions. The key step consists of showing that Lemma 6.10 and Corollary 6.11 can be deduced within $\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$. We need the following lemma.

Lemma 6.15 If p is a flat pCRL expression, b and c are boolean expressions, and x is a variable that does not occur in c, then

$$\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}} \vdash (\textstyle\sum_x p \lhd b \rhd \delta) \lhd c \rhd \delta \approx \sum_x p \lhd b \land c \rhd \delta.$$

Proof. We have that

$$\begin{aligned}(\textstyle\sum_x p \lhd b \rhd \delta) \lhd c \rhd \delta &\approx (\textstyle\sum_x p \lhd b \rhd \delta) \lhd c \rhd \textstyle\sum_x \delta && \text{by (CQ1)}\\ &\approx \textstyle\sum_x (p \lhd b \rhd \delta) \lhd c \rhd \delta && \text{by (CQ6)}\\ &\approx \textstyle\sum_x p \lhd b \wedge c \rhd \delta && \text{by (C3)},\end{aligned}$$

so the lemma is proved. □

Lemma 6.16 If d is a data expression and x is a variable, then

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p) \lhd \mathrm{eq}(x, d) \rhd \delta \approx \mathit{F}(p[x := \mathsf{d}]) \lhd \mathrm{eq}(x, d) \rhd \delta. \tag{6.6}$$

Proof. We proceed by structural induction on p.
Suppose p is an action expression, say $p = \mathrm{a}(d_0, \ldots, d_{n-1})$, and let $e_0, \ldots, e_{n-1}$ be such that $p[x := d] = \mathrm{a}(e_0, \ldots, e_{n-1})$, i.e., $e_i = d_i[x := d]$ $(0 \leq i < n)$. For any sequence $\vec{y} = y_0, \ldots, y_{n-1}$ of distinct variables without occurrence in the d_i

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p) \approx \textstyle\sum_{\vec{y}} \mathrm{a}(y_0, \ldots, y_{n-1}) \lhd \mathrm{eq}(\vec{y}, \vec{d}) \rhd \delta, \tag{6.7}$$

where $\mathrm{eq}(\vec{y}, \vec{d})$ is an abbreviation for $\mathrm{eq}(y_0, d_0) \wedge \cdots \wedge \mathrm{eq}(y_{n-1}, d_{n-1})$. Indeed, if p is not flat, then $\mathit{F}(p) = \mathrm{a}(y_0, \ldots, y_{n-1})\{\!\{y_0 := d_0\}\!\} \cdots \{\!\{y_{n-1} := d_{n-1}\}\!\}$, so that (6.7) follows by Lemma 6.15. On the other hand, if p happens to be flat, then $d_0, \ldots, d_{n-1}$ is a sequence of variables; using that $\mathbf{D} \models \beta((\exists \vec{y})\mathrm{eq}(\vec{y}, \vec{d})) \approx \top$, we deduce (6.7) as follows:

$$\begin{aligned}\mathit{F}(p) &\approx \mathrm{a}(d_0, \ldots, d_{n-1}) \lhd \top \rhd \delta && \text{by (C1)}\\ &\approx \mathrm{a}(d_0, \ldots, d_{n-1}) \lhd \beta((\exists \vec{y})\mathrm{eq}(\vec{y}, \vec{d})) \rhd \delta && \text{by (Bool)}\\ &\approx \textstyle\sum_{\vec{y}} \mathrm{a}(d_0, \ldots, d_{n-1}) \lhd \mathrm{eq}(\vec{y}, \vec{d}) \rhd \delta && \text{by (QE)}\\ &\approx \textstyle\sum_{\vec{y}} \mathrm{a}(y_0, \ldots, y_{n-1}) \lhd \mathrm{eq}(\vec{y}, \vec{d}) \rhd \delta && \text{by (EQ)}'.\end{aligned}$$

By similar reasoning, we also have, for any sequence $\vec{y} = y_0, \ldots, y_{n-1}$ of distinct variables that do not occur in the e_i $(0 \leq i < n)$,

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p) \approx \textstyle\sum_{\vec{y}} \mathrm{a}(y_0, \ldots, y_{n-1}) \lhd \mathrm{eq}(\vec{y}, \vec{e}) \rhd \delta, \tag{6.8}$$

where $\mathrm{eq}(\vec{y}, \vec{e})$ is an abbreviation for $\mathrm{eq}(y_0, e_0) \wedge \cdots \wedge \mathrm{eq}(y_{n-1}, e_{n-1})$. Since

$$\mathbf{D} \models \mathrm{eq}(\vec{y}, \vec{d}) \wedge \mathrm{eq}(x, d) \approx \mathrm{eq}(\vec{y}, \vec{e}) \wedge \mathrm{eq}(x, d),$$

(6.6) is easily obtained from (6.7) and (6.8) by Lemma 6.15 and (Bool).
If $p = \delta$, then (6.6) is trivial.
Note that for flat pCRL expressions Lemma 5.5 on p. 77, Lemma 5.6 on p. 78 and Lemma 5.33 on p. 92 hold with $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ instead of $\Pi(\mathcal{A}, \mathbf{D})$. Hence, since F distributes over $+$, $\cdot$, $\lhd\, b \,\rhd$ and $\sum_y$, the proof in these cases is analogous to that of Lemma 6.10. □

The following corollary is obtained from Lemma 6.16 in the same way as we have obtained Corollary 6.11 from Lemma 6.10.

Corollary 6.17 Let p be a pCRL expression, let d be a data expression and let x be a variable that does not occur in d; then

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p[x := d]) \approx \mathit{F}(p)\{\!|x := d|\!\}.$$

We are now in a position to prove that the deduction of an equation of flat pCRL expressions may be assumed to consist of flat pCRL expressions only.

Theorem 6.18 If $\mathbf{D}$ has equality and quantifier elimination, then

$$\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists} \vdash p \approx q \text{ if, and only if, } \Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash p \approx q$$

for all flat pCRL expressions p and q.

Proof. Clearly, every deduction within $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ is also a deduction within $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$, so we only need to prove the implication from left to right. For that, it suffices to prove that, for all pCRL expressions p and q, if $p \approx q$ is an axiom of $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$, then $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p) \approx \mathit{F}(q)$. For then any deduction within $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$ proving the validity of an equation $p \approx q$ may be transformed into a deduction within $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ proving the validity of $\mathit{F}(p) \approx \mathit{F}(q)$. From this the theorem follows, since $\mathit{F}(p) = p$ if p is already flat. In most cases, $p \approx q$ being an axiom of $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$ implies that $\mathit{F}(p) \approx \mathit{F}(q)$ is an axiom of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$; the only nontrivial cases are when $p \approx q$ is an instance of (Eq), (Data) or (Cq3).
For (Eq) we need to show that

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(\mathrm{a}(d_0, \ldots, d_{n-1})) \lhd \mathrm{eq}(\vec{d}, \vec{e}) \rhd \delta \approx \mathit{F}(\mathrm{a}(e_0, \ldots, e_{n-1})) \lhd \mathrm{eq}(\vec{d}, \vec{e}) \rhd \delta, \quad (6.9)$$

using $\mathrm{eq}(\vec{d}, \vec{e})$ as an abbreviation for $\mathrm{eq}(d_0, e_0) \wedge \cdots \wedge \mathrm{eq}(d_{n-1}, e_{n-1})$. Suppose that $\vec{y} = y_0, \ldots, y_{n-1}$ is a sequence of variables that do not occur in any of the d_i and e_i $(0 \leq i < n)$. Then, by Corollary 6.17 and Lemma 6.15,

$$\begin{aligned}&\mathit{F}(\mathrm{a}(d_0, \ldots, d_{n-1})) \lhd \mathrm{eq}(\vec{d}, \vec{e}) \rhd \delta \\ &\quad \approx \textstyle\sum_{\vec{y}} \mathrm{a}(y_0, \ldots, y_{n-1}) \lhd \mathrm{eq}(\vec{y}, \vec{d}) \wedge \mathrm{eq}(\vec{d}, \vec{e}) \rhd \delta;\end{aligned}$$

by (Bool), using $\mathbf{D} \models \mathrm{eq}(\vec{y}, \vec{d}) \wedge \mathrm{eq}(\vec{d}, \vec{e}) \approx \mathrm{eq}(\vec{y}, \vec{e}) \wedge \mathrm{eq}(\vec{d}, \vec{e})$,

$$\approx \textstyle\sum_{\vec{y}} \mathrm{a}(y_0, \ldots, y_{n-1}) \lhd \mathrm{eq}(\vec{y}, \vec{e}) \wedge \mathrm{eq}(\vec{d}, \vec{e}) \rhd \delta;$$

and by Lemma 6.15 and Corollary 6.17

$$\approx \mathit{F}(\mathrm{a}(e_0, \ldots, e_{n-1})) \lhd \mathrm{eq}(\vec{d}, \vec{e}) \rhd \delta.$$

This proves (6.9).

For (DATA) we need to show that if $\mathbf{D} \models d \approx e$, then

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p[x := d]) \approx \mathit{F}(p[x := e]). \tag{6.10}$$

First, observe that we may assume without loss of generality that x does not occur in d or in e. For if x happens to occur in d or in e, then we select a variable y distinct from x and with no occurrence in d, e and p; we note that $p[x := d] = p[x := y][y := d]$ and $p[x := e] = p[x := y][y := e]$; and we continue the proof with $p[x := y]$ instead of p, and with y instead of x. Now, given that x does not occur in d or e, we may apply Corollary 6.17 to both $\mathit{F}(p[x := d])$ and $\mathit{F}(p[x := e])$, and, since $\mathbf{D} \models d \approx e$ implies $\mathbf{D} \models \mathrm{eq}(x, d) \approx \mathrm{eq}(x, e)$, we obtain (6.10) by (BOOL).
It remains to consider (CQ3); we need to show that

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \textstyle\sum_x \mathit{F}(p) \approx \sum_x \mathit{F}(p) + \mathit{F}(p[x := d]). \tag{6.11}$$

Note that for flat pCRL expressions Lemma 5.31 on p. 91 holds with $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ instead of $\Pi(\mathcal{A}, \mathbf{D})$; we use it to conclude from $\mathbf{D} \models \mathrm{eq}(x, d) \preccurlyeq \top$ that

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p) \lhd \mathrm{eq}(x, d) \rhd \delta \preccurlyeq \mathit{F}(p) \lhd \top \rhd \delta,$$

so that by (C1)

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \mathit{F}(p) \approx \mathit{F}(p) + \mathit{F}(p) \lhd \mathrm{eq}(x, d) \rhd \delta. \tag{6.12}$$

We now first derive (6.11) for the special case that x has no occurrence in d:

$$\begin{aligned}
&\textstyle\sum_x \mathit{F}(p) \\
&\quad \approx \textstyle\sum_x (\mathit{F}(p) + \mathit{F}(p) \lhd \mathrm{eq}(x, d) \rhd \delta) && \text{by (6.12)} \\
&\quad \approx \textstyle\sum_x \mathit{F}(p) + \sum_x \mathit{F}(p) \lhd \mathrm{eq}(x, d) \rhd \delta && \text{by (CQ4)} \\
&\quad \approx \textstyle\sum_x \mathit{F}(p) + \mathit{F}(p[x := d]) && \text{by Cor. 6.17.}
\end{aligned}$$

For the general case, let y be a variable such that $y \neq x$, y has no occurrence in d and $y \notin \mathsf{FV}(p)$. By the first two conditions on y we may apply the special case to conclude

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \textstyle\sum_x \mathit{F}(p) \approx \sum_x \mathit{F}(p) + \mathit{F}(p[x := y]) \text{ and} \tag{6.13}$$

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \textstyle\sum_y \mathit{F}(p[x := y]) \approx \sum_y \mathit{F}(p[x := y]) + \mathit{F}(p[x := y][y := d]). \tag{6.14}$$

The third condition on y ensures that $y \notin \mathsf{FV}(\sum_x \mathit{F}(p))$, so if we apply $\sum_y$ to both sides of (6.13), and subsequently apply (CQ1) to the left-hand side and (CQ4) and (CQ1) to the right-hand side, then we get

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash \textstyle\sum_x \mathit{F}(p) \approx \sum_x \mathit{F}(p) + \sum_y \mathit{F}(p[x := y]). \tag{6.15}$$

Since $p[x := y][y := d] = p[x := d]$, the general case now follows by combining (6.14) and (6.15). □

From the set $\mathcal{P}_{\mathsf{flat}}$ of flat pCRL expressions we obtain an algebra $\mathsf{pCRL}(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ of flat pCRL expressions modulo provable equivalence, dividing out the congruence induced on $\mathcal{P}_{\mathsf{flat}}$ by $\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$. The construction is completely analogous to the one that turned $\mathcal{P}$ into $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$; we do not spell out the details. Combining Lemma 6.14 and Theorem 6.18, if p and q are pCRL expressions, then

$$\Pi(\mathcal{A},\mathbf{D})_{\mathsf{flat}} \vdash \digamma(p) \approx \digamma(q) \text{ if, and only if, } \Pi(\mathcal{A},\mathbf{D}) \vdash p \approx q.$$

Hence, and since $\digamma$ is surjective onto the set of flat pCRL expressions, the association

$$[\digamma(p)] \mapsto [p].$$

defines a surjective embedding from $\mathsf{pCRL}(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ into $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$. Therefore, we have the following corollary.

Corollary 6.19 Suppose that $\mathbf{D}$ has equality and quantifier elimination. The algebra $\mathsf{pCRL}(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ of flat pCRL expressions modulo provable equivalence is a basic process module over $\mathbf{B}$, and it is isomorphic to the algebra $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ of pCRL expressions modulo provable equivalence.

6.3 Dimension-restricted free basic process modules

Our goal in this section is to find a complete abstract algebraic characterisation of $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$. We have proved in the previous section that $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ is a basic process module over $\mathbf{B}$; to complete our characterisation we determine the distinctive properties of $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ in comparison with other basic process modules over $\mathbf{B}$.

With the transition from basic process algebras with deadlock to ω-dimensional basic process modules, we have added a notion of 'dimension'. The *dimension set* $\dim \mathsf{p}$ associated with an element p of an ω-dimensional basic process algebra $\mathbf{P}$ is the set of all $i < \omega$ such that $\mathbf{s}_i\mathsf{p} \neq \mathsf{p}$ in $\mathbf{P}$. In the case of the basic process modules $\mathbf{F}^*$ (discussed in Section 6.1) the dimension set $\dim F$ of an element F of $\mathbf{F}^*$ consists of all $i < \omega$ such that F is not uniform along i. In the case of the basic process modules $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ and $\mathsf{pCRL}(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ of pCRL expressions modulo provable equivalence (discussed in Section 6.2) the dimension set $\dim[p]$ consists of all $i < \omega$ such that $\sum_{x_i} p$ and p are not provably equivalent. Since $\mathsf{FV}(p)$ is finite for every pCRL expression p, and $x_i \notin \mathsf{FV}(p)$ implies that $\sum_{x_i} p$ and p are provably equivalent by an application of (CQ1), we conclude that $\dim[p]$ is finite for every pCRL expression p (both in $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ and $\mathsf{pCRL}(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$). This is a distinct property of $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ and $\mathsf{pCRL}(\mathcal{A},\mathbf{D})_{\mathsf{flat}}$ that not every basic process module has.

Definition 6.20 An ω-dimensional basic process module $\mathbf{P}$ is *locally finite* if $\dim \mathsf{p}$ is finite for all elements p of $\mathbf{P}$.

Remark 6.21 Our notions "dimension set" and "local finiteness" are analogous to the corresponding notions in the theory of cylindric algebras (cf. Henkin *et al.*, 1971). Naturally, a cylindric algebra $\mathbf{C}$ is locally finite if for every element b of $\mathbf{C}$ the equality $\mathbf{c}_i\mathsf{b} = \mathsf{b}$ is true in $\mathbf{C}$ for all but finitely many $i < \omega$. The cylindric algebra of formulas $\mathbf{Fm}/{\leftrightarrow_{\mathbf{D}}}$ is a locally finite cylindric algebra, so $\mathbf{B}$ is locally finite too, by Theorem 6.8.

Our remarks preceding Definition 6.20 establish the following lemma.

Lemma 6.22 $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is a locally finite basic process module over $\mathbf{B}$.

Thus, for the sake of our comparison of $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ with other basic process modules over $\mathbf{B}$, we now zoom in at the locally finite ones. The principal way to compare algebras is through their homomorphisms. Let $\mathbf{C}$ be an arbitrary cylindric algebra, and consider a homomorphism $h : \mathbf{P} \to \mathbf{Q}$ between two locally finite basic process modules $\mathbf{P}$ and $\mathbf{Q}$ over $\mathbf{C}$. Then h is *dimension-preserving* in the sense that

$$\dim h(\mathsf{p}) \subseteq \dim \mathsf{p} \text{ for all } \mathsf{p} \text{ in } \mathbf{P}.$$

Indeed, if $\mathbf{s}_i\mathsf{p} = \mathsf{p}$ in $\mathbf{P}$, then, since h is a homomorphism, $\mathbf{s}_i h(\mathsf{p}) = h(\mathbf{s}_i\mathsf{p}) = h(\mathsf{p})$ in $\mathbf{Q}$. Hence, if P_0 is a subset of $\mathbf{P}$ and $f : \mathsf{P}_0 \to \mathsf{Q}$ is an arbitrary mapping from P_0 into the universe Q of $\mathbf{Q}$ that extends to a homomorphism from $\mathbf{P}$ into $\mathbf{Q}$, then f must be dimension-preserving. From the fact that there exist locally finite basic process modules for which there is no finite upper bound on the dimensions of their elements ($\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ and $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ are examples), we conclude: if an algebra is free for the class of locally finite basic process modules over $\mathbf{C}$, then the dimension sets associated with its free generators must be infinite (and, in fact, equal to ω). In other words, there is no nontrivial free locally finite basic process module over $\mathbf{C}$ (the trivial basic process module over $\mathbf{C}$ is the one with $\{\delta\}$ as its universe).

This is a pity, because the free algebras in a class are often very useful for the description of the algebras belonging to the class. Fortunately, in the case of locally finite basic process modules, there is a way out that requires only a minor curtailment of freedom. Suppose that $\mathbf{P}$ is a locally finite basic process module over $\mathbf{C}$, and suppose that P_0 is a set of generators for $\mathbf{P}$. Then P_0 is *dimension-restricted free* for $\mathbf{P}$ if every dimension-preserving mapping from P_0 into the universe Q of any other locally finite basic process module over $\mathbf{C}$ can be extended to a homomorphism $h : \mathbf{P} \to \mathbf{Q}$. In this case we shall also say that $\mathbf{P}$ is *dimension-restricted free on* P_0. A locally finite basic process module is called dimension-restricted free if it has a dimension-restricted free set of generators.

Proposition 6.23 Let $\mathbf{C}$ be a cylindric algebra. If $\mathbf{P}$ and $\mathbf{Q}$ are locally finite basic process modules over $\mathbf{C}$, dimension-restricted free on P_0 and Q_0, respectively, and there exists a bijection $f : \mathsf{P}_0 \to \mathsf{Q}_0$ such that both f and f^{-1} are dimension-preserving, then $\mathbf{P}$ and $\mathbf{Q}$ are isomorphic.

Proof. Let $h_1 : \mathbf{P} \to \mathbf{Q}$ and $h_2 : \mathbf{Q} \to \mathbf{P}$ be the unique homomorphic extensions of f and f^{-1}, respectively. Then $h_2 \circ h_1$ homomorphically extends the identity mapping on P_0, and therefore it is the identity mapping on the universe of $\mathbf{P}$. Similarly, $h_1 \circ h_2$ homomorphically extends the identity mapping on Q_0 and therefore is the identity mapping on the universe of $\mathbf{Q}$. Consequently, the homomorphisms h_1 and h_2 are bijections, and hence isomorphisms. □

So, if we prove that $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is a dimension-restricted free locally finite basic process module over $\mathbf{B}$, then we have characterised it up to isomorphism, and this is the purpose of the remainder of this section. The first thing we should do is to select a set of generators for $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. The definition of pCRL expressions (see (3.4) on p. 33) suggests a candidate: the set of equivalence classes in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ that contain an action expression. Needless to say, it generates $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$, but Corollary 6.11 may be used to show that it cannot be dimension-restricted free.

Example 6.24 Let $\mathrm{a} \in \mathcal{A}$ be a unary parametrised action symbol, and suppose that the universe of $\mathbf{D}$ contains at least two distinct elements, say d_0 and d_1. Then, the action expressions $\mathrm{a}(x_0)$ and $\mathrm{a}(x_1)$ are not provably equivalent (if ν is a valuation such that $\nu(x_0) = \mathsf{d}_0$ and $\nu(x_1) = \mathsf{d}_1$, then $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \not\models \mathrm{a}(x_0) \approx \mathrm{a}(x_1)$). Now, observe that, by Corollary 6.11, $\mathbf{s}_0(\mathbf{d}_{01}{:}{\to}[\mathrm{a}(x_0)]) = [\mathrm{a}(x_1)]$ in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. Hence, a mapping f that maps the equivalence classes in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ that contain an action expression into another locally finite basic process module over $\mathbf{B}$ can only be extended to a homomorphism if $f[\mathrm{a}(x_1)] = \mathbf{s}_0(\mathbf{d}_{01}{:}{\to} f[\mathrm{a}(x_0)])$. Consequently, as soon as we have fixed a value for f at $[\mathrm{a}(x_0)]$, we no longer have any freedom at all in choosing a value for f at $[\mathrm{a}(x_1)]$.

The rationale of the above example is that a dimension-restricted free set of generators for $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ perhaps contains $[\mathrm{a}(x_0)]$ or $[\mathrm{a}(x_1)]$, but certainly not both. (It seems quite natural to prefer $[\mathrm{a}(x_0)]$ over $[\mathrm{a}(x_1)]$, $[\mathrm{a}(x_0, x_1)]$ over $[\mathrm{a}(x_6, x_9)]$, etc.) We prove below that

$$\{[\mathrm{a}(x_0, \ldots, x_{n-1})] \mid \mathrm{a} \in \mathcal{A} \text{ of arity } n\} \tag{6.16}$$

is a dimension-restricted free set of generators for $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. Caution: the sequence $x_0, \ldots, x_{n-1}$ consists of the first n variables in the enumeration of X fixed at the beginning of Section 6.1; henceforth, we shall call it the *nth initial segment* of X. Note that the association

$$\mathrm{a} \mapsto [\mathrm{a}(x_0, \ldots, x_{n-1})] \tag{6.17}$$

defines a bijection between $\mathcal{A}$ and the set in (6.16). It is notationally convenient to use the parametrised action symbol as a constant symbol denoting the corresponding equivalence class, and, more generally, to use $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ (i.e., terms built from the constant symbols $\mathcal{A}$ by means of another constant symbol δ, unary function symbols $\mathbf{s}_i$ $(i < \omega)$ and $[b]{:}{\to}$ $([b] \in \mathbf{B})$, and binary function symbols $+$ and $\cdot$) as formal expressions denoting equivalence classes of $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$.

Example 6.25 If $\mathrm{a} \in \mathcal{A}$, then $\mathbf{s}_0(\mathbf{d}_{01}{:}\rightarrow \mathrm{a})$ is an example of a $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$, which (if a's arity happens to be 2) denotes the equivalence class that contains the pCRL expression

$$\textstyle\sum_{x_0} \mathrm{a}(x_0, x_1) \lhd \mathrm{eq}(x_0, x_1) \rhd \delta.$$

If in and out are unary parametrised action symbols, and associated with $\mathbf{D}$ is a Boolean expression $0 \leq x_0$, then $\mathbf{s}_0([0 \leq x_0]{:}\rightarrow \mathrm{in} \cdot \mathrm{out})$ is a $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$ that denotes the equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ containing

$$\textstyle\sum_{x_0} \mathrm{in}(x_0)\mathrm{out}(x_0) \lhd 0 \leq x_0 \rhd \delta.$$

To say that the set in (6.16) generates $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ means that every equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is denoted by a $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$. To prove it, we shall define a mapping ξ from flat pCRL expressions to $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ such that $\xi(p)$ denotes the equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ that contains p (this is enough since every equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ contains a flat pCRL expression). The definition of ξ makes extensive use of the algebraic counterpart of substitution of the variable x_j for the variable x_i, expressed as a composition of a projective summation $\mathbf{s}_i$ and a guarded command $\mathbf{d}_{ij}{:}\rightarrow$. Therefore, it is convenient to introduce an abbreviation: if $\mathbf{C}$ is a cylindric algebra of dimension ω and $\mathbf{P}$ is a basic process module over $\mathbf{C}$, then we define for all $i, j < \omega$ a unary operation $\boldsymbol{\sigma}^i_j$ on the elements p of $\mathbf{P}$ by

$$\boldsymbol{\sigma}^i_j \mathsf{p} = \begin{cases} \mathbf{s}_i(\mathbf{d}_{ij}{:}\rightarrow \mathsf{p}) & \text{if } i \neq j; \text{ and} \\ \mathsf{p} & \text{if } i = j. \end{cases}$$

The following lemma is to record that $\boldsymbol{\sigma}^i_j$ behaves as expected on $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$.

Lemma 6.26 If $i, j < \omega$, then $\boldsymbol{\sigma}^i_j[p] = [p[x_i := x_j]]$ in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$.

Proof. If $i = j$, then this is trivial. If $i \neq j$, then, by the definition of cylindric summations and guarded commands in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$, the element $\boldsymbol{\sigma}^i_j[p]$ contains the pCRL expression

$$\textstyle\sum_{x_i} p \lhd \mathrm{eq}(x_i, x_j) \rhd \delta.$$

This expression is, by Corollary 6.11, provably equivalent to $p[x_i := x_j]$; hence $\boldsymbol{\sigma}^i_j[p]$ contains $p[x_i := x_j]$; this proves the lemma. □

Definition 6.27 We define ξ as the unique mapping from flat pCRL expressions to $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ such that

(i) if $p = \mathrm{a}(x_{i_0}, \ldots, x_{i_{n-1}})$ with $\mathrm{a} \in \mathcal{A}$ of arity n, and m is the least element of ω such that $m > n-1, i_0, \ldots, i_{n-1}$, then

$$\xi(p) = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a};$$

(ii) if p is a conditional, say $p = q \lhd b \rhd r$, then $\xi(p) = [b]{:}\rightarrow \xi(q) + \neg[b]{:}\rightarrow \xi(r)$.

(iii) ξ respects the other constructs of pCRL, i.e.,

$$\xi(\delta) = \delta, \qquad \xi(p \cdot q) = \xi(p) \cdot \xi(q), \text{ and}$$
$$\xi(p + q) = \xi(p) + \xi(q), \qquad \xi(\textstyle\sum_{x_i} p) = \mathbf{s}_i \xi(p).$$

Lemma 6.28 For every flat pCRL expression p, $\xi(p)$ denotes $[p]$ in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$.

Proof. The proof is by structural induction on p.
Suppose $p = \mathrm{a}(x_{i_0}, \ldots, x_{i_{n-1}})$ with $\mathrm{a} \in \mathcal{A}$ of arity n, and let $m > n-1, i_0, \ldots, i_{n-1}$, so that

$$\xi(p) = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a}.$$

By Lemma 6.26, the equivalence class denoted by $\boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a}$ contains the action expression $\mathrm{a}(x_0, \ldots, x_{n-1})[x_0 := x_m] \cdots [x_{n-1} := x_{m+n-1}]$, which is (syntactically) equal to

$$\mathrm{a}(x_m, \ldots, x_{m+n-1})$$

since the variables in the sequences $x_0, \ldots, x_{n-1}$ and $x_m, \ldots, x_{m+n-1}$ are all mutually distinct. Then, by Lemma 6.26, we find that $\xi(p)$ is the equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ that contains $\mathrm{a}(x_m, \ldots, x_{m+n-1})[x_{m+n-1} := x_{i_{n-1}}] \cdots [x_m := x_{i_0}]$. Since the variables in the sequence $x_m, \ldots, x_{m+n-1}$ are mutually distinct and also distinct from the variables in the sequence $x_{i_0}, \ldots, x_{i_{m-1}}$, this action expression is (syntactically) equal to

$$\mathrm{a}(x_{i_0}, \ldots, x_{i_{n-1}}) = p.$$

Suppose that $p = q \lhd b \rhd r$. By the induction hypothesis and the definition of guarded commands in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$, $\xi(p)$ contains the pCRL expression

$$q \lhd b \rhd \delta + r \lhd \neg b \rhd \delta,$$

which is by Lemma 5.5 on p. 77 provably equivalent to p.
For the other cases, the proof is straightforward. □

Since by Theorem 6.18 every element of $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ contains a flat pCRL expression, by the lemma just proved, every equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is denoted by an element from the set of $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$. Since the latter set is generated by $\mathcal{A}$, it follows that the set of equivalence classes denoted by the elements of $\mathcal{A}$, i.e., the set in (6.16), generates $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. Consequently, it now remains to establish that this set is dimension-restricted free, and for this, the set of $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ may also be of help. Namely, there is a straightforward way to transform it into a dimension-restricted free algebra. We define a deductive system $\mathbf{B}$-$\mathsf{BPM}_\omega(\mathcal{A})$ for equations of the form $t \approx u$, where t and u are $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$. As inference rules it has the rules of equational logic associated with basic process modules over $\mathbf{B}$. As axioms it has

(Cs7)	$\mathbf{s}_i \mathrm{a} \approx \mathrm{a}$ if $\mathrm{a} \in \mathcal{A}$ of arity $\leq i$.

Table 6.6: The dimension-restriction axioms for parametrised action symbols.

1. the equations generated by the axioms of basic process modules over $\mathbf{B}$ (cf. Tables 2.1 on p. 17, 6.1 on p. 108, and 6.4 on p. 111), replacing each occurrence of '=' by '$\approx$' and letting p, q and r range over $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$, and
2. the dimension-restriction axioms for the elements of $\mathcal{A}$ as generated by the schema (Cs7) in Table 6.6.

If $t \approx u$ has a deduction within this deductive system, then we call t and u *provably equivalent* and we write $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash t \approx u$. The set of $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ is naturally the universe of an algebra similar to basic process modules over $\mathbf{B}$ and provable equivalence is a congruence on this algebra. We shall denote the quotient, i.e., the algebra of $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ modulo provable equivalence, by $\mathbf{I}(\mathcal{A}, \mathbf{B})$. If t is a $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$, then we write $[t]$ for the equivalence class in $\mathbf{I}(\mathcal{A}, \mathbf{B})$ that contains it.

Proposition 6.29 The algebra $\mathbf{I}(\mathcal{A}, \mathbf{B})$ is a locally finite basic process module over $\mathbf{B}$, and the set $\{[\mathrm{a}] \mid \mathrm{a} \in \mathcal{A}\}$ is dimension-restricted free for $\mathbf{I}(\mathcal{A}, \mathbf{B})$.

Proof. Clearly, $\mathbf{I}(\mathcal{A}, \mathbf{B})$ is a basic process module over $\mathbf{B}$.

If t is a $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$, then we write $\dim t$ for the set of all $i < \omega$ such that $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \not\vdash \mathbf{s}_i t \approx t$. To prove that $\mathbf{I}(\mathcal{A}, \mathbf{B})$ is locally finite, it is enough to show that $\dim t$ is finite; we proceed by structural induction on t: If $t = \delta$, then $\dim t = \emptyset$ by (Cs6), and if t is a parametrised action symbol of arity n, then $\dim t \subseteq \{0, \ldots, n-1\}$ by (Cs7). If $t = u + v$ or $t = u \cdot v$, then $\dim t \subseteq (\dim u \cup \dim v)$ by (Cs4) and (Cs5), respectively; since $\dim u$ and $\dim v$ are finite by the induction hypothesis, $\dim t$ is finite too. If $t = [b]{:}{\rightarrow} t'$, then $\dim t \subseteq (\dim t' \cup \{i < \omega \mid \mathbf{c}_i[b] \neq [b] \text{ in } \mathbf{B}\})$ by (Gc9) or (Gc10). The set $\dim t'$ is finite by the induction hypothesis, and the set $\{i < \omega \mid \mathbf{c}_i[b] \neq [b] \text{ in } \mathbf{B}\}$ is finite since $\mathbf{B}$ is locally finite (cf. Remark 6.21). Hence $\dim t$ is finite. If $t = \mathbf{s}_i t'$, then $\dim t \subseteq (\dim t' - \{i\})$ according to (Cs2); since $\dim t'$ is finite by the induction hypothesis, also $\dim t$ is finite.

That the set $\{[\mathrm{a}] \mid \mathrm{a} \in \mathcal{A}\}$ generates $\mathbf{I}(\mathcal{A}, \mathbf{B})$ is clear; it remains to prove that it is dimension-restricted free for $\mathbf{I}(\mathcal{A}, \mathbf{B})$. So, suppose that $\mathbf{P}$ is an arbitrary locally finite basic process module over $\mathbf{B}$ and consider a dimension-preserving mapping

$$f : \{[\mathrm{a}] \mid \mathrm{a} \in \mathcal{A}\} \to \mathbf{P}.$$

We define a mapping g from the set of all $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ into the universe of $\mathbf{P}$ as the homomorphic extension of the association that sends $\mathrm{a} \in \mathcal{A}$ to $f[\mathrm{a}]$:

$$g(\mathrm{a}) = f[\mathrm{a}] \qquad g(t \cdot u) = g(t) \cdot g(u) \qquad g([b]{:}{\rightarrow} t) = [b]{:}{\rightarrow} g(t)$$
$$g(\delta) = \delta \qquad g(t + u) = g(t) + g(u) \qquad g(\mathbf{s}_i t) = \mathbf{s}_i g(t).$$

Since f is dimension-preserving, g maps each $\mathrm{a} \in \mathcal{A}$ of arity n to an element p of $\mathbf{P}$ such that $\dim \mathsf{p} \subseteq \{0, \ldots, n-1\}$; hence, $\mathbf{s}_i g(\mathrm{a}) = g(\mathrm{a})$ for all $i \geq n$. From this, and since $\mathbf{P}$ is a basic process module over $\mathbf{B}$, we get that if $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash t \approx u$, then $g(t) = g(u)$ in $\mathbf{P}$. That is, g maps all the elements of an equivalence class in $\mathbf{I}(\mathcal{A}, \mathbf{B})$ to the same element in $\mathbf{P}$, and therefore there exists a function h from $\mathbf{I}(\mathcal{A}, \mathbf{B})$ into $\mathbf{P}$ that sends $[t]$ to $g(t)$. It is immediate from the definition of g that h is a homomorphism and that it extends f. □

Remark 6.30 The theory of basic process algebras with deadlock is usually presented as an algebraic specification parametrised by a set of constant symbols (action symbols). Similarly, one might view the theory of basic process modules over $\mathbf{B}$ as an algebraic specification parametrised by a set of constant symbols (parametrised action symbols) and a set of axioms that specify the dimension of each of these constant symbols (the axioms generated by the schema (Cs7) in Table 6.6). The algebra $\mathbf{I}(\mathcal{A}, \mathbf{B})$ is the *initial algebra* associated with this algebraic specification.

The usefulness of Proposition 6.29 is in that we do not have to exhibit a homomorphism from $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ into every other locally finite basic process module over $\mathbf{B}$ to show that it is dimension-restricted free. It is now sufficient to exhibit an isomorphism between $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ and $\mathbf{I}(\mathcal{A}, \mathbf{B})$. This, in turn, can be achieved by proving the following completeness theorem: for all flat pCRL expressions p and q

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash p \approx q \text{ if, and only if, } \mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q).$$

We shall prove the completeness theorem below as Theorem 6.37. Its proof consists mainly in showing how the axioms of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$, when reformulated as equations of $\mathbf{B}\text{-}\mathsf{BPM}_\omega$-terms over $\mathcal{A}$, can be deduced by means of the axioms of basic process modules and the dimension-restriction axioms. Before we come to that, however, we shall make a few preparations. In particular, we shall demonstrate that the syntactic notions '$x_i \notin \mathsf{FV}(p)$' and '$p[x_i := x_j]$', which play a prominent rôle in the axioms of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$, are adequately represented by their algebraic counterparts in the theory of basic process modules (see Lemma 6.33 and Lemma 6.35, respectively).

First, we establish a few facts about the unary substitution operations $\boldsymbol{\sigma}^i_j$ on basic process modules. For the formulation thereof it is convenient to also define them on cylindric algebras: let $\mathbf{C}$ be a cylindric algebra of dimension ω; we define for all $i, j < \omega$ a unary operation $\boldsymbol{\sigma}^i_j$ on elements b of $\mathbf{C}$ by

$$\boldsymbol{\sigma}^i_j \mathsf{b} = \begin{cases} \mathbf{c}_i(\mathbf{d}_{ij} \wedge \mathsf{b}) & \text{if } i \neq j; \text{ and} \\ \mathsf{b} & \text{if } i = j. \end{cases}$$

In the proof of following lemma we shall make use of certain derived identities of cylindric algebras; they are all established in Henkin *et al.* (1971).

Lemma 6.31 Let $\mathbf{C}$ be a cylindric algebra of dimension ω, let $i, j, k, l < \omega$, and let $\mathsf{b} \in \mathbf{C}$; then, in every basic process module over $\mathbf{C}$,

(i) $\boldsymbol{\sigma}^i_j\delta = \delta$, $\boldsymbol{\sigma}^i_j$ distributes over $+$ and $\cdot$, and $\boldsymbol{\sigma}^i_j(\mathsf{b}{:\to}\,\mathsf{p}) = (\boldsymbol{\sigma}^i_j\mathsf{b}){:\to}\,\boldsymbol{\sigma}^i_j\mathsf{p}$;

(ii) $\mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_i\mathsf{p} = \mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_j\mathsf{p}$;

(iii) if $k \neq i, j$, then $\mathbf{s}_k\boldsymbol{\sigma}^i_j\mathsf{p} = \boldsymbol{\sigma}^i_j\mathbf{s}_k\mathsf{p}$;

(iv) $\boldsymbol{\sigma}^i_j\mathbf{s}_i\mathsf{p} = \mathbf{s}_i\mathsf{p}$;

(v) $\boldsymbol{\sigma}^k_i\boldsymbol{\sigma}^j_k\mathbf{s}_k\mathsf{p} = \boldsymbol{\sigma}^j_i\mathbf{s}_k\mathsf{p}$;

(vi) $\boldsymbol{\sigma}^k_i\boldsymbol{\sigma}^j_k\mathbf{s}_k\mathbf{s}_l\mathsf{p} = \boldsymbol{\sigma}^l_i\boldsymbol{\sigma}^j_l\mathbf{s}_l\mathbf{s}_k\mathsf{p}$; and

(vii) if $k \neq i, j$, then $\boldsymbol{\sigma}^i_j\boldsymbol{\sigma}^k_l\mathsf{p} = \begin{cases} \boldsymbol{\sigma}^k_j\boldsymbol{\sigma}^i_j\mathsf{p} & \text{if } i = l\text{; and} \\ \boldsymbol{\sigma}^k_l\boldsymbol{\sigma}^i_j\mathsf{p} & \text{otherwise.} \end{cases}$

Proof.

(i) This is trivial if $i = j$, so we assume $i \neq j$. We then obtain $\boldsymbol{\sigma}^i_j\delta = \delta$ by (GC5) and (CS6), and $\boldsymbol{\sigma}^i_j(\mathsf{p}+\mathsf{q}) = \boldsymbol{\sigma}^i_j\mathsf{p} + \boldsymbol{\sigma}^i_j\mathsf{q}$ by (GC6) and (CS4). Furthermore, we have

$$\begin{aligned} \boldsymbol{\sigma}^i_j(\mathsf{p}\cdot\mathsf{q}) &= \mathbf{s}_i((\mathbf{d}_{ij}{:\to}\,\mathsf{p})\cdot(\mathbf{d}_{ij}{:\to}\,\mathsf{q})) && \text{by (GC7), (GC8)} \\ &= \mathbf{s}_i((\mathbf{d}_{ij}{:\to}\,\mathsf{p})\cdot(\mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^i_j\mathsf{q})) && \text{by (GC11)} \\ &= \boldsymbol{\sigma}^i_j\mathsf{p}\cdot\boldsymbol{\sigma}^i_j\mathsf{q} && \text{by (GC8), (CS5);} \end{aligned}$$

and $\boldsymbol{\sigma}^i_j(\mathsf{b}{:\to}\,\mathsf{p}) = (\boldsymbol{\sigma}^i_j\mathsf{b}){:\to}\,\boldsymbol{\sigma}^i_j\mathsf{p}$ can be deduced with (GC4), (GC9), (GC11) and commutativity, associativity and idempotency of $\wedge$.

(ii) If $i = j$, then, trivially, $\mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_i\mathsf{p} = \mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_j\mathsf{p}$.
If $i \neq j$, but $i = k$, then $\mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_i\mathsf{p} = \mathbf{d}_{ij}{:\to}\,\mathsf{p} = \mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_j\mathsf{p}$ by (GC11), and the case that $i \neq j$, but $j = k$, is similar.
So, suppose that i, j and k are distinct; then

$$\begin{aligned} &(*)\quad \mathbf{c}_k\mathbf{d}_{ij} = \mathbf{d}_{ij} \text{ and} \\ &(**)\ \mathbf{d}_{ij} \wedge \mathbf{d}_{ki} = \mathbf{d}_{ij} \wedge \mathbf{d}_{kj} \end{aligned} \qquad \text{(Henkin } \textit{et al.}\text{, 1971),}$$

so

$$\begin{aligned} \mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_i\mathsf{p} &= \mathbf{d}_{ij}{:\to}\,\mathbf{s}_k(\mathbf{d}_{ki}{:\to}\,\mathsf{p}) \\ &= \mathbf{s}_k(\mathbf{d}_{ij}{:\to}(\mathbf{d}_{ki}{:\to}\,\mathsf{p})) && \text{by (GC10) and (*)} \\ &= \mathbf{s}_k(\mathbf{d}_{ij}{:\to}(\mathbf{d}_{kj}{:\to}\,\mathsf{p})) && \text{by (GC4) and (**)} \\ &= \mathbf{d}_{ij}{:\to}\,\mathbf{s}_k(\mathbf{d}_{kj}{:\to}\,\mathsf{p}) && \text{by (GC10) and (*)} \\ &= \mathbf{d}_{ij}{:\to}\,\boldsymbol{\sigma}^k_j\mathsf{p}. \end{aligned}$$

(iii) If $k \neq i, j$, then $\mathbf{s}_k\boldsymbol{\sigma}^i_j\mathsf{p} = \boldsymbol{\sigma}^i_j\mathbf{s}_k\mathsf{p}$ by (CS1), (GC10) and (*).

(iv) If $i = j$, then $\boldsymbol{\sigma}^i_j\mathbf{s}_i\mathsf{p} = \mathbf{s}_i\mathsf{p}$ is by definition. If $i \neq j$, then, since $\mathbf{c}_i\mathbf{d}_{ij} = \top$ (cf. Henkin *et al.*, 1971), $\boldsymbol{\sigma}^i_j\mathbf{s}_i\mathsf{p} = \mathbf{s}_i\mathsf{p}$ is by (GC9) and (GC1).

(v) If $k = i$ or $k = j$, then $\boldsymbol{\sigma}_i^k \boldsymbol{\sigma}_k^j \mathbf{s}_k \mathsf{p} = \boldsymbol{\sigma}_i^j \mathbf{s}_k \mathsf{p}$ by definition. If $k \neq i, j$, then we obtain $\boldsymbol{\sigma}_i^k \boldsymbol{\sigma}_k^j \mathbf{s}_k \mathsf{p} = \boldsymbol{\sigma}_i^j \mathbf{s}_k \mathsf{p}$ from (ii)–(iv):

$$\begin{aligned}
\boldsymbol{\sigma}_i^k \boldsymbol{\sigma}_k^j \mathbf{s}_k \mathsf{p} &= \boldsymbol{\sigma}_i^k \boldsymbol{\sigma}_i^j \mathbf{s}_k \mathsf{p} && \text{by (ii)} \\
&= \boldsymbol{\sigma}_i^k \mathbf{s}_k \boldsymbol{\sigma}_i^j \mathsf{p} && \text{by (iii)} \\
&= \mathbf{s}_k \boldsymbol{\sigma}_i^j \mathsf{p} && \text{by (iv)} \\
&= \boldsymbol{\sigma}_i^j \mathbf{s}_k \mathsf{p} && \text{by (iii).}
\end{aligned}$$

(vi) That $\boldsymbol{\sigma}_i^k \boldsymbol{\sigma}_k^j \mathbf{s}_k \mathbf{s}_l \mathsf{p} = \boldsymbol{\sigma}_i^l \boldsymbol{\sigma}_l^j \mathbf{s}_l \mathbf{s}_k \mathsf{p}$ follows from (v) and (Cs1).

(vii) If $k \neq i, j$, then, by (iii) and (i), $\boldsymbol{\sigma}_j^i \boldsymbol{\sigma}_l^k \mathsf{p} = \mathbf{s}_k((\boldsymbol{\sigma}_j^i \mathbf{d}_{kl}) {:}\rightarrow \boldsymbol{\sigma}_j^i \mathsf{p})$.

If $i = l$, then, since $\boldsymbol{\sigma}_j^i \mathbf{d}_{ki} = \mathbf{d}_{kj}$ (cf. Henkin *et al.*, 1971), the right-hand side is equivalent to $\boldsymbol{\sigma}_j^k \boldsymbol{\sigma}_j^i \mathsf{p}$.

On the other hand, if $i \neq l$, then, since also $i \neq k$, $\boldsymbol{\sigma}_j^i \mathbf{d}_{kl} = \mathbf{d}_{kl}$ (cf. Henkin *et al.*, 1971), so the right-hand side is equivalent to $\boldsymbol{\sigma}_l^k \boldsymbol{\sigma}_j^i \mathsf{p}$. □

Note that the proof of Lemma 6.31 does not involve any applications of (Cs3). As an amusing and instructive aside, we can use Lemma 6.31(v) to prove that (Cs3) is superfluous in the definition of *locally finite* ω-dimensional basic process modules. We establish this in the following corollary. (Clearly, our definitions of 'dimension set' and 'local finiteness' make sense for every algebraic structure with a sequence $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_k, \ldots$ $(k < \omega)$ of unary operations.)

Corollary 6.32 Let $\mathbf{C}$ be a cylindric algebra, and let $\mathbf{P}$ be an algebraic structure similar to basic process modules over $\mathbf{C}$. The following are equivalent:

(i) $\mathbf{P}$ is locally finite, and satisfies the equalities in Table 2.1 on p. 17, the equalities (Cs1), (Cs2) and (Cs4)–(Cs6) from Table 6.1 on p. 108, and the equalities in Table 6.4 on p. 111.

(ii) $\mathbf{P}$ is a locally finite basic process module over $\mathbf{C}$.

Proof. That (ii) implies (i) is immediate; we prove that (i) implies (ii).

Let p be an element of $\mathbf{P}$; we prove that $\mathbf{s}_i \mathsf{p} = \mathbf{s}_i \mathsf{p} + \mathsf{p}$ for all $i < \omega$. Using that $\mathbf{P}$ is locally finite, let $j < \omega$ be such that $j \neq i$ and $j \notin \dim \mathsf{p}$. We conclude that

$$\mathbf{s}_i \mathsf{p} = \mathbf{s}_i \mathsf{p} + \mathbf{s}_j \boldsymbol{\sigma}_j^i \mathsf{p} \tag{6.18}$$

from the following deduction:

$$\begin{aligned}
\mathbf{s}_i \mathsf{p} &= \mathbf{s}_i \mathbf{s}_j \mathsf{p} && j \notin \dim \mathsf{p} \\
&= \mathbf{s}_i \mathbf{s}_j((\mathbf{d}_{ij} \vee \neg\, \mathbf{d}_{ij}) {:}\rightarrow \mathsf{p}) && \text{by (Gc1)} \\
&= \mathbf{s}_i \mathbf{s}_j(\mathbf{d}_{ij} {:}\rightarrow \mathsf{p} + \neg\, \mathbf{d}_{ij} {:}\rightarrow \mathsf{p}) && \text{by (Gc3)} \\
&= \mathbf{s}_i \mathbf{s}_j((\mathbf{d}_{ij} {:}\rightarrow \mathsf{p} + \neg\, \mathbf{d}_{ij} {:}\rightarrow \mathsf{p}) + \mathbf{d}_{ij} {:}\rightarrow \mathsf{p}) && \text{by (A1)–(A3)} \\
&= \mathbf{s}_i \mathbf{s}_j(\mathsf{p} + \mathbf{d}_{ij} {:}\rightarrow \mathsf{p}) && \text{by (Gc3), (Gc1)} \\
&= \mathbf{s}_i \mathsf{p} + \mathbf{s}_i \mathbf{s}_j\, \mathbf{d}_{ij} {:}\rightarrow \mathsf{p} && \text{by (Cs4), } j \notin \dim \mathsf{p} \\
&= \mathbf{s}_i \mathsf{p} + \mathbf{s}_j \boldsymbol{\sigma}_j^i \mathsf{p} && \text{by (Cs1).}
\end{aligned}$$

Then, observe that by omitting all occurrences of $\mathbf{s}_j$ in the above deduction, we get a deduction that proves $\mathbf{s}_i\mathsf{p} = \mathbf{s}_i\mathsf{p} + \boldsymbol{\sigma}_i^j\mathsf{p}$, whence

$$\mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p} = \mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p} + \boldsymbol{\sigma}_i^j\boldsymbol{\sigma}_j^i\mathsf{p}.$$

Since $j \notin \dim \mathsf{p}$ we get by Lemma 6.31(v) that $\boldsymbol{\sigma}_i^j\boldsymbol{\sigma}_j^i\mathsf{p} = \boldsymbol{\sigma}_i^j\boldsymbol{\sigma}_j^i\mathbf{s}_j\mathsf{p} = \mathbf{s}_j\mathsf{p} = \mathsf{p}$, so

$$\mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p} = \mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p} + \mathsf{p}. \tag{6.19}$$

Hence,

$$\begin{aligned}
\mathbf{s}_i\mathsf{p} &= \mathbf{s}_i\mathsf{p} + \mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p} && \text{by (6.18)}\\
&= \mathbf{s}_i\mathsf{p} + (\mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p} + \mathsf{p}) && \text{by (6.19)}\\
&= (\mathbf{s}_i\mathsf{p} + \mathbf{s}_j\boldsymbol{\sigma}_j^i\mathsf{p}) + \mathsf{p} && \text{by (A2)}\\
&= \mathbf{s}_i\mathsf{p} + \mathsf{p} && \text{by (6.18).}
\end{aligned}$$

Consequently, $\mathbf{P}$ is a locally finite ω-dimensional basic process module over $\mathbf{C}$. □

When we demonstrated the local finiteness of $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$, we established that $x_i \notin \mathsf{FV}(p)$ implies that $\sum_{x_i} p$ and p are provably equivalent, whence $i \notin \dim[p]$ in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$. The proof involved an application of (CQ1) (cf. our remarks preceding Definition 6.20). We now deduce this same property again, but this time we only use axioms of basic process modules and dimension-restriction axioms. This shows that the algebraic notion of 'dimension' adequately represents the syntactic notion of 'free variables in an expression'.

Lemma 6.33 If p is a flat pCRL expression such that $x_i \notin \mathsf{FV}(p)$, then

$$\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \mathbf{s}_i\xi(p) \approx \xi(p). \tag{6.20}$$

Proof. We assume $x_i \notin \mathsf{FV}(p)$ and prove (6.20) by structural induction on p. Suppose that $p = \mathrm{a}(x_{i_0}, \ldots, x_{i_{n-1}})$ with $\mathrm{a} \in \mathcal{A}$ of arity n, let m be the least element of ω such that $m > n-1, i_0, \ldots, i_{n-1}$, so that

$$\xi(p) = \boldsymbol{\sigma}_{i_0}^m \cdots \boldsymbol{\sigma}_{i_{n-1}}^{m+n-1}\boldsymbol{\sigma}_{m+n-1}^{n-1} \cdots \boldsymbol{\sigma}_m^0\mathrm{a}.$$

Note that $i \notin \{i_0, \ldots, i_{n-1}\}$ by the assumption that $x_i \notin \mathsf{FV}(p)$. We now distinguish three cases:
If $i < n$, then we may interchange $\mathbf{s}_i$ and $\boldsymbol{\sigma}_{i_0}^m \cdots \boldsymbol{\sigma}_{i_{n-1}}^{m+n-1}\boldsymbol{\sigma}_{m+n-1}^{n-1} \cdots \boldsymbol{\sigma}_{m+i+1}^{i+1}$ in the left-hand side of (6.20) by Lemma 6.31(iii), and subsequently delete $\mathbf{s}_i$ with an application of (Cs2); this gives the right-hand side of (6.20).
Likewise, if $m \leq i < m+n$, say $i = m+j$ with $0 \leq j < n$, then starting from the left-hand side of (6.20) we interchange $\mathbf{s}_i$ and $\boldsymbol{\sigma}_{i_0}^m \cdots \boldsymbol{\sigma}_{i_{j-1}}^{i-1}$, so that we may subsequently delete $\mathbf{s}_i$ to obtain the right-hand side of (6.20).
In the case that remains, $n < i < m$ or $i \geq m+n$, so by Lemma 6.31(iii) and (Cs7),

$$\mathbf{s}_i\xi(p) \approx \boldsymbol{\sigma}_{i_0}^m \cdots \boldsymbol{\sigma}_{i_{m+n-1}}^{m+n-1}\boldsymbol{\sigma}_{m+n-1}^{n-1} \cdots \boldsymbol{\sigma}_m^0\mathbf{s}_i\mathrm{a} \approx \xi(p).$$

Suppose that p is a conditional, say $p = q \lhd b \rhd r$. Clearly, $x_i \notin \mathsf{FV}(p)$ implies that x_i does not occur in b, and therefore $\mathbf{c}_i[b] = [b]$ and $\mathbf{c}_i[\neg\, b] = [\neg\, b]$ in $\mathbf{B}$. We now prove (6.20) with the following deduction:

$$\begin{aligned}
\mathbf{s}_i\xi(p) &= \mathbf{s}_i(\mathbf{c}_i[b]{:}\rightarrow \xi(q) + \mathbf{c}_i[\neg\, b]{:}\rightarrow \xi(r)) \\
&\approx \mathbf{s}_i(\mathbf{c}_i[b]{:}\rightarrow \xi(q)) + \mathbf{s}_i(\mathbf{c}_i[\neg\, b]{:}\rightarrow \xi(r)) && \text{by (Cs4)} \\
&\approx \mathbf{c}_i[b]{:}\rightarrow \mathbf{s}_i\xi(q) + \mathbf{c}_i[\neg\, b]{:}\rightarrow \mathbf{s}_i\xi(r) && \text{by (Gc10)} \\
&\approx \mathbf{c}_i[b]{:}\rightarrow \xi(q) + \mathbf{c}_i[\neg\, b]{:}\rightarrow \xi(r) && \text{by (IH)} \\
&= \xi(p).
\end{aligned}$$

If $p = \delta$, then $\mathbf{s}_i\xi(p) \approx \xi(p)$ by (Cs6).
If $p = q + r$, then $\mathbf{s}_i\xi(p) \approx \mathbf{s}_i\xi(q) + \mathbf{s}_i\xi(r)$ by (Cs4), so $\mathbf{s}_i\xi(p) \approx \xi(p)$ follows by the induction hypothesis.
If $p = q \cdot r$, then $\xi(r) \approx \mathbf{s}_i\xi(r)$ by the induction hypothesis,

$$\mathbf{s}_i\xi(p) \approx \mathbf{s}_i(\xi(q) \cdot \mathbf{s}_i\xi(r)) \approx \mathbf{s}_i\xi(q) \cdot \mathbf{s}_i\xi(r)$$

by (Cs5), and $\mathbf{s}_i\xi(p) \approx \xi(p)$ by another two applications of the induction hypothesis.
If $p = \sum_{x_j} q$, then there are two cases: if $x_i = x_j$, then $\mathbf{s}_i\xi(p) \approx \xi(p)$ by (Cs2); otherwise $x_i \notin \mathsf{FV}(q)$, whence $\mathbf{s}_i\xi(q) \approx \xi(q)$ by the induction hypothesis, and hence $\mathbf{s}_i\xi(p) \approx \mathbf{s}_j\mathbf{s}_i\xi(q) \approx \xi(p)$ by (Cs1). □

Intuitively, the sequence of natural numbers $m, \ldots, m+n-1$ in the definition of ξ (Definition 6.27) refers to a sequence of 'fresh' variables $x_m, \ldots, x_{m+n-1}$ ('fresh' here means 'without an occurrence in the sequence $x_0, \ldots, x_{n-1}$ or in the sequence $x_{i_0}, \ldots, x_{i_{n-1}}$'). Freshness is ensured by choosing m larger than $n-1$ and $i_0, \ldots, i_{n-1}$; that we took for m the *least* such number, was a quite arbitrary choice. The following lemma shows that any $m > n-1, i_0, \ldots, i_{n-1}$ does the job.

Lemma 6.34 If $a = \mathrm{a}(x_{i_0}, \ldots, x_{i_{n-1}})$ is a flat action expression, then

$$\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(a) \approx \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a}$$

for all $m > n-1, i_0, \ldots, i_{n-1}$.

Proof. By definition

$$\xi(a) = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{i_{n-1}}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a},$$

where m is the *least* element of ω that is greater than $n-1, i_0, \ldots, i_{n-1}$. We need to show that, in fact, m may be *any* element of ω greater than $n-1, i_0, \ldots, i_{n-1}$, and this is easily obtained as a consequence of the following claim.

Claim Let $\mathbf{P}$ be a basic process module and suppose that p is an element of $\mathbf{P}$; if $m, \ldots, m+n \notin \dim \mathsf{p}$ and $m > i_0, \ldots, i_{n-1}$, then

$$\boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathsf{p} = \boldsymbol{\sigma}^{m+1}_{i_0} \cdots \boldsymbol{\sigma}^{m+n}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n} \cdots \boldsymbol{\sigma}^0_{m+1} \mathsf{p}.$$

If $n = 0$, then there is nothing to prove.
Assume, inductively, that $n > 0$ and $m, \ldots, m+n \notin \dim \mathsf{p}$. Then

$$\begin{aligned}
&\boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathsf{p} \\
&\quad = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \\
&\qquad\quad \boldsymbol{\sigma}^{n-2}_{m+n-2} \cdots \boldsymbol{\sigma}^0_m \mathbf{s}_{m+n-1} \mathbf{s}_{m+n} \mathsf{p} && m, \ldots, m+n \notin \dim \mathsf{p} \\
&\quad = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n-1} \mathbf{s}_{m+n-1} \mathbf{s}_{m+n} \\
&\qquad\quad \boldsymbol{\sigma}^{n-2}_{m+n-2} \cdots \boldsymbol{\sigma}^0_m \mathsf{p} && \text{by Lem. 6.31(iii)} \\
&\quad = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-2}_{i_{n-2}} \boldsymbol{\sigma}^{m+n}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n} \mathbf{s}_{m+n} \mathbf{s}_{m+n-1} \\
&\qquad\quad \boldsymbol{\sigma}^{n-2}_{m+n-2} \cdots \boldsymbol{\sigma}^0_m \mathsf{p} && \text{by Lem. 6.31(vi)} \\
&\quad = \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-2}_{i_{n-2}} \boldsymbol{\sigma}^{m+n}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n} \boldsymbol{\sigma}^{n-2}_{m+n-2} \cdots \boldsymbol{\sigma}^0_m \mathsf{p} && \text{by Lem. 6.31(iii)} \\
&\quad = \boldsymbol{\sigma}^{m+n}_{i_{n-1}} \boldsymbol{\sigma}^m_{i_0} \cdots \boldsymbol{\sigma}^{m+n-2}_{i_{n-2}} \boldsymbol{\sigma}^{n-2}_{m+n-2} \cdots \boldsymbol{\sigma}^0_m \boldsymbol{\sigma}^{n-1}_{m+n} \mathsf{p} && \text{by Lem. 6.31(vii)},
\end{aligned}$$

whence, since $m, \ldots, m+n-1 \notin \dim(\boldsymbol{\sigma}^{n-1}_{m+n} \mathsf{p})$ by Lemma 6.31(iii),

$$\begin{aligned}
&\quad = \boldsymbol{\sigma}^{m+n}_{i_{n-1}} \boldsymbol{\sigma}^{m+1}_{i_0} \cdots \boldsymbol{\sigma}^{m+n-1}_{i_{n-2}} \\
&\qquad\quad \boldsymbol{\sigma}^{n-2}_{m+n-1} \cdots \boldsymbol{\sigma}^0_{m+1} \boldsymbol{\sigma}^{n-1}_{m+n} \mathsf{p} && \text{by (IH)} \\
&\quad = \boldsymbol{\sigma}^{m+1}_{i_0} \cdots \boldsymbol{\sigma}^{m+n}_{i_{n-1}} \boldsymbol{\sigma}^{n-1}_{m+n} \cdots \boldsymbol{\sigma}^0_{m+1} \mathsf{p} && \text{by Lem. 6.31(vii)}.
\end{aligned}$$

The claim has been proved, and the lemma follows from it. □

We can now prove that the syntactic notion of 'substituting the variable x_j for the free occurrences of the variable x_i' is adequately represented by its algebraic counterpart, the unary operation $\boldsymbol{\sigma}^i_j$.

Lemma 6.35 For every flat pCRL expression p,

$$\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \boldsymbol{\sigma}^i_j \xi(p) \approx \xi(p[x_i := x_j]). \tag{6.21}$$

Proof. If $i = j$, then the lemma is trivial. If $x_i \notin \mathsf{FV}(p)$, then $p = p[x_i := x_j]$ and $\xi(p) \approx \mathbf{s}_i \xi(p)$ by Lemma 6.33, and hence (6.21) follows by Lemma 6.31(iv). For the remainder of the proof, we therefore assume that $i \neq j$ and that $x_i \in \mathsf{FV}(p)$; we proceed by structural induction on p.
Suppose that $p = \mathrm{a}(x_{i_0}, \ldots, x_{i_{n-1}})$ and $p[x_i := x_j] = \mathrm{a}(x_{j_0}, \ldots, x_{j_{n-1}})$; clearly, $j_k = i_k$ if $i_k \neq i$, and $j_k = j$ if $i_k = i$ $(0 \leq k \leq n-1)$. Let us now fix

$$m > n-1, i_0, \ldots, i_{n-1}, j_0, \ldots, j_{n-1}.$$

In particular $m > i$, since $i \in \{i_0, \ldots, i_{n-1}\}$ by our assumption that $x_i \in \mathsf{FV}(p)$, so, with a few applications of Lemma 6.31(iii) and a subsequent application of either (Cs2) or (Cs7), we easily get

$$\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a} \approx \mathbf{s}_i \boldsymbol{\sigma}^{n-1}_{m+n-1} \cdots \boldsymbol{\sigma}^0_m \mathrm{a}. \tag{6.22}$$

We now deduce (6.21) as follows:

$$\begin{aligned}
\boldsymbol{\sigma}^i_j\xi(p) &\approx \boldsymbol{\sigma}^i_j\boldsymbol{\sigma}^m_{i_0}\cdots\boldsymbol{\sigma}^{m+n-1}_{i_{n-1}}\boldsymbol{\sigma}^{n-1}_{m+n-1}\cdots\boldsymbol{\sigma}^0_m\mathsf{a} && \text{by Lem. 6.34}\\
&\approx \boldsymbol{\sigma}^m_{j_0}\cdots\boldsymbol{\sigma}^{m+n-1}_{j_{n-1}}\boldsymbol{\sigma}^i_j\boldsymbol{\sigma}^{n-1}_{m+n-1}\cdots\boldsymbol{\sigma}^0_m\mathsf{a} && \text{by Lem. 6.31(vii)}\\
&\approx \boldsymbol{\sigma}^m_{j_0}\cdots\boldsymbol{\sigma}^{m+n-1}_{j_{n-1}}\boldsymbol{\sigma}^i_j\mathbf{s}_i\boldsymbol{\sigma}^{n-1}_{m+n-1}\cdots\boldsymbol{\sigma}^0_m\mathsf{a} && \text{by (6.22)}\\
&\approx \boldsymbol{\sigma}^m_{j_0}\cdots\boldsymbol{\sigma}^{m+n-1}_{j_{n-1}}\mathbf{s}_i\boldsymbol{\sigma}^{n-1}_{m+n-1}\cdots\boldsymbol{\sigma}^0_m\mathsf{a} && \text{by Lem. 6.31(iv)}\\
&\approx \boldsymbol{\sigma}^m_{j_0}\cdots\boldsymbol{\sigma}^{m+n-1}_{j_{n-1}}\boldsymbol{\sigma}^{n-1}_{m+n-1}\cdots\boldsymbol{\sigma}^0_m\mathsf{a} && \text{by (6.22)}\\
&\approx \xi(p[x_j := x_i]) && \text{by Lem. 6.34.}
\end{aligned}$$

If $p = \delta$, then (6.21) is immediate by Lemma 6.31(i).
If $p = p_1 + p_2$, $p = p_1 \cdot p_2$, or $p = p_1 \lhd b \rhd p_2$, then (6.21) is easily deduced from Lemma 6.31(i) and the induction hypothesis.
If $p = \sum_{x_k} p'$, then $k \neq j$ by our assumption about substitutions (see p. 33). Moreover, $k \neq i$ by our assumption that $x_i \in \mathsf{FV}(p)$. By the induction hypothesis, we get that $\boldsymbol{\sigma}^i_j\xi(p') \approx \xi(p'[x_i := x_j])$, so (6.21) follows by Lemma 6.31(iii).
The proof is complete. □

We shall prove the completeness theorem by establishing a correspondence between the axioms of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ and the axioms of basic process modules. To prove that the equivalents of the axioms for the binary conditionals of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ can be derived with the axioms for guarded commands in basic process modules will be to a large extent straightforward. The deduction of the equivalent of (C6) is a minor exception, and it is convenient to prove it ahead of things, as a separate lemma.

Lemma 6.36 Let $\mathbf{C}$ be a cylindric algebra, and let b be an element of $\mathbf{C}$; then

$$(\mathsf{b}{:}\!\rightarrow \mathsf{p} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{q}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{r} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{s}) = \mathsf{b}{:}\!\rightarrow(\mathsf{p} \cdot \mathsf{r}) + \neg\,\mathsf{b}{:}\!\rightarrow(\mathsf{q} \cdot \mathsf{s})$$

for all elements p, q, r and s of a basic process module over $\mathbf{C}$.

Proof. We get

$$(\mathsf{b}{:}\!\rightarrow \mathsf{p}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{q} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{r}) = \mathsf{b}{:}\!\rightarrow(\mathsf{p} \cdot \mathsf{q}); \tag{6.23}$$

for, since $\mathsf{b} \wedge \mathsf{b} = \mathsf{b}$ and $\mathsf{b} \wedge \neg\,\mathsf{b} = \bot$ in $\mathbf{C}$, we have the following deduction:

$$\begin{aligned}
&(\mathsf{b}{:}\!\rightarrow \mathsf{p}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{q} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{r})\\
&\quad = (\mathsf{b}{:}\!\rightarrow \mathsf{p}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{q} + \bot{:}\!\rightarrow \mathsf{r}) && \text{by (Gc8), (Gc6) and (Gc4)}\\
&\quad = (\mathsf{b}{:}\!\rightarrow \mathsf{p}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{q} + \delta) && \text{by (Gc2)}\\
&\quad = (\mathsf{b}{:}\!\rightarrow \mathsf{p}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{q}) && \text{by (A6)}\\
&\quad = \mathsf{b}{:}\!\rightarrow(\mathsf{p} \cdot \mathsf{q}) && \text{by (Gc8) and (Gc7).}
\end{aligned}$$

Then, by (A4) and (A1), and since $\neg\,\neg\,\mathsf{b} = \mathsf{b}$ in $\mathbf{C}$,

$$\begin{aligned}
&(\mathsf{b}{:}\!\rightarrow \mathsf{p} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{q}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{r} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{s}) =\\
&\qquad (\mathsf{b}{:}\!\rightarrow \mathsf{p}) \cdot (\mathsf{b}{:}\!\rightarrow \mathsf{r} + \neg\,\mathsf{b}{:}\!\rightarrow \mathsf{s}) + (\neg\,\mathsf{b}{:}\!\rightarrow \mathsf{q}) \cdot (\neg\,\mathsf{b}{:}\!\rightarrow \mathsf{s} + \neg\,\neg\,\mathsf{b}{:}\!\rightarrow \mathsf{r}).
\end{aligned}$$

The lemma follows by applying (6.23) to the right-hand side. □

We now prove the completeness theorem.

Theorem 6.37 Suppose that $\mathbf{D}$ has equality and quantifier elimination. Then

$$\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash p \approx q \text{ if, and only if, } \mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$$

for all flat pCRL expressions p and q.

Proof. We first prove that if $p \approx q$ is an instance of an axiom of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$, then $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$. Note that ξ distributes over the operations $+$, $\cdot$ and δ. Hence, since every basic process module satisfies (A1)–(A7), if $p \approx q$ is an instance of one of (A1)–(A7), then $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$. Next, we treat the instances of (C1)–(C6):

(C1) Consider $\xi(p \lhd \top \rhd q) = [\top]{:}{\rightarrow}\xi(p) + \neg[\top]{:}{\rightarrow}\xi(q)$. By (GC1) the first summand of the right-hand side is provably equal to $\xi(p)$, and, since $\neg[\top] = [\bot]$ in $\mathbf{B}$, the second summand is provably equal to δ by (GC2), whence may be removed by (A6).

(C2) Consider $\xi(p \lhd b \rhd q) = [b]{:}{\rightarrow}\, p + \neg[b]{:}{\rightarrow}\, q$, and interchange the summands in the right-hand side with an application of (A1); since $[b] = \neg[\neg\, b]$ in $\mathbf{B}$ the result is $\xi(q \lhd \neg\, b \rhd p)$.

(C3) Consider

$$\begin{aligned}\xi((p \lhd b \rhd q) \lhd c \rhd q)\\ = [c]{:}{\rightarrow}([b]{:}{\rightarrow}\xi(p) + \neg[b]{:}{\rightarrow}\xi(q)) + \neg[c]{:}{\rightarrow}\xi(q);\end{aligned}$$

in the right-hand side, distribute $[c]{:}{\rightarrow}$ over the alternative composition with (GC6), and combine the guards with (GC4) to obtain

$$([c \wedge b]{:}{\rightarrow}\xi(p) + [c \wedge \neg\, b]{:}{\rightarrow}\xi(q)) + \neg[c]{:}{\rightarrow}\xi(q).$$

Rearrange the summands with (A2) and combine the guards $[c \wedge \neg\, b]$ and $\neg[c]$ with (GC3); since $[c \wedge \neg\, b] \vee \neg[c] = \neg[b \wedge c]$ in $\mathbf{B}$, the result is

$$[b \wedge c]{:}{\rightarrow}\xi(p) + \neg[b \wedge c]{:}{\rightarrow}\xi(q) = \xi(p \lhd b \wedge c \rhd q).$$

(C4) Consider

$$\begin{aligned}\xi((p + q) \lhd b \rhd (r + s))\\ = [b]{:}{\rightarrow}(\xi(p) + \xi(q)) + \neg[b]{:}{\rightarrow}(\xi(r) + \xi(s));\end{aligned}$$

in the right-hand side, with applications of (GC6) distribute the guards over the alternative compositions, and rearrange summands with (A1) and (A2); the result is

$$\begin{aligned}([b]{:}{\rightarrow}\xi(p) + \neg[b]{:}{\rightarrow}\xi(r)) + ([b]{:}{\rightarrow}\xi(q) + \neg[b]{:}{\rightarrow}\xi(s))\\ = \xi(p \lhd b \rhd r + q \lhd \neg\, b \rhd s).\end{aligned}$$

(C5) Consider $\xi(p \lhd b \vee c \rhd \delta) = [b \vee c]{:}{\rightarrow}\,\xi(p) + \neg[b \vee c]{:}{\rightarrow}\,\delta$. The first summand of the right-hand side is provably equal to $[b]{:}{\rightarrow}\,\xi(p) + [c]{:}{\rightarrow}\,\xi(p)$ by (Gc3), and the second summand may be deleted by (Gc5) and (A6).

(C6) That $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi((p \lhd b \rhd q)\cdot(r \lhd b \rhd s)) \approx \xi(p\cdot r \lhd b \rhd q\cdot s)$ follows from Lemma 6.36.

If $\mathbf{D} \models b \approx c$, then $[b] = [c]$, so if $p \approx q$ is an instance of (Bool), then $\xi(p) = \xi(q)$.

With the instances of (Cq1), (Cq2) and (Cq4)–(Cq6) (recall that (Cq3) was omitted from the deductive system $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$) we deal as follows:

(Cq1) Immediate by Lemma 6.33.

(Cq2) If $i = j$, then $\xi(\sum_{x_i} p) = \xi(\sum_{x_j} p[x_i := x_j])$; so assume that $i \neq j$. If $x_j \notin \mathsf{FV}(p)$, then $\xi(p) \approx \mathbf{s}_j\xi(p)$ by Lemma 6.33; hence, since $\mathbf{d}_{ij} = \mathbf{d}_{ji}$ in every cylindric algebra,

$$\begin{aligned}
\xi(\textstyle\sum_{x_i} p) &\approx \mathbf{s}_i\boldsymbol{\sigma}^j_i\xi(p) && \text{by Lem. 6.31(iv)}\\
&\approx \mathbf{s}_j\boldsymbol{\sigma}^i_j\xi(p) && \text{by (Cs1)}\\
&\approx \xi(\textstyle\sum_{x_j} p[x_i := x_j]) && \text{by Lem. 6.35.}
\end{aligned}$$

(Cq4) Immediate by (Cs4).

(Cq5) If $x_i \notin \mathsf{FV}(q)$, then $\xi(q) \approx \mathbf{s}_i\xi(q)$ by Lemma 6.33, so we get

$$\xi((\textstyle\sum_{x_i} p)\cdot q) \approx \mathbf{s}_i\xi(p)\cdot\mathbf{s}_i\xi(q) \approx \xi(\textstyle\sum_{x_i} p\cdot q)$$

with an application of (Cs5).

(Cq6) If x_i does not occur in b, then $[b] = \mathbf{c}_i[b]$ and $\neg[b] = \mathbf{c}_i\neg[b]$ in $\mathbf{B}$; so

$$\begin{aligned}
&\xi(\textstyle\sum_{x_i} p \lhd b \rhd \sum_{x_i} q)\\
&\quad= \mathbf{c}_i[b]{:}{\rightarrow}\,\mathbf{s}_i\xi(p) + \mathbf{c}_i\neg[b]{:}{\rightarrow}\,\mathbf{s}_i\xi(q)\\
&\quad\approx \mathbf{s}_i\,\mathbf{c}_i[b]{:}{\rightarrow}\,\xi(p) + \mathbf{s}_i\,\mathbf{c}_i\neg[b]{:}{\rightarrow}\,\xi(q) && \text{by (Gc10)}\\
&\quad= \xi(\textstyle\sum_{x_i}(p \lhd b \rhd q)).
\end{aligned}$$

For the instances of (Qe), note that if $x_i \notin \mathsf{FV}(p)$, then (*) $\xi(p) \approx \mathbf{s}_i\xi(p)$ by Lemma 6.33; hence, using that $\mathbf{c}_i[b] = [\beta((\exists x_i)b)]$ in $\mathbf{B}$, we obtain

$$\begin{aligned}
\xi(\textstyle\sum_{x_i} p \lhd b \rhd \delta) &\approx \mathbf{s}_i([b]{:}{\rightarrow}\,\mathbf{s}_i\xi(p)) && \text{by (*)}\\
&\approx \mathbf{c}_i[b]{:}{\rightarrow}\,\mathbf{s}_i\xi(p) && \text{by (Gc9)}\\
&\approx \xi(p \lhd \beta((\exists x_i)b) \rhd \delta) && \text{by (*).}
\end{aligned}$$

Suppose that $p \approx q$ is an instance of (Eq)′, say

$$\begin{aligned}
p &= \mathrm{a}(x_{i_0},\ldots,x_{i_{n-1}}) \lhd \mathrm{eq}(x_{i_0},x_{j_0}) \wedge \cdots \wedge \mathrm{eq}(x_{i_{n-1}},x_{j_{n-1}}) \rhd \delta, \text{ and}\\
q &= \mathrm{a}(x_{j_0},\ldots,x_{j_{n-1}}) \lhd \mathrm{eq}(x_{i_0},x_{j_0}) \wedge \cdots \wedge \mathrm{eq}(x_{i_{n-1}},x_{j_{n-1}}) \rhd \delta.
\end{aligned}$$

We fix $m > n-1, i_0, \ldots, i_{n-1}, j_0, \ldots, j_{n-1}$; then

$$\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \mathbf{d}_{i_0 j_0} \wedge \cdots \wedge \mathbf{d}_{i_{n-1} j_{n-1}} :\rightarrow \boldsymbol{\sigma}_{i_0}^m \cdots \boldsymbol{\sigma}_{i_{n-1}}^{m+n-1} t \approx \mathbf{d}_{i_0 j_0} \wedge \cdots \wedge \mathbf{d}_{i_{n-1} j_{n-1}} :\rightarrow \boldsymbol{\sigma}_{j_0}^m \cdots \boldsymbol{\sigma}_{j_{n-1}}^{m+n-1} t$$

for every $\mathbf{B}$-BPM_ω-term t over $\mathcal{A}$, by Lem. 6.31(ii), (vii), (GC4) and a straightforward induction on n. From this, $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$ is obtained with Lemma 6.34.

Hence, if $p \approx q$ is an axiom of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$, then $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$. Moreover, it is at once clear that an application of an inference rule of $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ translates to an application of the corresponding rule of equational logic in the setting of basic process modules. It follows that any deduction within $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ proving the validity of an arbitrary equation $p \approx q$ of flat pCRL expressions p and q can be transformed into a deduction that proves $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$; this concludes the proof of the implication from left to right.

For the other implication, note that the dimension-restriction axioms are true in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ under the interpretation of $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ as equivalence classes of $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ induced by the association in (6.17). Furthermore, by Theorem 6.9 $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ is a basic process module over $\mathbf{B}$, and satisfies, a fortiori, the instances of the axioms of basic process modules over $\mathbf{B}$ with respect to this particular interpretation. That is, if $\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(p) \approx \xi(q)$, then the equivalence class in $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ denoted by $\xi(p)$ must be the same as the equivalence class denoted by $\xi(q)$. By Lemma 6.28, $\xi(p)$ denotes $[p]$ and $\xi(q)$ denotes $[q]$, and from $[p] = [q]$ we conclude that $\Pi(\mathcal{A}, \mathbf{D})_{\exists}^{\mathrm{eq}} \vdash p \approx q$. Hence, by Theorem 6.18 $\Pi(\mathcal{A}, \mathbf{D})_{\mathsf{flat}} \vdash p \approx q$. This concludes the proof of the implication from right to left, and the proof of the theorem. □

To conclude that the set in (6.16) is a dimension-restricted free set of generators for $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ we still need to close one small gap. By Theorem 6.37 it is now clear that the mapping ξ induces an embedding from $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})_{\mathsf{flat}}$ into $\mathbf{I}(\mathcal{A}, \mathbf{D})$, and hence, by Theorem 6.18, that

$$[p] \mapsto [\xi(p)] \tag{6.24}$$

defines an embedding from $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ into $\mathbf{I}(\mathcal{A}, \mathbf{D})$. What we need, however, is an isomorphism, i.e., a surjective embedding, and one that extends the association

$$[\mathrm{a}(x_0, \ldots, x_{n-1})] \mapsto [\mathrm{a}]. \tag{6.25}$$

Actually, since we already know that the sets $\{[\mathrm{a}(x_0, \ldots, x_{n-1})] \mid \mathrm{a} \in \mathcal{A} \text{ of arity } n\}$ and $\{[\mathrm{a}] \mid \mathrm{a} \in \mathcal{A}\}$ generate $\mathsf{pCRL}(\mathcal{A}, \mathbf{D})$ and $\mathbf{I}(\mathcal{A}, \mathbf{D})$, respectively, it suffices that the embedding defined in (6.24) extends (6.25), and this is a consequence of the following lemma.

Lemma 6.38 If a is a parametrised action symbol of arity n, then

$$\mathbf{B}\text{-}\mathsf{BPM}_\omega(\mathcal{A}) \vdash \xi(\mathrm{a}(x_0, \ldots, x_{n-1})) \approx \mathrm{a}.$$

Proof. According to Definition 6.27,

$$\xi(\mathrm{a}(x_0,\ldots,x_{n-1})) = \boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{n-1}^{2n-1}\boldsymbol{\sigma}_{2n-1}^{n-1}\cdots\boldsymbol{\sigma}_n^0\mathrm{a}.$$

We prove by induction on $j < n$ that

$$\boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-1}^{n+j-1}\boldsymbol{\sigma}_{n+j-1}^{j-1}\cdots\boldsymbol{\sigma}_n^0\mathrm{a} \approx \mathrm{a}.$$

If $j = 0$, then this is immediate; if $j > 0$, then $n + j - 1 > n - 1$, and we deduce

$$\begin{aligned}
&\boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-1}^{n+j-1}\boldsymbol{\sigma}_{n+j-1}^{j-1}\cdots\boldsymbol{\sigma}_n^0\mathrm{a}\\
&\quad\approx \boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-1}^{n+j-1}\boldsymbol{\sigma}_{n+j-1}^{j-1}\cdots\boldsymbol{\sigma}_n^0\mathbf{s}_{n+j-1}\mathrm{a} && \text{by (Cs7)}\\
&\quad\approx \boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-1}^{n+j-1}\boldsymbol{\sigma}_{n+j-1}^{j-1}\mathbf{s}_{n+j-1}\boldsymbol{\sigma}_{n+j-2}^{j-2}\cdots\boldsymbol{\sigma}_n^0\mathrm{a} && \text{by Lem. 6.31(iii)}\\
&\quad\approx \boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-2}^{n+j-2}\mathbf{s}_{n+j-1}\boldsymbol{\sigma}_{n+j-2}^{j-2}\cdots\boldsymbol{\sigma}_n^0\mathrm{a} && \text{by Lem. 6.31(v)}\\
&\quad\approx \boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-2}^{n+j-2}\boldsymbol{\sigma}_{n+j-2}^{j-2}\cdots\boldsymbol{\sigma}_n^0\mathbf{s}_{n+j-1}\mathrm{a} && \text{by Lem. 6.31(iii)}\\
&\quad\approx \boldsymbol{\sigma}_0^n \cdots \boldsymbol{\sigma}_{j-2}^{n+j-2}\boldsymbol{\sigma}_{n+j-2}^{j-2}\cdots\boldsymbol{\sigma}_n^0\mathrm{a} && \text{by (Cs7)}\\
&\quad\approx \mathrm{a} && \text{by (IH).}
\end{aligned}$$

This completes the proof of the lemma. □

So, the association in (6.24) defines an isomorphism between $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ and $\mathbf{I}(\mathcal{A},\mathbf{D})$ that extends (6.25), and hence by Proposition 6.29:

Corollary 6.39 The algebra $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ is a locally finite basic process module over $\mathbf{B}$, and the set $\{[\mathrm{a}(x_0,\ldots,x_{n-1})] \mid \mathrm{a} \in \mathcal{A} \text{ of arity } n\}$ is a dimension-restricted free set of generators for $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$.

We have characterised the algebra $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ up to isomorphism, roughly, by proving that the axioms of basic process modules constitute an axiomatisation of the ground equational theory of $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ (expanded with parametrised action symbols as constant symbols). An interesting question is whether there are still some identities of $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ (equations between terms built from variables and the operations of basic process modules over $\mathbf{B}$ that are valid in $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$) that are specific to $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ in comparison with other locally finite basic process modules over $\mathbf{B}$. We conjecture that this is not the case:

Conjecture 6.40 If t and u are $\mathbf{B}$-BPM_ω-terms in variables from some countably infinite set, then $t \approx u$ is an identity of $\mathsf{pCRL}(\mathcal{A},\mathbf{D})$ if, and only if, $t \approx u$ identically holds in every locally finite basic process module over $\mathbf{B}$.

Bibliographic notes

For the material of this chapter we have drawn inspiration from a branch of mathematics called "Algebraic Logic". Excellent introductions to the subject are by Halmos (1956b), and Halmos and Givant (1998); they concentrate on algebraising (monadic) first-order predicate logic. For a general theory about associating an

algebraic semantics to a (not necessarily classical) logic we refer to a monograph by Blok and Pigozzi (1989). In establishing the correspondence between our formal system and basic process modules, we have borrowed techniques from the two principal algebraic versions of first-order predicate logic: *cylindric algebras* (Henkin *et al.*, 1971, 1985) and *polyadic algebras* (Halmos, 1956a).

The idea that a pCRL expression describes a function from the Cartesian power D^ω into an arbitrary generalised basic process algebra $\mathbf{P}$ resembles Halmos' point of view that a *propositional function* is a function from some Cartesian power into an arbitrary Boolean algebra (of propositions). The set of all such functions is again a Boolean algebra, with respect to pointwise operations, on which in addition one may define existential quantifiers (based on the infinite joins of the Boolean algebra of propositions) and substitution operations. A Boolean subalgebra of propositional functions that is moreover closed under existential quantifiers and unary substitution operations is an example of a polyadic algebra.

Our transition from the deductive system $\Pi(\mathcal{A}, \mathbf{D})^{\mathrm{eq}}_{\exists}$ to the deductive system associated with $\mathbf{B}$-BPM_ω-terms over $\mathcal{A}$ parallels a series of three articles in a 1965 issue of the Archiv für Mathematische Logik und Grundlagenforschung by Tarski (1965), by Kalish and Montague (1965), and by Monk (1965).

Tarski considers a system of first-order predicate logic with equality but without operation symbols and individual constants. He observes that $\varphi[x := y]$, the result of substituting a variable y for the free occurrences of a variable x in the first-order formula φ, is equivalent to $(\forall x)(\mathrm{eq}(x, y) \to \varphi)$ (cf. our Corollary 6.11). He then uses this observation to simplify his system, eliminating the notion of a variable occurring free in a given formula and replacing the general notion of substitution by 'replacement of one variable for another in an atomic formula'. Kalish and Montague extend Tarski's result to a system of first-order predicate logic with operation symbols and individual constants. Taking their articles together, one obtains a simplification of first-order predicate logic with equality comparable to the simplification that was achieved by means of our transition from pCRL to flat pCRL.

Monk subsequently proves that, without loss of expressivity or demonstrative power, it is possible to work exclusively with formulas in which every non-logical predicate is followed by some fixed sequence of variables. Then, atomic formulas may be thought of as constants, and this ultimately leads to dimension-restricted free cylindric algebras (see Henkin *et al.*, 1971, 1985). Compare this to our function ξ that explains how a flat pCRL expression may be translated to a $\mathbf{B}$-BPM_ω-term over $\mathcal{A}$. The idea behind the translation is to interpret an element $\mathrm{a} \in \mathcal{A}$ of arity n as a constant that denotes the action expresion $\mathrm{a}(x_0, \ldots, x_{n-1})$, where $x_0, \ldots, x_{n-1}$ is the (fixed) nth initial segment of variables.

An advantage of the theory of basic process modules is that, as opposed to pCRL, it does not involve binders. It is well-known that binders introduce a considerable amount of complexity into a syntax. They ensue the need for a distinction between free and bound occurrences of a variable, and for a more complicated notion of substitution (see Chapter 3). The λ-calculus (Barendregt, 1984) gives a systematic treatment of such things and is often incorporated in a formal language to organise the variable binding aspects. Alternatively, when there is a desire to

stay within the realm of purely equational logic, Combinatory Logic (Curry, 1930) may be incorporated for this purpose. In the context of process algebra, this approach is taken by Bergstra *et al.* (1994). They define an extension of ACP with unary operations $\sum$ on processes that are similar to our choice quantifiers, except that they have no binding effect themselves. In their setting, a binding effect is simulated by means of the incorporated (typed) Combinatory Logic.

7

Concluding remarks

We have investigated how the choice quantifiers of μCRL fit in with process algebra in the style of Bergstra and Klop (1984). Our starting point was their theory BPA_δ of basic process algebras with deadlock.

In Chapter 2 we have introduced the theory GBPA_δ, extending BPA_δ with an abstract algebraic definition of generalised summation, a partial operation from sets of processes to processes satisfying a few requirements. These requirements, formulated as axioms in the form of equations, are to ensure that generalised summation indeed generalises alternative composition, and that sequential composition distributes from the right over it. We have proved that our abstract algebraic definition of generalised summation coincides with the natural generalisation of binary alternative composition in algebras of transition trees, and we conclude from this that our axioms are rightly chosen.

In Chapter 3 we have employed the theory GBPA_δ to formalise our intuition that choice quantification is a syntactic abbreviation mechanism, used to denote sums of large (possibly infinite) sets of processes. The precise formalisation of the correspondence between choice quantification and generalised summation turned out to be a complex task. One source of discomfort was that we had to fix a data domain D to be able to say precisely which sum is denoted by a choice quantifier. As a consequence, the whole subsequent theory about pCRL is parametrised by this data domain. However, it hardly plays a meaningful role in our general theory about pCRL.

Recall that we have advertised to separate the specification of relevant data from the specification of a process. Our results in Chapter 4 may be so interpreted that in pCRL this separation is not achieved completely. Although a first-order assertion about the data can always be expressed as an equation of pCRL expressions, it is not necessarily expressible as an equation of data expressions. One might call this an anomaly in the design of pCRL, and at least from a theoretical point of view, it is another source of discomfort. For instance, a relatively complete axiomatisation of the equational theory of pCRL can only be obtained under additional assumptions with respect to the expressiveness of the data language (cf. Chapter 5).

In Chapter 6, we have used the results of the earlier chapters to improve the presentation of the theory of choice quantification. Recall that a data algebra consists of two parts: a Boolean algebra of conditions and a data part to serve as domain for the choice quantifiers. In the theory of basic process modules, the Boolean part is still present, in the form of the imported cylindric algebra. The

data domain has been eliminated altogether from the general theory. By taking a cylindric algebra (i.e., a Boolean algebra with existential quantifiers and equality) of conditions, we have achieved a complete separation of pure data aspects (in the cylindric algebra) from pure process aspects (in the ω-dimensional basic process algebra).

A further advantage of the theory of basic process modules is that it defines a variety of algebras in the universal algebraic sense, i.e., consisting of a universe and an indexed set of finitary operations on this universe. Using the theory of universal algebra, our definition yields at the same time, and in a manner that is completely standard, a semantic class of algebras and a formal system to reason about the elements of these algebras. Whereas the introduction of the language pCRL, its semantics and its deductive system is lengthy and complex, the theory of basic process modules provides an elegant shortcut in the form of an abstract algebraic theory of parametrised processes.

Bibliography

Aceto, L., Fokkink, W. J., and Verhoef, C. (2001). Structural operational semantics. In Bergstra *et al.* (2001), chapter 3, pages 197–292.

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

Baeten, J. C. M. and Bergstra, J. A. (1994). On sequential composition, action prefixes and process prefix. *Formal Aspects of Computing*, **6**(3), 250–268.

Baeten, J. C. M. and Weijland, W. P. (1990). *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Barendregt, H. P. (1984). *The Lambda Calculus — its syntax and semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics*. North-Holland, Amsterdam New-York Oxford, revised edition.

Bergstra, J. A. and Klop, J. W. (1984). Process algebra for synchronous communication. *Information and Control*, **60**(1–3), 109–137.

Bergstra, J. A. and Klop, J. W. (1985). Algebra of communicating processes with abstraction. *Theoretical Computer Science*, **37**(1), 77–121.

Bergstra, J. A., Heering, J., and Klint, P., editors (1989). *Algebraic specification*. Frontier Series. ACM Press, New York.

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

Bergstra, J. A., Ponse, A., and Smolka, S. A., editors (2001). *Handbook of Process Algebra*. North-Holland.

Blok, W. J. and Pigozzi, D. (1989). Algebraizable logics. *Memoirs of the American Mathematical Society*, **77**(396).

Blom, S., Fokkink, W., Groote, J. F., van Langevelde, I., Lisser, B., and van de Pol, J. (2001). μCRL: a toolset for analysing algebraic specifications. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, volume 2102 of *Lecture Notes in Computer Science*, pages 250–254. Springer.

Bolognesi, T. and Brinksma, E. (1987). An introduction to the ISO specification language LOTOS. *Computer Networks and ISDN System*, **14**(1), 25–59.

Bradfield, J. and Stirling, C. (2001). Modal logics and mu-calculi: An introduction. In Bergstra *et al.* (2001), chapter 4, pages 293–330.

Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W. (1984). A theory of communicating sequential processes. *Journal of the ACM*, **31**, 560–599.

Burris, S. and Sankappanavar, H. P. (1981). *A Course in Universal Algebra*. Number 78 in Graduate Texts in Mathematics. Springer-Verlag, New York Heidelberg Berlin.

Chang, C. C. and Keisler, H. J. (1990). *Model Theory*, volume 73 of *Studies in logic and the foundations of mathematics*. North-Holland, Amsterdam - New York - Oxford - Tokyo, 3rd edition.

Curry, H. B. (1930). Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, **52**, 509–536, 789–834.

Davis, M. (1982). *Computability and Unsolvability*. Dover Publications, Inc.

Fokkink, W. and Klusener, S. (1995). An effective axiomatization for real time ACP. *Information and Computation*, **122**(2), 286–299.

Fokkink, W. J. (2000). *Introduction to Process Algebra*. Texts in Theoretical Computer Science. Springer.

Fokkink, W. J. and Luttik, S. P. (2000). An ω-complete equational specification of interleaving. In U. Montanari, J. D. Rolim, and E. Welzl, editors, *Proceedings of the 27th Colloquium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *LNCS*, pages 729–743, Geneva, Switzerland. Springer.

Van Glabbeek, R. J. and Weijland, W. P. (1996). Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, **43**(3), 555–600.

Goguen, J. A. and Meseguer, J. (1985). Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, **11**(3), 307–334.

Groote, J. F. and Luttik, S. P. (1998a). Undecidability and completeness results for process algebras with alternative quantification over data. Report SEN-R9806, CWI, The Netherlands. Available from `http://www.cwi.nl/`.

Groote, J. F. and Luttik, S. P. (1998b). A complete axiomatisation of branching bisimulation for process algebras with alternative quantification over data. Report SEN-R9830, CWI, The Netherlands. Available from `http://www.cwi.nl/`.

Groote, J. F. and Ponse, A. (1994). Proof theory for μCRL: A language for processes with data. In D. J. Andrews, J. F. Groote, and C. A. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specification Languages*, Workshops in Computing, pages 232–251, Utrecht, The Netherlands. Springer-Verlag.

Groote, J. F. and Ponse, A. (1995). The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes*, Workshops in Computing, pages 26–62, Utrecht, The Netherlands. Springer-Verlag.

Groote, J. F. and Reniers, M. A. (2001). Algebraic process verification. In Bergstra *et al.* (2001), chapter 17, pages 1151–1208.

Groote, J. F., Reniers, M. A., Van Wamel, J. J., and Van der Zwaag, M. B. (2000). Completeness of timed μCRL. Report SEN-R0034, CWI. Available from `http://www.cwi.nl/`.

Groote, J. F., Ponse, A., and Usenko, Y. S. (2001). Linearization in parallel pCRL. *Journal of Logic and Algebraic Programming*, **48**, 39–70.

Halmos, P. and Givant, S. (1998). *Logic as Algebra*. Number 21 in Dolciani Mathematical Expositions. Mathematical Association of America, Washington, DC.

Halmos, P. R. (1956a). Algebraic logic, II. Homogeneous locally finite polyadic Boolean algebras of infinite degree. *Fundamenta Mathematicae*, **43**, 255–325.

Halmos, P. R. (1956b). The basic concepts of algebraic logic. *American Mathematical Monthly*, **53**, 363–387.

Halmos, P. R. (1974). *Naive Set Theory*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 2 edition. First edition (1960) published by D. Van Nostrand Co., Princeton, N.J.-Toronto-London-New York.

Henkin, L., Monk, J. D., and Tarski, A. (1971). *Cylindric Algebras – Part I*, volume 64 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company.

Henkin, L., Monk, J. D., and Tarski, A. (1985). *Cylindric Algebras – Part II*, volume 115 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company.

Hennessy, M. (1985). Acceptance trees. *Journal of the ACM*, **32**(4), 896–928.

Hennessy, M. (1991). A proof system for communicating processes with value-passing. *Formal Aspects of Computing*, **3**, 346–366.

Hennessy, M. and Lin, H. (1995). Symbolic bisimulations. *Theoretical Computer Science*, **138**(2), 353–389.

Hennessy, M. and Lin, H. (1996). Proof systems for message-passing process algebras. *Formal Aspects of Computing*, **8**(4), 379–407.

Hennessy, M. and Lin, H. (1997). Unique fixpoint induction for message-passing process calculi. In *Proceedings of CATS'97*, Australia Computer Science Communications, pages 122–131, Sidney.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, London.

Hollenberg, M. (1998). *Logic and Bisimulation*. Ph.D. thesis, Utrecht University.

Hungerford, T. W. (1974). *Algebra*, volume 73 of *Graduate Texts in Mathematics*. Springer.

Kalish, D. and Montague, R. (1965). On Tarski's formalization of predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, **7**, 81–101.

Koppelberg, S. (1989). Elementary arithmetic. In J. D. Monk and R. Bonnet, editors, *Handbook of Boolean Algebras (Vol. I)*, pages 5–46. North-Holland.

Loeckx, J., Ehrich, H.-D., and Wolf, M. (1996). *Specification of abstract data types*. John Wiley & Sons Ltd., Chichester.

Luttik, B. (2000). A note on unique factorisation of communicating processes. Available from `http://www.cwi.nl/~luttik/`.

Luttik, B. and Rodenburg, P. (1996). Transformations of reduction systems. Report P9615, Programming Research Group, University of Amsterdam.

Luttik, B. and Visser, E. (1997). Specification of rewriting strategies. In M. Sellink, editor, *Proceedings of the 2nd International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF'97)*, Electronic Workshops in Computing, pages 1–16, Berlin. Springer-Verlag.

Luttik, S. P. (1997). Description and formal specification of the link layer of P1394. In I. Lovrek, editor, *Proceedings of the 2nd International Workshop on Applied Formal Methods in System Design*, Zagreb, Croatia.

Luttik, S. P. (1999a). Complete axiomatisations of weak-, delay- and eta-bisimulation for process algebras with alternative quantification over data. Report SEN-R9914, CWI. Available from `http://www.cwi.nl/`.

Luttik, S. P. (1999b). Cylindric process algebras with conditionals give substitutionless pcrl. Report SEN-R9912, CWI. Available from `http://www.cwi.nl/`.

Luttik, S. P., Rodenburg, P. H., and Verma, R. M. (1998). Correctness criteria for transformations of rewrite systems (with an application to Thatte's transformation). Revision of (Luttik and Rodenburg, 1996); available from `http://www.cwi.nl/~luttik`.

Manes, E. G. (1985). Guard modules. *Algebra Universalis*, **21**, 103–110.

Mauw, S. and Veltink, G. J. (1990). A process specification formalism. *Fundamenta Informaticae*, **XIII**, 85–139.

Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Prentice-Hall.

McKenzie, R. N., McNulty, G. F., and Taylor, W. F. (1987). *Algebras, Lattices, Varieties — Volume I*. Wadsworth & Brooks/Cole, Monterey, California.

Milner, R. (1980). *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer.

Milner, R. (1983). Calculi for synchrony and asynchrony. *Theoretical Computer Science*, **28**(3), 267–310.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs.

Milner, R. (1999). *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, I and II. *Information and Computation*, **100**, 1–77.

Monk, D. (1965). Substitutionless predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, **7**, 102–121.

Myhill, J. (1955). Creative sets. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, **1**, 97–108.

Parrow, J. and Sangiorgi, D. (1995). Algebraic theories for name-passing calculi. *Inform. and Comput.*, **120**(2), 174–197.

Parrow, J. and Victor, B. (1998). The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS'98*, pages 176–185. IEEE Computer Society Press.

Ponse, A. (1991). Process expressions and Hoare's logic: Showing an irreconcilability of context-free recursion with Scott's induction rule. *Information and Computation*, **95**, 192–217.

Ponse, A. (1996). Computable processes and bisimulation equivalence. *Formal Aspects of Computing*, **8**(6), 648–678.

Ponse, A. and Usenko, Y. S. (2001). Equivalence of recursive specifications in process algebra. *Information Processing Letters*, **80**, 59–65.

Rasiowa, H. and Sikorski, R. (1963). *The mathematics of metamathematics*. Państwowe wydawnictwo naukowe, Warszawa, Poland.

Rathke, J. (1997). Unique fixpoint induction for value-passing processes. In *Proceedings of LICS'97, 12^{th} Annual Symposium on Logic in Computer Science, Warsaw*, pages 140–148. IEEE Computer Society Press.

Rodenburg, P. H. (2000). On adding certain constants to basic process algebra. Unpublished manuscript.

Rogers, Jr., H. (1992). *Theory of Recursive Functions and Effective Computability*. The MIT Press. Paperback edition. Original edition published by McGraw-Hill Book Company, 1967.

Shankland, C. and Van der Zwaag, M. B. (1998). The tree identify protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, **10**(5–6), 509–531.

Shoenfield, J. R. (1967). *Mathematical Logic*. Addison-Wesley Publishing Company.

Stirling, C. (2001). *Modal and Temporal Properties of Processes*. Graduate Texts in Computer Science. Springer.

Tarski, A. (1951). *A decision method for elementary algebra and geometry*. University of California Press, Berkeley and Los Angeles, Calif. 2nd ed.

Tarski, A. (1965). A simplified formalization of predicate logic with identity. *Archiv für Mathematische Logik und Grundlagenforschung*, **7**(3–4), 61–79.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, **42**, 230–265. corrections in Ibid, vol. 43, pp. 544–546.

Van der Zwaag, M. B. (2000). Time-stamped actions in pCRL algebras. Report SEN-R0002, CWI. Available from `http://www.cwi.nl/`.

Index of notations

SYMBOL	MEANING	PAGE
$p + q$	alternative composition	17, 33
$p \cdot q$	sequential composition	17, 33
δ	deadlock	17, 33
$\sum P$	generalised summation	19
$\mathrm{a}(d_1, \ldots, d_n)$	action expression	33
$p \lhd b \rhd q$	conditional	33
$\sum_x p$	choice quantifier	33
$\mathrm{a}?x_1, \ldots, x_n.p$	input prefix	47
$\mathrm{a}!x_1, \ldots, x_n.p$	output prefix	47
$\mathsf{s}_i\mathsf{p}$	projective summation	108
$\mathsf{b}{:}\!\rightarrow \mathsf{p}$	guarded command	112
$\boldsymbol{\sigma}^i_j\mathsf{p}$	substitution operation	126
$\mathsf{p} \leq \mathsf{q}$	partial order induced by $+$ on a BPA_δ	18
$p \approx q$	pCRL equation	38
$p \preccurlyeq q$	pCRL summand inclusion	38
$\top$	true	31, 54
$\bot$	false	31, 54
$\neg\, b$, $\neg\, \varphi$	complement, negation	31, 52
$b \vee c$, $\varphi \vee \psi$	join, disjunction	31, 52
$b \wedge c$, $\varphi \wedge \psi$	meet, conjunction	31, 54
$(\exists x)\varphi$	existential quantifier	52
$\mathrm{eq}(x, y)$	equality relation	54
$\varphi \rightarrow \psi$	implication	54
$\varphi \leftrightarrow \psi$	bi-implication	54
$(\forall x)\varphi$	universal quantifier	54
$\bigvee_{m \leq i \leq m} \varphi_i$	generalised disjunction	54
$\beta(\varphi)$	first-order formula φ conceived as a Boolean expression	84
$\mathbf{c}_i\mathsf{b}$	cylindrification	110
$\mathbf{d}_{ij}\mathsf{b}$	diagonal element	110
$\boldsymbol{\sigma}^i_j\mathsf{b}$	substitution operation	129
X, $\mathcal{D}$, $\mathcal{B}$	set of variables, data expressions, Boolean expressions	32
$\mathsf{FV}(p)$	set of variables with a free occurrence in p	33

Index of subjects

Keuzekwantificatie in procesalgebra

Samenvatting (Dutch summary)

In dit proefschrift bestuderen we een fragment van de processpecificatietaal μCRL. Deze taal is ontworpen voor de formele specificatie en verificatie van het *gedrag* van complexe systemen, met name van systemen die bestaan uit een aantal parallel executerende componenten. Een belangrijk aspect aan μCRL is dat het de mogelijkheid biedt om bij de specificatie van gedrag gebruik te maken van *abstracte datatypen*, apart gedefinieerd middels een meer-soortige algebraïsche specificatie. In het eerste deel van hoofdstuk 1 bespreken we het conceptuele voordeel van deze mogelijkheid, en introduceren we informeel de constructie uit μCRL die de hoofdrol speelt in de rest van dit werk: *keuzekwantificatie*.

In het tweede deel van hoofdstuk 1 komen een aantal aspecten van de procestheorie aan de orde. In het bijzonder brengen we de voordelen van de algebraïsche benadering onder de aandacht. De meeste constructies van μCRL zijn ontleend aan de algebraïsche procestheorie ACP. Het ligt dus voor de hand om deze theorie te gebruiken om μCRL-specificaties van een semantiek te voorzien. We beargumenteren dat dit een generalisatie vereist van de notie van *keuze* zoals die in ACP is bevat. Een voorkomen van de keuzekwantor uit μCRL kan namelijk aanleiding geven tot een keuze tussen oneindig veel alternatieven, terwijl met de operaties van ACP alleen keuzes tussen eindig veel alternatieven uitdrukbaar zijn. In het derde deel van hoofdstuk 1 belichten we kort de onderwerpen van de latere hoofdstukken.

In hoofdstuk 2 beschouwen we theorie BPA_δ, het fragment van ACP dat gaat over een binaire operatie $+$ voor *keuze*, een binaire operatie $\cdot$ voor *sequentiële compositie*, en een constante δ die *deadlock* representeert. We definiëren de theorie GBPA_δ, een uitbreiding van BPA_δ met *gegeneraliseerde sommatie*. Gegeven een universum P van processen is dit een operatie

$$\sum : \mathcal{D} \to \mathsf{P}, \text{ met } \mathcal{D} \subseteq \{\mathsf{P}' \mid \mathsf{P}' \subset \mathsf{P}\}$$

die aan elke (mogelijkerwijs oneindige) verzameling van processen in $\mathcal{D}$ weer een proces toekent, zo dat een drietal axioma schema's geldt. Twee van deze schema's drukken tezamen uit dat de gegeneraliseerde som van een verzameling processen de kleinste bovengrens is met betrekking tot de partiële ordening die de binaire operatie $+$ induceert op het universum van processen. Het derde schema zegt dat de binaire operatie $\cdot$ van rechts distribueert over gegeneraliseerde sommatie. Om onze nieuwe axioma schema's te motiveren, beschouwen we algebra's van transitiebomen waarvan bekend is dat ze worden geaxiomatiseerd door BPA_δ. We laten zien dat de natuurlijke uitbreidingen van deze algebra's met gegeneraliseerde

sommatie geaxiomatiseerd worden door GBPA_δ.

In de eerste helft van hoofdstuk 3 geven we een precieze definitie van pCRL, het fragment van μCRL waar het ons in de rest van dit proefschrift om gaat. Het is geparametriseerd met een *data-algebra*, een verzameling met functies en relaties. Voor de specificatie van gedrag bevat het de constructies van BPA_δ, en daarnaast: acties geparametriseerd met dataexpressies, een conditional, en keuzekwantificatie. Deze laatste constructie kwantificeert over het universum van de data-algebra. We voorzien de taal pCRL van een semantiek door een precies verband te leggen met de operaties van GBPA_δ. Keuzekwantificatie wordt daarbij opgevat als een vorm van gegeneraliseerde sommatie. Twee pCRL-expressies heten equivalent als ze in elk geschikt model van de theorie GBPA_δ hetzelfde proces aanduiden. Equivalente pCRL-expressies duiden dus dezelfde transitieboom aan, maar ook het omgekeerde blijkt het geval: als twee pCRL-expressies dezelfde transitieboom aanduiden, dan zijn ze equivalent.

In de tweede helft van hoofdstuk 3 presenteren we een tweetal hulpresultaten die betrekking hebben op de syntactische structuur van pCRL-expressies. Ten eerste definiëren we *boomvormen*, pCRL-expressies die aan bepaalde syntactische eisen voldoen. We bewijzen dat er voor elke pCRL-expressie een equivalente *boomvorm* bestaat. Ten tweede geven we een vertaling van het eindige, sequentiële fragment van 'value-passing CCS' naar pCRL. De pCRL-expressies in het bereik van deze vertaling noemen we '*input/output*'-expressies. 'Value-passing CCS' heeft niet een aparte constructie voor keuzekwantificatie, maar is gebaseerd op het zogenaamde '*input prefix*'-mechanisme, een combinatie van keuzekwantificatie en een beperkte vorm van sequentiële compositie. Voor elke 'input/output'-expressie bestaat er natuurlijk weer een equivalente boomvorm, en het blijkt dat die boomvorm nog aan een extra syntactische eis voldoet die we *expliciete instantiatie* noemen.

In hoofdstuk 4 leggen we een verband tussen de equivalentie van pCRL-expressies enerzijds en de geldigheid van eerste-orde beweringen over de data-algebra anderzijds. Zo is het altijd mogelijk om, gegeven een tweetal pCRL-expressies p en q, een eerste-orde formule met betrekking tot de data-algebra te vinden die waar is dan en slechts dan als p en q equivalent zijn. Er geldt bovendien dat het altijd mogelijk is om, gegeven een eerste-orde formule φ met betrekking tot de data algebra, een tweetal pCRL-expressies te vinden die equivalent zijn dan en slechts dan als φ waar is. Het blijkt dat keuze kwantificatie bij deze correspondentie verantwoordelijk is voor de simulatie in pCRL van zowel de universele als de existentiële kwantificatie uit de eerste-orde logica. Het 'input prefix'-mechanisme van 'value-passing CCS' is minder expressief dan keuzekwantificatie. We concluderen dit uit het feit dat vergelijkingen tussen 'input/output'-expressies corresponderen met *universele* eerste-orde beweringen over de data. Bijgevolg kan existentiële kwantificatie in principe niet worden gesimuleerd met de constructies van 'value-passing CCS'.

Om het rekenen met pCRL expressies te vergemakkelijken, presenteren we in de eerste helft van hoofdstuk 5 een deductiesysteem voor pCRL. De axioma's van dit systeem drukken fundamentele eigenschappen van de constructies van pCRL uit; ze zeggen bijvoorbeeld dat keuzekwantificatie distribueert over alternatieve compositie. De afleidingsregels van dit systeem zijn gebaseerd op de equationele logica. Aangezien de axioma's en de afleidingsregels geldig zijn met betrekking tot onze

semantiek van pCRL-expressies, kan een afleiding van ons deductiesysteem worden gezien als een volledig syntactisch bewijs dat twee pCRL-expressies equivalent zijn.

Vervolgens zou men de vraag kunnen stellen of ons deductiesysteem ook volledig is, dat wil zeggen, of ons deductiesysteem krachtig genoeg is om elke equivalentie van een dergelijk syntactisch bewijs te voorzien. De expressiviteitsresultaten uit hoofdstuk 4 laten onmiddellijk zien dat dit niet het geval kan zijn. Als er namelijk een volledig deductiesysteem voor pCRL-equivalenties zou bestaan, dan zou er ook voor elke data-algebra een algoritme zijn dat de geldigheid van een eerste-orde bewering met betrekking tot deze data-algebra kan vaststellen. In het bijzonder zou er dan volgen dat de eerste-orde theorie van de natuurlijke getallen met optelling, vermenigvuldiging en een kleiner-dan relatie beslisbaar is, en dit is in strijd met de onvolledigheidsstelling van Gödel.

De volgende vraag die zich opwerpt, is voor welke deelklasse van data-algebra's ons systeem dan wel volledig is. Deze vraag komt aan de orde in de tweede helft van hoofdstuk 5. We formuleren een drietal algemene eisen op data-algebra's, namelijk

1. dat ze ω-volledig algebraïsch moeten zijn gespecificeerd,
2. dat ze een gelijkheidspredicaat moeten bevatten, en
3. dat ze eliminatie van kwantoren moeten toelaten.

Ons deductiesysteem blijkt volledig te zijn, mits de data-algebra voldoet aan deze drie eisen, en na toevoeging van nog twee extra axiomaschema's. Verder concluderen we dat met een subtiele verzwaring van de derde eis de toevoeging van één van deze twee extra axiomaschema's overbodig is.

Wat opvalt aan de in hoofdstukken 2, 3 en 5 ontwikkelde theorie, is dat er een duidelijk onderscheid is tussen een syntactisch gedeelte (de taal pCRL, het bijbehorende deductiesysteem) en een semantisch gedeelte (de algebraïsche theorie GBPA_δ). Het verband tussen beide delen, en met name de interpretatie van keuze kwantificatie als een speciaal soort gegeneraliseerde sommatie, is complex. Hierover kan het volgende worden opgemerkt. Enerzijds is gegeneraliseerde sommatie een operatie is met mogelijkerwijs oneindig veel argumenten, en daardoor niet geschikt als constructie van een formele taal. Anderzijds is keuzekwantificatie weliswaar een geschikte constructie voor een formele taal, maar ook afhankelijk van de syntactische structuur van zijn argument, en daardoor niet geschikt als operatie van een algebraïsche theorie.

Door deze scheiding van syntax en semantiek, mist de theorie de wiskundige elegantie van haar voorganger, de algebraïsche theorie BPA_δ. In hoofdstuk 6 definiëren we de theorie van de basis procesmodules met als doel de syntax, het deductiesysteem en de semantiek van pCRL in één algebraïsche theorie te verenigen. We geven een vertaling van pCRL-expressies naar termen in de signatuur van de basis procesmodules. We bewijzen vervolgens dat twee pCRL-expressies equivalent zijn dan en slechts dan als hun vertalingen equivalent zijn volgens de axioma's van basis procesmodules.

Dit proefschrift eindigt, in hoofdstuk 7, met enige conclusies.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes.* Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search.* Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection.* Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases.* Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics.* Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems.* Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods.* Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems.* Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules.* Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System.* Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars.* Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction.* Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic.* Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms.* Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes.* Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols.* Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor.* Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant.* Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs.* Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language.* Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure.* Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication.* Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle.* Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection.* Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-04