

On the Expressiveness of Choice Quantification

Bas Luttik¹

CWI, The Netherlands

Abstract

We define process algebras with a generalised operation \sum for choice. For every infinite cardinal κ , we prove that the algebra of transition trees with branching degree $< \kappa$ is free in the class of process algebras in which \sum is defined for all subsets with a cardinality $< \kappa$. We explain how the expressions of a fragment of the specification language μCRL may be used to denote elements of our process algebras. In particular, we explain how choice quantifiers may be used to denote infinite sums. We show that choice quantifiers can simulate both the existential and the universal quantifiers of first-order logic, while the input prefix mechanism can only simulate the universal quantifier.

Key words: process algebra, choice quantification, input prefix mechanism, transition tree

1991 MSC: 03D40, 08A50, 68Q90

1 Introduction

This paper is about modeling input in process algebras. Suppose that $c?x.p$ denotes the process that receives along channel c an arbitrary element \mathbf{d} from a domain \mathbf{D} of data values and subsequently proceeds as $p[x := \mathbf{d}]$, say

$$c?x.p \xrightarrow{c?\mathbf{d}} p[x := \mathbf{d}], \text{ for all } \mathbf{d} \in \mathbf{D}. \quad (1)$$

Email address: `luttik@cs.vu.nl` (Bas Luttik).

¹ Present address: Faculty of Sciences, Vrije Universiteit, De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands.

Furthermore, suppose that $c!d.q$ denotes the process that sends a value $d \in D$ along c , i.e.,

$$c!d.q \xrightarrow{c!d} q.$$

In the event that the actions $c?d$ and $c!d$ occur simultaneously, we speak of a *synchronisation*. Such a synchronisation may take place, e.g., when we put the processes $c?x.p$ and $c!d.q$ in parallel; we write in this case

$$c?x.p \mid c!d.q \longrightarrow p[x := d] \mid q.$$

Intuitively, the synchronisation causes the value d to *pass* through channel c from the parallel component at the right to the component at the left.

Milner (1983) (see also Milner, 1989) observed that, according to his semantics (now usually referred to as *early semantics*, see, e.g., Ingólfssdóttir and Lin (2001)), the notion of input is definable in terms of the operation for *choice* in the following way: for a set of processes P , denote by $\sum P$ the process that reflects a choice between the elements of P ; then

$$c?x.p = \sum \{c?d.p[x := d] \mid d \in D\}. \quad (2)$$

An operation for choice belongs to the standard equipment of process algebras, albeit usually in its binary form and denoted by $+$. Hence, if D is a finite set, say $D = \{d_1, \dots, d_n\}$, then the process $c?x.p$ is an element of every process algebra with actions $c?d_1, \dots, c?d_n$ and processes $p[x := d_1], \dots, p[x := d_n]$, viz. the element $c?d_1.p[x := d_1] + \dots + c?d_n.p[x := d_n]$. If D is infinite, then the availability of the process $c?x.p$ in a process algebra varies. For instance, if D is countably infinite, then by (1) the process $c?x.p$ gives rise to an infinitely branching transition tree. So it is certainly not an element of the algebra of finitely branching transition trees, but it could be an element of the algebra of countably branching transition trees (this still depends on the availability of the actions $c?d$ and of the processes $p[x := d]$).

Infinite sums Milner observed (2) as a property of his semantics. In this paper we take it as an axiom, so that the question whether some process algebra is suitable to model input over some domain D amounts to asking which sums it has. In Section 2 we consider process algebras with binary operations $+$ and \cdot for alternative and sequential composition, respectively, and with an explicit deadlocked process δ (i.e., we consider the models of the theory BPA_δ of Bergstra and Klop (1984b)). We extend such process algebras with an operation \sum that assigns sums to (possibly infinite) sets of processes. In the case of transition trees, the sum $\sum T$ of a set T of transition trees is the transition tree that we get by identifying the roots of the trees in T . Clearly, $\sum T$ 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 general, a $\text{BPA}_\delta \mathbf{P}$ with universe \mathbf{P} will be fitted with an operation

$$\sum : \mathcal{D} \rightarrow \mathbf{P}, \text{ with } \mathcal{D} \subseteq \{\mathbf{P}' \mid \mathbf{P}' \subseteq \mathbf{P}\}$$

satisfying certain axioms (see Table 2 below). We motivate our axiomatic definition by proving that it coincides with the concrete definition in the algebra of transition trees (Theorem 11).

Since our operation \sum takes subsets of \mathbf{P} into \mathbf{P} , one may think of it as an operation with variable (and possibly infinite) arity. This entails a departure from the standard theory of abstract algebras, which is about operations with a fixed (and finite) arity. Rasiowa and Sikorski (1963) have shown that many of the central notions in abstract algebra (e.g., subalgebra, homomorphism and congruence) generalise in a straightforward manner. There is one place, though, where things get considerably more complex: terms over the signature of our process algebras are generally not finite. For the benefit of formal reasoning it is often convenient to have finite expressions denoting certain infinite sums. The process expressions of the specification language μCRL ² defined by Groote and Ponse (1995) (for a survey, see Groote and Reniers (2001)) may serve this purpose.

A language with choice quantifiers The general goal of μCRL (and of similar languages such as PSF (Mauw and Veltink, 1990) and LOTOS (Bolognesi and Brinksma, 1987)) is to combine in a single framework the formal specification of concurrent processes and the data that are passed in synchronisations. Data are defined by means of a many-sorted algebraic specification. For the specification of processes μCRL has the usual operations of process algebra (more specifically, it has the operations of Bergstra and Klop's ACP (Bergstra and Klop, 1984b)), with three additional mechanisms for the combination of data and processes:

- (1) *parametrised actions*: if c is a parametrised action of arity n and x_1, \dots, x_n are variables ranging over data values, then the expression $c(x_1, \dots, x_n)$ denotes an action for every instance with data values for the variables;
- (2) a *conditional construct*: if b is a Boolean expression, then $p \triangleleft b \triangleright q$ denotes the same process as p when b evaluates to true, and it denotes the same process as q otherwise; and
- (3) *choice quantifiers*: if p is a process expression, possibly containing a free variable x that ranges over data, then the expression $\sum_x p$ denotes the

² micro Common Representation Language

choice between all instances of p with a data value for x :

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

In Section 3 we define a recursion-free, sequential fragment of μCRL , called pCRL , that includes all the aforementioned constructs. We explain how pCRL expressions may be used to denote elements of process algebras with infinite sums, and in particular how they may be used to denote infinitely branching transition trees. The input mechanism is expressible in pCRL via

$$c?x.p = \sum_x c(x) \cdot p. \tag{3}$$

Note that the expression $c?x.p$ displays two dependencies between the process p and the input action $c?x$. Firstly, the process p cannot execute before the input action $c?x$ has occurred. Secondly, p depends on the value that is received in the execution of the input action. In pCRL these dependencies are expressed by separate constructs: sequential composition ensures that p comes after $c?x$; choice quantification establishes a link between the variable x in $c?x$ and free occurrences of x in p .

Expressiveness In Section 4 we study the expressiveness of the constructs of pCRL by measuring the logical complexity of pCRL equations, equations of (finite) pCRL expressions. Not surprisingly, the logical complexity of a pCRL equation depends on the data that occurs in it. The results that we obtain are therefore relative to the expressiveness of the data. We prove that any pCRL equation can be effectively transformed into an equivalent first-order assertion about the data (Theorem 33), and that, conversely, any first-order assertion about the data gives rise to an equivalent pCRL equation (Theorem 40). Hence, pCRL is precisely as expressive as first-order logic with respect to the data model. To a large extent, pCRL owes its expressiveness to the choice quantifiers. They account for the simulation of the universal as well as the existential quantifiers of first-order logic.

In Section 5, we consider the recursion-free fragment of *value-passing CCS* (Milner, 1989; Hennessy, 1991), which includes the input mechanism $c?x.p$ as a primitive construction and does not have the general form of choice quantification. Using (3), the expressions of this language may be translated into pCRL expressions, and the translation gives rise to a particular fragment of pCRL , obtained by imposing a syntactic restriction on the use of choice quantification. Roughly, we define that a pCRL expression has *explicit instantiation* if every occurrence of a choice quantifier \sum_x is immediately followed by an action that has x as a parameter (note that the pCRL expression of (3) satisfies this condition). It turns out that an equation of pCRL expressions with explicit instantiation has the content of a universal first-order assertion over the data

that occurs in it (Corollary 49). We thus conclude the input mechanism itself only accounts for the simulation of the universal quantifiers of first-order logic.

This article ends, in Section 6, with some concluding remarks and some discussion of related work.

2 Process algebras with infinite sums

A *basic process algebra with deadlock* is an algebra³ $\mathbf{P} = \langle \mathbf{P}, +, \cdot, \delta \rangle$ that satisfies for all $\mathbf{p}, \mathbf{q}, \mathbf{r} \in \mathbf{P}$ the equalities in Table 1 (Bergstra and Klop, 1984b). The class of all basic process algebras with deadlock is denoted by \mathbf{BPA}_δ . The elements of a basic process algebra with deadlock we shall call *processes*. Intuitively, a process $\mathbf{p} \in \mathbf{P}$ is a collection of behaviours that we shall refer to as the *alternatives* in \mathbf{p} .

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

Table 1

The axioms of basic process algebras with deadlock.

The operation $+$ stands for *alternative composition* (or: *choice*); if \mathbf{p} and \mathbf{q} are processes, then $\mathbf{p} + \mathbf{q}$ is the process that executes either an alternative in \mathbf{p} or an alternative in \mathbf{q} . According to (A1)–(A3), the structure $\langle \mathbf{P}, + \rangle$ 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 \mathbf{p} and \mathbf{q} are processes, then $\mathbf{p} \cdot \mathbf{q}$ is the process that starts with executing an alternative in \mathbf{p} , and if this execution terminates, then it proceeds with executing an alternative in \mathbf{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

³ 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).

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 illustrate this by means of an example.

Example 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 Fig. 1). The sender has a connection to the medium that

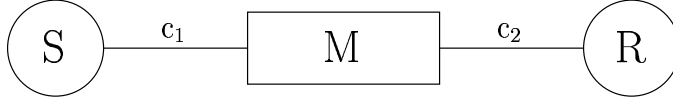


Fig. 1. A simple protocol for the acknowledged transmission of messages.

we call c_1 and the receiver has a connection to the medium that we call c_2 . The sender sends a message \mathbf{m} into the medium along c_1 . In the medium the contents of \mathbf{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 \mathbf{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 (1) to the sender if it has received a valid message; otherwise it sends a negative acknowledgment (0). The sending party may be modeled as the following process:

$$S = s_1(\mathbf{m}) \cdot (r_1(0) + r_1(1)),$$

where $s_1(\mathbf{m})$ denotes the action of sending \mathbf{m} along c_1 , $r_1(0)$ denotes the action of receiving 0 along c_1 and $r_1(1)$ denotes the action of receiving 1 along c_1 . It is understood here that an action $r_1(a)$ ($a \in \{0, 1\}$) synchronises with an action $s_1(a)$ from the medium. Thus, the choice between $r_1(0)$ or $r_1(1)$ is not made by the sender. It is determined by the medium, and it is not made before the action $s_1(\mathbf{m})$ has occurred. So, we want that

$$s_1(\mathbf{m}) \cdot (r_1(0) + r_1(1)) \neq s_1(\mathbf{m}) \cdot r_1(0) + s_1(\mathbf{m}) \cdot r_1(1).$$

2.1 Generalised basic process algebras with deadlock

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

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

Deadlock is the least element with respect to this partial order, and any two processes \mathbf{p} and \mathbf{q} have a least upper bound $\mathbf{p} + \mathbf{q}$. Thus, in a basic process

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

Table 2

The axioms for generalised summation.

algebra with deadlock any finite set $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ of processes has a least upper bound $\mathbf{p}_1 + \dots + \mathbf{p}_n$.

In Example 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 $\mathbf{D} = \{\mathbf{d}_1, \dots, \mathbf{d}_n\}$ is any arbitrary finite data domain, then we may model the receipt of an arbitrary element of \mathbf{D} as the process $r(\mathbf{d}_1) + \dots + r(\mathbf{d}_n)$, i.e., as the least upper bound of the set of actions that stand for the receipt of a particular element of \mathbf{D} . Taking this a little further, if \mathbf{D} happens to be an infinite set, then the process that models the receipt of an arbitrary element from \mathbf{D} would be the least upper bound of the set of processes that represent the receipt of any particular element of \mathbf{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 operation in an algebra. Let \mathbf{A} be a set; a *generalised operation* \mathbf{O} on \mathbf{A} is a partial mapping from the subsets of \mathbf{A} to \mathbf{A} . That is, $\mathbf{O} : \mathcal{D} \rightarrow \mathbf{A}$, where \mathcal{D} is a set of subsets of \mathbf{A} . The class \mathcal{D} is called the *domain* of the generalised operation \mathbf{O} and the sets in \mathcal{D} are called the *admissible sets* of the operation \mathbf{O} . Then, Rasiowa and Sikorski proceed to define a *generalised (abstract) algebra* as a structure $\langle \mathbf{A}, o_1, \dots, o_m, O_1, \dots, O_n \rangle$, where \mathbf{A} is a set, o_1, \dots, o_m is a sequence of finitary operations on \mathbf{A} and O_1, \dots, O_n is a sequence of generalised operations on \mathbf{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

$$\sum : \mathcal{D} \rightarrow \mathbf{P},$$

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

Suppose that P' is admissible for \sum . If \sum satisfies (GA1) of Table 2, then $\sum P'$ is an upper bound of P' with respect to \leq . If \sum also satisfies (GA2) of

Table 2, then $\Sigma P'$ is the least upper bound of P' with respect to \leq . If (GA1) and (GA2) hold for all admissible P' , then we say that Σ *generalises* $+$. We have the following lemma.

Lemma 2. *If Σ generalises $+$, then $\Sigma\{p_1, \dots, p_n\} = p_1 + \dots + p_n$ for all finite sets $\{p_1, \dots, p_n\}$ that are admissible for Σ .*

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

Note that if $P' = \{p_1, \dots, p_n\}$ and $\{p \cdot q \mid p \in P'\}$ are both admissible for Σ , then by (A4) and Lemma 2

$$\Sigma P' \cdot q = (p_1 + \dots + p_n) \cdot q = p_1 \cdot q + \dots + p_n \cdot q = \Sigma\{p \cdot q \mid p \in P'\}.$$

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

Example 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

$$\cup : \{\Gamma \subseteq \Omega \mid |\Gamma| \leq \aleph_0\} \rightarrow \Omega.$$

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

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

We shall consider extensions of basic process algebras with deadlock with an operation Σ that generalises $+$ in such a way that \cdot distributes from the right over Σ . Distributivity from the right of \cdot over Σ is formulated as (GA3) in Table 2, which is to be interpreted in the sense that if one side of the equality is defined, then so is the other.

Definition 4. A *generalised* basic process algebra with deadlock is a generalised algebra $\mathbf{P} = \langle P, +, \cdot, \delta, \Sigma \rangle$ such that

- (i) $\langle P, +, \cdot, \delta \rangle$ is a basic process algebra with deadlock;
- (ii) $P' \subseteq P$ is admissible if, and only if, $\{p \cdot q \mid p \in P'\}$ is admissible for all $q \in P$;

(iii) (GA1)–(GA3) of Table 2 hold for all admissible $P' \subseteq P$ and for all $q \in P$.

We denote by $GBPA_\delta$ the class of generalised basic process algebras with deadlock.

Suppose that $P = \langle P, +, \cdot, \delta \rangle$ is an arbitrary basic process algebra with deadlock. If we want to extend it with a generalised operation Σ that satisfies the axioms in Table 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 Σ as having no admissible sets at all (the *trivial generalisation* of P), or as having as admissible sets precisely the finite subsets of P (the *finitary generalisation* of P). But mostly, we shall be interested in the *maximal generalisation* of P in which the domain of Σ is the largest set of subsets of P such that $\langle P, +, \cdot, \delta, \Sigma \rangle$ is a generalised basic process algebra with deadlock.

Example 5. Suppose that the messages of Example 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:

$$R = \Sigma(\{r_2(\mathbf{m}) \cdot s_2(1) \mid \mathbf{m} \in M \text{ \& \; } \mathbf{m} \text{ is valid}\} \cup \{r_2(\mathbf{m}) \cdot s_2(0) \mid \mathbf{m} \in M \text{ \& \; } \mathbf{m} \text{ is not valid}\}).$$

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 1 we have specified the process S by explaining how it is obtained from the actions $s_1(\mathbf{m})$, $r_1(0)$ and $r_1(1)$ by means of the operations for sequential and alternative composition. In Example 5 we have specified the process R by explaining how it is obtained from the actions $r_2(\mathbf{m})$, $s_2(0)$ and $s_2(1)$ 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 $Q \subseteq P$ is *closed* under the generalised operation Σ if $\Sigma P' \in Q$ for every $P' \subseteq Q$ that is admissible for Σ in P . A generalised basic process algebra with deadlock $Q = \langle Q, +, \cdot, \delta, \Sigma \rangle$ is a *subalgebra* of P if:

- (i) $\langle Q, +, \cdot, \delta \rangle$ is a subalgebra of $\langle P, +, \cdot, \delta \rangle$;
- (ii) Q is closed under Σ ; and
- (iii) Σ in Q is the restriction to Q of Σ in P .

A set $P_0 \subseteq P$ is a set of *generators* for P if the least subalgebra of P that contains P_0 is P itself. Let $P = \langle P, +, \cdot, \delta, \Sigma \rangle$ be a generalised basic process algebra with deadlock, and let us fix a set $A \subseteq P$ of *actions*. The least subalgebra

that contains \mathbf{A} consists of those elements of \mathbf{P} that can be obtained from the actions in \mathbf{A} by means of applications of the fundamental operations of generalised basic process algebras with deadlock. Hence, if \mathbf{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 urelements⁴ that we shall call *labels* and we define the set $T_\kappa(\mathcal{L})$ of *transition trees with branching degree* $< \kappa$ as the least set such that

- (i) $\{\ell\} \in T_\kappa(\mathcal{L})$ for all $\ell \in \mathcal{L}$;
- (ii) if $\ell \in \mathcal{L}$ and $\mathbf{t} \in T_\kappa(\mathcal{L})$, then $\{\langle \ell, \mathbf{t} \rangle\} \in T_\kappa(\mathcal{L})$; and
- (iii) if $T' \subseteq T_\kappa(\mathcal{L})$ and $|T'| < \kappa$, then $\bigcup T' \in T_\kappa(\mathcal{L})$.

The elements of a tree we shall call *branches*. Clearly, if \mathbf{b} is a branch, then either $\mathbf{b} \in \mathcal{L}$ or there exists a label ℓ and a tree \mathbf{t} such that $\mathbf{b} = \langle \ell, \mathbf{t} \rangle$. If $\mathbf{t} \in T_\kappa(\mathcal{L})$, then \mathbf{t} has less than κ branches. The elements of $T_{\aleph_0}(\mathcal{L})$ we shall call *finitely branching* (\aleph_0 denotes the cardinality of ω); they may be pictured as in Fig. 2.

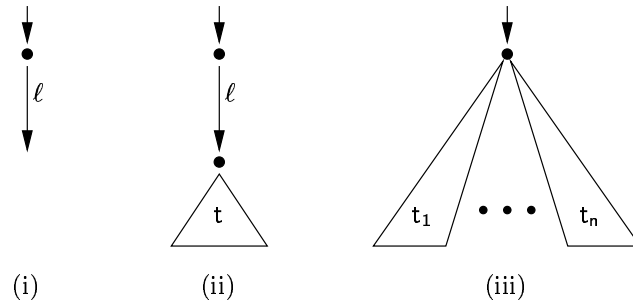


Fig. 2. The finitely branching transition trees as constructed in (i)–(iii).

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

$$\mathbf{t} \cdot \mathbf{u} = \bigcup_{\mathbf{b} \in \mathbf{t}} (\mathbf{b} \odot \mathbf{u}),$$

where $\langle \ell, \mathbf{t}' \rangle \odot \mathbf{u} = \langle \ell, \mathbf{t}' \cdot \mathbf{u} \rangle$ and $\ell \odot \mathbf{u} = \langle \ell, \mathbf{u} \rangle$ ($\ell \in \mathcal{L}$, $\mathbf{t}' \in T_\kappa(\mathcal{L})$).

⁴ 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.

Let us denote by \mathcal{D}_κ 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

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

Proposition 6. *The algebra $\mathsf{T}_\kappa(\mathcal{L}) = \langle \mathsf{T}_\kappa(\mathcal{L}), +, \cdot, \delta, \Sigma \rangle$ 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 $\bigcup \mathsf{T}' = \Sigma \mathsf{T}'$ 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 $\mathsf{T}_\kappa(\mathcal{L})$. It is immediate from the definitions that (GA3) holds. Since $\Sigma \emptyset = \emptyset = \delta$, (A7) is a special case of (GA3). Since Σ generalises $+$ and every two-element set of trees is admissible for Σ , it follows from Lemma 2 that (A4) is also a special case of (GA3). So, it remains to show that (A5) holds, i.e., that $(\mathbf{t} \cdot \mathbf{u}) \cdot \mathbf{v} = \mathbf{t} \cdot (\mathbf{u} \cdot \mathbf{v})$ for all \mathbf{t} , \mathbf{u} and \mathbf{v} ; we proceed by induction on the rank⁵ of \mathbf{t} :

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

The proof of the proposition is complete. \square

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 $\mathbf{a} \cdot \mathbf{t}$, where \mathbf{a} is an action, and the branches of the form $\langle \ell, \mathbf{t} \rangle$. 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

⁵ We mean the rank of \mathbf{t} as a *set*. In Zermelo-Fraenkel set theory, the formation of sets may be thought of as proceeding in *stages*. At each stage exactly one new ordinal is constructed (cf., e.g., Shoenfield, 1967), and the rank of the set \mathbf{t} is the ordinal that belongs to the stage in which \mathbf{t} is first formed. It so happens that if x and y are sets such that $x \in y$, then the rank of x is strictly less than the rank of y . Therefore, using the standard definition $\langle x, y \rangle = \{\{x\}, \{x, y\}\}$ of ordered pair, we get that if $\langle \ell, \mathbf{t}' \rangle$ is a branch of \mathbf{t} , then $\mathbf{t}' \in \{\ell, \mathbf{t}'\} \in \langle \ell, \mathbf{t}' \rangle \in \mathbf{t}$, whence that the rank of \mathbf{t}' is less than the rank of \mathbf{t} .

Fig. 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).

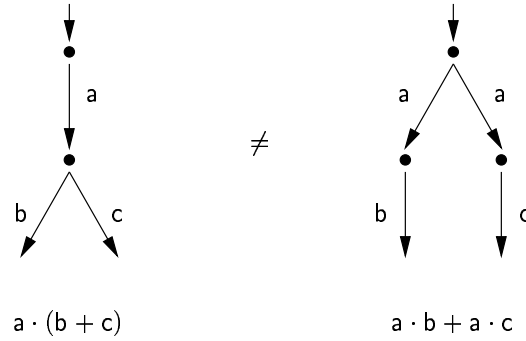


Fig. 3. In $\mathbf{T}_\kappa(\mathcal{L})$, sequential composition does not distribute from the left over alternative composition.

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

Lemma 7. *If a and b are tree actions, then*

- (i) $a \not\leq \delta$, and $a \cdot t \not\leq \delta$ for all trees t ;
- (ii) for all trees t , u and v :
 - (a) $a \leq t + u$ if, and only if, $a \leq t$ or $a \leq u$, and
 - (b) $a \cdot t \leq u + v$ if, and only if, $a \cdot t \leq u$ or $a \cdot t \leq v$;
- (iii) for all admissible $\mathsf{T}' \subseteq \mathbf{T}_\kappa(\mathcal{L})$:
 - (a) $a \leq \sum \mathsf{T}'$ if, and only if, there exists $t' \in \mathsf{T}'$ such that $a \leq t'$, and
 - (b) $a \cdot t \leq \sum \mathsf{T}'$ if, and only if, there exists $t' \in \mathsf{T}'$ such that $a \cdot t \leq t'$;
- (iv) $a \not\leq b \cdot t$ and $a \cdot t \not\leq b$, for all trees t ;
- (v) $a \leq b$ if, and only if, $a = b$; and
- (vi) $a \cdot t \leq b \cdot u$ if, and only if, $a = b$ and $t = u$, for all trees t and u .

2.3 Free generalised basic process algebras with deadlock

Let $\mathbf{P} = \langle \mathbf{P}, +, \cdot, \delta, \Sigma \rangle$ 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 $\mathbf{Q} = \langle \mathbf{Q}, +, \cdot, \delta, \Sigma \rangle$. Then, we define a mapping

$$h : \mathbf{P} \rightarrow \mathbf{Q}$$

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

- (i) $h(\mathbf{P}')$ is admissible for Σ in \mathbf{Q} and
- (ii) $h(\Sigma \mathbf{P}') = \Sigma h(\mathbf{P}')$,

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

$$f : \mathbf{A} \rightarrow \mathbf{Q}.$$

If \mathbf{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 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 \mathbf{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 $X, Y \in \mathbf{L}$; and we define $\Sigma \mathbf{L}' = \bigcup \mathbf{L}'$ for all $\mathbf{L}' \subseteq \mathbf{L}$. The generalised algebra $\mathbf{L} = \langle \mathbf{L}, +, \cdot, \delta, \Sigma \rangle$ is a generalised basic process algebra with deadlock and it is generated by the set $\mathbf{L}_0 = \{\{\ell\} \mid \ell \in \mathcal{L}\}$. 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 $\mathbf{Q} = \langle \mathbf{Q}, +, \cdot, \delta, \Sigma \rangle$ is a generalised basic process algebra with deadlock and every mapping $f : \mathbf{L}_0 \rightarrow \mathbf{Q}$ extends to a homomorphism $h : \mathbf{L} \rightarrow \mathbf{Q}$, 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 Fig. 3), it follows that not every mapping $f : \mathbf{L}_0 \rightarrow \mathbf{T}_\kappa(\mathcal{L})$ extends to a homomorphism $h : \mathbf{L} \rightarrow \mathbf{T}_\kappa(\mathcal{L})$.

Suppose that \mathcal{K} is any subclass of \mathbf{GBPA}_δ and let \mathbf{P}_0 be a set of generators for \mathbf{P} ; then \mathbf{P} is *free for* \mathcal{K} *over* \mathbf{P}_0 if every mapping $f : \mathbf{P}_0 \rightarrow \mathbf{Q}$ from \mathbf{P}_0 into the universe \mathbf{Q} of an element \mathbf{Q} of \mathcal{K} can be extended to a homomorphism $h : \mathbf{P} \rightarrow \mathbf{Q}$. We say that \mathbf{P} is *free in* \mathcal{K} *over* \mathbf{P}_0 if $\mathbf{P} \in \mathcal{K}$ and \mathbf{P} is free for \mathcal{K} over \mathbf{P}_0 . If \mathbf{P} is free in \mathcal{K} over \mathbf{P}_0 , then \mathbf{P}_0 is called a *free generating set for* \mathbf{P} , and \mathbf{P} is said to be *freely generated by* \mathbf{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 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 9. Let \mathbf{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 \mathbf{a}^n ($n \geq 1$) inductively as follows: $\mathbf{a}^1 = \mathbf{a}$ and $\mathbf{a}^{n+1} = \mathbf{a} \cdot \mathbf{a}^n$. Clearly, the set

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

is admissible for Σ in $\mathbf{T}_{\aleph_1}(\mathcal{L})$. With Lemma 7 and induction on m and n it is easily verified that $\mathbf{a}^m = \mathbf{a}^n$ implies $m = n$, so $|\mathbf{T}'| = \aleph_0$. Consequently, \mathbf{T}' is not admissible for Σ 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 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 Σ , 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 Σ of $\mathbf{T}_\kappa(\mathcal{L})$ consists precisely of the subsets of $\mathbf{T}_\kappa(\mathcal{L})$ that have cardinality less than κ .

Definition 10. A generalised basic process algebra with deadlock with universe \mathbf{P} is κ -complete if every $\mathbf{P}' \subseteq \mathbf{P}$ with $|\mathbf{P}'| < \kappa$ is admissible for Σ .

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 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 $\mathbf{T}_0 = \{\{\ell\} \mid \ell \in \mathcal{L}\}$.*

Proof. Clearly, $\mathbf{T}_\kappa(\mathcal{L})$ is κ -complete, and it is generated by \mathbf{T}_0 . Let \mathbf{P} be any κ -complete generalised basic process algebra, and suppose $f : \mathbf{T}_0 \rightarrow \mathbf{P}$. We

define a mapping $h : \mathbf{T}_\kappa(\mathcal{L}) \rightarrow \mathbf{P}$ by induction on the rank of transition trees:

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

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

Note that $h(\delta) = \sum \emptyset = \delta$. It is clear that h preserves \sum , whence, by Lemma 2, h also preserves $+$. To prove that $h(\mathbf{t} \cdot \mathbf{u}) = h(\mathbf{t}) \cdot h(\mathbf{u})$ we do induction on the rank of \mathbf{t} :

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

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

Remark 12. For the reader familiar with the theory of basic process algebras with deadlock we note that the above theorem is a generalisation of the completeness theorem for basic process algebras with deadlock (see Baeten and Weijland (1990)). This can be seen as follows. Consider the free algebra $\mathbf{T}_{\aleph_0}(\mathcal{L})$. Clearly, it is isomorphic to the algebra of finite acyclic process graphs modulo bisimulation. Since, by Lemma 2, the operation \sum is a defined operator in $\mathbf{T}_{\aleph_0}(\mathcal{L})$, it must also be a free algebra in the class of basic process algebras with deadlock. Hence, it is isomorphic to the initial algebra of BPA_δ -terms with actions from \mathbf{T}_0 .

3 The syntax and semantics of pCRL

In the previous section 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 facilitate an explicit treatment of infinite sums. In this section, 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 $\mathbf{P} = \langle \mathbf{P}, +, \cdot, \delta, \sum \rangle$ 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 \mathbf{A} of *action symbols* (e.g., $s_1(\mathbf{m})$, $r_1(\mathbf{0})$) and a mapping

$$\text{act} : \mathbf{A} \rightarrow \mathbf{P}$$

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 $s_1(\mathbf{m})$, $r_1(\mathbf{0})$ and $r_1(\mathbf{1})$ as actions of \mathbf{P} , we may describe another element of \mathbf{P} with the expression $s_1(\mathbf{m}) \cdot (r_1(\mathbf{0}) + r_1(\mathbf{1}))$ (see Example 1). Similarly, if we already have an expression for each element of $\mathbf{P}' \subseteq \mathbf{P}$ and \mathbf{P}' is a finite set, then we could describe the least upper bound of the elements in \mathbf{P}' writing the symbol Σ and listing the expressions for the elements of \mathbf{P}' between brackets. For instance, we could describe the least upper bound of the set consisting of three actions denoted by $r(\mathbf{m}_1)$, $r(\mathbf{m}_2)$ and $r(\mathbf{m}_3)$ with the expression $\Sigma\{r(\mathbf{m}_1), r(\mathbf{m}_2), r(\mathbf{m}_3)\}$.

When \mathbf{P}' is an infinite set, listing the expressions for the elements of \mathbf{P}' is not an option. We need a method to denote the least upper bound of an infinite set \mathbf{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 \mathbf{D} can be modeled as the least upper bound of the set of actions that model the receipt of a particular element of \mathbf{D} , i.e., as the process $\Sigma\{r(\mathbf{d}) \mid \mathbf{d} \in \mathbf{D}\}$. Note how we make use of the intuitive structure of the expression $r(\mathbf{d})$ 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

$$\mathbf{A} = \{a(\mathbf{d}_1, \dots, \mathbf{d}_n) \mid a \in \mathcal{A} \text{ of arity } n \text{ and } \mathbf{d}_1, \dots, \mathbf{d}_n \in \mathbf{D}\}. \quad (4)$$

Then, certain infinite sums are expressible using quantification over \mathbf{D} . Let x be a variable that ranges over \mathbf{D} ; we denote the process $\Sigma\{r(\mathbf{d}) \mid \mathbf{d} \in \mathbf{D}\}$ with the expression $\Sigma_x r(x)$. The symbol Σ_x will be called a *choice quantifier*.

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

Example 13. Let \mathbf{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 $\text{sqr} : \mathbf{N} \rightarrow \mathbf{N}$ is such that $n \mapsto n^2$, then

$$\Sigma_x \text{in}(x) \cdot \text{out}(\text{sqr}(x)) = \Sigma\{\text{in}(n) \cdot \text{out}(n^2) \mid n \in \mathbf{N}\}.$$

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

Example 14. Consider the receiving party R of the protocol described in the previous section (see Example 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 $V(m)$ holds if, and only if, m is valid. Writing $r_2(x) \cdot s_2(1) \triangleleft V(x) \triangleright r_2(x) \cdot s_2(0)$ for the set

$$\{r_2(m) \cdot s_2(1) \mid m \in M \ \& \ V(m)\} \cup \{r_2(m) \cdot s_2(0) \mid m \in M \ \& \ \text{not } V(m)\},$$

we may denote the receiving party R with the expression

$$\sum_x (r_2(x) \cdot s_2(1) \triangleleft V(x) \triangleright r_2(x) \cdot s_2(0)).$$

Let us now turn to the technicalities.

3.1 Data

We assume that data are given as a model for a first-order language. The following definitions are standard (see, e.g., Chang and Keisler, 1990). A first-order language \mathcal{L} is a collection of symbols with fixed associated arities, subdivided into a collection \mathcal{F} of *function symbols* and a collection \mathcal{R} of *relation symbols*. A *model* M for \mathcal{L} consists of a *domain* M , together with an interpretation function that assigns to each n -ary function symbol f an n -ary function

$$f_M : M^n \rightarrow M,$$

and to each n -ary relation symbol r an n -ary relation

$$r_M \subseteq M^n.$$

In the following example we present a first-order language and a model for it; in the course of this article, we shall return to this model a few times.

Example 15. Consider the language with one binary relation symbol \leq , two binary function symbols $+$ and \cdot , a unary function symbol $-$ and nullary function symbols 0 and 1 . We get a model \mathbf{R} for this language by taking as domain the set \mathbf{R} of real numbers, interpreting \leq as the binary relation $\leq_{\mathbf{R}}$ defined by

$$r_1 \leq_{\mathbf{R}} r_2 \text{ if, and only if, } r_1 \text{ is at most } r_2,$$

and interpreting $+$ as addition of real numbers, \cdot as multiplication, $-$ as taking the additive inverse, and 0 and 1 as the respective

It is convenient to fix, for the remainder of this article, a model \mathbf{D} for a countable, but otherwise arbitrary, first-order language. The domain of \mathbf{D} is denoted by \mathbf{D} . We refer to \mathbf{D} as the *data model*, and to its domain \mathbf{D} as the *data domain*. We also fix a countably infinite set X of (*data*) *variables*. With respect to the language of \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

$$d ::= x \mid f(d, \dots, d), \quad (5)$$

where x is a variable, f is a function symbol of arity n and d, \dots, d is a sequence of length n . Data expressions will be used to specify the parameters of actions.

The set \mathcal{B} of *Boolean expressions* associated with \mathbf{D} is generated by

$$b ::= d \approx e \mid r(d_1, \dots, d_n) \mid \neg b \mid b \vee c, \quad (6)$$

where d and e are data expressions, r is a relation symbol of arity n , and d_1, \dots, d_n are data expressions. We further introduce \top to abbreviate the Boolean expression $x \approx x$ (for some variable $x \in X$), and \perp to abbreviate the Boolean expression $\neg \top$.

A *valuation* is a mapping from the set of variables X into the data domain \mathbf{D} . Let us fix a valuation $\nu : X \rightarrow \mathbf{D}$; we denote by $\bar{\nu}$ its unique homomorphic extension, i.e., its extension to a mapping from \mathcal{D} into \mathbf{D} such that

$$\begin{aligned} \bar{\nu}(x) &= \nu(x) \text{ and} \\ \bar{\nu}(f(d_1, \dots, d_n)) &= f_{\mathbf{D}}(\bar{\nu}(d_1), \dots, \bar{\nu}(d_n)). \end{aligned}$$

Thus, the mapping $\bar{\nu}$ assigns to every data expression d its unique *value under* ν , the element $\bar{\nu}(d)$ of \mathbf{D} . Next, we inductively define the *satisfaction relation* $\mathbf{D}, \nu \models b$, which assigns to every Boolean expression its *truth value under* ν :

- (i) $\mathbf{D}, \nu \models d \approx e$ if, and only if, $\bar{\nu}(d) = \bar{\nu}(e)$;
- (ii) $\mathbf{D}, \nu \models r(d_1, \dots, d_n)$ if, and only if, $r_{\mathbf{D}}(\bar{\nu}(d_1), \dots, \bar{\nu}(d_n))$;
- (iii) $\mathbf{D}, \nu \models \neg b$ if, and only if, $\mathbf{D}, \nu \not\models b$; and
- (iv) $\mathbf{D}, \nu \models b \vee c$ if, and only if, $\mathbf{D}, \nu \models b$ or $\mathbf{D}, \nu \models c$.

If $\mathbf{D}, \nu \models b$ for all valuations ν , then we write $\mathbf{D} \models b$.

Remark 16. In μCRL , data is defined by means of a many-sorted algebraic specification, which must at least include the specification of a sort `Bool` with nullary function symbols \top and \perp . It is usually assumed that \top and \perp denote distinct elements of the sort `Bool`, and that these are, in fact, the only elements of `Bool`. Terms of sort `Bool` are then used to specify the conditions. Apart from

these special assumptions about the sort `Bool`, it is treated as just another data sort, so that, e.g., action symbols may take parameters of sort `Bool`, and also choice quantification can be over the sort `Bool`. Moreover, it is allowed to define functions that take arguments of sort `Bool` and have any other specified data sort as domain; this facilitates, e.g., the specification of an *if-then-else* construct for each data sort.

The notion of data defined here deviates in several ways from the notion of data associated with μCRL specifications.

Firstly, we restrict our attention to a data model that is essentially single-sorted (although one may want to conceive it as a two-sorted data *algebra*, treating relations as functions into a two-element Boolean algebra; cf. the author's dissertation (Luttik, 2002)). This deviation is to simplify notation; our results may straightforwardly be generalised to a many-sorted setting.

Secondly, in our setting conditions are specified by quantifier-free first-order formulas with respect to the data model. Thus we presuppose, apart from the expressions for \top and \perp , the Boolean connectives \neg and \vee , and an equality predicate \approx on the data domain. Although the presence of these additional constructs is not required in a μCRL specification, it is common practice to include them. Our main result in Section 4 is to establish a precise correspondence between universal and existential quantification of first-order logic on the one hand, and choice quantification on the other hand. Presupposing that conditions are specified with quantifier-free first-order formulas, facilitates a convenient formulation of this result.

3.2 The language `pCRL`

Now, suppose that \mathcal{A} is a nonempty countable set of symbols with fixed associated arities; we call the elements of \mathcal{A} *parametrised action symbols*. The set \mathcal{P} of `pCRL expressions` associated with \mathbf{D} and \mathcal{A} is generated by the following grammar:

$$p ::= a(d_1, \dots, d_n) \mid \delta \mid p + p \mid p \cdot p \mid p \triangleleft b \triangleright p \mid \sum_x p \quad (7)$$

where a is a parametrised action symbol of arity n , d_1, \dots, 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:

$$+ < \sum_x < \triangleleft b \triangleright < \cdot ,$$

i.e., $+$ binds weakest and \cdot binds strongest. The construct $\triangleleft b \triangleright$ 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 $\text{FV}(p)$. 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 $\sigma : X \rightarrow \mathcal{D}$ is *correct* for p if, for all $x \in \text{FV}(p)$, 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

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

Let $\vec{x} = x_1, \dots, x_n$ be a sequence of variables, and let $\vec{d} = d_1, \dots, d_n$ 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 $i = 1, \dots, n$, 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 $y \notin \text{FV}(p')$; 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 17. 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 18. Suppose that variables range over the set \mathbf{R} of real numbers. According to our remarks at the beginning of this section, with the expression $\sum_x \text{in}(x)$ we mean the process that inputs an arbitrary real number. This is the infinite sum

$$\sum\{\text{in}(r) \mid r \in \mathbf{R}\}$$

in a generalised basic process algebra with deadlock with for every real number r a process that is denoted with the action symbol $\text{in}(r)$ and that models the action of inputting r . There may not be a **pCRL** expression to denote the process $\text{in}(r)$; e.g., with respect to the language of \mathbf{R} in Example 15, $\sqrt{2}$ is not a data expression, and hence $\text{in}(\sqrt{2})$ is not a **pCRL** expression. Also note that the set of **pCRL** expressions associated with \mathbf{R} is countable, while the **pCRL** expression $\sum_x \text{in}(x)$ refers to the least upper bound of a continuum of alternatives (there is an action $\text{in}(r)$ for every real number $r \in \mathbf{R}$).

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 $\sum\{p[x := d] \mid d \in \mathbf{D}\}$, where $p[x := d]$ is obtained by replacing the free occurrences of x in p by the element $d \in \mathbf{D}$. To get a formalisation that reflects our intuition, we introduce expressions of the form $p[x := d]$ as an auxiliary notion.

The set $\text{Pol}_{\mathcal{D}}(\mathbf{D})$ of *data polynomials* associated with \mathbf{D} consists of the data expressions over the language of \mathbf{D} extended with, for every element $d \in \mathbf{D}$, a new nullary function symbol called the *name* of d . Let us agree to denote the name of an element d of \mathbf{D} by the same symbol d , so that the set $\text{Pol}_{\mathcal{D}}(\mathbf{D})$ is generated by

$$dpol ::= x \mid d \mid f(dpol, \dots, dpol),$$

where x is a variable, d is an element of \mathbf{D} , f is a function symbol of arity n and d_{pol}, \dots, d_{pol} is a sequence of length n (cf. (5)).

Example 19. With respect to the data algebra \mathbf{R} of Example 15, a data expression $d(x_1, \dots, x_n)$ is a polynomial in n indeterminates over \mathbf{R} with natural coefficients, whereas a data polynomial $d_{pol}(x_1, \dots, x_n)$ is a polynomial in n indeterminates over \mathbf{R} with real coefficients. Further 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 $\text{Pol}_{\mathcal{B}}(\mathbf{D})$ of *Boolean polynomials* associated with \mathbf{D} is generated by the grammar in (6) by letting d_1, \dots, d_n range over data polynomials instead of over data expressions. The set $\text{Pol}_{\mathcal{P}}(\mathbf{D})$ of *pCRL polynomials* associated with \mathbf{D} and \mathcal{A} is generated by the grammar in (7) by letting d_1, \dots, d_n range over data polynomials and b over Boolean polynomials. The set $\text{Pol}_{\mathcal{P}}(\mathbf{D})$ is the universe of a generalised algebra similar to generalised basic process algebras with deadlock:

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

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

$$P = \{p[x := d] \mid d \in \mathbf{D}\}, \quad (8)$$

and we define

$$\Sigma P = \Sigma_x p.$$

Remark 20. Examples 18 and 19 illustrate why we have taken ‘pCRL expression’ as the fundamental notion in our language and treat ‘pCRL polynomial’ as auxiliary: we wish to reason about the least upper bound of a continuum of alternatives (e.g., the pCRL expression $\Sigma_x \text{in}(x)$ 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 the data model is 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 model is minimal.

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

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

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 symbols \mathbf{A} and a mapping $\text{act} : \mathbf{A} \rightarrow \mathbf{P}$ that interprets action symbols as elements of \mathbf{P} . The action symbols in \mathbf{A} , we have argued, should be thought of as having a particular structure (see (4)). We have, as we may now observe, assumed that \mathbf{A} consists of a special kind of pCRL polynomial. The elements of \mathbf{A} are of the form $a(d_1, \dots, d_n)$, with $a \in \mathcal{A}$ and $d_1, \dots, d_n \in \mathbf{D}$. Henceforth, we call such polynomials pCRL *actions*. The mapping

$$\text{act} : \mathbf{A} \rightarrow \mathbf{P}$$

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

In accordance with McKenzie *et al.* (1987), we denote by $\text{Sg}(\mathbf{A})$ the subuniverse of $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ generated by \mathbf{A} (i.e., $\text{Sg}(\mathbf{A})$ is the least set that contains \mathbf{A} and is closed under the operations of generalised basic process algebras with deadlock); we define

$$\mathbf{Act}(\mathcal{A}, \mathbf{D}) = \langle \text{Sg}(\mathbf{A}), +, \cdot, \delta, \Sigma \rangle.$$

The \mathbf{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 $\overline{\text{act}} : \text{Sg}(\mathbf{A}) \rightarrow \mathbf{P}$

be the maximal extension of \mathbf{act} to a partial mapping from $\text{Sg}(\mathbf{A})$ to \mathbf{P} that respects the operations of generalised basic process algebras with deadlock. Since \mathbf{A} generates $\text{Sg}(\mathbf{A})$, $\overline{\mathbf{act}}$ is unique.

Definition 21. Let \mathbf{P} be a generalised basic process algebra with deadlock with an associated \mathbf{A} -interpretation \mathbf{act} . We say that \mathbf{P} is *pCRL-complete* with respect to \mathbf{act} if the following closure condition holds for all pCRL polynomials $p(x)$ in one variable:

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

Example 22. 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 \mathbf{act} , can only be satisfied if \mathbf{P} is pCRL-complete. On the other hand, if \mathbf{P} is pCRL-complete, then \mathbf{act} uniquely extends to a homomorphism

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

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 are now going to associate with every valuation ν a homomorphism

$$\llbracket _ \rrbracket_\nu : \mathbf{Pol}(\mathcal{A}, \mathbf{D}) \rightarrow \mathbf{Act}(\mathcal{A}, \mathbf{D}).$$

First, let us agree that from now on $\bar{\nu}$ denotes the unique homomorphic extension of ν to a mapping from the set $\mathbf{Pol}_{\mathcal{D}}(\mathbf{D})$ of data polynomials into \mathbf{D} (this is the obvious extension of our earlier definition of $\bar{\nu}$ on p. 18 so that it assigns a value in \mathbf{D} to every data *polynomial*). Subsequently, let us assume that in the definition of the satisfaction relation on p. 18, d , e and d_1, \dots, d_n range over data polynomials, and that b and c range over Boolean polynomials. We thus get an inductive definition of a satisfaction relation $\mathbf{D}, \nu \models bpol$, which assigns every Boolean *polynomial* its truth value under ν . Then, we can define

$\llbracket _ \rrbracket_\nu$ as follows:

$$\begin{aligned}
\llbracket a(dp_{ol_1}, \dots, dp_{ol_n}) \rrbracket_\nu &= a(\bar{\nu}(dp_{ol_1}), \dots, \bar{\nu}(dp_{ol_n})); \\
\llbracket \delta \rrbracket_\nu &= \delta; \\
\llbracket p + q \rrbracket_\nu &= \llbracket p \rrbracket_\nu + \llbracket q \rrbracket_\nu; \\
\llbracket p \cdot q \rrbracket_\nu &= \llbracket p \rrbracket_\nu \cdot \llbracket q \rrbracket_\nu; \\
\llbracket p \triangleleft bpol \triangleright q \rrbracket_\nu &= \begin{cases} \llbracket p \rrbracket_\nu & \text{if } \mathbf{D}, \nu \models bpol \\ \llbracket q \rrbracket_\nu & \text{if } \mathbf{D}, \nu \not\models bpol; \text{ and} \end{cases} \\
\llbracket \sum_x p \rrbracket_\nu &= \sum \{ \llbracket p[x := d] \rrbracket_\nu \mid d \in \mathbf{D} \}.
\end{aligned}$$

The homomorphism $\llbracket _ \rrbracket_\nu$ maps $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ onto its \mathbf{A} -generated subalgebra. We define the interpretation homomorphism ι_ν generated by the valuation ν as the composition of $\overline{\text{act}}$ and $\llbracket _ \rrbracket_\nu$:

$$\begin{array}{ccc}
\mathbf{Pol}(\mathcal{A}, \mathbf{D}) & & \\
\downarrow \llbracket _ \rrbracket_\nu & \searrow \iota_\nu = \overline{\text{act}} \circ \llbracket _ \rrbracket_\nu & \\
\mathbf{Act}(\mathcal{A}, \mathbf{D}) & \xrightarrow{\overline{\text{act}}} & \mathbf{P}.
\end{array}$$

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 $p \preceq q$, where p and q are pCRL expressions. If $\iota_\nu(p) \leq \iota_\nu(q)$, then we say that ν *satisfies* $p \preceq q$ in \mathbf{P} (notation: $\mathbf{P}, \nu \models p \preceq q$). If every valuation satisfies $p \preceq q$ in \mathbf{P} , then we say that $p \preceq q$ is *valid* in \mathbf{P} , and we write $\mathbf{P} \models p \preceq q$. Note that it follows from the definition of \leq on p. 6 that

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

3.4 pCRL trees

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

$$\text{act} : \mathbf{A} \rightarrow \mathbf{T}_0 = \{ \{ \ell \} \mid \ell \in \mathcal{L} \}$$

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

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

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

Example 23. If we take as data the additive group of integers ordered by \leq , then the \mathbf{pCRL} expression

$$\sum_x r(x)s(x) \triangleleft 0 \leq x \triangleright r(x)s(-x),$$

may be pictured as the tree in Fig. 4.

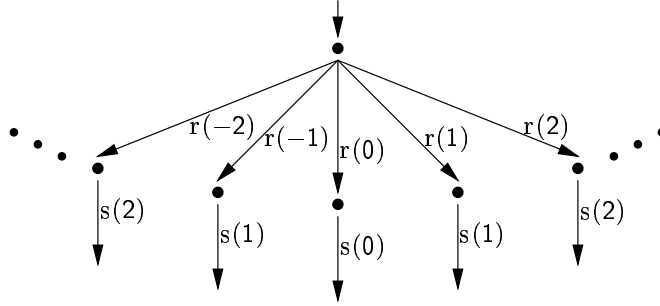


Fig. 4. The transition tree associated with the expression of Example 23.

Their interpretation as transition trees induces an equivalence on the \mathbf{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 \mathbf{pCRL} polynomials modulo this equivalence. First, we need to generalise the notion of congruence. Suppose that ϑ is a congruence of an algebra $\langle \mathbf{P}, +, \cdot, \delta \rangle$ similar to basic process algebras with deadlock. As usual, with \mathbf{p}/ϑ we shall denote the congruence class with respect to ϑ that contains \mathbf{p} , i.e., $\mathbf{p}/\vartheta = \{\mathbf{q} \mid \langle \mathbf{q}, \mathbf{p} \rangle \in \vartheta\}$, and if $\mathbf{P}' \subseteq \mathbf{P}$, then

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

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

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

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

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

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

Now, consider the homomorphism $\overline{\mathbf{act}} : \mathbf{Act}(\mathcal{A}, \mathbf{D}) \rightarrow \mathbf{T}_\kappa(\mathcal{L})$, induced by the

bijection act . The *kernel* of this homomorphism is the relation

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

it is a congruence on $\mathbf{Act}(\mathcal{A}, \mathbf{D})$.⁶ We denote the generalised quotient algebra by $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$, i.e.,

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

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

$$\mathbf{a}(d_1, \dots, d_n) \mapsto \mathbf{a}(d_1, \dots, d_n)/\vartheta.$$

With respect to this \mathbf{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*.

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

$$\sum_{\vec{x}} a \triangleleft b \triangleright \delta \text{ or of the form } \sum_{\vec{x}} ap \triangleleft b \triangleright \delta, \quad (9)$$

where $\sum_{\vec{x}}$ abbreviates the sequence $\sum_{x_1} \cdots \sum_{x_n}$ for a sequence $\vec{x} = x_1, \dots, x_n$ of variables, a is an action expression, b is a Boolean expression and p is a pCRL expression. If in (9) 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 $\sum_{\vec{x}} ap \triangleleft b \triangleright \delta$.

Definition 24. The set \mathcal{T} of *tree forms* is generated by

$$t ::= \delta \mid \sum_{\vec{x}} a \triangleleft b \triangleright \delta \mid \sum_{\vec{x}} at \triangleleft b \triangleright \delta \mid t + t,$$

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

⁶ 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.

Example 25. With the pCRL expression of Example 14 we may associate the tree form

$$\sum_x r_2(x)s_2(1) \triangleleft V(x) \triangleright \delta + \sum_x r_2(x)s_2(0) \triangleleft \neg V(x) \triangleright \delta.$$

With the pCRL expression of Example 23 we may associate the tree form

$$\sum_x r(x)s(x) \triangleleft 0 \leq x \triangleright \delta + \sum_x r(x)s(-x) \triangleleft \neg(0 \leq x) \triangleright \delta;$$

the first simple expression describes the right half of the transition tree in Fig. 4 and that the second simple expression describes the left half.

Proposition 26. *There is a recursive function $\theta : \mathcal{P} \rightarrow \mathcal{T}$ that associates with every pCRL expression a tree form $\theta(p)$ such that $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx \theta(p)$.*

Proof. See Appendix A. □

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

$$t = t_1 + \cdots + t_m + t_{m+1} + \cdots + t_n, \tag{10}$$

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

4 A correspondence between pCRL and first-order logic

The language pCRL is parametrised with the data model \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 Fig. 4 on p. 26 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.

Not surprisingly, 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 symbol, then

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

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

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \begin{cases} p \triangleleft b \triangleright q \approx p & \text{if } \mathbf{D} \models b; \text{ and} \\ p \triangleleft b \triangleright q \approx q & \text{if } \mathbf{D} \not\models b. \end{cases}$$

This means that if it is undecidable whether $\mathbf{D} \models b$, then the validity in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ of pCRL equations is also undecidable. But even if $\mathbf{D} \models b$ is decidable, the validity of a pCRL equation in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ may still be undecidable.

Example 28. 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

$$T(z, x, y) = \top \text{ if, and only if, } y \text{ encodes a computation } ^7 \text{ of } Z \text{ on } x. ^8$$

Kleene's T -predicate is a decidable relation on the natural numbers. Now, consider the pCRL expression

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

If Z has a successful computation on input x , then $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models p(z, x) \approx c$; otherwise $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models p(z, x) \approx \delta$. So $p(z, x) \approx c$ holds in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ 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 $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ is undecidable.

Although existential quantifiers are not part of our definition of Boolean expressions, they pop up when we consider validity in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$. Example 28 shows that the validity in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ of a pCRL equation may be undecidable if there

⁷ 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.

⁸ In the recursion theory literature (e.g., Davis, 1982; Rogers, Jr., 1992) one finds the predicates $T_n(z, x_1, \dots, x_n, y)$, where Z takes the sequence x_1, \dots, x_n as input; we shall only use T_1 and drop the subscript.

exist undecidable first-order assertions about the data. We shall see below that it is necessary and sufficient for the decidability of validity in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$ that *all* first-order assertions about the data are decidable.

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

$$\varphi ::= d \approx e \mid r(d_1, \dots, d_n) \mid \neg \varphi \mid \varphi \vee \psi \mid (\exists x)\varphi, \quad (11)$$

where d_1, \dots, d_n are data expressions, r is a relation symbol of arity n , and x is a variable. We see that the syntax of first-order formulas (11) is obtained by adding, for every variable $x \in X$, an existential quantifier $(\exists x)$ to the syntax of Boolean expressions (cf. (6) on p. 18). The construct $(\exists x)$ is a binder; it binds the variable x in its argument. As such, it comes with a notion of α -conversion; we adopt Convention 17 also for first-order formulas. It is convenient to introduce a few standard abbreviations: if φ and ψ are first-order formulas, then $\varphi \wedge \psi$ abbreviates the first-order formula $\neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi$ abbreviates $\neg\varphi \vee \psi$, $\varphi \leftrightarrow \psi$ abbreviates $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$, and $(\forall x)\varphi$ abbreviates $\neg(\exists x)\neg\varphi$. Furthermore, if $m \geq 1$, $n \geq 0$ and φ_i is a first-order formula for all $m \leq i \leq n$, then $\bigvee_{m \leq i \leq n} \varphi_i$ abbreviates the first-order formula that is inductively given by:

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

Given a valuation $\nu : X \rightarrow \mathbf{D}$, we define the *satisfaction relation* $\mathbf{D}, \nu \models \varphi$ by letting b and c in the definition of the satisfaction relation for Boolean expressions on p. 18 range over first-order formulas, and by adding an extra clause for existential quantification:

- (v) $\mathbf{D}, \nu \models (\exists x)\varphi$ if, and only if, there exists an element $\mathbf{d} \in \mathbf{D}$ such that $\mathbf{D}, \nu[x := \mathbf{d}] \models \varphi$, where $\nu[x := \mathbf{d}]$ is the valuation such that

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

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 $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models p \approx q$. 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) $\phi : \mathcal{P} \times \mathcal{P} \rightarrow \Phi$ 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) $\eta : \Phi \rightarrow \mathcal{P} \times \mathcal{P}$ 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 4.1 (see Theorem 33); the function η will be defined in Section 4.2 (see Theorem 40).

4.1 The definition of ϕ

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

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

We shall then obtain ϕ from ϕ_{\preceq} 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,

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models \delta \preceq u. \tag{12}$$

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,

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t' + t'' \preceq u \text{ if, and only if, } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t' \preceq u, t'' \preceq u. \tag{13}$$

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

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

(The inversion is for convenience of notation; e.g., we have, for a sequence of variables $\vec{x} = x_1, \dots, x_n$, that

$$\iota_\nu(\sum_{\vec{x}} p) = \sum \{ \iota_\nu(p[\vec{x} := \vec{d}]) \mid \vec{d} = d_1, \dots, d_n \in \mathbf{D} \},$$

also if some variable occurs more than once in \vec{x} .)

Lemma 29. *Suppose that $\vec{x} = x_1, \dots, x_n$ is a sequence of variables such that $\{\vec{x}\} \cap \mathbf{FV}(u) = \emptyset$; then*

$$\begin{aligned} \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models \sum_{\vec{x}} t^* \triangleleft b \triangleright \delta \preceq u \text{ if, and only if,} \\ \text{for all sequences } \vec{d} = d_1, \dots, d_n \in \mathbf{D} \\ \mathbf{D}, \nu[\vec{x} := \vec{d}] \models b \text{ implies } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x} := \vec{d}] \models t^* \preceq u. \end{aligned} \quad (14)$$

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

$$\iota_\nu(t) = \sum \{ \iota_\nu(t^*[\vec{x} := \vec{d}]) \mid \vec{d} = d_1, \dots, d_n \in \mathbf{D} \text{ s.t. } \mathbf{D}, \nu[\vec{x} := \vec{d}] \models b \}.$$

(To find the expression on the right-hand side of the equation we have used that $\mathbf{D}, \nu \models b[\vec{x} := \vec{d}]$ if, and only if, $\mathbf{D}, \nu[\vec{x} := \vec{d}] \models b$, which is easily established by induction on the structure of the Boolean expression b .)

If $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t \preceq u$, then, by (GA1),

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

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

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

Hence, if $\mathbf{D}, \nu[\vec{x} := \vec{d}] \models b$, then $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x} := \vec{d}] \models t^* \preceq u$.

Conversely, suppose that

$$\mathbf{D}, \nu[\vec{x} := \vec{d}] \models b \text{ implies } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x} := \vec{d}] \models t^* \preceq u \text{ for all } \vec{d};$$

then, by (GA2),

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x} := \vec{d}] \models t \preceq u.$$

Hence, since $\iota_\nu(u) = \iota_\nu(u[\vec{x} := \vec{d}])$, $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t \preceq u$. \square

So, if $t = t_1 + \dots + t_n$ and t_i is simple for all $1 \leq i \leq n$, then, by (12) and (13), whether a statement of the form $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t \preceq u$ is true is determined by whether statements of the form $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t_i \preceq u$ are true. Furthermore, if

$t_i = \sum_{\vec{x}} t_i^* \triangleleft b \triangleright \delta$, then, by (14), whether the statement $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t_i \preceq u$ is true is determined by whether a statement of the form $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t_i^* \preceq u$ 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 $t^* \preceq u$ in $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$; again we distinguish cases, this time according to the form of u :

Suppose that $u = \delta$; then, by Lemma 7(i),

$$\text{if } t^* = a \text{ or } t^* = at', \text{ then } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \not\models t^* \preceq \delta. \quad (15)$$

Suppose that $u = u' + u''$; then, by Lemma 7(ii),

$$\begin{aligned} &\text{if } t^* = a \text{ or } t^* = at', \text{ then} \\ &\quad \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t^* \preceq u' + u'' \text{ if, and only if,} \\ &\quad \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t^* \preceq u' \text{ or } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t^* \preceq u''. \end{aligned} \quad (16)$$

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

Lemma 30. *Suppose $t^* = a$ or $t^* = at'$, and let $\vec{x} = x_1, \dots, x_n$ be a sequence of variables such that $\{\vec{x}\} \cap \text{FV}(t^*) = \emptyset$; then*

$$\begin{aligned} &\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t^* \preceq \sum_{\vec{x}} u^* \triangleleft b \triangleright \delta \text{ if, and only if,} \\ &\quad \text{there is a sequence } \vec{d} = d_1, \dots, d_n \in \mathbf{D} \text{ such that} \\ &\quad \mathbf{D}, \nu[\vec{x} := \vec{d}] \models b \text{ and } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu[\vec{x} := \vec{d}] \models t^* \preceq u^*. \end{aligned} \quad (17)$$

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

$$\iota_\nu(u) = \sum \{ \iota_\nu(u^*[\vec{x} := \vec{d}]) \mid \vec{d} = d_1, \dots, d_n \in \mathbf{D} \text{ s.t. } \mathbf{D}, \nu[\vec{x} := \vec{d}] \models b \}.$$

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 7(iii) that $\iota_\nu(t^*) \leq \iota_\nu(u)$ if, and only if, there exists $\vec{d} = d_1, \dots, d_n \in \mathbf{D}$ such that $\mathbf{D}, \nu[\vec{x} := \vec{d}] \models b$ and $\iota_\nu(t^*) \leq \iota_\nu(u^*[\vec{x} := \vec{d}])$; the lemma follows. \square

Now, suppose that u is a simple expression, say $u = \sum_{\vec{x}} u^* \triangleleft b \triangleright \delta$ 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 7(v),

$$\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t^* \preceq u^* \text{ if, and only if, } \mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models t^* \approx u^*; \quad (18)$$

if $t^* = at'$ and $u^* = a'u'$, then, by Lemma 7(vi),

$$\mathbf{T}_D(\mathcal{A}), \nu \models t^* \preceq u^* \text{ if, and only if, } \mathbf{T}_D(\mathcal{A}), \nu \models a \approx a', t' \approx u'; \quad (19)$$

if $t^* = at'$ and $u^* = a'$, or $t^* = a$ and $u^* = a'u'$, then, by Lemma 7(iv),

$$\mathbf{T}_D(\mathcal{A}), \nu \not\models t^* \preceq u^*. \quad (20)$$

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

- (1) $\mathbf{D}, \nu \models b$, 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) $\mathbf{T}_D(\mathcal{A}), \nu \models t' \preceq u'$ and $\mathbf{T}_D(\mathcal{A}), \nu \models u' \preceq t'$, where t' and u' are continuations of simple expressions in t and u , respectively.

Associating an appropriate first-order formula with a statement of the first form is easy: just take b . The following definition enables us to associate an appropriate first-order formula with a statement of the second form.

Definition 31. Let $a = a(d_1, \dots, d_m)$ and $a' = a'(e_1, \dots, e_n)$ be action expressions, we define a Boolean expression $a \approx a'$ as follows:

$$a \approx a' = \begin{cases} d_1 \approx e_1 \wedge \dots \wedge d_m \approx e_n & \text{if } a = a' \text{ and } m = n; \text{ and} \\ \perp & \text{otherwise} \end{cases}$$

In the following lemma we prove that our definition is correct.

Lemma 32. *If a and a' are action expressions, then*

$$\mathbf{T}_D(\mathcal{A}), \nu \models a \approx a' \text{ if, and only if, } \mathbf{D}, \nu \models a \approx a'.$$

Proof. Suppose that $a = a(d_1, \dots, d_m)$ and $a' = a'(e_1, \dots, e_n)$; we have

$$\begin{aligned} \mathbf{T}_D(\mathcal{A}), \nu \models a \approx a' & \\ \Leftrightarrow a(\bar{\nu}(d_1), \dots, \bar{\nu}(d_m)) &= a'(\bar{\nu}(e_1), \dots, \bar{\nu}(e_n)) \\ \Leftrightarrow a = a', m = n \text{ and } \bar{\nu}(d_i) &= \bar{\nu}(e_i) \text{ for all } 1 \leq i \leq n \\ \Leftrightarrow a = a', m = n \text{ and } \mathbf{D}, \nu \models d_i &\approx e_i \text{ for all } 1 \leq i \leq n \\ \Leftrightarrow a = a', m = n \text{ and } \mathbf{D}, \nu \models d_1 &\approx e_1 \wedge \dots \wedge d_n \approx e_n \\ \Leftrightarrow \mathbf{D}, \nu \models a \approx a', & \end{aligned}$$

by which the lemma is proved. □

With statements of the third form we are going to deal recursively. First, we associate with every tree form t a natural number $|t|$:

$$\begin{aligned} |\delta| &= 0; & |\sum_{\bar{x}} a \triangleleft b \triangleright \delta| &= 1; \\ |t' + t''| &= |t'| + |t''|; & |\sum_{\bar{x}} at' \triangleleft b \triangleright \delta| &= |t'| + 1. \end{aligned}$$

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 $\mathbf{T}_D(\mathcal{A}), \nu \models t \preceq u$ is equivalent to a first-order combination of expressions of the first two forms.

Algorithm 1 reflects our analysis, except that it applies (14) and (17) without verifying the provisos of Lemmas 29 and 30. Let us say that Algorithm 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 .

Algorithm 1 yields a partial recursive function $\phi_{\preceq} : \mathcal{T}_o \times \mathcal{T}_o \rightarrow \Phi$ 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 17 also denoted by ϕ_{\preceq} ; we have that

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

Since $\mathbf{T}_D(\mathcal{A}), \nu \models p \approx q$ if, and only if, $\mathbf{T}_D(\mathcal{A}), \nu \models p \preceq q$ and $\mathbf{T}_D(\mathcal{A}), \nu \models q \preceq p$, and by Corollary 27, we get that

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

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 $\phi_{\preceq}(\theta_o(t), \theta_o(q)) = \phi_{\preceq}(\theta_o(t + \delta), \theta_o(q))$ for all q .) We obtain a one-one function as follows. Let $\ulcorner _ \urcorner : \mathcal{P} \rightarrow (\omega - \{0\})$ 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 $(\perp)^n$ by $(\perp)^1 = \perp$ and $(\perp)^{n+1} = (\perp)^n \vee \perp$; note that $\mathbf{D}, \nu \models \varphi \vee (\perp)^n$ if, and only if, $\mathbf{D}, \nu \models \varphi$, for all formulas φ . Now, let $\phi : \mathcal{P} \times \mathcal{P} \rightarrow \Phi$ be such that for all p and q

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

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

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

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

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

case

$t = \delta$:

return \top .

$t = \sum_{\vec{x}} a \triangleleft b \triangleright \delta$:

return

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

$t = \sum_{\vec{x}} a \cdot t' \triangleleft b \triangleright \delta$:

compute $\phi_{\preceq}(t', u'_i)$ **for all** $1 \leq i \leq m$;

compute $\phi_{\preceq}(u'_i, t')$ **for all** $1 \leq i \leq m$;

return

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

$t = t' + t''$:

compute $\phi_{\preceq}(t', u)$;

compute $\phi_{\preceq}(t'', u)$;

return $\phi_{\preceq}(t', u) \wedge \phi_{\preceq}(t'', u)$.

end.

Algorithm 1.

Theorem 33. *There exists a one-one recursive function $\phi : \mathcal{P} \times \mathcal{P} \rightarrow \Phi$ such that for all pCRL expressions p and q*

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

4.2 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 $\mathbf{T}_{\mathbf{D}}(\mathcal{A}), \nu \models p \approx q$. Clearly, if φ is quantifier-free, then it is a Boolean expression, so that it may be used as a condition in a conditional.

Lemma 34. *If φ is a quantifier-free first-order formula and c is a closed action expression, then*

$$\mathbf{T}_D(\mathcal{A}), \nu \models c \triangleleft \varphi \triangleright \delta \approx c \text{ if, and only if, } \mathbf{D}, \nu \models \varphi.$$

Proof. If $\mathbf{D}, \nu \models \varphi$, then $\iota_\nu(c \triangleleft \varphi \triangleright \delta) = \iota_\nu(c)$; otherwise $\iota_\nu(c \triangleleft \varphi \triangleright \delta) = \delta$. Since $\iota_\nu(c) \neq \delta$ the lemma follows. \square

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

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

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

Lemma 35. *There exists a recursive function $\pi : \Phi \rightarrow \Phi$ 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). \square

By the above lemma, we may now concentrate on prenex forms. Lemma 34 shows how the matrix of a prenex form can be expressed as a pCRL equation. To deal with the prefix, we shall prove that universal and existential quantifiers can be expressed as transformations on pairs of pCRL expressions. From this, we shall conclude that every prenex form is expressible as a pCRL equation, defining a 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 36. Suppose that $\mathbf{t}_1, \mathbf{t}_2, \mathbf{u}_1$ and \mathbf{u}_2 are trees. We wish to construct trees \mathbf{t} and \mathbf{u} such that $\mathbf{t} = \mathbf{u}$ if, and only if, $\mathbf{t}_1 = \mathbf{u}_1$ and $\mathbf{t}_2 = \mathbf{u}_2$. Let \mathbf{a}_1 and \mathbf{a}_2 be distinct tree actions; we define $\mathbf{t} = \mathbf{a}_1 \cdot \mathbf{t}_1 + \mathbf{a}_2 \cdot \mathbf{t}_2$ and $\mathbf{u} = \mathbf{a}_1 \cdot \mathbf{u}_1 + \mathbf{a}_2 \cdot \mathbf{u}_2$ (see Fig. 5).

By Lemma 7(vi) $\mathbf{a}_1 \cdot \mathbf{t}_1 = \mathbf{a}_1 \cdot \mathbf{u}_1$ if, and only if, $\mathbf{t}_1 = \mathbf{u}_1$, and also, since $\mathbf{a}_1 \neq \mathbf{a}_2$, $\mathbf{a}_1 \cdot \mathbf{t}_1 \neq \mathbf{a}_2 \cdot \mathbf{u}_2$. Hence by Lemma 7(ii) $\mathbf{a}_1 \cdot \mathbf{t}_1 \leq \mathbf{u}$ if, and only if, $\mathbf{t}_1 = \mathbf{u}_1$. Similarly it follows that $\mathbf{a}_2 \cdot \mathbf{t}_2 \leq \mathbf{u}$ if, and only if, $\mathbf{t}_2 = \mathbf{u}_2$, so $\mathbf{t} \leq \mathbf{u}$ if, and only if, $\mathbf{t}_1 = \mathbf{u}_1$ and $\mathbf{t}_2 = \mathbf{u}_2$. By a symmetric argument it also follows that $\mathbf{u} \leq \mathbf{t}$ if, and only if, $\mathbf{t}_1 = \mathbf{u}_1$ and $\mathbf{t}_2 = \mathbf{u}_2$; we get $\mathbf{t} = \mathbf{u}$ if, and only if, $\mathbf{t}_1 = \mathbf{u}_1$ and $\mathbf{t}_2 = \mathbf{u}_2$.

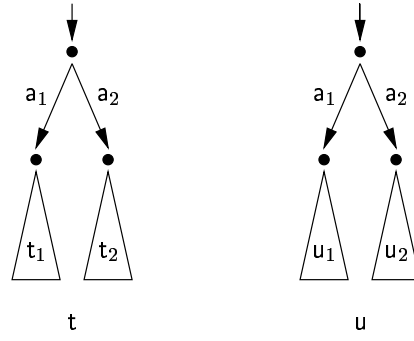


Fig. 5. $t = u$ if, and only if, $t_1 = u_1$ and $t_2 = u_2$.

Let a be a unary parametrised action symbol; we define

$$\begin{aligned} (\forall x)_1 \langle p, q \rangle &= \sum_x a(x)p; \text{ and} \\ (\forall x)_2 \langle p, q \rangle &= \sum_x a(x)q. \end{aligned}$$

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

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

Compare this to the use of a_1 and a_2 in Fig. 5: it follows from $a_1 \neq a_2$ that $a_1 \cdot t_1 \neq a_2 \cdot u_2$.

Lemma 37 (\forall -introduction). *If p and q are pCRL expressions, then*

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

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

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

so by, Lemma 7(iii,vi), for every $\mathbf{d}_1 \in \mathbf{D}$ there exists $\mathbf{d}_2 \in \mathbf{D}$ such that $a(\mathbf{d}_1) = a(\mathbf{d}_2)$ and $\iota_\nu(p[x := \mathbf{d}_1]) = \iota_\nu(q[x := \mathbf{d}_2])$. Since $a(\mathbf{d}_1) = a(\mathbf{d}_2)$ implies $\mathbf{d}_1 = \mathbf{d}_2$, it follows that

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

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

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

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

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

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

Example 38. Suppose that t_1, t_2, u_1 and u_2 are trees. We wish to construct trees t and u such that $t = u$ if, and only if, $t_1 = u_1$ or $t_2 = u_2$. Let a_1, a_2 and c be distinct tree actions; we define $t = c \cdot (a_1 \cdot t_1 + a_2 \cdot u_2) + c \cdot (a_1 \cdot u_1 + a_2 \cdot t_2)$ and $u = c \cdot (a_1 \cdot t_1 + a_2 \cdot u_2) + c \cdot (a_1 \cdot u_1 + a_2 \cdot t_2) + c \cdot (a_1 \cdot t_1 + a_2 \cdot t_2)$ (see Fig. 6).

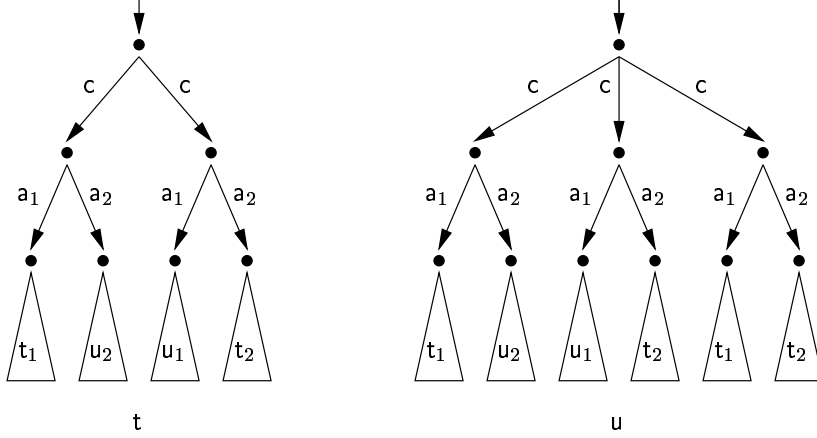


Fig. 6. $t = u$ if, and only if, $t_1 = u_1$ or $t_2 = u_2$.

Clearly, $t \leq u$ and $c \cdot (a_1 \cdot t_1 + a_2 \cdot u_2) + c \cdot (a_1 \cdot u_1 + a_2 \cdot t_2) \leq t$; so $t = u$ if, and only if, $c \cdot (a_1 \cdot t_1 + a_2 \cdot t_2) \leq t$. Hence, by Lemma 7(ii,vi), $t = u$ if, and only if, $t_1 = u_1$ or $t_2 = u_2$.

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

$$\begin{aligned} (\exists x)_1 \langle p, q \rangle &= \sum_x c \cdot (\sum_x a(x)p + a(x)q); \text{ and} \\ (\exists x)_2 \langle p, q \rangle &= (\exists x)_1 \langle p, q \rangle + c \cdot (\sum_x a(x)p). \end{aligned}$$

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 $a(x)q$, while the second occurrence binds the variable x in $a(x)p$. Intuitively, by executing c an instance $a(\mathbf{d}) \cdot q[x := \mathbf{d}]$ of $a(x)q$ is fixed, but from the execution of c it cannot be seen which particular element of \mathbf{D} is selected. Compare this to the function of the tree action c in Fig. 6: by executing c a choice is made between $a_i \cdot t_i$ and $a_i \cdot u_i$ for $i = 1, 2$.

Lemma 39 (\exists -introduction). *If p and q are pCRL expressions, then*

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

Proof. Note that

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

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

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

The proof of the lemma is complete. \square

Theorem 40. *There exists a one-one recursive function $\eta : \Phi \rightarrow \mathcal{P} \times \mathcal{P}$ such that for every first-order formula φ*

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

(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 quantifier-free formula, then

$$P(\varphi) = c \triangleleft \varphi \triangleright \delta \text{ and } Q(\varphi) = c,$$

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 34, 37 and 39 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 $\ulcorner _ \urcorner : \Phi \rightarrow (\omega - \{0\})$ of Φ into the set of positive natural numbers; we define $\eta : \Phi \rightarrow \mathcal{P} \times \mathcal{P}$ by

$$\varphi \mapsto \langle P(\pi(\varphi)) + (\delta)^{\ulcorner \varphi \urcorner}, Q(\pi(\varphi)) \rangle,$$

where $(\delta)^1 = \delta$ and $(\delta)^{n+1} = \delta \cdot \delta^n$ for $n \geq 1$.

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

By Theorem 33 the pCRL theory of \mathbf{D} is one-one reducible to the first-order theory of \mathbf{D} , and Theorem 40 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 41. *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).*

5 The universal fragment: input prefixing

We have proved that 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, Algorithm 1 yields a quantifier-free formula when applied to tree forms t and u without choice quantifiers, and with any quantifier-free formula Lemma 34 associates a pCRL expression without choice quantifiers. In the examples we have always used the choice quantifier to model input. Instead of introducing a choice quantifier, one could add the input mechanism directly, as a special kind of action. This approach is taken in *value-passing CCS* (Milner, 1989; Hennessy, 1991). In that language, actions that model the input of values are distinguished from actions that model the output of values. Suppose that $a \in \mathcal{A}$ is an n -ary parametrised action. An *input prefix* is an expression of the form

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

An *output prefix* is an expression of the form

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

Let us now consider the finite, sequential fragment of value-passing CCS without the internal action τ : the set \mathcal{IO} of *input/output expressions* is defined

by

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

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

Example 42. We consider again the protocol that we took as an example in Sections 2 and 3. In value-passing CCS the sending party (see Example 1) is denoted by

$$S = c_1!m.c_1?x;$$

the receiving party (see Examples 5 and 14, and also Example 25) by

$$R = c_2?x.(V(m) \rightarrow c_2!1 + \neg V(m) \rightarrow c_2!0).$$

To provide the input/output expressions with a semantics, we inductively associate a **pCRL** expression with each of them:

$$\begin{aligned} \text{nil} &\mapsto \delta; \text{ }^9 \\ \text{if } io &\mapsto p, \text{ then } a?x_1, \dots, x_n.io \mapsto \sum_{x_1, \dots, x_n} a(\vec{x})p; \\ \text{if } io &\mapsto p, \text{ then } a!d_1, \dots, d_n.io \mapsto a(d_1, \dots, d_n)p; \\ \text{if } io_1 &\mapsto p_1 \text{ and } io_2 \mapsto p_2, \text{ then } (io_1 + io_2) \mapsto p_1 + p_2; \text{ and} \\ \text{if } io &\mapsto p, \text{ then } (b \rightarrow io) \mapsto p \triangleleft b \triangleright \delta. \end{aligned}$$

We shall generally not make the distinction between input/output expressions and the **pCRL** expressions associated with them; in particular, we shall frequently call a **pCRL** expression p an input/output expression if there is one associated with it. The *input/output theory* of \mathbf{D} is the set of all equations between input/output expressions p and q 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 34.

Lemma 43. *If φ is a quantifier-free first-order formula and c is a closed output action, then $\mathbf{T}_{\mathbf{D}}(\mathcal{A}, \nu \models (\varphi \rightarrow c) \approx c$ 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

⁹ Actually, nil and δ are processes with different properties: nil is a neutral element for parallel composition, while δ is not. But in our setting it only matters that they are both neutral elements for $+$.

also input/output expressions:

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

Hence, we have the following lemma.

Lemma 44 (\forall -introduction). *Suppose that p and q are pCRL expressions. Then*

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

A first-order formula is *universal* if it is in prenex form and all quantifiers in its prefix are universal; we denote by Φ_U the set of universal formulas. From Lemmas 43 and 44 we straightforwardly get a variant of Theorem 40.

Theorem 45. *There exists a one-one recursive function $\eta_{io} : \Phi_U \rightarrow \mathcal{IO} \times \mathcal{IO}$ such that for every universal first-order formula φ*

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

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

The transformation $\langle (\exists x)_1, (\exists x)_2 \rangle$ defined in Section 4.2 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

$$\sum_x c(\sum_x a(x)p + a(x)q)$$

does not occur in the action expression c that immediately follows it. Recall that, intuitively, by executing c an instance $a(\mathbf{d}) \cdot q[x := \mathbf{d}]$ of $a(x)q$ is fixed, but from the execution of c it cannot be seen which particular element of D is selected. We shall see below that if p is an input/output expression, then in $\theta_o(p)$, the ordered tree form associated to p , the construct \sum_x only occurs in a special way.

Definition 46. Let $t = t_1 + \dots + t_m + t_{m+1} + \dots + t_n$ be an ordered tree form with

$$t_i = \begin{cases} \sum_{\vec{x}_i} a_i t'_i \triangleleft b_i \triangleright \delta & 1 \leq i \leq m; \\ \sum_{\vec{x}_i} a_i \triangleleft b_i \triangleright \delta & m < i \leq n. \end{cases}$$

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

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

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

Lemma 47. *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 $a?x_1, \dots, x_n.io$, i.e., suppose that p' is the pCRL expression associated with io and let

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

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

$$\theta_o(p) = \sum_{x_1, \dots, x_n} a(x_1, \dots, x_n)\theta_o(p') \triangleleft \top \triangleright \delta$$

has explicit instantiation.

- (2) Suppose p is associated with $b \rightarrow io$ and let p' be the pCRL expression associated with io ; then

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

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 $\theta_{\text{cnd}}(\theta_o(p'), b)$ has explicit instantiation. It follows that $\theta_o(p)$ has explicit instantiation. \square

We shall now prove that all existential quantifiers can be eliminated from the formula $\phi_{\preceq}(t, u)$ if t and u are ordered tree forms with explicit instantiation.

Theorem 48. *If t and u are ordered tree forms with explicit instantiation, then there exists a universal first-order formula φ such that $\mathbf{D} \models \phi_{\preceq}(t, u) \leftrightarrow \varphi$.*

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

$$((\forall x)\varphi \wedge \psi) \leftrightarrow (\forall x)(\varphi \wedge \psi), \text{ provided that } x \notin \text{FV}(\psi); \quad (21)$$

$$((\forall x)\varphi \vee \psi) \leftrightarrow (\forall x)(\varphi \vee \psi), \text{ provided that } x \notin \text{FV}(\psi); \quad (22)$$

$$(\varphi \rightarrow (\forall x)\psi) \leftrightarrow (\forall x)(\varphi \rightarrow \psi), \text{ provided that } x \notin \text{FV}(\varphi); \quad (23)$$

$$(\exists x)(x \approx d \wedge \varphi) \leftrightarrow \varphi[x := d]. \quad (24)$$

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 $\phi_{\preceq}(t, u) = \top$, which is a universal formula.

If $t = t' + t''$, then by the induction hypothesis $\phi_{\preceq}(t', u)$ and $\phi_{\preceq}(t'', u)$ are equivalent to universal first-order formulas, say

$$\begin{aligned}\phi_{\preceq}(t', u) &\leftrightarrow (\forall x_1) \dots (\forall x_k) \varphi' \text{ and} \\ \phi_{\preceq}(t'', u) &\leftrightarrow (\forall y_1) \dots (\forall y_l) \varphi''.\end{aligned}$$

Without loss of generality we may assume that $x_i \neq y_j$, $x_i \notin \text{FV}(\varphi'')$ and $y_j \notin \text{FV}(\varphi')$, for all $1 \leq i \leq k$ and $1 \leq j \leq l$. Hence by (21)

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

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

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

Then

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

Now consider the subformula $(\exists \vec{x}_i) (b_i \wedge a \approx a_i \wedge \phi_{\preceq}(t', u'_i) \wedge \phi_{\preceq}(u'_i, t'))$; by (22) and (23) it suffices to prove that it is equivalent to a universal formula. By the induction hypothesis $\phi_{\preceq}(t', u'_i)$ and $\phi_{\preceq}(u'_i, t')$ are equivalent to universal formulas, say $(\forall x_1) \dots (\forall x_k) \varphi$ and $(\forall y_1) \dots (\forall y_l) \psi$. If $|\vec{x}_i| = 0$, then the theorem follows immediately from (21), and if $a \approx a_i = \perp$, then the theorem follows since $(\exists x) \perp \leftrightarrow \perp$. Otherwise a and a_i are instances of the same parametrised action and, since u has explicit instantiation, $a_i = a(\vec{x}_i)$. Let $a = a(\vec{d})$, where \vec{d} is a sequence of data expressions with $|\vec{x}_i| = |\vec{d}|$. Then,

$$a \approx a_i = x_{i1} \approx d_1 \wedge \dots \wedge x_{ik} \approx d_k,$$

whence by (24)

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

From this the theorem follows, since by (21) the right-hand side is equivalent to a universal formula. \square

Hence, the universal fragment of the first-order theory of \mathbf{D} is one-one reducible to the input-output theory of \mathbf{D} , and from Lemma 47 and Theorem 48 we get the converse. Hence, the input/output theory of \mathbf{D} and the universal fragment of the first-order theory of \mathbf{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 49. *The input/output theory of \mathbf{D} and the universal fragment of the first-order theory of \mathbf{D} are recursively isomorphic (provided that there exist a closed output action and a parametrised action with arity > 0).*

6 Concluding remarks and related work

Our generalised basic process algebras with deadlock provide an abstract definition of what is an infinite sum in an arbitrary process algebra. Our motivation for having an explicit treatment of infinite sums was the observation of Milner that they may be used to model input over an infinite domain. Furthermore, we have explained how the expressions of a fragment of μCRL may be used to reason formally about the elements of GBPA_δ 's. Thus, we have established a correspondence between μCRL and process algebra, which was until now only suggested by the choice of the constructs.

Value-passing CCS and its successor the π -calculus (Milner, 1999) include an input prefix mechanism, which may be considered an extension of the action prefix mechanism of pure CCS with binding parameters. ACP has, instead of the action prefix mechanism of CCS , a binary operation \cdot for sequential composition. In combination with recursion, the operation for sequential composition enhances the expressive power of the system; e.g., with sequential composition the process behaviour of a stack can be specified in finitely many equations, which is not possible in pure CCS (see Bergstra and Klop, 1984a). On the other hand, since binary sequential composition is associative, the input mechanism cannot be incorporated in ACP by simply giving actions binding parameters, for this yields a scoping ambiguity: if $c?x \cdot (p \cdot q) \approx (c?x \cdot p) \cdot q$, then $c?x$ cannot bind x in q (cf. Baeten and Bergstra, 1994).

This scoping ambiguity is solved in μCRL (and in PSF) with the introduction of an additional construct: the choice quantifier. Thus, in μCRL the binding aspect of the input mechanism is separated from the action. This accounts for greater expressivity compared to value-passing CCS . For instance, in μCRL we can specify

- *restricted input*: if x ranges over natural numbers and the predicate $\text{even}(x)$

holds if, and only if, x is even, then the expression

$$\sum_x \text{in}(x) \cdot p \triangleleft \text{even}(x) \triangleright \delta$$

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

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

$$X(N') = \sum_x \text{out}(x) \cdot X(N' - \{x\}) \triangleleft x \in N' \triangleright \delta$$

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 46).

In μCRL we see a gain in expressive power with respect to CCS by the inclusion as separate constructs of two aspects of the input mechanism. A similar phenomenon occurs in the *fusion calculus* of Parrow and Victor (1998), which has the polyadic π -calculus as a subcalculus. There, a special kind of actions, called *fusion actions*, keep track of certain identifications of names, and input actions do not bind names. The fusion calculus has one binder, which is called the *scope operator*; it is in most circumstances similar to the restriction operator of the π -calculus. Fusion actions and scope operator 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.

We have established a correspondence between the pCRL theory of the data model and its first-order theory. Our Algorithm 1, which associates with every pCRL equation a first-order data formula, produces an open formula when applied to tree forms without choice quantifiers. From this, we conclude that the contribution of choice quantifiers to μCRL corresponds to the contribution of universal and existential quantifiers to first-order logic. A particular instance of our result says that if we take as data the natural numbers with Kleene's T -predicate (see Example 28), then any arithmetical relation may be reduced to a pCRL equation.

Ponse (1996) has investigated the complexity of another fragment of μCRL . He considers data models with only recursive functions and relations, and, with respect to our fragment, 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_1^0 . So, approximately, the contribution of data-parametric recursion to μCRL corresponds to the contribution of universal quantifiers to first-order logic.

We have also proved that a choice quantifier used to express input, behaves as a universal quantifier. Part of this result was already obtained by Hennessy and Lin (1995) for value-passing CCS. They give an algorithm that associates to each pair of finite value-passing processes a universal formula that holds if, and only if, the processes are (symbolic) bisimilar. We have added to this the result that the universal quantifiers introduced by their algorithm cannot be eliminated. More importantly, our result is about a more expressive calculus than value-passing CCS; e.g., it also allows expressions with restricted input and nondeterministic output.

Acknowledgments The author is grateful to Jan Friso Groote for initiating this research and for the many stimulating discussions about the subject of this paper. Further, the author has benefited from discussions with and suggestions by Jan Bergstra, Sjouke Mauw, Julian Rathke, Jaco van de Pol, Alban Ponse, Vincent van Oostrom, Simona Orzan, Piet Rodenburg, Yaroslav Usenko, Mark van der Zwaag, and an anonymous referee.

References

- 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. (1984a). The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings of the 11th International Colloquium on Automata Languages and Programming (ICALP 1984)*, volume 172 of *Lecture Notes in Computer Science*, pages 82–94, New York/Berlin. Springer-Verlag.
- Bergstra, J. A. and Klop, J. W. (1984b). Process algebra for synchronous communication. *Information and Control*, **60**(1–3), 109–137.
- Bergstra, J. A., Ponse, A., and Smolka, S. A., editors (2001). *Handbook of Process Algebra*. North-Holland.

- Bolognesi, T. and Brinksma, E. (1987). An introduction to the ISO specification language LOTOS. *Computer Networks and ISDN System*, **14**(1), 25–59.
- 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.
- Davis, M. (1982). *Computability and Unsolvability*. Dover Publications, Inc.
- Groote, J. F. and Ponse, A. (1995). The syntax and semantics of μ CRL. 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 μ CRL. Report SEN-R0034, CWI. Available from <http://www.cwi.nl/>.
- 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.
- 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.
- Ingólfssdóttir, A. and Lin, H. (2001). A symbolic approach to value-passing processes. In Bergstra *et al.* (2001), chapter 7, pages 427–478.
- Luttik, B. (2002). *Choice Quantification in Process Algebra*. Ph.D. thesis, University of Amsterdam.
- Mauw, S. and Veltink, G. J. (1990). A process specification formalism. *Fundamenta Informaticae*, **XIII**, 85–139.
- McKenzie, R. N., McNulty, G. F., and Taylor, W. F. (1987). *Algebras, Lattices, Varieties — Volume I*. Wadsworth & Brooks/Cole, Monterey, California.
- 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.
- Myhill, J. (1955). Creative sets. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, **1**, 97–108.
- 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. (1996). Computable processes and bisimulation equivalence. *Formal Aspects of Computing*, **8**(6), 648–678.
- Rasiowa, H. and Sikorski, R. (1963). *The mathematics of metamathematics*. Państwowe wydawnictwo naukowe, Warszawa, Poland.
- 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.
- 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.

A Tree Forms

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

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

$$\begin{aligned}
\theta_{\text{seq}}(\delta, t) &= \delta; \\
\theta_{\text{seq}}(\sum_{\vec{x}} a \triangleleft b \triangleright \delta, t) &= \sum_{\vec{x}} a \cdot t \triangleleft b \triangleright \delta && (\{\vec{x}\} \cap \text{FV}(t) = \emptyset); \\
\theta_{\text{seq}}(\sum_{\vec{x}} a \cdot t \triangleleft b \triangleright \delta, u) &= \sum_{\vec{x}} a \cdot \theta_{\text{seq}}(t, u) \triangleleft b \triangleright \delta && (\{\vec{x}\} \cap \text{FV}(u) = \emptyset); \\
\theta_{\text{seq}}(t + u, v) &= \theta_{\text{seq}}(t, v) + \theta_{\text{seq}}(u, v).
\end{aligned}$$

Suppose t and u are tree forms; $\theta_{\text{seq}}(t, u)$ 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 17 also denoted by θ_{seq} .

Lemma 50. $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \theta_{\text{seq}}(t, u) \approx t \cdot u$.

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 $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$; we show that $\iota_{\nu}(\theta_{\text{seq}}(t, u)) = \iota_{\nu}(t \cdot u)$.

If $t = \delta$, then, with an application of (A7),

$$\iota_\nu(\theta_{\text{seq}}(t, u)) = \delta = \delta \cdot \iota_\nu(u) = \iota_\nu(t \cdot u).$$

Suppose $t = \sum_{\vec{x}} a \triangleleft b \triangleright \delta$, with $\vec{x} = x_1, \dots, x_n$. By our assumption on the variables in t and u , $\{\vec{x}\} \cap \text{FV}(u) = \emptyset$, so $u[\vec{x} := \vec{d}] = u$ for all $\vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \in \mathbf{D}$. Hence, by (GA3)

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

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

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

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

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

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

$$\begin{aligned} \iota_\nu(\theta_{\text{seq}}(t, u)) &= \sum \{ \iota_\nu(a[\vec{x} := \vec{d}]) \cdot \iota_\nu(\theta_{\text{seq}}(t', u)[\vec{x} := \vec{d}]) \mid \\ &\quad \vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \in \mathbf{D} \text{ s.t. } \mathbf{D}, \nu \models b[\vec{x} := \vec{d}] \} \\ &= \sum \{ \iota_\nu(a[\vec{x} := \vec{d}]) \cdot \iota_\nu(t'[\vec{x} := \vec{d}]) \mid \\ &\quad \vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \in \mathbf{D} \text{ s.t. } \mathbf{D}, \nu \models b[\vec{x} := \vec{d}] \} \cdot \iota_\nu(u) \\ &= \iota_\nu(t \cdot u). \end{aligned}$$

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

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

The proof of the lemma is complete. \square

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

$$\begin{aligned} \theta_{\text{cnd}}(\delta, b) &= \delta; \\ \theta_{\text{cnd}}(\sum_{\vec{x}} a \triangleleft c \triangleright \delta, b) &= \sum_{\vec{x}} a \triangleleft b \wedge c \triangleright \delta && (\{\vec{x}\} \cap \text{FV}(b) = \emptyset); \\ \theta_{\text{cnd}}(\sum_{\vec{x}} a \cdot t \triangleleft c \triangleright \delta, b) &= \sum_{\vec{x}} a \cdot t \triangleleft b \wedge c \triangleright \delta && (\{\vec{x}\} \cap \text{FV}(b) = \emptyset); \\ \theta_{\text{cnd}}(t + u, b) &= \theta_{\text{cnd}}(t, b) + \theta_{\text{cnd}}(u, b). \end{aligned}$$

Suppose t is a tree form and b is a boolean expression; $\theta_{\text{cnd}}(t, b)$ 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 17 also denoted by θ_{cnd} .

Lemma 51. $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \theta_{\text{cnd}}(t, b) \approx t \triangleleft b \triangleright \delta$.

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 $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$; we show that $\iota_\nu(\theta_{\text{cnd}}(t, b)) = \iota_\nu(t \triangleleft b \triangleright \delta)$.

If $t = \delta$, then $\iota_\nu(\theta_{\text{cnd}}(t, b)) = \delta = \iota_\nu(\delta \triangleleft b \triangleright \delta) = \iota_\nu(t \triangleleft b \triangleright \delta)$.

Suppose that $t = \sum_{\vec{x}} t^* \triangleleft c \triangleright \delta$, with $\vec{x} = x_1, \dots, x_n$. By our assumption on the variables in t and b , $\{\vec{x}\} \cap \text{FV}(b) = \emptyset$, so there are two cases:

- (1) Suppose that $\mathbf{D}, \nu \models b[\vec{x} := \vec{d}]$ for all sequences $\vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \in \mathbf{D}$. Then it holds that $\mathbf{D}, \nu \models b$, which enables us to conclude $\iota_\nu(t \triangleleft b \triangleright \delta) = \iota_\nu(t)$. Furthermore, it also holds that $\mathbf{D}, \nu \models b[\vec{x} := \vec{d}] \wedge c[\vec{x} := \vec{d}]$ if, and only if, $\mathbf{D}, \nu \models c[\vec{x} := \vec{d}]$, which enables us to conclude

$$\begin{aligned} \iota_\nu(\theta_{\text{cnd}}(t, b)) &= \sum \{ \iota_\nu(t^*[\vec{x} := \vec{d}]) \mid \\ &\quad \vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \text{ s.t. } \mathbf{D}, \nu \models b[\vec{x} := \vec{d}] \wedge c[\vec{x} := \vec{d}] \} \\ &= \sum \{ \iota_\nu(t^*[\vec{x} := \vec{d}]) \mid \vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \text{ s.t. } \mathbf{D}, \nu \models c[\vec{x} := \vec{d}] \} \\ &= \iota_\nu(t). \end{aligned}$$

Hence, $\iota_\nu(\theta_{\text{cnd}}(t, b)) = \iota_\nu(t \triangleleft b \triangleright \delta)$.

- (2) Suppose that $\mathbf{D}, \nu \not\models b[\vec{x} := \vec{d}]$ for all sequences $\vec{d} = \mathbf{d}_1, \dots, \mathbf{d}_n \in \mathbf{D}$. Then it holds that $\mathbf{D}, \nu \not\models b$, which enables us to conclude $\iota_\nu(t \triangleleft b \triangleright \delta) = \delta$. Furthermore, it also holds that $\mathbf{D}, \nu \not\models b[\vec{x} := \vec{d}] \wedge c[\vec{x} := \vec{d}]$ for all

sequences $\vec{d} = d_1, \dots, d_n \in \mathbf{D}$, so, with applications of (A6) and (GA2),

$$\begin{aligned} & \iota_\nu(\theta_{\text{cnd}}(t, b)) \\ &= \sum \{ \iota_\nu(t^*[\vec{x} := \vec{d}]) \mid \\ & \quad \vec{d} = d_1, \dots, d_n \text{ s.t. } \mathbf{D}, \nu \models b[\vec{x} := \vec{d}] \wedge c[\vec{x} := \vec{d}] \} \\ &= \sum \emptyset = \delta. \end{aligned}$$

Hence $\iota_\nu(\theta_{\text{cnd}}(t, b)) = \iota_\nu(t \triangleleft b \triangleright \delta)$.

If $t = t' + t''$, then by the induction hypothesis

$$\begin{aligned} \iota_\nu(\theta_{\text{cnd}}(t, u)) &= \iota_\nu(\theta_{\text{cnd}}(t', u)) + \iota_\nu(\theta_{\text{cnd}}(t'', u)) \\ &= \iota_\nu(t' \triangleleft b \triangleright \delta) + \iota_\nu(t'' \triangleleft b \triangleright \delta). \end{aligned}$$

So, if $\mathbf{D}, \nu \models b$, then $\iota_\nu(\theta_{\text{cnd}}(t, u)) = \iota_\nu(t' + t'') = \iota_\nu(t \triangleleft b \triangleright \delta)$; and if $\mathbf{D}, \nu \not\models b$, then, with an application of (A3), $\iota_\nu(\theta_{\text{cnd}}(t, u)) = \delta + \delta = \delta = \iota_\nu(t \triangleleft b \triangleright \delta)$. \square

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

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

Lemma 52. $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \theta_{\text{sum}}(x, t) \approx \sum_x t$.

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 $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$; we show that $\iota_\nu(\theta_{\text{sum}}(x, t)) = \iota_\nu(\sum_x t)$.

If $t = \delta$, then, since $\delta[x := \mathbf{d}] = \delta$ and by (GA1) and (GA2)

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

If t is a simple expression, then $\theta_{\text{sum}}(x, t) = \sum_x t$ by definition.

If $t = t' + t''$, then by the induction hypothesis

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

and

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

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

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

so by (GA2)

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

hence $\iota_\nu(\theta_{\text{sum}}(x, t)) \leq \iota_\nu(\sum_x t)$.

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

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

and similarly,

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

so that

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

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

Now, we define θ as follows:

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

Lemma 53 (Tree forms). *The function $\theta : \mathcal{P} \rightarrow \mathcal{T}$ associates with every pCRL expression p a tree form $\theta(p)$ such that $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models \theta(p) \approx p$.*

Proof. Clearly, $\theta(p)$ is a tree form for every pCRL expression p . To prove that $\mathbf{T}_{\mathbf{D}}(\mathcal{A}) \models p \approx \theta(p)$, we fix an arbitrary valuation ν and an interpretation homomorphism ι_ν generated by ν from $\mathbf{Pol}(\mathcal{A}, \mathbf{D})$ into $\mathbf{T}_{\mathbf{D}}(\mathcal{A})$, 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 $\bar{\nu}(\top) = \top$,

$$\iota_\nu(\theta(p)) = \iota_\nu(p \triangleleft \top \triangleright \delta) = \iota_\nu(p).$$

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

$$\begin{aligned} \iota_\nu(\theta(p)) &= \iota_\nu(\theta(p')) \cdot \iota_\nu(\theta(p'')) && \text{by Lemma 50} \\ &= \iota_\nu(p') \cdot \iota_\nu(p'') = \iota_\nu(p) && \text{by (IH).} \end{aligned}$$

If $p = p' \triangleleft b \triangleright p''$, then

$$\begin{aligned} \iota_\nu(\theta(p)) &= \iota_\nu(\theta(p') \triangleleft b \triangleright \delta) + \iota_\nu(\theta(p'') \triangleleft \neg b \triangleright \delta) && \text{by Lemma 51} \\ &= \iota_\nu(p' \triangleleft b \triangleright \delta) + \iota_\nu(p'' \triangleleft \neg b \triangleright \delta) && \text{by (IH).} \end{aligned}$$

We now distinguish cases: if $\mathbf{D}, \nu \models b$, then $\iota_\nu(\theta(p)) = \iota_\nu(p') + \delta = \iota_\nu(p)$ by (A6); and if $\mathbf{D}, \nu \not\models b$, then $\iota_\nu(\theta(p)) = \delta + \iota_\nu(p'') = \iota_\nu(p)$ by (A1), (A6).

If $p = \sum_x p'$, then by Lemma 52 $\iota_\nu(\theta(p)) = \iota_\nu(\sum_x \theta(p'))$, and from the induction hypothesis we get $\iota_\nu(\theta(p')[x := \mathbf{d}]) = \iota_\nu(p'[x := \mathbf{d}])$ for all $\mathbf{d} \in \mathbf{D}$; hence,

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

This completes the proof of the lemma. □