Acta Informatica 38, 155–234 (2001)

Process algebra with action dependencies

Arend Rensink[1], **Heike Wehrheim**[2]

[1] Department of Computer Science, University of Twente, The Netherlands (e-mail: rensink@cs.utwente.nl)
[2] Abteilung Semantik, Fachbereich Informatik, Universität Oldenburg, Postfach 2503, 26111 Oldenburg, Germany (e-mail: wehrheim@informatik.uni-oldenburg.de)

Received: 19 November 1998 / 18 July 2001

Abstract. In this paper, we present a process algebra with a minimal form of semantics for actions given by *dependencies*. Action dependencies are interpreted in the Mazurkiewicz sense: independent actions should be able to commute, or (from a different perspective) should be unordered, whereas dependent actions are always ordered. In this approach, the process algebra operators are used to describe the *conceptual* behavioural structure of the system, and the action dependencies determine the minimal necessary orderings and thereby the additionally possible parallelism within this structure.

In previous work on the semantics of specifications using Mazurkiewicz dependencies, the main interest has been on linear time. We present in this paper a branching time semantics, both operationally and denotationally. For this purpose, we introduce a process algebra that incorporates, besides some standard operators, also an operator for *action refinement*. For interpreting the operators in the presence of action dependencies, a new concept of *partial termination* has to be developed. We show consistency of the operational and denotational semantics; furthermore, we give a axiomatisation of bisimilarity, which is complete for finite terms. Some small examples demonstrate the flexibility of this process algebra in the design of distributed reactive systems.

Contents

1 Introduction

Process algebras are languages for structurally building specifications out of basic entities called *actions*, using composition operators like sequential composition, choice or parallel composition. The application area of process algebras is the specification and verification of reactive systems. Typical representatives are CCS [49], TCSP [14,35], ACP [7] or LOTOS [11].

Actions in process algebras are usually just names for basic system observables. No further interpretation is given to them; the choice for a particular name of an action does not influence the semantics of a specification. The parallelism or ordering of actions within a system is completely fixed by the composition operators used in the specification. In this paper, we take a different approach. The actions in the process algebra will carry a limited amount of semantic information, in the form of a so-called *dependency relation* among them. Intuitively, actions are dependent if they share some common resource on which only single access at a time is possible. Such a resource can for instance be a variable, a database entry, a channel, a printer or a processor. Given such a dependency relation, the additional information about the actions can be used in the interpretation of the composition operators. The idea is that actions which are using the same resource (are dependent) have to be ordered (since the conflicting accesses to this resource have to be resolved somehow), whereas independent actions never have to be ordered and thus never have to wait for one another to proceed.

The language we introduce here allows to specify the order of execution of system components in a rather abstract way, focussing on the conceptual structure of the system; the dependency relation guarantees that still as much parallelism as possible is achieved. If two interactions of the system in principle occur in a sequential order, but there are some small independent parts, the designer can actually specify them as sequentially composed and still obtain an overlapping of their executions. When writing specifications, the designer does not have to figure out all possible concurrency in order to get the most efficient (maximally parallel), specification; he just has to fix the dependencies.

In this setting, we also take another look at the concept of *action refinement*, in the shape of an operator that is not present in the standard algebras, but which has been the subject of extensive research: for instance, [3,4, 22,69,32,60,70,71]. The action refinement operator enables a designer to decompose abstract actions that are regarded as atomic, i.e., whose execu-

tion is modelled as an indivisible step, into more concrete behaviour that is no longer indivisible, but rather composed of many steps. This can be used as a tool in the top-down design of complex systems. Unfortunately (as noted first in [16]), action refinement cannot be modelled in a standard interleaving framework; indeed, this is the main theme of research in the papers cited above. However, as it turns out, one of the attractive consequences of a global dependency relation as considered in this paper is that it provides sufficient additional information to allow action refinement within an interleaving model (under some assumptions that effectively require that the refinement of actions respects their dependencies).

The idea of action dependencies regulating orderings has been suggested and intensively studied by Mazurkiewicz [46–48] and others (see for instance [26]). The basic concept in Mazurkiewicz' work are *traces*, which are equivalence classes of sequences of actions, factorised by a permutation equivalence: in a sequence, adjacent independent actions may be commuted. System behaviour is described as a set of traces, constituting the possible runs of the system. Thus, traces are essentially a *linear time* model for behaviour: the moments of choice in a behaviour are not represented. A process algebraic, linear time setting with action dependencies has been developed by Janssen, Poel and Zwiers in [39,40]. In particular, they also introduce an operator for action refinement which takes action dependencies into account.

In the present paper, we develop a *branching time* semantics for a process algebra based on action dependencies, containing most of the standard features, such as sequential composition, parallel composition, choice and recursion, as well as action refinement. This finalises previous work reported in [72,63,64]. The language and basic definitions are given in Sect. 2. In Sect. 3, we give a structural operational semantics such that, by the format of the rules, bisimulation equivalence is a congruence. The model generated by this semantics is an ordinary labelled transition system – which, as mentioned above, is surprising, since in the usual approach, this model is not strong enough to be compositional for action refinement. The most innovative feature of the operational semantics is the notion of *partial termination* that was developed to capture the interplay between action dependencies and choice.

Next, in Sect. 4, we give a denotational counterpart, which is *not* based on an interleaving model but instead on an event-based formalism developed in [57]. We have chosen this type of model here because it allows us to use (variations on) standard constructions, especially for action refinement; see, e.g., [22,69,60]. In other words, our denotational model is certainly distinctive enough to capture all relevant behavioural effects of the operators of our algebra, including action refinement. We use a technique from metric semantics (see [21]) for the denotation of recursive behaviour. We then

show that the operational and denotational semantics coincide, i.e., give rise to equivalent (namely, bisimilar) models; therefore, the strength of the denotational model serves as a strong argument in favour of the correctness of the operational semantics.

As a next step, in Sect. 5 we develop an axiomatisation for bisimilarity over our language that is complete for the finite fragment. The main difficulty here was to find suitable axioms for sequential composition, which of the operators of our algebra is the one most influenced by action dependencies. Moreover, the concept of partial termination also complicates the picture. As for the more unusual operator for that action refinement, it turns out that this can be captured by a quite straightforward and small set of axioms, which in fact merely specify some distribution properties.

The paper ends with a few application examples in order to demonstrate the practical usefulness of our approach; see Sect. 6. First, we show that a protocol initially designed as consisting of three successive phases can be implemented, through action refinement, in such a way that the phases partially overlap, controlled by the appropriate action dependencies. Then, we show that this principle can be applied more generally, by formulating a version of the *communications closed layers* law proposed by Elrad and Francez [27] and promoted (in a linear time setting) by Zwiers et al., for instance in [39]. Finally, we also give an example from the field of data bases, showing the decomposition of an atomic query.

2 Language

We start with the introduction of our process algebra and give a first informal discussion of its semantics.

Act denotes a set of actions, ranged over by $a, b, c, \ldots$. The dependencies among these entities are modelled by a *dependency relation* $D \subseteq Act \times Act$ which is reflexive and symmetric. The inverse notion of *independency* is the complement of D: $I = (Act \times Act) \setminus D$. The set of all actions b an action a depends on is called its *dependency class* and is defined as $[a]_D = \{b \mid b\ D\ a\}$. This notion can be extended to sets A of actions by letting $[A]_D$ be $\bigcup_{a \in A}[a]_D$. Similarly the *independence class* of an action is $[a]_I = \{b \mid b\ I\ a\}$, and of a set of actions, $[A]_I = \bigcap_{a \in A}[a]_I$.

To model recursion and thus infinite behaviour of systems, a set of process names $\mathbf{X}$ is used. It is ranged over by X, Y, Z. Each name $X \in \mathbf{X}$ has an implicit associated *alphabet* $\mathcal{A}_X \subseteq Act$. More about alphabets later.

The *specification language* $\mathbf{L}$ consists of all terms B generated by the following productions:

$$B ::= \mathbf{0}_A \mid a \mid B \cdot B \mid B + B \mid B \parallel_A B \mid B[r] \mid X \mid recX.\, B$$

where $a \in Act$, $A \subseteq Act$, $r\colon Act \to \mathbf{L}$ is a refinement function which is the identity almost everywhere, and $X \in \mathbf{X}$. We use $B, C, \ldots, B_1, B_2, \ldots$ to range over $\mathbf{L}$. We refer to the language without refinement as the *flat* fragment of the language, and without recursion as the *finite* fragment. The finite fragment of $\mathbf{L}$ is denoted $\mathbf{L}_{fin}$.

An occurrence of a variable X in a term B is called *bound* if it only occurs in the scope of a rec-operator, otherwise it is *free*. The free variables of B are collected in $fv(B)$. B is called *closed* if it contains no free variables ($fv(B) = \emptyset$), otherwise it is *open*. Refinement functions are assumed always to map to closed terms. The substitution of a term C for a (free) variable X in B is denoted $B\langle C/X\rangle$.

Well-formedness. Without discussing them at this point, we list the additional restrictions imposed upon terms in the course of this paper. We call a term *well-formed* if it satisfies all of these conditions. Unless explicitly stated otherwise, we assume all terms to be well-formed. The set of well-formed terms is denoted $\mathbf{L}^{wf}$.

- Refinement maps only to closed terms; i.e., $fv(r(a)) = \emptyset$ for all $a \in Act$.
- Refinement is strongly D-consistent (see Definition 3.5).
- The alphabet is well-defined; in particular, $\mathcal{A}(B) \subseteq \mathcal{A}_X$ for all sub-terms $recX.\, B$ (see Table 1).
- Recursion is dependently guarded (see Definition 3.7).

Notation. The family $(\mathbf{0}_A)_{A \subseteq Act}$ stands for empty processes, *deadlocked* on the actions in A (see below). We let $\mathbf{0}$ stand for $\mathbf{0}_{Act}$ (complete deadlock), $\mathbf{1}$ for $\mathbf{0}_\emptyset$ (proper termination) and $B_1 \,|||\, B_2$ for $B_1 \parallel_\emptyset B_2$.

Instead of using the operator $recX.\_$, we sometimes equivalently assume that recursive behaviour of processes is specified by means of a set of equations $X_i = B_i$ (i in some finite index set), and when speaking of solutions to recursion terms, we mean solutions to this set of equations.

To avoid brackets in expressions, we fix the following priorities among the operators of the language. The rank of the operators from highest to lowest is: Refinement, sequential composition, parallel composition, choice.

We will often need the set of actions which syntactically occur in a term, called the *alphabet* $\mathcal{A}(B)$ of a term. It is inductively defined in Table 1. Note that this relies on the implicit alphabet of process names, introduced above. The alphabet of recursive terms is only defined under the assumption that the (calculated) alphabet of the body of the recursion is a subset of the (implicit) alphabet of the process name used as a recursion variable. (Note that we have restricted $\mathbf{L}^{wf}$, the set of well-formed terms, to those for which the alphabet *is* well-defined.) Furthermore, we will consider syntactic substitution $B\langle C/X\rangle$ to be defined only if $\mathcal{A}(C) \subseteq \mathcal{A}_X$. It is easy to see that then $\mathcal{A}(B\langle C/X\rangle) \subseteq \mathcal{A}(B)$.

Table 1. Alphabet of a term

$\mathcal{A}(\mathbf{0}_A)$	$:=$	A
$\mathcal{A}(a)$	$:=$	$\{a\}$, where $a \in Act$
$\mathcal{A}(B_1\ op\ B_2)$	$:=$	$\mathcal{A}(B_1) \cup \mathcal{A}(B_2)$, where $op \in \{+, \cdot, \|_A\}$
$\mathcal{A}(B[r])$	$:=$	$\bigcup_{a \in \mathcal{A}(B)} \mathcal{A}(r(a))$
$\mathcal{A}(X)$	$:=$	$\mathcal{A}_X$
$\mathcal{A}(recX.\ B)$	$:=$	$\mathcal{A}_X$ if $\mathcal{A}(B) \subseteq \mathcal{A}_X$

2.1 Discussion of the operators

The language covers most of the standard process algebra operators, such as sequential composition, parallel composition, choice and recursion, and one more rarely used operator: action refinement. These operators, however, will not get quite the interpretation they traditionally have in process algebras. The basic idea is to *conceptually* keep the meaning of the operators, but use the dependency relation to determine the ordering in the occurrence of actions more precisely; especially to find possibilities for concurrency which go beyond the ones specified by the operators in a term. For instance, the phases of a protocol may conceptually follow one another (and are thus specified to occur in a sequential order) while nevertheless there can be some slight overlap and this overlap may be derived via the independencies of the actions in the phases. Independent actions should never have to wait for one another to proceed, even if they are sequentially composed, while dependent actions always have to be ordered somehow, even if composed in parallel. Parallel composition thus never allows simultaneous execution of dependent actions. The interpretation of all our operators has to adhere to this basic principle. In the following, we give an informal description of the intended semantics and describe the differences to the standard approaches.

Actions and recursion. $a \in Act$ describes a process which executes the *action* a and then terminates successfully. Even before a is executed, the process is terminated for actions independent of a.

Process variables and the operator $recX.\ B$ are used to model infinite behaviour of processes; the latter is interpreted by unrolling it to $B\langle recX.\ B/X\rangle$. When developing the operational semantics, we will discuss recursion (and the problems it introduces in our setting) in greater detail.

Sequential composition and termination. Instead of strong sequential composition, action dependencies give rise to a notion of *weak* sequential composition, which we denote "$\cdot$". The crucial point for the semantics of weak

sequential composition is that an ordering of *dependent* actions of the first and second operand has to be achieved, however without introducing unnecessary orderings of *independent* actions. This is modelled by introducing a special notion of termination, which we call *partial termination*. For standard ("strong") sequential composition, a process can either successfully terminate or deadlock and depending on this, a process sequentially following it may or may not start (see Baeten and Van Glabbeek [6]). This concept is now replaced by partial termination: a process can either be terminated with respect to a particular action or not and depending on this, another process sequentially following it may or may not execute this action. As an example, let

$$B_1 = a \cdot b$$
$$B_2 = c \,|||\, d$$

with dependencies $a\ D\ b$, $c\ I\ a$, $c\ I\ b$ and $d\ D\ a$, $d\ I\ b$. B_1 can execute a and afterwards b and is initially already terminated for all actions which are independent of *both* a and b. Thus $B_1 \cdot B_2$ can immediately execute c but not d.

This particular notion of termination is also reflected in the language constants representing empty processes: instead of two constants for complete termination and deadlock (like δ and ε in ACP, see [7]), we need a family of constants $(\mathbf{0}_A)_{A \subseteq Act}$ representing all possible partial terminations: the index A represents the actions on which the process is deadlocked; it is terminated only for actions $b \in Act$ that are independent of all $a \in A$. As an example for empty processes in connection with weak sequential composition: if $a\ D\ b$ then $\mathbf{0}_{\{a\}} \cdot b$ cannot execute any action, whereas if $a\ I\ b$, then $\mathbf{0}_{\{a\}} \cdot b$ can perform b.

Choice. $B_1 + B_2$ denotes the *nondeterministic choice* between B_1 and B_2. Our choice operator is similar to the CCS choice; however, partial termination may resolve choices. The reason for this can best be seen in connection with a sequential composition. As an example consider the term

$$B = (a + b) \cdot c,$$

where a and c are dependent but b and c independent. A process is terminated for an action a if there is one run of it which is independent of a. For instance, the term $a + b$ is terminated for c since $b\ I\ c$. Thus, B may start with c; however, this resolves the choice and only b is left. In fact, if b were still possible after c then the specified order between dependent actions a and c would be violated. Thus, partial termination, like global termination in [6, 7], may resolve choices.

Parallel composition. The family of operators $\{\|_A\}_{A \subseteq Act}$ stands for TCSP-like *parallel composition* with synchronisation on actions within A. In the process $B_1 \|_A B_2$, actions from A may only be executed as *joint* events of B_1 and B_2. In order to respect the action dependencies, we additionally need some ordering of dependent actions of the first and second component. Dependent actions of parallel components have to be executed in a nondeterministically chosen order, whereas independent actions can be executed concurrently. Thus, a kind of mutual exclusion is modelled. The idea is that a parallel composition combines two system components of which we are not interested in a particular order of execution, but these components still may not access the same resource at the same time. As an example: $a \,|||\, b$, $a \; D \; b$, denotes a process that can execute a and b in any order but not in parallel.

Refinement. Action refinement [3,4,52,68,9,42,18,70,9,58] is used to support top-down design of distributed systems. Starting with an abstract specification, step-by-step more concrete specifications are developed by replacing actions by more complex processes. Syntactically, this is formulated by means of a refinement function $r\colon Act \to \mathbf{L}$ describing the replacement of actions by complex processes. We assume that $r(a)$ is unequal to a only for a finite number of actions a; moreover, as noted above, refinement is only well-formed if $r(a)$ is closed for all a.

In contrast to standard action refinement, in our setting, the inheritance of abstract orderings by concrete actions of the refinement is driven by the dependencies between the latter. Therefore, the refinement of ordered abstract actions may result in processes which partially overlap in their execution. This leads to a much more flexible refinement concept.

Example 2.1 Let $B = a \cdot b$ with $a \; D \; b$, and $r\colon a \mapsto a_1 \cdot a_2$ with $a_1 \; D \; a_2$, and $b \mapsto b_1 \cdot b_2$ with $b_1 \; D \; b_2$, such that $a_1 \; D \; b_1$ and $a_2 \; D \; b_2$ but $a_2 \; I \; b_1$ and $a_1 \; I \; b_2$. The only allowed execution of $B[r]$ by standard refinement concepts would be $a_1a_2b_1b_2$, where the entire refinement of b has to wait for a to complete. With dependency-based refinement we get the following, depicted as a partial order. The runs are all possible interleavings. As can be seen, an overlapping execution $a_1b_1a_2b_2$ is allowed.

$$\begin{array}{ccc} a_1 & \to & a_2 \\ \downarrow & & \downarrow \\ b_1 & \to & b_2 \end{array}$$

There are, however, limits to the flexibility of refinement with respect to the action dependencies. In particular, if two actions are dependent, it is natural to expect that their refinements are as well, in an appropriate sense. For instance, it would not be correct if in the above example, both b_i were

to be independent of both a_j. This and similar considerations lead to the concept of *strong D-consistency* of refinement functions, discussed in the next section (Definition 3.5). Note that well-formedness also requires strong D-consistency for all refinement functions.

Some of the ideas on the semantics of process algebra operators in the presence of a dependency relation sketched above have already appeared in previous work; however, always in a linear time setting. Dependency based sequential composition first appeared in the work of Mazurkiewicz [48] (where sequential composition is trace concatenation); in Janssen, Poel and Zwiers [74,41,28] both weak sequential composition (called layer composition) and dependency-based refinement are defined; and in Gaifman [30] weak sequential composition also appears, there called D-local concatenation. The necessity of attaching information about partial termination also arises in the setting of Mazurkiewicz traces, as Diekert has investigated [24, 25]: Concatenation of two *infinite* traces is only possible when the alphabet of the first trace is known.

3 Operational semantics

In this section, we present a structural operational semantics (SOS) in the style of Plotkin [56] for $\mathbf{L}$, which will allow the derivation of a labelled transition system for every closed term $B \in \mathbf{L}$. In the usual way, this will give rise to a labelled transition system modelling the behaviour of the terms of $\mathbf{L}$.

Definition 3.1 *A* labelled transition system *is a tuple* $\langle \Lambda, S, \rightarrow, \iota \rangle$ *such that*

- Λ *is a set of* labels*;*
- S *is a set of* states*;*
- $\rightarrow \subseteq S \times \Lambda \times S$ *is a* transition relation*;*
- $\iota \in S$ *is the* initial state.

The class of all labelled transition systems will be denoted $\mathbf{LTS}$. Throughout this paper, we will have $\Lambda = Act \cup \checkmark_{Act}$, where $\checkmark_{Act} = \{\checkmark_a \mid a \in Act\}$ is a set of *partial termination labels*. Transitions $s \xrightarrow{a} s'$ for $a \in Act$ stand for ordinary action occurrences, whereas $s \xrightarrow{\checkmark_a} s'$ for $\checkmark_a \in \checkmark_{Act}$ denotes the partial termination of s with respect to a ($\in Act$). We will henceforth omit the component Λ.

The operational semantics for $\mathbf{L}$ is given by the rules in Table 2. Below we will show that in order to obtain a well-defined transition system, we have to restrict recursion to *dependently guarded* process variables (Definition 3.7); furthermore, some other sensibility criteria force us to restrict refinement to a subclass of *strongly D-consistent* refinement functions (Definition 3.5). This reduces the language to that of well-formed terms, $\mathbf{L}^{wf}$, introduced in Sect. 2.

Definition 3.2 (operational semantics) *The operational semantics for a term $B \in \mathbf{L}^{wf}$ is the transition system $lts(B) = \langle \mathbf{L}^{wf}, \rightarrow, B \rangle$, where $\rightarrow$ is the smallest set of transitions agreeing with the rules in Table 2.*

Note that we have *not* restricted the operational semantics to closed terms; there is even an operational rule explicitly dealing with process variables. This is mainly for technical convenience in dealing with recursion; see below.

Table 2. Operational semantics of **L**

deadlock	$\dfrac{a \; I \; A}{\mathbf{0}_A \xrightarrow{\surd_a} \mathbf{0}_A}\,\mathrm{R}_1$
action	$\dfrac{}{a \xrightarrow{a} \mathbf{1}}\,\mathrm{R}_2 \qquad \dfrac{a \; I \; b}{b \xrightarrow{\surd_a} b}\,\mathrm{R}_3$
choice	$\dfrac{x \xrightarrow{a} x'}{\begin{array}{l} x + y \xrightarrow{a} x' \\ y + x \xrightarrow{a} x' \end{array}}\,\mathrm{R}_4 \qquad \dfrac{x \xrightarrow{\surd_a} x' \;\; y \xrightarrow{\surd_a}}{\begin{array}{l} x + y \xrightarrow{\surd_a} x' \\ y + x \xrightarrow{\surd_a} x' \end{array}}\,\mathrm{R}_5 \qquad \dfrac{x \xrightarrow{\surd_a} x' \;\; y \xrightarrow{\surd_a} y'}{x + y \xrightarrow{\surd_a} x' + y'}\,\mathrm{R}_6$
sequential composition	$\dfrac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}\,\mathrm{R}_7 \qquad \dfrac{x \xrightarrow{\surd_a} x' \quad y \xrightarrow{\alpha} y' \quad \alpha \in \{a, \surd_a\}}{x \cdot y \xrightarrow{\alpha} x' \cdot y'}\,\mathrm{R}_8$
parallel composition	$\dfrac{x \xrightarrow{a} x' \quad a \notin A}{\begin{array}{l} x \parallel_A y \xrightarrow{a} x' \parallel_A y \\ y \parallel_A x \xrightarrow{a} y \parallel_A x' \end{array}}\,\mathrm{R}_9 \qquad \dfrac{x \xrightarrow{\alpha} x' \quad y \xrightarrow{\alpha} y' \quad \alpha \in A \cup \surd_{Act}}{x \parallel_A y \xrightarrow{\alpha} x' \parallel_A y'}\,\mathrm{R}_{10}$
refinement	$\dfrac{x \xrightarrow{a} x' \quad r(a) \xrightarrow{b} y}{x[r] \xrightarrow{b} y \cdot x'[r]}\,\mathrm{R}_{11} \qquad \dfrac{x \xrightarrow{\surd_a} x'}{x[r] \xrightarrow{\surd_a} x'[r]}\,\mathrm{R}_{12}$
variables	$\dfrac{a \; I \; \mathcal{A}_X}{X \xrightarrow{\surd_a} X}\,\mathrm{R}_{13}$
recursion	$\dfrac{y \xrightarrow{\alpha} y' \quad \alpha \notin \{\surd_a \mid a \; I \; \mathcal{A}_X\}}{recX.\, y \xrightarrow{\alpha} y' \langle recX.\, y / X \rangle}\,\mathrm{R}_{14} \qquad \dfrac{a \; I \; \mathcal{A}_X}{recX.\, y \xrightarrow{\surd_a} recX.\, y}\,\mathrm{R}_{15}$

The main difference with standard process algebra lies in the treatment of *termination*. While usually a term can be either terminated or not terminated, weak sequential composition needs a more general concept of *partial termination*, i.e. termination with respect to certain actions. The concept of partial termination lies at the heart of all non-standard rules of our SOS semantics.

- The constant $\mathbf{0}_A$ stands for an empty process deadlocked on actions in A, thus it is terminated for all actions independent of A.
- An action $a \in Act$ is terminated for all independent actions $b \in Act$, i.e., such that $a\ I\ b$.
- Like ordinary complete termination used in languages with sequential composition (see for instance [6]), partial termination may resolve choices.[1] Note the *negative premise* in the partial termination rules for choices. Negative premises are known to be potentially troublesome (see Groote [34] and Van Glabbeek [67]); indeed, we have to restrict recursion to dependently guarded variables to avoid problems (see below).
- In a weak sequential composition, the second operand may start executing actions when the first operand is terminated for them. Unlike strong sequential composition, such activities of the second operand do not discard the first operand.
- Action refinement and recursion require a more extensive discussion; see below.

3.1 Action refinement

As shown in Example 2.1, dependency-based action refinement allows the concurrent execution of independent parts of the refinements of actions that are themselves (on the abstract level) dependent and hence ordered. However, as mentioned before, the allowed overlap is subject to some limitations due to the intuition that the action and its refinement should still describe the same entity, and hence their dependencies with respect to other actions should be consistent.

- Dependencies should be inherited from the abstract to the concrete level to some degree. If two abstract actions are dependent, some dependency should still exist after refinement: in particular, the refinement $r(a)$ of a given action a should not be (partially) terminated for another, dependent action $b\ D\ a$. Technically, $a\ D\ b$ should imply $r(a) \stackrel{\surd b}{\not\longrightarrow}$.
- Independency should also be inherited. Since an abstract action does not change during an independent activity, its refinement should also be completely unaffected. That is, $a\ I\ b$ should imply $\mathcal{A}(r(a))\ I\ b$.

A refinement function is called *D-consistent* if it satisfies both of these criteria. A similar property was defined in [39]. Note that it follows that for all $a \in Act$, $[a]_I = [\mathcal{A}(r(a))]_I$ and $r(a) \stackrel{\surd b}{\longrightarrow} B$ implies $b\ I\ \mathcal{A}(r(a))$; indeed, these two properties form a sufficient condition for D-consistency.

[1] In fact, ordinary termination does not resolve choices in all process algebras, for instance Aceto and Hennessy [3] define choice terms to be terminated only if both operands are.

Example 3.3

- If $a\ D\ b$, $a_1\ D\ b$ and $a_2\ I\ b$ such that $r\colon a \mapsto a_1 + a_2$, then $r(a) \xrightarrow{\surd_b}$; hence r does not preserve the dependence of a and b in the required sense, meaning that r is not D-consistent.
- If $a\ I\ b$, $a_1\ I\ b$ and $a_2\ D\ b$ such that $r\colon a \mapsto a_1 + a_2$, then $r(a) \xrightarrow{\surd_b} B$ with $B = a_1 \neq r(a)$; hence r does not preserve the independence of a and b in the required sense, meaning that r is not D-consistent.
- If r is actually a *renaming* function, that is, $r(a) \in Act$ for all $a \in Act$, then r is D-consistent if and only if $a\ D\ b \iff r(a)\ D\ b$ for all $a, b \in Act$.

Rule R_{11} in Table 2 specifies the execution of actions from an r-image: if the abstract system can execute some action a and the refinement of a can start with some action b, then b is also possible for the refined system. Afterwards the refinement of a may proceed, but also new refinements may start if independent of the remaining a-refinement. To come back to Example 2.1, the overlapping execution of the refinement of a and b is thus derivable:

$$(a \cdot b)[r] \xrightarrow{a_1} a_2 \cdot b[r] \xrightarrow{b_1} a_2 \cdot b_2 \cdot \mathbf{1}[r] \xrightarrow{a_2} \mathbf{1} \cdot b_2 \cdot \mathbf{1}[r] \xrightarrow{b_2} \mathbf{1} \cdot \mathbf{1} \cdot \mathbf{1}[r]$$

Rule R_{12} is straightforward, and reflects the intuition behind D-consistency: If the abstract system x is terminated w.r.t. an action a, then the refined system is also terminated for a.

Unfortunately, a complication occurs as soon as we reconsider the original motivation behind the dependency relation. Intuitively, we would at least expect the Mazurkiewicz property of partial commutativity to hold; that is, if $B \xrightarrow{a}\xrightarrow{b}$ for $a\ I\ b$, we also expect $B \xrightarrow{b}\xrightarrow{a}$ (see [48]). As the following example shows, this property can be destroyed by certain refinement functions.

Example 3.4 Consider $B = a \cdot b$ with $a\ D\ b$, and $r\colon a \mapsto a$, $b \mapsto b_1 \cdot b_2$ such that $a\ I\ b_1$ and $a\ D\ b_2$. Note that r is D-consistent. $B[r]$ has the following operational behaviour:

$$B[r] \xrightarrow{a} \mathbf{1} \cdot (\mathbf{1} \cdot b)[r] \xrightarrow{b_1} \mathbf{1} \cdot b_2 \cdot (\mathbf{1} \cdot \mathbf{1})[r] \xrightarrow{b_2} \mathbf{1} \cdot \mathbf{1} \cdot (\mathbf{1} \cdot \mathbf{1})[r]$$

However, $B[r] \not\xrightarrow{b_1}$; hence the partial commutativity of traces does not hold. The reason is that the only initial action of B is a, and therefore $B[r]$ cannot start with an action not coming from $r(a)$.

The problem demonstrated by this example can be traced to a feature of the refinement function r: although $a\ D\ b$, there is an initial action of $r(b)$ for which $r(a)$ is terminated, or in other words (due to D-consistency) that is independent of a. Disallowing this kind of situation is necessary and sufficient to guarantee partial commutativity. For this purpose we strengthen

the notion of D-consistency and require all well-formed terms to satisfied this strengthened requirement.

Definition 3.5 (strong D-consistency) *A refinement function $r\colon Act \to \mathbf{L}$ is called* strongly D-consistent *if it satisfies the following properties for all $a, b \in Act$:*

- *$a\ D\ b$ implies (i) $r(a) \xrightarrow{\checkmark_b}\!\!\!\!\!\!\rightarrow$, and (ii) $a\ D\ c$ for all $r(b) \xrightarrow{c}$;*
- *$a\ I\ b$ implies $a\ I\ \mathcal{A}(r(b))$.*

(The difference with D-consistency lies in the second condition.) The necessity of strong D-consistency follows from the fact that an example along the lines of Example 3.4 can always be constructed for a refinement function that is not strongly D-consistent. The sufficiency, i.e., the fact that for all $a\ I\ b$, $B \xrightarrow{a}\xrightarrow{b}$ implies $B \xrightarrow{b}\xrightarrow{a}$ if all refinement functions in B are strongly D-consistent, is a consequence of Proposition 3.16 below.

It is worth noting that the operational refinement rules are simpler by far than the ones obtained in other approaches. For instance, we do not rely on auxiliary operators of any kind and do not have to enhance the labels of the transition relation, as is for instance done in [22,60,15] for modelling standard action refinement. Of course, this is achieved at the cost of imposing strong D-consistency.

3.2 Recursion

The operator $recX.\ \_$ looks quite standard and at first sight one does not expect any surprises from it. However, the situation is complicated by the negative premise in the termination rules of the choice operator (R_5 and R_6 of Table 2). The standard operational rule for recursion is the following (with $\alpha \in \Lambda$):

$$\frac{y\langle recX.\ y/X\rangle \xrightarrow{\alpha} y'}{recX.\ y \xrightarrow{\alpha} y'} \tag{1}$$

However, the following example shows that this is not satisfactory for the derivation of the partial terminations:

Example 3.6 Assume $\mathcal{A}_X = \{a, b\}$ and $B = recX.\ a \cdot X + b$.

1. First consider $a\ I\ c$ and $b\ I\ c$. Let us consider the partial c-termination for B. Intuitively we should get $B \xrightarrow{\checkmark_c} B$. This is however not derivable since this would involve an infinite unfolding of the recursion.
2. Now consider a different dependency relation: $a\ I\ c$ and $b\ D\ c$. Even if we could derive that B is terminated for c, it would not be easy to determine B' such that $B \xrightarrow{\checkmark_c} B'$; it is certainly not equal to B, since the action b should not be possible anymore.

3. Finally, consider $a \ D \ c$ and $b \ I \ c$. In this case, there is no problem deriving the expected termination $B \xrightarrow{\surd_c} b$, since $a \cdot X \xrightarrow{\surd_c}\!\!\!\!\!\!\rightarrow$ is already clear from $a \xrightarrow{\surd_c}\!\!\!\!\!\!\rightarrow$; X does not have to be tested with respect to its a-termination.

The problem is essentially due to the fact that the variable X in the body of the recursive term B is tested again in the course of deriving the transitions of B; in combination with the negative premise in R_5, this gives rise to circular reasoning. Since the negative premise is very much inherent in our approach, the only way to avoid problems of this kind is to restrict ourselves to terms where this kind of circular reasoning cannot occur. A standard way to achieve this is to replace (1) by

$$\frac{y \xrightarrow{\alpha} y'}{recX.\, y \xrightarrow{\alpha} y'\langle recX.\, y/X\rangle} \tag{2}$$

(see for instance [5]). This has the advantage that the source term of the transition is syntactically simpler than the source term of the conclusion; since this is already true of all other operational rules, it follows that there can be no infinite proofs of a positive transition; hence negative transitions are unambiguously decidable. (In terms of Groote [34], a stratification trivially exists.)

In general, however, this alternative rule may limit the derivable transitions, since the source term of the premise now contains a free variable, for which (usually) no transitions are derivable. For instance, if $a \ I \ b$ then $recX.(a \cdot X + b) \xrightarrow{b} a \cdot \mathbf{1}$ can be derived using (1) but not using (2). In order to avoid this effect, one simultaneously restricts recursion to *guarded* terms. In general, a free variable of term is called guarded if it only occurs in so-called *sleeping positions* (see Vaandrager [66]), where a sleeping position of a given operator is one which is not tested by the rules of the operational semantics. The effect is that, at least for the the initial transitions of the term, it makes no difference what is substituted for a guarded variable.

Unfortunately, no operator of our language has a proper sleeping position. In particular, the second position of a weak sequential composition is not a sleeping position, since it is tested by Rule R_8. For that reason, X cannot be considered guarded in the sub-term $a \cdot X$ of B in Example 3.6 and of $recX.\,(a \cdot X + b) \xrightarrow{b} a \cdot \mathbf{1}$ above. It follows that the usual notion of guardedness is not directly applicable. Our solution is to define an alternative notion of *dependent guardedness*, in combination with Rule R_{13} which actually defines some (termination) transitions for free process variables.

Definition 3.7 (dependent guardedness) *Let* $B \in \mathbf{L}$.

- X *is called dependently guarded in* B *if every free occurrence of* X *is within the operand* B_2 *of a subterm* $B_1 \cdot B_2$ *of* B *such that* $B_1 \xrightarrow{\checkmark_a}$ *implies* $a \; I \; \mathcal{A}(B_2)$.
- B *is called dependently guarded if for all subterms* $recX.\, C$ *of* B, X *is dependently guarded in* C.

Note that in the condition $B_1 \xrightarrow{\checkmark_a}$, it is possible that $X \in fv(B_1)$; hence the definition relies on the fact that the operational semantics is defined for open terms. For instance, X is dependently guarded in $(a \cdot X) \cdot X$ iff $[a]_I = [\mathcal{A}_X]_I$.

Dependent guardedness may alternatively be characterised inductively on the structure of terms, such that X is dependently guarded in all terms B (except for $B = X$) if it is dependently guarded in all operands of B; and moreover, X is also dependently guarded in $B_1 \cdot B_2$ if X is dependently guarded in B_1 and $B_1 \xrightarrow{\checkmark_a}$ implies $a \; I \; \mathcal{A}(B_2)$.

In principle, this solves the problems associated with recursion. However, (2) has the annoying consequence of *always* unfolding recursive terms, even to derive termination transitions of completely independent actions. It therefore generates non-finite-state models even for quite harmless processes; e.g., if $\mathcal{A}_X = \{a\}$ and $a \; I \; b$ then

$$recX.\, a \cdot X \xrightarrow{\checkmark_b} a \cdot recX.\, a \cdot X \xrightarrow{\checkmark_b} a \cdot a \cdot recX.\, a \cdot X \xrightarrow{\checkmark_b} \cdots$$

This effect is avoided by disallowing (2) for $\checkmark_a$-transitions with $a \; I \; \mathcal{A}_X$, and adding a rule stating that $recX.\, B \xrightarrow{\checkmark_a} recX.\, B$ for such transitions: Rules R_{14} and R_{15} of Table 2, respectively.

3.3 Properties of the operational semantics

We first clarify the relation between the alphabet of a term and its operational semantics: all a-transitions belong to the alphabet, which, moreover, can only grow smaller during execution (Clause 1); $\checkmark_a$-transitions reduce the alphabet of a term to (at most) the actions independent of a (Clause 2); and every term is $\checkmark_a$-terminated for every a that is completely independent of its alphabet, without being affected in any way (Clause 3). The proof is a straightforward induction on the structure of B, here omitted.

Proposition 3.8 (alphabet) *Let* $B \in \mathbf{L}^{wf}$.

1. *If* $B \xrightarrow{a} B'$, *then* $\mathcal{A}(B') \cup \{a\} \subseteq \mathcal{A}(B)$;
2. *If* $B \xrightarrow{\checkmark_a} B'$, *then* $\mathcal{A}(B') \subseteq \mathcal{A}(B) \cap [a]_I$.
3. $B \xrightarrow{\checkmark_a}$ *if and only if* $a \; I \; \mathcal{A}(B)$.

The following proposition provides evidence that the notion of dependent guardedness is in some sense correct. The notion of correctness is the aforementioned property that a dependently guarded variable is unable to influence the initial transition of a term. The proposition captures the relation between syntactic substitution, operational semantics and dependent guardedness; again, it crucially relies on the operational semantics of open terms.

Proposition 3.9 (substitution) *Let $B, C \in \mathbf{L}^{wf}$ with $\mathcal{A}(C) \subseteq \mathcal{A}_X$.*

1. *If $B \xrightarrow{\alpha} B'$, then $B\langle C/X\rangle \xrightarrow{\alpha} B'\langle C/X\rangle$.*
2. *If $B\langle C/X\rangle \xrightarrow{\alpha} B'$ and X is dependently guarded in B, then $B' = B''\langle C/X\rangle$ for some B'' such that $B \xrightarrow{\alpha} B''$.*

The proof can be found in Appendix A. As an added bonus, dependent guardedness of X in B guarantees that $X = B$ has a unique solution (up to bisimulation); see Theorem 3.15 below.

Diamond closure. In the presence of a dependency relation $D \subseteq Act \times Act$, it is often required that transition systems satisfy some *diamond closure* properties (see [26]); e.g., for all $a\ I\ b$:

1. If $s \xrightarrow{a} s' \xrightarrow{b} s''$, then $s \xrightarrow{b} s'_0 \xrightarrow{a} s''$ for some s'_0;
2. If $s \xrightarrow{a} s'$ and $s \xrightarrow{b} s''$, then $s' \xrightarrow{b} s''_0$ and $s'' \xrightarrow{a} s''_0$ for some s''_0.

Our operational semantics satisfies neither of the above properties; the first, however, can be recaptured as soon as we interpret the transition system modulo bisimulation (see below).

Example 3.10

1. Consider $B = a \,|||\, b$ and $r\colon\ a \mapsto a_1 \cdot a_2,\ b \mapsto b_1 \cdot b_2$, where all a-actions are independent of all b-actions. $B[r]$ displays the following operational behaviour:

$$B[r] \xrightarrow{a_1} a_2 \cdot (\mathbf{1} \,|||\, b)[r] \xrightarrow{b_1} a_1 \cdot b_2 \cdot (\mathbf{1} \,|||\, \mathbf{1})[r]$$
$$B[r] \xrightarrow{b_1} b_2 \cdot (a \,|||\, \mathbf{1})[r] \xrightarrow{a_1} b_2 \cdot a_2 \cdot (\mathbf{1} \,|||\, \mathbf{1})[r]$$

   Although the (independent) actions a_1 and b_1 can indeed be executed in either order, as required by the first diamond closure property, the resulting end states are not the same —even though they are bisimilar, as we will see below.
2. The second diamond closure property is circumvented quite easily by specifying a choice between independent actions: If $a\ I\ b$, then $B = a + b$ is a well-formed term such that $B \xrightarrow{a} \mathbf{1}$ and $B \xrightarrow{b} \mathbf{1}$, but $\mathbf{1} \not\xrightarrow{a}$ and $\mathbf{1} \not\xrightarrow{b}$.

Partial termination. Diamond closure applies to non-termination transitions; partial termination has its own logic. First of all, it should not come as a surprise that termination is deterministic: although choices may be resolved in the course of a termination transition, there is only *one way* in which they can be resolved.

On the other hand, due to the fact that termination may resolve choices, it is not necessarily true that a term B that is partially terminated for either a or b can also terminate for *both* a and b in succession.

Example 3.11 Assume $a\ I\ b$ and let $B = a + b$. It follows that $B \xrightarrow{\surd_a} b$ and $B \xrightarrow{\surd_b} a$, hence B can terminate for *either* a or b. However, $B \xrightarrow{\surd_a} \xrightarrow{\surd_b}$ does *not* hold, hence B cannot terminate for *both* a and b.

If, however, a term can still display a certain activity after partial termination, then that activity and the termination could as well be reversed; that is, there is a partial commutation in the sense that $B \xrightarrow{\surd_a} \xrightarrow{\alpha} B'$ implies $B \xrightarrow{\alpha} \xrightarrow{\surd_a} B'$. If α itself is actually also a termination transition, then of course the commutation is total.

The properties discussed above are formalised in the following proposition. The proof is a straightforward induction on the term structure, here omitted.

Proposition 3.12 (termination) *Let* $B \in \mathbf{L}^{wf}$.

1. *If* $B \xrightarrow{\surd_a} B'$ *and* $B \xrightarrow{\surd_a} B''$, *then* $B' = B''$.
2. *If* $B \xrightarrow{\surd_a} \xrightarrow{\alpha} B'$, *then* $B \xrightarrow{\alpha} \xrightarrow{\surd_a} B'$.

3.4 Bisimulation

Transition systems as a semantic model for process terms are usually too informative: they distinguish processes that one would normally consider as being equal, such as, for instance, $recX.a \cdot X$ and $recX.a \cdot a \cdot X$. Hence, as a second step an equivalence notion is introduced which additionally equates some processes with distinct transition systems. The standard equivalence notion for transition systems is *bisimulation* [54].

Definition 3.13 (bisimulation) *Let* $T_i = \langle S_i, \rightarrow, \iota_i \rangle$, $i = 1, 2$, *be labelled transition systems.* T_1 *and* T_2 *are* bisimilar *(*$T_1 \sim T_2$*) if there exists a relation* $\rho \subseteq S_1 \times S_2$ *such that* $(\iota_1, \iota_2) \in \rho$ *and whenever* (s_1, s_2) *is in* ρ, *then for all* $\alpha \in \Lambda$

1. $s_1 \xrightarrow{\alpha} s_1'$ *implies* $\exists s_2' : s_2 \xrightarrow{\alpha} s_2'$ *and* $(s_1', s_2') \in \rho$ *and*
2. $s_2 \xrightarrow{\alpha} s_2'$ *implies* $\exists s_1' : s_1 \xrightarrow{\alpha} s_1'$ *and* $(s_1', s_2') \in \rho$.

As usual, two terms are called bisimilar if their corresponding transition systems are.

Congruence. An immediate question arising whenever a language's behavioural model is to be interpreted up to some equivalence relation is whether the semantics of the language remains well-defined; in other words, whether the equivalence is a *congruence* with respect to the operators of the language. For $\mathbf{L}^{wf}$ and $\sim$, we have

Theorem 3.14 (congruence of bisimulation) *Bisimulation is a congruence for all operators of* $\mathbf{L}^{wf}$ *(including recursion).*

The proof can be found in Appendix A (Page 210). For all operators except recursion, the proof relies on the so-called *GSOS format* of [10]; to prove congruence of recursion, the *up-to* technique used in [49] can be applied, since our rules contain no look-ahead (see [61]).

In particular, the result that bisimulation is a congruence for (dependency-based) refinement is interesting, since the fact that it does *not* hold for standard refinement (see [16]) has been the starting point of almost all papers on action refinement cited before. Again, of course, this fact comes at the price of restricting action refinement to strongly D-consistent refinement functions.

The congruence result also implies that in principle it is possible to develop an equational proof system for bisimulation over $\mathbf{L}^{wf}$. In Sect. 5 we indeed develop such a (sound and complete) proof system.

Unique fixpoints. Using the congruence result, we can now also establish that, in $\mathbf{L}^{wf}$, recursion yields unique fixpoints up to bisimilarity. This is formulated in the following theorem. Apart from being interesting in its own right, this result is important in the proof of correspondence of the operational and denotational semantics, in the next section.

Theorem 3.15 (unique fixpoints) *If* $B \in \mathbf{L}^{wf}$ *with* $fv(B) \subseteq \{X\}$*, then* $recX.\, B$ *is the unique solution of* $X = B$ *in* $\mathbf{L}^{wf}$ *modulo* $\sim$.

The proof can be found in Appendix A (Page 210). It is important to note that the proof does not use the concrete definition of $\mathbf{L}^{wf}$ but only the properties established in Proposition 3.9 and Theorem 3.14. Hence the theorem remains valid if we extend the language in such a way that these properties are not violated. This fact is used in the proof of correspondence of the operational and denotational semantics, in the next section.

Diamond closure revisited. Finally, is is noteworthy that $\mathbf{L}^{wf}$ satisfies a weaker form of the first diamond closure property discussed above. Let us call a transition system *partially commutative up to bisimulation* if whenever $a\ I\ b$ and $s \xrightarrow{a} \xrightarrow{b} s'$ there exists some s'' such that $s \xrightarrow{b} \xrightarrow{a} s''$ and $s' \sim s''$.

Proposition 3.16 (partial commutativity) *For all* $B \in \mathbf{L}^{wf}$*,* $lts(B)$ *is partially commutative up to bisimulation.*

This will follow directly from the denotational characterisation in the next section (Theorem 4.22). Note that partial commutativity up to bisimulation is strictly stronger than partial trace commutativity in the sense of Example 3.4.

4 Denotational semantics

In this section, we develop a denotational semantics for $\mathbf{L}^{wf}$. In a sense, this semantics will be a "soundness check" for the operational semantics: in contrast to common wisdom, operationally we have characterised action refinement in the rather poor model of standard labelled transition systems (albeit with the additional assumption about global action dependencies). The basis for the denotational semantics, on the other hand, will be a very rich *event-based model*; and the constructions defined for the operators of $\mathbf{L}^{wf}$ are variations on known constructions on, for instance, prime event structures [43], stable event structures [73] and families of posets [60]. We then give a mapping from the denotational to the operational model showing the consistency of the two (up to bisimulation); this shows that the poor model is yet rich enough to capture the usual semantics of action refinement. (Note that, in contrast to the usual case, our denotational semantics is not strictly more abstract than the operational.)

Unfortunately, due to the action dependencies and the corresponding special features of weak sequential composition and refinement, none of the existing event-based models mentioned above is immediately suitable. Instead, we use an extension of the family-of-posets model. We assume a universe Evt of *events*, ranged over by d, e, which are used to model the occurrences of actions. We also use a special element $* \notin Evt$; we denote $Evt_* = Evt \cup \{*\}$. We require Evt to be closed under pairing in the sense that $Evt_* \times Evt_* \subseteq Evt$. The events are implicitly labelled; that is, there is a global labelling function $\ell\colon Evt \to Act$, which satisfies

$$\ell(e_1, e_2) = \begin{cases} \ell(e_1) \text{ if } e_2 = * \\ \ell(e_2) \text{ otherwise.} \end{cases}$$

We extend the dependency relation $D \subseteq Act \times Act$ to Evt by writing $d\ D\ e$ iff $\ell(d)\ D\ \ell(e)$.

Definition 4.1 (system runs) *A* system run *is a tuple* $p = \langle E, \leq, T \rangle$ *where*

- $E \subseteq Evt$ *is a finite set of events.*
- $\leq \subseteq E \times E$ *is a reflexive, cycle-free causal ordering of the events, such that* $D \cap (E \times E) = \leq \cup \geq$*, i.e., events are ordered iff they are dependent. (It follows that* $\leq$ *is not necessarily transitive.) We sometimes use* $<$ *to denote the irreflexive sub-relation of* $\leq$*.*
- $T \subseteq Act$ *is a set of actions with respect to which the run is* terminated.

Intuitively, the elements of T are independent of all actions in the "future" of this run, i.e., those that the system yet has to do when it reaches the state modelled by the current run. We often use E_p, $\leq_p$ and T_p to denote the components of a system run p; i.e., $p = \langle E_p, \leq_p, T_p \rangle$. We also use $\mathcal{A}_p = \ell(E_p)$ to denote the set actions occurring in p. The class of all system runs is denoted $\mathbf{P}$.

In the terminology of event structures, the system runs correspond to configurations, where the causal ordering of the events is included "locally" in each configuration. This makes for a richer model than most other event-based models (see [58] for a discussion); we will see below that this richness is actually necessary to model some of the features of $\mathbf{L}^{wf}$. The termination sets provide additional information used to model the partial termination properties: a system run p is terminated w.r.t. an action a iff $a \in T_p$. A useful intuition is that T_p is independent of all the actions that are yet to occur.

We use the following additional notations for system runs:

$$
\begin{aligned}
\varepsilon_T &= \langle \emptyset, \emptyset, T \rangle \\
\langle e, T \rangle &= \langle \{e\}, (e,e), T \rangle \\
\langle e \rangle &= \langle e, \emptyset \rangle \\
p \restriction E &= \langle E_p \cap E, \leq_p \cap (E \times E), T_p \rangle \quad \text{for } E \subseteq Evt \\
p \setminus E &= p \restriction (E_p \setminus E) \quad \text{for } E \subseteq Evt \\
p - q &= p \setminus E_q \\
p \cup T &= \langle E_p, \leq_p, T_p \cup T \rangle \quad \text{for } T \subseteq Act \\
p \cap T &= \langle E_p, \leq_p, T_p \cap T \rangle \quad \text{for } T \subseteq Act \\
p \setminus T &= \langle E_p, \leq_p, T_p \setminus T \rangle \quad \text{for } T \subseteq Act
\end{aligned}
$$

We also define a *prefix relation* over system runs. If a system run p is a prefix of q, this means that the latter provides more information about the system behaviour (the events that may occur and the corresponding termination properties) than the former.

Definition 4.2 (system run prefix) *A system run p is said to be a* prefix *of another system run q, denoted $p \preceq q$, if the following conditions hold:*

- *$E_p \subseteq E_q$, i.e., fewer events have occurred in p than in q;*
- *$\leq_p = \leq_q \cap (E_q \times E_p)$, i.e., the ordering of the events is the same in p and q, and moreover, all $\leq_q$-predecessors of events in p are also in p.*
- *$T_p \subseteq T_q \setminus [\mathcal{A}_{q-p}]_D$; i.e., an action is terminated in p only if it is terminated in q and independent of all actions occurring between p and q.*

System runs are sometimes depicted in the form $\boxed{F}T$, where

- F is a graphical representation of the first two components of the run, in the form of event nodes connected by arrows indicating the direct orderings between events (and not those that can be derived due to cycle-freedom and ordering of dependent events).
- T is the termination set; if $T = \emptyset$ it is sometimes omitted.

We often assume $Act \times \mathbb{N} \subseteq Evt$ with $\ell(a, i) = a$ in such figures and use the shorthand notation ${}^{i}a$ for (a, i) and ${}^{ij}a$ for $((a, i), (a, j))$, ${}^{i*}a$ for $((a, i), *)$ and ${}^{*j}a$ for $(*, (a, j))$; if an example includes only one occurrence of each action we sometimes also use $Act \subseteq Evt$. (Hence, for instance, $\boxed{{}^{i}a}T = \langle {}^{i}a, T\rangle$ and $\boxed{{}^{i}a} = \langle {}^{i}a\rangle$.)

Example 4.3 Assume $Act = \{a, b, c, d\}$ with $a\ D\ b\ D\ c\ D\ d$ (and all other actions independent).

- System runs: $\boxed{\begin{matrix} {}^{1}a \to {}^{2}b \\ \nearrow \\ {}^{3}c \to {}^{4}d \end{matrix}}\{a, c\}$ and $\boxed{\begin{matrix} b \to b \to a \\ d \end{matrix}}\{b, c\}$.
- Prefixes: $\boxed{\begin{matrix} {}^{1}a \\ \searrow \\ {}^{3}c \to {}^{2}b \end{matrix}}\{a\} \preceq \boxed{\begin{matrix} {}^{1}a \to {}^{2}b \\ \nearrow \\ {}^{3}c \to {}^{4}d \end{matrix}}\{a, c\}$ and $\boxed{\begin{matrix} {}^{1}a \\ {}^{3}c \to {}^{4}d \end{matrix}}\emptyset \preceq \boxed{\begin{matrix} {}^{1}a \to {}^{2}b \\ \nearrow \\ {}^{3}c \to {}^{4}d \end{matrix}}\{a, c\}$.
  (Note that the termination sets of the runs on the left hand side of $\preceq$ are *maximal* for these relations to hold.)
- Generally, $p \preceq p \cup A$ and $\varepsilon_{\emptyset} \preceq p$ for all $p \in \mathbf{P}$ and $A \subseteq Act$.

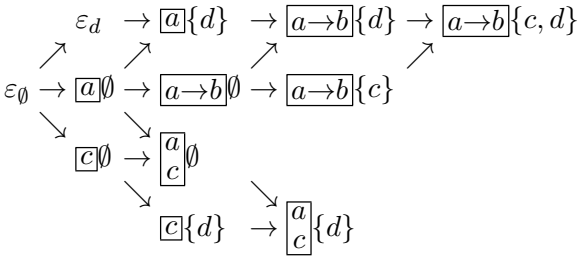
The structure $\langle \mathbf{P}, \preceq \rangle$ has some interesting order-theoretic properties, which, however, play no further role in this paper: it is a consistently complete prime algebraic domain, with (as the last item of the above example shows) $\varepsilon_{\emptyset}$ as a bottom element.

Definition 4.4 (denotational models) *A* system model *is a nonempty set $\mathcal{P} \subseteq \mathbf{P}$, such that $p \preceq q \in \mathcal{P}$ implies $p \in \mathcal{P}$; i.e., $\mathcal{P}$ is prefix closed.*

We use $E_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} E_p$ to denote the set of events used in $\mathcal{P}$, and $\mathbf{M}$ to denote the set of system models. System models are interpreted up to isomorphism; a function $\phi\colon E_{\mathcal{P}} \to E_{\mathcal{Q}}$ is an isomorphism between $\mathcal{P}$ and $\mathcal{Q}$ (denoted $\phi\colon \mathcal{P} \cong \mathcal{Q}$ or simply $\mathcal{P} \cong \mathcal{Q}$ if ϕ is irrelevant) if it preserves labelling ($\ell(\phi(e)) = \ell(e)$ for all $e \in E_{\mathcal{P}}$) and it is a bijection such that $\mathcal{Q} = \phi(\mathcal{P}) = \{\phi(p) \mid p \in \mathcal{P}\}$, where $\phi(p)$ is the result of renaming the events of p according to ϕ in the natural way. The following sub-sections present the constructions on $\mathbf{M}$ used to model the operators of $\mathbf{L}^{wf}$.

Example 4.5 Let $Act = \{a, b, c, d\}$ and $a\ D\ b\ D\ c\ D\ d$ as in Example 4.3. The following graph represents a system model. Since there is only one instance of every action, we have omitted event annotations. The arrows

between system runs represent the prefix ordering.



(As we will see below, this models the behaviour of $a \cdot (b \cdot \mathbf{0}_a + c \cdot \mathbf{0}_b)$.) Note that $\boxed{a}\{d\} \not\preceq \boxed{\begin{matrix}a\\c\end{matrix}}\{c,d\}$.

Note that the non-emptiness and prefix closure of system runs together imply that $\varepsilon_\emptyset \in \mathcal{P}$ for all $\mathcal{P} \in \mathbf{M}$.

4.1 Constructions

We first introduce and discuss the model constructions used to implement the operators of $\mathbf{L}^{wf}$, and afterwards establish their formal properties, such as the fact that all constructions stay within $\mathbf{M}$, and are well-defined with respect to isomorphism.

Deadlock constants. Deadlock is modelled by empty runs whose termination set is independent of the deadlock alphabet; i.e., to model $\mathbf{0}_A$ we use the set of all $\varepsilon_T \in \mathbf{P}$ where $T\ I\ A$. Note that if $p \preceq \varepsilon_T$ with $T\ I\ A$, then $p = \varepsilon_{T_p}$ with $T_p \subseteq T$, hence $T_p\ I\ A$; thus, $\mathcal{P}$ is a valid system model.

Action constants. To model a single action a denotationally, we need a single event e labelled by that action; no proper causal ordering (except for the reflexive $e \leq e$) is possible. There are two types of runs: the empty ones, where nothing has happened yet, and the complete ones, where the action has occurred. In the former, the termination sets are independent of a; in the latter, the termination sets are arbitrary. This gives rise to $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ where $\mathcal{P}_1$ equals the model of $\mathbf{0}_{\{a\}}$ (see above) and $\mathcal{P}_2$ consists of all system runs $\langle e, T\rangle$ for a given event $e \in Evt$ with $\ell(e) = a$ and $T \subseteq Act$ arbitrary.

Choice. The construction for choice is entirely analogous to the usual one in event-based modelling: It consists of a simple union of the operands, with the proviso that the events used in those operands must be disjoint. That is, we define

$$\mathcal{P}_1 + \mathcal{P}_2 = \mathcal{P}_1 \cup \mathcal{P}_2 \quad \text{if } E_{\mathcal{P}_1} \cap E_{\mathcal{P}_2} = \emptyset.$$

Note that the disjointness side condition can always be fulfilled by choosing appropriate isomorphic representatives.

Parallel composition. The model construction for parallel composition basically consists of combining all pairs of runs from the operands by gluing them together at the events labelled by synchronising actions, whenever this yields a valid system run. To accomplish the gluing together, we use the following construction on event sets $E_1, E_2 \subseteq Evt$:

$$\begin{aligned} E_1 \parallel_A E_2 = \; & \{(e, *) \in E_1 \times \{*\} \mid \ell(e) \notin A\} \\ & \cup \{(*, e) \in \{*\} \times E_2 \mid \ell(e) \notin A\} \\ & \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \ell(e_1) = \ell(e_2) \in A\} \end{aligned}$$

In comparison with the standard case (i.e., without action dependencies), the construction is complicated slightly by two things: all dependent events in the synchronised run have to be ordered, even if they stem from different operands and are unsynchronised; and the termination sets have to be computed. W.r.t. the first complication, consider the following example:

Example 4.6 Assume $Act = \{a, b, c\}$ with $c \; D \; \{a, b\}$. The synchronisation of the system runs $\boxed{\begin{matrix} {}^{1}a \rightarrow {}^{2}a \\ {}^{3}b \end{matrix}}$ and $\boxed{{}^{4}a \rightarrow {}^{5}c \rightarrow {}^{6}a}$ on the action a (omitting termination sets) yields one of the runs $\boxed{\begin{matrix} {}^{14}a & \searrow & \\ {}^{3*}b & \rightarrow & {}^{*5}c \rightarrow {}^{26}a \end{matrix}}$ and $\boxed{\begin{matrix} {}^{14}a \rightarrow {}^{*5}c & \rightarrow & {}^{26}a \\ & \searrow & {}^{3*}b \end{matrix}}$ (where ij denotes the pair (i, j)).

For $i = 1, 2$ let $\pi_i \colon Evt \rightharpoonup Evt$ be the partial function defined by $\pi_i(d) = e_i$ if $d = (e_1, e_2)$ and $e_i \neq *$. The parallel composition of system models is then defined as follows:

$$\begin{aligned} \mathcal{P}_1 \parallel_A \mathcal{P}_2 = \{q \in \mathbf{P} \mid \; & \exists p_1 \in \mathcal{P}_1, \, p_2 \in \mathcal{P}_2 \colon E_q \subseteq E_{p_1} \parallel_A E_{p_2}, \\ & <_{p_i} = \{(\pi_i(d), \pi_i(e)) \mid d <_q e\} \quad \text{for } i = 1, 2, \\ & T_q = T_{p_1} = T_{p_2}\} \end{aligned}$$

Note that, given two system runs $p_i \in \mathcal{P}_i$ for $i = 1, 2$, if there exists any valid synchronisation of p_1 and p_2 at all, it is unambiguous which events of p_1 synchronise with which of p_2, due to the fact that identically labelled events are dependent and thus ordered in both p_1 and p_2; but it is not always pre-determined how the non-synchronising events with dependent labels are ordered in q. See also Example 4.6.

Weak sequential composition. A run of a system obtained by sequential composition consists of a run of the first operand, followed by a run of the second, where the "followed by" is modulo the dependency relation. Moreover, we cannot combine arbitrary pairs of runs: rather, the alphabet of the second run should be part of the termination set of the first. This gives

rise to the following construction, where again (as for choice) we require $E_{\mathcal{P}_1} \cap E_{\mathcal{P}_2} = \emptyset$:

$$\begin{aligned}\mathcal{P}_1 \cdot \mathcal{P}_2 = \{q \mid\ & \exists p_1 \in \mathcal{P}_1,\, p_2 \in \mathcal{P}_2 \colon \mathcal{A}_{p_2} \subseteq T_{p_1}, \\ & \leq_q = \leq_{p_1} \cup ((E_{p_1} \times E_{p_2}) \cap D) \cup \leq_{p_2}, \\ & T_q = T_{p_1} \cap T_{p_2}\}\end{aligned}$$

Refinement. Denotationally, refinement consists of a system model $\mathcal{P}$ to be refined and a function $\mathcal{R} \colon Act \to \mathbf{M}$ that maps each action to another system model. In accordance with the notion of strong D-consistency on syntactic refinement functions (Definition 3.5), we also impose a requirement on the function $\mathcal{R}$:

Definition 4.7 (denotational D-consistency) *A semantic refinement function $\mathcal{R} \colon Act \to \mathbf{M}$ is called* denotationally D-consistent *if for all $a, b \in Act$:*

- *$a\ D\ b$ implies (i) $\varepsilon_b \notin \mathcal{R}(a)$ and (ii) $a\ D\ \ell(e)$ for all $\langle e \rangle \in \mathcal{R}(b)$;*
- *$a\ I\ b$ implies $\mathcal{A}_p\ I\ b$ and $p \cup b \in \mathcal{R}(a)$ for all $p \in \mathcal{R}(a)$.*

We only consider denotationally D-consistent $\mathcal{R}$. The runs of the refined system are obtained by taking a run p of $\mathcal{P}$, and refining each event d of p by some non-empty run $w(d)$ of $\mathcal{R}(a)$, where a is the label of d. Thus, we each time use a so-called *witness* function $w \colon E_p \to \mathbf{M}$ that selects nonempty runs from the refinement function $\mathcal{R}$.

$$\begin{aligned}\mathcal{P}[\mathcal{R}] = \{q \mid\ & \exists p \in \mathcal{P} \colon \forall d \in E_p \colon \exists w(d) \in \mathcal{R}(\ell(d)) \setminus \varepsilon_T \colon \\ & E_q = \left\{(d, e) | d \in E_p, e \in E_{w(d)}\right\}, \\ & \leq_q = \{((d_1, e_1), (d_2, e_2)) | d_1 <_p d_2, e_1\ D\ e_2 \text{ or} \\ & d_1 = d_2, e_1 <_{w(d_1)} e_2\}, \\ & T_q = T_p \cap \textstyle\bigcap_{d \in E_p} T_{w(d)}\}\end{aligned}$$

The condition that w only selects non-empty runs is there to ensure that we can by and large reconstruct p and w from each $q \in \mathcal{P}[\mathcal{R}]$ (with the exception of the termination sets).

Note that, due to denotational D-consistency of $\mathcal{R}$, the images of w respect the causal ordering of events in p: if $d \not\leq_p d'$ then $d\ I\ d'$ and hence the labels of $w(d)$ and $w(d')$ are independent and, moreover, the labels of $w(d')$ can be added to the termination set of $w(d)$. (This is a compatibility property similar to the one used above for weak sequential composition.)

Example 4.8 Assume $Act = \{a, a_1, a_2, b, b_1, b_2, c, c_1, c_2\}$ and let the independencies be given by $a_1\ \ I\ \ \{b_2, c_2\}$, $a_2\ \ I\ \ \{b_1, b_2, c_1\}$ and $\{b, b_1, b_2\}\ I\ \{c, c_1, c_2\}$. Take the system run $p = \boxed{\begin{smallmatrix} & & b \\ & \nearrow & \\ a & \to & c \end{smallmatrix}} Act$ and refine according to

$$w\colon \quad a \mapsto \boxed{a_1 \to a_2} Act \quad b \mapsto \boxed{b_1 \to b_2} Act \quad c \mapsto \boxed{c_1 \to c_2} Act$$

This results in the run $q = \boxed{\begin{array}{ccc} & & b_1{\to}b_2 \\ & \nearrow & \\ a_1 & \to & a_2 \\ & \searrow & \searrow \\ & & c_1{\to}c_2 \end{array}} Act$. The prefix $\boxed{\begin{array}{ccc} & & b_1{\to}b_2 \\ & \nearrow & \\ a_1 & & \\ & \searrow & \\ & & c_1 \end{array}} \{b_1, b_2\}$ of q, on the other hand, is obtained by refining p above according to

$$w\colon \quad a \mapsto \boxed{a_1}\{b_1, b_2, c_1\} \quad b \mapsto \boxed{b_1{\to}b_2}Act \quad c \mapsto \boxed{c_1}\{a_1, b, b_1, b_2\} \ .$$

Recursion. As usual, the semantics of a recursive term $recX.\, B$ is the fixpoint of a function over $f\colon \mathbf{M} \to \mathbf{M}$, where f is derived from B — essentially, f is the semantic counterpart of substituting a term for X in B. We will use the theory of metric spaces to show that this fixpoint is uniquely defined; moreover, any solution of f *up to isomorphism*, i.e., such that $f(\mathcal{P}) \cong \mathcal{P}$, is also isomorphic to the fixpoint of f.

Even if the semantics of recursion will be shown to *satisfy* this fixpoint property, it is not how we *define* the semantics. For the definition of $[\![recX.\, B]\!]$, instead, we take the limit (i.e., the union) of the semantic models of a sequence of approximants $(B^i_X)_{i\in\mathbb{N}}$ defined by

$$\begin{aligned} B^0_X &= \mathbf{0}_{\mathcal{A}_X} \\ B^{i+1}_X &= B\langle B^i_X / X\rangle \ . \end{aligned}$$

We will see that the aforementioned f is monotonic such that $[\![B^{i+1}_X]\!] = f([\![B^i_X]\!])$ for all $i \in \mathbb{N}$; since $[\![B^0_X]\!] = \{\varepsilon_T \mid T\ I\ \mathcal{A}_X\}$ is the smallest element of the sub-space of $\mathbf{M}$ in which the fixpoint of f must lie, it follows that $\bigcup_{i\in\mathbb{N}} [\![B^i_X]\!]$ is indeed a fixpoint of f. In Sect. 4.3 we provide the necessary theory.

4.2 Well-definedness

We defined a number of constructions above, without considering whether $\mathbf{M}$ is closed under them, i.e., whether the structures thus constructed are again system models (in the sense of Definition 4.4). In particular, one has to check that prefix closure is preserved. The following proposition states that this is indeed the case.

Proposition 4.9 $\mathbf{M}$ *is closed under the constructions defined above.*

The proof can be found in Appendix A (Page 211). A noteworthy point is that the denotational D-consistency of the refinement functions (Definition 4.7) is essential in the proof.

Table 3. Denotational semantics for closed terms of $\mathbf{L}^{wf}$

$$
\begin{aligned}
[\![\mathbf{0}_A]\!] &= \{\varepsilon_T \mid T\ I\ A\} \\
[\![a]\!] &= [\![\mathbf{0}_a]\!] \cup \{\langle e, T\rangle \mid T \subseteq Act\} \quad \text{where } a = \ell(e) \\
[\![B_1 + B_2]\!] &= \iota_1([\![B_1]\!]) + \iota_2([\![B_2]\!]) \\
[\![B_1 \parallel_A B_2]\!] &= [\![B_1]\!] \parallel_A [\![B_2]\!] \\
[\![B_1 \cdot B_2]\!] &= \iota_1([\![B_1]\!]) \cdot \iota_2([\![B_2]\!]) \\
[\![B[r]]\!] &= [\![B]\!][a \mapsto [\![r(a)]\!] \mid a \in Act] \\
[\![recX.\ B]\!] &= \textstyle\bigcup_{i\in\mathbb{N}} [\![B_X^i]\!] \quad \text{where } B_X^0 = \mathbf{0}_{\mathcal{A}_X},\ B_X^{i+1} = B\langle B_X^i / X\rangle
\end{aligned}
$$

A further result is that the constructions are well-defined modulo isomorphism. This is immediate from the definitions, since any bijective renaming of the events of the operands can easily be turned into a bijective renaming of the events of the constructed model. This is formulated in the following proposition, the proof of which is straightforward and hence omitted.

Proposition 4.10 *The above constructions over* $\mathbf{M}$ *are well-defined up to* $\cong$.

With the help of the injections $\iota_i \colon Evt \to Evt$ defined by $\iota_1(e) = (e, *)$ and $\iota_2(e) = (*, e)$ for all $e \in Evt$, we can now define a denotational $\mathbf{M}$-semantics for closed terms of $\mathbf{L}^{wf}$, in the form of a function $[\![-]\!] \colon \mathbf{L}^{wf} \to \mathbf{M}$. It is given in Table 3.

The immediate question is if the denotational semantics is well-defined; in particular, if every refinement function constructed in Table 3 is indeed denotationally D-consistent. This turns out to be indeed the case.

Proposition 4.11 $[\![B]\!] \in \mathbf{M}$ *for all* $B \in \mathbf{L}^{wf}$.

In the course of the proof — which is worked out in full in Appendix A (from Page 213) — the following property is also shown to hold:

Proposition 4.12 *If* $r \colon Act \to \mathbf{L}^{wf}$ *is strongly D-consistent, then the function* $Act \to \mathbf{M}$ *defined by* $a \mapsto [\![r(a)]\!]$ *for all* $a \in Act$ *is denotationally D-consistent.*

This in turn relies on a certain relation between operational and denotational concepts, established by the following proposition.

Proposition 4.13 *For all closed* $B \in \mathbf{L}^{wf}$*, the following holds:*

1. $a\ I\ \mathcal{A}(B)$ *implies* $a\ I\ \mathcal{A}_p$ *and* $p \cup a \in [\![B]\!]$ *for all* $p \in [\![B]\!]$.
2. $B \xrightarrow{\checkmark_a}$ *iff* $\varepsilon_a \in [\![B]\!]$.
3. $B \xrightarrow{a}$ *iff* $\langle e\rangle \in [\![B]\!]$ *with* $\ell(e) = a$.

Somewhat unexpectedly, the inverse implications of Propositions 4.13.1 and (hence) 4.12 do *not* hold. This is due to the fact that, in some cases, there are actions which are semantically independent of a system model but not syntactically independent of a term giving rise to that model. As an example, consider the term $B = a + \mathbf{0}_b$ where $a\ I\ b$: we have $[\![B]\!] \cong [\![a]\!]$, showing that $\mathbf{0}_b$ does not contribute anything to the behaviour of this term and hence $b\ I\ \mathcal{A}_p$ and $p \cup b \in [\![B]\!]$ for all $p \in [\![B]\!]$; yet $b\ D\ \mathcal{A}(B)$.

4.3 Recursion as a unique fixpoint

Table 3 contains a definition for the denotational semantics of recursive terms; however, we have yet to demonstrate that this semantics is reasonable, in the sense that it satisfies the criteria usually imposed upon the concept of recursion. In this subsection, we show that the set of models $\mathbf{M}$ actually forms a metric space and recursive terms correspond to unique fixpoints of contractions over $\mathbf{M}$. We only deal with simple, non-nested recursion, i.e., terms $recX.\ B$ in which B itself is finite; the general case is a standard generalisation that is notationally much more complex but presents no essential novelties.

Denotationally, the semantics of a well-formed recursive term $recX.\ B$ (where $B \in \mathbf{L}_{fin}$ and $fv(B) \subseteq \{X\}$ due to the restriction to simple recursion) is expected to solve the equation $X = B$, interpreted in $\mathbf{M}$ modulo $\cong$. That is, a solution of this equation is a system model $\mathcal{P} \in \mathbf{M}$ such that $\mathcal{P} \cong [\![B]\!](\mathcal{P})$, where $[\![B]\!](-)\colon \mathbf{M} \to \mathbf{M}$ is a function derived from B by extending the definitions in Table 3 with a parameter:

$$\begin{aligned}
[\![\mathbf{0}_A]\!](\mathcal{P}) &= \{\varepsilon_T \mid T\ I\ A\} \\
[\![a]\!](\mathcal{P}) &= [\![\mathbf{0}_a]\!] \cup \{\langle e, T\rangle \mid T \subseteq Act\} \\
[\![B_1 + B_2]\!](\mathcal{P}) &= \iota_1([\![B_1]\!](\mathcal{P})) + \iota_2([\![B_2]\!](\mathcal{P})) \\
[\![B_1 \parallel_A B_2]\!](\mathcal{P}) &= [\![B_1]\!](\mathcal{P}) \parallel_A [\![B_2]\!](\mathcal{P}) \\
[\![B_1 \cdot B_2]\!](\mathcal{P}) &= \iota_1([\![B_1]\!](\mathcal{P})) \cdot \iota_2([\![B_2]\!](\mathcal{P})) \\
[\![B[r]]\!](\mathcal{P}) &= [\![B]\!](\mathcal{P})[a \mapsto [\![r(a)]\!] \mid a \in Act] \\
[\![X]\!](\mathcal{P}) &= \mathcal{P}\ .
\end{aligned}$$

(Recall that refinement functions in well-formed terms map to closed terms only, hence $[\![r(a)]\!]$ is well-defined.) This effectively defines a semantic counterpart to syntactic substitution.

Proposition 4.14 *For all $B \in \mathbf{L}_{fin}^{wf}$ and $C \in \mathbf{L}^{wf}$ such that $fv(B) \subseteq \{X\}$, C is closed and $\mathcal{A}(C) \subseteq \mathcal{A}_X$, $[\![B\langle C/X\rangle]\!] = [\![B]\!]([\![C]\!])$.*

We first show that $[\![B]\!](-)$ itself has a unique fixpoint, which equals $[\![recX.B]\!]$; this is certainly a solution of $X = B$. We then show that all $\mathcal{P}$ solving the

equation up to isomorphism, i.e., such that $[\![B]\!](\mathcal{P}) \cong \mathcal{P}$, are isomorphic to $[\![recX.\ B]\!]$.

Global termination. In order to obtain unique fixpoints, we have to be more precise about the desired termination properties. In the operational semantics (Proposition 3.8), all terms are globally terminated with respect to actions that are independent of the term's alphabet. This is a property that we also want the denotational semantics to reflect, and without which fixpoints are *not* unique, as the following example shows.

Example 4.15 Consider the function $f = [\![a \cdot X]\!](-)$ (i.e., $f\colon \mathcal{P} \mapsto \iota_1([\![a]\!]) \cdot \iota_2(\mathcal{P})$ for all $\mathcal{P}$), and consider the following two models:

$$\mathcal{P} = \{\langle a^n, T\rangle \mid n \in \mathbb{N}, T\ I\ a\}$$
$$\mathcal{Q} = \{\langle a^n, \emptyset\rangle \mid n \in \mathbb{N}\}$$

(where $\langle a^n, T\rangle$ denotes a system run consisting of n consecutive occurrences of the action a and termination set T). It is not difficult to see that $\mathcal{P} \cong f(\mathcal{P})$ and $\mathcal{Q} \cong f(\mathcal{Q})$; yet $\mathcal{P} \ncong \mathcal{Q}$. According to Table 3, $[\![recX.\ a \cdot X]\!] = \mathcal{P}$, and indeed we consider this the "appropriate" semantics; it can be depicted by

$$\varepsilon_{[a]_I} \to \boxed{a}[a]_I \to \boxed{a \to a}[a]_I \to \boxed{a \to a \to a}[a]_I \to \cdots$$

$\mathcal{Q}$ is not terminated for any $b\ I\ a$, i.e., $\varepsilon_b \notin \mathcal{Q}$; this contradicts the operational intuition that a term should always be terminated for all actions independent of its alphabet.

The desired property is captured by the following definition:

Definition 4.16 (global termination) *Let $\mathcal{P} \in \mathbf{M}$ and $T \subseteq Act$. $\mathcal{P}$ is called* globally terminated *for T if $T\ I\ \mathcal{A}_p$ and $p \cup T \in \mathcal{P}$ for all $p \in \mathcal{P}$.*

The class of all models globally terminated for T will be denoted $\mathbf{M}_T$. Note that $\mathbf{M}_T \subseteq \mathbf{M}_{T'}$ if $T \supseteq T'$. For instance, in the above example $\mathcal{P}$ is globally terminated for $[a]_I$, whereas $\mathcal{Q}$ is not. In the semantics defined by Table 3, all $B \in \mathbf{L}^{wf}$ are globally terminated for $[\mathcal{A}(B)]_I$; that is, $[\![B]\!] \in \mathbf{M}_{[\mathcal{A}(B)]_I}$ for all closed $B \in \mathbf{L}^{wf}$. (This follows from Proposition 4.17 below.)

Accordingly, we will in fact interpret $[\![B]\!](-)$ as a *partial* function $\mathbf{M} \to \mathbf{M}$, defined only on $\mathcal{P} \in \mathbf{M}_{[\mathcal{A}_X]_I}$; or alternatively as a (total) function $\mathbf{M}_{[\mathcal{A}_X]_I} \to \mathbf{M}_{[\mathcal{A}(B)]_I}$. The following proposition (proof omitted) implies that this interpretation is valid.

Proposition 4.17 *If $B \in \mathbf{L}^{wf}_{fin}$ with $fv(B) \subseteq \{X\}$, then $[\![B]\!](\mathcal{P}) \in \mathbf{M}_{[\mathcal{A}(B)]_I}$ for all $\mathcal{P} \in \mathbf{M}_{[\mathcal{A}_X]_I}$.*

It follows that any closed recursive term $recX.\, B \in \mathbf{L}^{wf}$ (which satisfies $\mathcal{A}(B) \subseteq \mathcal{A}_X$, see Table 1) gives rise to a function $[\![B]\!](-)\colon \mathbf{M}_T \to \mathbf{M}_T$ with $T = [\mathcal{A}_X]_I$. We now show that $[\![recX.\, B]\!]$ as defined in Table 3 is the unique fixpoint of $[\![B]\!](-)$ in $\mathbf{M}_T$ — even if it is not unique in $\mathbf{M}$, as Example 4.15 shows.

A complete metric space. In order to achieve this, we use the theory of metric spaces; cf. [21] for an exposition of the basic theory. First we turn $\mathbf{M}_T$ into a complete metric space, where the distance between two models is determined by the largest depth up to which they coincide. For arbitrary $p \in \mathbf{P}$, the depth of p is determined by its longest $<_p$-chain, as follows:

$$depth(p) = \max\{n \mid \exists (e_i)_{1 \le i \le n} \subseteq E_p \colon \forall 1 \le i < n \colon e_i <_p e_{i+1}\}$$

Furthermore, for arbitrary $\mathcal{P} \in \mathbf{M}$ and $n \in \mathbb{N}$, the "prefix" of $\mathcal{P}$ *up to* depth n is defined by

$$\mathcal{P}{\uparrow}n = \{p \in \mathcal{P} \mid depth(p) \le n\}$$

This gives rise to the following distance function $\delta\colon \mathbf{M} \times \mathbf{M} \to \mathbb{R}$:

$$\delta(\mathcal{P}, \mathcal{Q}) = 2^{-\sup\{n+1 \mid \mathcal{P}{\uparrow}n = \mathcal{Q}{\uparrow}n\}}\ .$$

The "$+1$" in the supremum is there to ensure that if $\mathcal{P}{\uparrow}0 = \mathcal{Q}{\uparrow}0$ (meaning that the termination properties of $\mathcal{P}$ and $\mathcal{Q}$ coincide) then $\delta(\mathcal{P}, \mathcal{Q}) < 1$. Since $\sup \emptyset = 0$ and $\sup \mathbb{N} = \infty$, it follows that $\delta(\mathcal{P}, \mathcal{Q}) = 1$ iff $(\varepsilon_T \in \mathcal{P}) \not\Leftrightarrow (\varepsilon_T \in \mathcal{Q})$ for some $T \subseteq Act$, and $\delta(\mathcal{P}, \mathcal{Q}) = 0$ iff $\mathcal{P} = \mathcal{Q}$.

The above definitions of δ and $depth$ are standard — for event-based models, very similar ones can be found in [43] — and so is the (proof of the) following theorem.

Theorem 4.18 *For all $T \subseteq Act$, $\langle \mathbf{M}_T, \delta \rangle$ is a complete metric space.*

In fact, the limit $\mathcal{P}$ of a Cauchy sequence $(\mathcal{P}_i)_i$ in $\langle \mathbf{M}, \delta \rangle$ is given by $\bigcup_{i \in \mathbb{N}} \bigcap_{j \ge i} \mathcal{P}_i$. In general, due to the properties of complete metric spaces, any contracting function $f\colon \mathbf{M}_T \to \mathbf{M}_T$ has a unique fixpoint (see [21]). We now prove that functions of the form $[\![B]\!](-)$ are contracting if X is dependently guarded in B. (Note that this is not true for arbitrary B; as an extreme case, $[\![X]\!](-)$ is clearly not contracting.)

Proposition 4.19 *Let $B \in \mathbf{L}^{wf}_{fin}$ with $fv(B) \subseteq \{X\}$.*

1. *$[\![B]\!](-)$ is non-increasing;*
2. *If X is dependently guarded in B, then $[\![B]\!](-)$ is contracting.*

The proof can be found in Appendix A (Page 216). This gives rise to the following theorem.

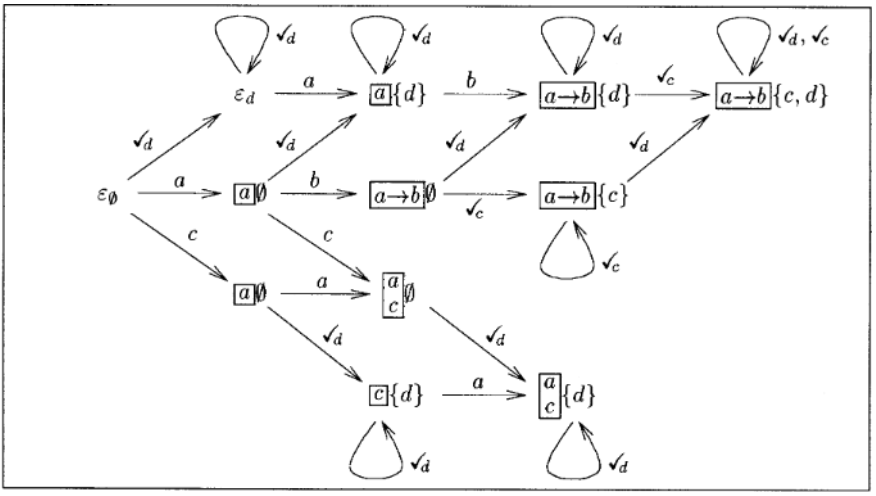


Fig. 1. A model of $a \cdot (b \cdot \mathbf{0}_a + c \cdot \mathbf{0}_b)$, where $Act = \{a, b, c, d\}$ and $a\ D\ b\ D\ c\ D\ d$

Theorem 4.20 *If $B \in \mathbf{L}^{wf}$ with $fv(B) \subseteq X$, and X is dependently guarded in B, then $[\![recX.B]\!]$ is the unique solution of $X = B$ modulo $\cong$ in $\mathbf{M}_{[\mathcal{A}_X]_I}$.*

Note that this not only states that $\mathcal{P} = [\![B]\!](\mathcal{P})$ implies $\mathcal{P} = [\![recX.\ B]\!] = \mathcal{P}$, but also that $\mathcal{P} \cong [\![B]\!](\mathcal{P})$ implies $\mathcal{P} \cong [\![recX.\ B]\!]$. The proof can be found in Appendix A (Page 218).

4.4 Transitions

To show the correspondence between operational and denotational semantics, we turn each system system model $\mathcal{P}$ into an $Act \cup \surd_{Act}$-labelled transition system $\langle \mathcal{P}, \rightarrow, \varepsilon_\emptyset \rangle$; i.e., the states of the transition system are given by the system runs. A similar construction can be found in, e.g., [65]

Intuitively, in $\mathcal{P}$ there is an a-labelled transition from each run of $\mathcal{P}$ to any run of which it is a prefix differing only by a single a-labelled event; furthermore, there is a $\surd_a$-labelled transition between any two system runs p and $p \cup a$. Formally, $\rightarrow$ is the smallest relation such that for all $p \in \mathcal{P}$:

- $p \setminus e \xrightarrow{\ell(e)} p$ if $e \in \max E_p$ and $\ell(e)\ I\ T_p$;
- $p \xrightarrow{\surd_a} p \cup a$ if $p \cup a \in \mathcal{P}$.

Note that if $e \in \max E_p$ and $\ell(e)\ I\ T_p$ then $p \setminus e \preceq p$, and hence $p \setminus e \in \mathcal{P}$. For instance, Fig. 1 represents the system model of Example 4.5 as a transition system.

An immediate observation is that system model isomorphism implies bisimilarity. (The bisimulation relation is given by $\{(p, \phi(p)) | p \in \mathcal{P}\}$ where $\phi\colon \mathcal{P} \cong \mathcal{Q}$.)

Proposition 4.21 *If* $\mathcal{P} \cong \mathcal{Q}$*, then* $\langle \mathcal{P}, \rightarrow, \varepsilon_\emptyset \rangle \sim \langle \mathcal{Q}, \rightarrow, \varepsilon_\emptyset \rangle$.

One of the main results of this paper is the following theorem, which states the correspondence between operational and denotational semantics.

Theorem 4.22 *For all closed* $B \in \mathbf{L}^{wf}$, $B \sim lts([\![B]\!])$.

The proof of this theorem proceeds by induction on the structure of B. For the refinement operator, the correspondence crucially relies on strong D-consistency of refinement functions (Definition 3.5) and its denotational counterpart (see Definition 4.7 and Proposition 4.12). For the case of recursion, we use the uniqueness of fixpoints modulo $\sim$, stated in Theorem 3.15: essentially, because the denotational semantics of $[\![recX.\ B]\!]$ yields a solution of $X = B$, it must be bisimilar to $recX.\ B$. However, a technical problem in this argument is that we have proved Theorem 3.15 on the syntactical level (if two *terms* are solutions of $X = B$, then they are bisimilar) and so it is not directly applicable to system models like $[\![recX.\ B]\!]$. To circumvent this, for the purpose of proving Theorem 4.22 we introduce additional constants $t_\mathcal{P}$ for every $\mathcal{P} \in \mathbf{M}$ to $\mathbf{L}^{wf}$. As remarked in Sect. 3.4, the proof of Theorem 3.15 is not invalidated if we extend $\mathbf{L}$ in this way.

Moreover, the proof actually uses a different (bisimilar) representation for the operational semantics of system models. Namely, instead of turning each individual system model $\mathcal{P}$ into a transition system of which the runs (the elements of $\mathcal{P}$) are the states, we turn the *class* of system models, $\mathbf{M}$, into a transition system of which the *system models* (the elements of $\mathbf{M}$) are the states. Full details of the proof, including this alternative representation, are worked out in Appendix A (from Page 220).

5 Axiomatisation

Next we will develop an axiomatisation of bisimilarity. We give a finite equational theory T such that for all closed terms B_1, B_2 of the finite language $\mathbf{L}^{wf}_{fin}$,

$$\mathsf{T} \vdash B_1 = B_2 \text{ if and only if } B_1 \sim B_2,$$

that is, T will be sound for bisimilarity in $\mathbf{L}^{wf}$ and complete for bisimilarity in $\mathbf{L}^{wf}_{fin}$. Within T, the rules for equational reasoning (reflexivity, symmetry, transitivity, substitution and instantiation) can be used to deduce equality of terms from given equations.

5.1 Auxiliary operators

As might be expected, the unusual behaviour of weak sequential composition forces some modifications to the standard axioms for bisimilarity.

The axiomatisation of ACP [8] for instance contains a rule for the right-distributivity of sequential composition over choice: $(x + y)z = xz + yz$ (where juxtaposition is sequential composition). For weak sequential composition, however, this is not valid: for instance, if $a \ I \ c \ I \ b$, then $(a + b) \cdot c \xrightarrow{c} (a + b) \cdot \mathbf{1}$ which can still do both a and b; however, if $a \cdot c + b \cdot c \xrightarrow{c} B$, then either $B = a \cdot \mathbf{1}$ or $B = b \cdot \mathbf{1}$, neither of which can do both a and b. It follows that $(a + b) \cdot c \not\sim a \cdot c + b \cdot c$.

Our axiomatisation is inspired by Aceto, Bloom and Vaandrager [2], who developed a general method for deriving complete axiomatisations for strong bisimilarity directly from GSOS rules. Their work is not directly applicable to our system in its current form (even for the part without recursion), but we closely follow their ideas. The problem for the applicability lies in the fact that although the recursion-free language can only describe finite behaviour, still in a technical sense it allows infinite computations to be specified: for instance, if $a \ I \ b$ then $b \xrightarrow{\surd_a} b \xrightarrow{\surd_a} \cdots$. This means that the technique of [2] fails to induce a normal form.

Nevertheless, a complete axiomatisation does exist. As usual, it requires the addition of auxiliary operators to $\mathbf{L}$; they are collected in Table 4. The language including all auxiliary operators is denoted $\mathbf{L}^+$, and the well-formed sub-language $\mathbf{L}^{wf+}$. In particular, $recX.\ B$ is only well-formed if $\mathcal{A}(B) \subseteq \mathcal{A}_X$; for that purpose, $\mathcal{A}(B)$ must at least be defined, implying that B may contain no occurrences of the residue operator, $B \downarrow a$.

Left merge and communication merge. $\lfloor\!\lfloor_A$ is the standard *left merge* from ACP, adapted to our notion of synchronisation and extended to deal with termination. $|_A$ is the relevant version of the *communication merge*. The idea is that $B \lfloor\!\lfloor_A C$ captures part of the behaviour of $B \parallel_A C$, namely the cases where the first action that occurs comes from the left hand operand, B, and does not have to synchronise. $B \mid_A C$, on the other hand, captures the part where the first action must arise from a synchronisation, i.e., must be an element of A and performed simultaneously by B and C. Parallel composition can then be split up according to the following axiom:

$$x \parallel_A y = x \lfloor\!\lfloor_A y + y \lfloor\!\lfloor_A x + x \mid_A y \ .$$

Left and right sequential and residue. We use operators $\leftarrow$ and $\rightarrow$, called *left sequential* and *right sequential*, which serve a similar purpose with respect to sequential composition as left and communication merge with respect to synchronisation. That is, $B \leftarrow C$ captures the part of $B \cdot C$ where the first action to occur must come from B, whereas $B \rightarrow C$ specifies that it must come from C, in which case B must terminate for that action. Both operators are right-associative. Sequential composition can then be split up as follows:

$$x \cdot y = x \leftarrow y + x \rightarrow y \ .$$

Table 4. Operational semantics of auxiliary operators

left merge	$\dfrac{x \xrightarrow{a} x' \quad a \notin A}{x \between_A y \xrightarrow{a} x' \parallel_A y}\,\mathrm{R}_{16}$	$\dfrac{x \xrightarrow{\surd a} x' \quad y \xrightarrow{\surd a} y'}{x \between_A y \xrightarrow{\surd a} x' \between_A y'}\,\mathrm{R}_{17}$
communication merge	$\dfrac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y' \quad a \in A}{x \mid_A y \xrightarrow{a} x' \parallel_A y'}\,\mathrm{R}_{18}$	$\dfrac{x \xrightarrow{\surd a} x' \quad y \xrightarrow{\surd a} y'}{x \mid_A y \xrightarrow{\surd a} x' \mid_A y'}\,\mathrm{R}_{19}$
left sequential	$\dfrac{x \xrightarrow{a} x'}{x \leftarrow y \xrightarrow{a} x' \cdot y}\,\mathrm{R}_{20}$	$\dfrac{x \xrightarrow{\surd a} x' \quad y \xrightarrow{\surd a} y'}{x \leftarrow y \xrightarrow{\surd a} x' \leftarrow y'}\,\mathrm{R}_{21}$
right sequential	$\dfrac{x \xrightarrow{\surd a} x' \quad y \xrightarrow{a} y'}{x \rightarrow y \xrightarrow{a} x' \cdot y'}\,\mathrm{R}_{22}$	$\dfrac{x \xrightarrow{\surd a} x' \quad y \xrightarrow{\surd a} y'}{x \rightarrow y \xrightarrow{\surd a} x' \rightarrow y'}\,\mathrm{R}_{23}$
residue	$\dfrac{x \xrightarrow{\surd a} x' \xrightarrow{\beta} x''}{x \downarrow a \xrightarrow{\beta} x''}\,\mathrm{R}_{24}$	
deadlock	$\dfrac{x \xrightarrow{\surd a} x' \quad a \in [A]_I}{\delta_A(x) \xrightarrow{\surd a} \delta_A(x')}\,\mathrm{R}_{25}$	

Note that, unlike parallel composition, sequential composition is not symmetrical, and hence left and right sequential cannot be covered by a single operator. Left sequential in fact coincides with *action prefix* when the first operand is a single action. Action prefix plays its usual role as one of the basic operators of normal forms (see [2]). For right sequential, on the other hand, the situation is more complicated.

The following distributivity and associativity properties are straightforward to establish:

$$(x + y)\leftarrow z = x\leftarrow z + y\leftarrow z$$
$$x\rightarrow(y + z) = x\rightarrow y + x\rightarrow z$$

$$(x\leftarrow y)\leftarrow z = x\leftarrow(y \cdot z)$$
$$x\rightarrow(y\leftarrow z) = (x\rightarrow y)\leftarrow z$$

Disregarding deadlock for the moment, the important remaining problem for the axiom system is to deal with terms of the form $B\rightarrow a$, i.e., right sequential with a single action as its right hand operand. Intuitively, this specifies that a must occur first, followed by the part of B that is left after terminating for a; or, if B does not terminate for a at all, $B\rightarrow a$ is deadlocked. Unfortunately, however, there is no easy way to capture this with the operators discussed so far; for instance, if B is a choice, say between B_1 and

Table 5. Alphabet and partial termination set of $\mathbf{L}^+$ except residue

B	$\mathcal{A}(B)$	$\mathcal{T}(B)$	
$\mathbf{0}_A$	A	$[A]_I$	
a	$\{a\}$	$[a]_I$	
$B_1 + B_2$	$\mathcal{A}(B_1) \cup \mathcal{A}(B_2)$	$\mathcal{T}(B_1) \cup \mathcal{T}(B_2)$	
$B_1 \diamond B_2$	$\mathcal{A}(B_1) \cup \mathcal{A}(B_2)$	$\mathcal{T}(B_1) \cap \mathcal{T}(B_2)$	if $\diamond \in \{\|\|_A, \lfloor\lfloor_A, \lfloor_A, \cdot, \leftarrow, \rightarrow\}$
$\delta_A(B_1)$	$A \cup \mathcal{A}(B_1)$	$[A]_I \cap \mathcal{T}(B_1)$	
$B_1[r]$	$\bigcup_{a \in \mathcal{A}(B_1)} \mathcal{A}(r(a))$	$\mathcal{T}(B_1)$	
X	$\mathcal{A}_X$	$[\mathcal{A}_X]_I$	
$recX.\ B_1$	$\mathcal{A}_X$	$\mathcal{T}(B_1)$	

B_2, then $B \rightarrow a$ cannot be rewritten without a rather extensive case distinction to determine whether B_1 and B_2 terminate or do not terminate for a. (As we saw above, weak sequential composition does not right-distribute over choice; it is precisely the right sequential that is the problem in this regard.)

We solve this problem by introducing, into our axiom system, two further auxiliary notions that precisely capture the partial termination relation. First, Table 4 defines a *residue* operator: $B \downarrow a$ denotes the residue of the term B after it has terminated for a, or a deadlocked term if B cannot terminate for a. Second, Table 5 defines a partial function $\mathcal{T}\colon \mathbf{L}^+ \rightharpoonup \mathbf{2}^{Act}$ that returns the set of actions for which a term, which may itself not contain the residue operator, is partially terminated. In addition, Table 5 extends the function $\mathcal{A}$ returning the alphabet of a term (Table 1) to $\mathbf{L}^+$, again with the exception of the residue operator.

Note that there is a relation between $\mathcal{T}(B)$ and $\mathcal{A}(B)$, in that $[\mathcal{A}(B)]_I \subseteq \mathcal{T}(B)$ for all B, expressing that a term is certainly terminated for all completely independent actions. The following proposition states the crucial property of the partial termination set and the residue operator.

Proposition 5.1 *Let $B \in \mathbf{L}^+$ without residue operator and $a \in Act$. $a \in \mathcal{T}(B)$ if and only if $B \xrightarrow{\surd_a} B'$ for some B', in which case $B' \sim B \downarrow a$.*

Forced deadlock. In the above discussion, we have ignored the termination and deadlock properties of terms. Because of the special nature of partial termination, this is another area that deserves careful attention. For instance, even for completely deadlocked terms (that are unable to perform an action themselves), the alphabet of a term alone is not sufficient to determine its termination properties: one can easily find non-bisimilar deadlocked terms with the same alphabet. Thus, axioms like $x + \mathbf{0}_{\mathcal{A}(x)} = x$ are in general not sound.

Example 5.2 Let $B = \mathbf{0}_{\{a,b\}} + \mathbf{0}_{\{b,c\}}$ and $C = \mathbf{0}_{\{a,c\}} + \mathbf{0}_{\{b,c\}}$ with $d\ I\ \{a, b\}$ and $d\ D\ c$. It follows that B and C are both deadlocked, with $\mathcal{A}(B) = \mathcal{A}(C) = \{a, b, c\}$. However $B \xrightarrow{\checkmark_d}$ whereas $C \not\xrightarrow{\checkmark_d}$; hence $B \not\sim C$.

What we need is some finer method, which also takes choices into account. For this, we introduce a further auxiliary operator, called *deadlock* (also defined in Table 4). The deadlock operator serves a purpose similar to the encapsulation operator of ACP [8]; however, the use of the index set is different. In general, δ_A transforms a term B into a deadlocked term (i.e., $\delta_A(B) \not\xrightarrow{a}$ for all $a \in Act$) with the same termination behaviour as B, except that A is added to the alphabet. The following proposition (proof omitted) formalises this property:

Proposition 5.3 *If $B \in \mathbf{L}^+$ is deadlocked (i.e. $\forall a \in Act\colon B \not\xrightarrow{a}$), then $\delta_A(B) \sim \mathbf{0}_A \cdot B$.*

Using this forced deadlock operator, we can use $x + \delta_A(x) = x$ in place of the unsound $x + \mathbf{0}_{\mathcal{A}(x)} = x$.

Properties of the extended language. We use $\mathbf{L}^{wf+}$ to denote the subset of $\mathbf{L}^+$ for which the same well-formedness conditions hold as for $\mathbf{L}^{wf}$ (see Sect. 2). $\mathbf{L}^{wf+}$ enjoys many of the properties we proved in Sect. 3 for $\mathbf{L}^{wf}$. In particular, the following hold:

- The properties of the alphabet are preserved; that is, Proposition 3.8 can be extended from $\mathbf{L}^{wf}$ to $\mathbf{L}^{wf+}$.
- The properties of partial termination are preserved; that is, Proposition 3.12 can be extended from $\mathbf{L}^{wf}$ to $\mathbf{L}^{wf+}$.
- The congruence property of bisimulation is preserved; that is, Theorem 3.14 can be extended from $\mathbf{L}^{wf}$ to $\mathbf{L}^{wf+}$. The same argument as before can be used (see the proof of Theorem 3.14, Page 210), except for the case of the residue operator, of which the operational rule R_{24} uses look-ahead in the premise. Since this falls outside the GSOS format, for this case we have to revert to the ntyxt/ntyft format proposed in [34]. The congruence argument for the recursion operator, too, is invalid in the presence of look-ahead (see [61]); however, the situation is saved since (as mentioned above) the well-formedness of recursive terms implies the absence of the residue operator inside recursion.

5.2 The equational theory

The set of all relevant axioms is collected in Table 6. Note that several axioms contain side conditions referring to the alphabet or the termination set of terms; such axioms are therefore only applicable in the absence of the residue operator.

Table 6. The equational theory T

$\mathbf{0}_A = \mathbf{0}_{A'}$	if $[A]_I = [A']_I$	C1	$(x + y)\leftarrow z = (x\leftarrow z) + (y\leftarrow z)$		LS1
$x + y = y + x$		C2	$(x\leftarrow y)\leftarrow z = x\leftarrow(y \cdot z)$		LS2
$x + (y + z) = (x + y) + z$		C3	$\mathbf{0}_A\leftarrow x = \delta_A(x)$		LS3
$x + x = x$		C4	$x\rightarrow(y + z) = x\rightarrow y + x\rightarrow z$		RS1
$x + \delta_A(x) = x$		C5	$x\rightarrow(y\leftarrow z) = (x\rightarrow y)\leftarrow z$		RS2
$x \parallel_A y = x \mathbin{\lfloor\!\lfloor}_A y + y \mathbin{\lfloor\!\lfloor}_A x + x \mid_A y$		P	$x\rightarrow a = a\leftarrow(x \downarrow a) + \delta_{\{a\}}(x)$		
$(x + y) \mathbin{\lfloor\!\lfloor}_A z = (x \mathbin{\lfloor\!\lfloor}_A z) + (y \mathbin{\lfloor\!\lfloor}_A z)$		LM1		if $a \in \mathcal{T}(x)$	RS3
$a\leftarrow x \mathbin{\lfloor\!\lfloor}_{A\setminus\{a\}} y = a\leftarrow(x \parallel_{A\setminus\{a\}} y)$		LM2	$x\rightarrow a = \delta_{\{a\}}(x)$	if $a \notin \mathcal{T}(x)$	RS4
$a\leftarrow x \mathbin{\lfloor\!\lfloor}_{A\cup\{a\}} y = \delta_{\{a\}}(x \parallel_{A\cup\{a\}} y)$		LM3	$x\rightarrow \mathbf{0}_A = \delta_A(x)$		RS5
$\mathbf{0}_{A'} \mathbin{\lfloor\!\lfloor}_A x = \delta_{A'}(x)$		LM4	$(x + y) \downarrow a = x \downarrow a + y \downarrow a$		RD1
$x \mid_A y = y \mid_A x$		CM1	$b\leftarrow x \downarrow a = b\leftarrow(x \downarrow a)$		
$(x + y) \mid_A z = x \mid_A z + y \mid_A z$		CM2		if $a \in \mathcal{T}(b\leftarrow x)$	RD2
$a\leftarrow x \mid_{A\cup\{a\}} a\leftarrow y = a\leftarrow(x \parallel_{A\cup\{a\}} y)$		CM3	$b\leftarrow x \downarrow a = \mathbf{0}$	if $a \notin \mathcal{T}(b\leftarrow x)$	RD3
$a\leftarrow x \mid_{A\cup\{a\}} b\leftarrow y = \delta_{\{a,b\}}(x \parallel_{A\cup\{a\}} y)$			$\mathbf{0}_A \downarrow a = \mathbf{0}_A$	if $a\ I\ A$	RD4
	if $a \neq b$	CM4	$\mathbf{0}_A \downarrow a = \mathbf{0}$	if $a\ D\ A$	RD5
$a\leftarrow x \mid_{A\setminus\{a\}} y = \delta_{\{a\}}(x \parallel_{A\setminus\{a\}} y)$		CM5	$\delta_A(\mathbf{0}_{A'}) = \mathbf{0}_{A\cup A'}$		D1
$\mathbf{0}_{A'} \mid_A x = \delta_{A'}(x)$		CM6	$\delta_A(a) = \mathbf{0}_{A\cup\{a\}}$		D2
$\mathbf{1} \cdot x = x$		S1	$\delta_A(x \diamond y) = \delta_A(x) \diamond \delta_A(y)$		
$x \cdot \mathbf{1} = x$		S2		where $\diamond \in \{+, \leftarrow\}$	D3
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$		S3	$a[r] = r(a)$		RF1
$x \cdot y = x \parallel_\emptyset y$	if $\mathcal{A}(x)\ I\ \mathcal{A}(y)$	S4	$\mathbf{0}_A[r] = \mathbf{0}_A$		RF2
$x \cdot y = x\leftarrow y + x\rightarrow y$		S5	$(x_1 \diamond x_2)[r] = x[r] \diamond y[r]$		
				where $\diamond \in \{+, \cdot, \leftarrow\}$	RF3

In addition to those already discussed at the introduction of the auxiliary operators, Table 6 includes several more axioms concerning sequential composition. For instance, S1–S3 express that sequential composition imposes a monoid structure on the language, with $\mathbf{1}$ as a neutral element; this is as expected and coincides with the standard axioms for ACP (see [7]). Another interesting axiom is S4, which expresses that for two terms whose alphabet is independent, sequential composition and (synchronisation free) parallel composition have the same effect.

The following theorem states the soundness of the axioms. It implies that we may use the equational theory T generated by Table 6 together with

the usual equational proof rules to derive bisimilarity of terms in $\mathbf{L}^{wf+}$. The proof can be found in Appendix A (Page 226).

Theorem 5.4 (soundness) *The axioms of Table 6 are sound w.r.t. bisimilarity.*

Our axiom system is also *complete*, i.e., rich enough to derive all bisimilarities of finite terms. We prove this in the usual way, by reducing all terms to a small core language, which in this case consists of action prefix, choice and termination constants, and showing completeness for this core language. We denote the core language by $\mathbf{L}_t$ (for *tree* language); it is generated by the following grammar:

$$B ::= \mathbf{0}_A \mid a{\leftarrow}B \mid B + B \ .$$

Completeness of the core language comes down to the following property.

Proposition 5.5 *For all $B_1, B_2 \in \mathbf{L}_t$, $B_1 \sim B_2$ implies $\mathsf{T} \vdash B_1 = B_2$.*

The proof, the full details of which can be found in Appendix A (Page 230), uses a sum normal form for terms of $\mathbf{L}_t$. Using D1–D3, it is not difficult to show that for all $B \in \mathbf{L}_t$, $\mathsf{T} \vdash \delta_{Act}(B) = \mathbf{0}_{Act}$; hence with the help of C5 one can deduce $B + \mathbf{0} = B$. Therefore, C2–C5 together imply that we may use the standard notation for sums: $\sum_{i\in I} B_i$ (where I is finite) denotes the choice between all B_i, which equals B_n if $I = \{n\}$ and $\mathbf{0}$ if $I = \emptyset$. It follows that each term $B \in \mathbf{L}_t$ can be written in the form

$$B = \textstyle\sum_{i\in I} a_i{\leftarrow}B_i + \sum_{j\in J} \mathbf{0}_{A_j} \ ,$$

where $B_i \in \mathbf{L}_t$ for all $i \in I$. The proof of Proposition 5.5 proceeds by induction on the *depth* of tree terms in sum form, which is defined by

$$\mathit{depth}\left(\textstyle\sum_{i\in I} a_i{\leftarrow}B_i + \sum_{j\in J} \mathbf{0}_{A_j}\right) = \max\{1 + \mathit{depth}(B_i) \mid i \in I\}$$

(where $\max\emptyset$ is assumed to equal 0). As an intermediate step in the proof of Proposition 5.5, we first prove a special case for deadlock constants:

Lemma 5.6 *Let $B \in \mathbf{L}_t$. If $B + \mathbf{0}_A \sim B$, then $\mathsf{T} \vdash B + \mathbf{0}_A = B$.*

Using this, one can show that for arbitrary $B_1, B_2 \in \mathbf{L}_t$, $B_1 \sim B_2$ implies $\mathsf{T} \vdash B_1 + B_2 = B_2$ (by induction on $\mathit{depth}(B_1 + B_2)$). By symmetry, this implies Proposition 5.5.

Normalisation. The second part of the completeness result consists of showing that every term of $\mathbf{L}^{wf}$ can be rewritten (using the equational theory T) to a term of the core language. This is a consequence of the following proposition.

Proposition 5.7 *Let $B_1, B_2 \in \mathbf{L}_t$, and let $r\colon Act \to \mathbf{L}_t$ be strongly D-consistent.*

1. $\mathsf{T} \vdash a = a \leftarrow \mathbf{1}$.
2. $\mathsf{T} \vdash B_1 \cdot B_2 = C$ *for some* $C \in \mathbf{L}_t$.
3. $\mathsf{T} \vdash B_1 \parallel_A B_2 = C$ *for some* $C \in \mathbf{L}_t$.
4. $\mathsf{T} \vdash B_1[r] = C$ *for some* $C \in \mathbf{L}_t$.

In order to prove this, we first have to establish similar results for the (other) auxiliary operators:

Lemma 5.8 *Let $B \in \mathbf{L}_t$ be arbitrary.*

1. $\mathsf{T} \vdash \delta_A(B) = C$ *for some* $C \in \mathbf{L}_t$ *with* $depth(C) = 0$.
2. $\mathsf{T} \vdash B \downarrow a = C$ *for some* $C \in \mathbf{L}_t$ *with* $depth(C) \leq depth(B)$.
3. $\mathsf{T} \vdash B \rightarrow a = C$ *for some* $C \in \mathbf{L}_t$ *with* $depth(C) \leq 1 + depth(B)$.

The proof of Lemma 5.8 and Proposition 5.7 can be found in Appendix A (Page 230). The main completeness result is now straightforward to prove.

Theorem 5.9 (completeness) *For all $B_1, B_2 \in \mathbf{L}^{wf}$, $B_1 \sim B_2$ implies* $\mathsf{T} \vdash B_1 = B_2$.

Proof. According to Proposition 5.7, there are terms $B_1', B_2' \in \mathbf{L}_t$ such that $\mathsf{T} \vdash B_1 = B_1'$ and $\mathsf{T} \vdash B_2 = B_2'$. Since all axioms of T are sound (Theorem 5.4), it follows that $B_1' \sim B_2'$; hence $\mathsf{T} \vdash B_1' = B_2'$ according to Proposition 5.5. Combining these equalities, we obtain $\mathsf{T} \vdash B_1 = B_2$. □

6 Applications

We discuss some small examples from the fields of protocols and data-bases to illustrate the applicability of our approach, especially the usefulness of weak sequential composition and action refinement, and the interaction of sequential composition with choice. First, we show that the data transfer and release phases of a toy protocol can be specified sequentially and nevertheless be allowed a potential overlap, due to the fact that the release is not instantaneous and data packets may still be underway. Then, we reconstruct the *communication closed layers* principle from [27] (advocated in [41]) in our setting and apply it to another toy version of a data transfer protocol. Finally, we show how to refine a data base update action without blocking simultaneous requests – an example which was inspired by [13].

6.1 Connection release

We consider a small protocol for connection-oriented data transfer between two parties. The example is inspired by Goltz and Götz [33]. The protocol consists of three phases: *connection establishment*, *data transfer* and

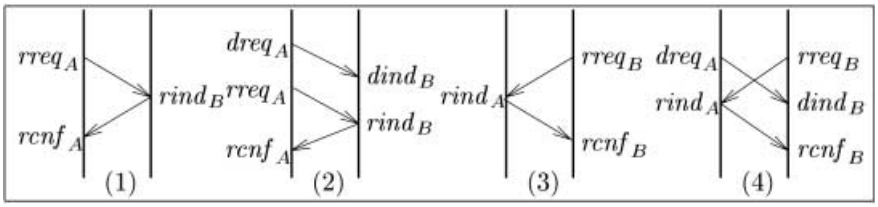


Fig. 2. Possible interactions of data and release phase

connection release. Here we concern ourselves only with the interaction between the data transfer and release phases, respectively specified by terms $Data$ and Rel. On the top level, the specification is given by

$$Prot = Data \cdot Rel \ .$$

This reflects the idea that there is a natural ordering, based on the fact that after connection release no data can be transferred any more. However, because of the typically distributed nature of protocol systems, it is in general very difficult to rule out that there are still packets underway when the connection release is initiated; hence some actions from the data phase may take place only after the release phase has started (but not after it has finished). In a traditional process algebra, such an overlap would contradict the specified sequential ordering of the two phases.

In order to keep the formalisation within bounds, let us assume that data is only transferred from party A to party B, and the transfer of one data item consists of two actions, $dreq_A$ and $dind_B$ for *data request* and *data indication* taking place at A and B, respectively. Data transfer is unconfirmed. As a further simplification, we model just one possible data transfer. The release phase can be initiated by either A or B by a *release request* $rreq_A$ or $rreq_B$, which is indicated at the other end by a *release indication* $rind_B$ or $rind_A$, and confirmed by a *release confirm* $rcnf_A$ or $rcnf_B$. This behaviour can be specified by the following terms:

$$\begin{aligned} Data &= \mathbf{1} + dreq_A \cdot dind_B \\ Rel &= rreq_A \cdot rind_B \cdot rcnf_A + rreq_B \cdot rind_A \cdot rcnf_B \ . \end{aligned}$$

The four possible global scenarios for the behaviour of this system are depicted in Fig. 2, in the form of message sequence charts. Note that in scenario (4), the data indication $dind_B$ can take place before or after the release request $rreq_B$; however, after a release confirm, no data can arrive any more.

Consider the dependencies between the actions. The local actions of each party are dependent with the exception of $dind_B$ and $rreq_B$; the idea here

is that party B cannot know if there is an incoming data indication or not, and hence this cannot influence whether or not B will request release. In addition, each indication should be dependent on the corresponding request, and the confirmation on the indication.

Now we can analyse the behaviour of this protocol. Its first transition is either a data request (by A) or a release request (by A or B). If it is a data request then

$$Prot = Data \cdot Rel \xrightarrow{dreq_A} \mathbf{1} \cdot dind_B \cdot Rel$$

which corresponds to scenario (2) or (4) of Fig. 2. Which of these two is chosen depends on who initiates the release. Despite the syntactic structure of the specification, it is not necessarily the case that $dind_B$ be the next action to occur. In fact, both $rreq_A$ and $rreq_B$ are already enabled in $dind_B \cdot Rel$, and so $dind_B$ can be delayed for several steps:

$$dind_B \cdot Rel \xrightarrow{rreq_B} dind_B \cdot \mathbf{1} \cdot rind_A \cdot rcnf_B \xrightarrow{rind_A} dind_B \cdot \mathbf{1} \cdot \mathbf{1} \cdot rcnf_B \ .$$

At this stage, finally, the data indication must take place, followed by the release confirmation.

On the other hand, if the first action of $Prot$ is the release request from B, then (because $Data \xrightarrow{\checkmark_{rreq_B}} Data$) the choice in the data phase is not resolved by this: we get

$$Prot \xrightarrow{rreq_B} Data \cdot \mathbf{1} \cdot rind_A \cdot rcnf_B \ ,$$

corresponding to scenario (3) or (4) in Fig. 2. The next action will decide between these scenarios: it is either $dreq_A$ or $rind_A$, the latter of which *does* decide the choice in $Data$:

$$Data \cdot rind_A \cdot rcnf_B \xrightarrow{rind_A} \mathbf{1} \cdot \mathbf{1} \cdot rcnf_B \ .$$

Note that the absence of right-distributivity of choice over (weak) sequential composition is important here. If we distribute Rel over the choice in $Data$, we obtain the alternative protocol

$$Prot' = Rel + dreq_A \cdot dind_B \cdot Rel \ .$$

This specifies a different protocol: in $Prot'$, an initial $rreq_B$-action automatically resolves the choice and implies that no data transfer takes place, and thus $dreq_A$ may be refused afterwards:

$$Prot' \xrightarrow{rreq_B} \mathbf{1} \cdot rind_A \cdot rcnf_B \ .$$

An initial $rreq_B$-action in $Prot$ on the other hand leaves two possibilities for the future behaviour: either the release request is immediately indicated by

party A or party A may still send new data which then has to be accepted by B. To further illustrate this difference, we use the axiomatisation to rewrite $Prot$ into a form which makes this visible:

$$\begin{aligned} Prot = {} & rreq_A \cdot rind_B \cdot rcnf_A && \text{(i)} \\ & + dreq_A \cdot dind_B \cdot rreq_A \cdot rind_B \cdot rcnf_A && \text{(ii)} \\ & + rreq_B \cdot (rind_A \cdot rcnf_B + dreq_A \cdot dind_B \cdot rind_A \cdot rcnf_B) && \text{(iii)} \end{aligned}$$

We show only part of the derivation of this equality. We start by proving that all weak sequential composition operators in $Prot$ can be replaced by left sequential. For this, we show that in general $a \cdot b = a \leftarrow b$ if $a \; D \; b$. First note that

$$\begin{aligned} a \rightarrow b &\overset{\text{RS4}}{=} \delta_{\{b\}}(a) \overset{\text{D2}}{=} \mathbf{0}_{\{a,b\}} \overset{\text{D1}}{=} \delta_{\{a\}}(\mathbf{0}_{\{b\}}) \\ &\overset{\text{LS3}}{=} \mathbf{0}_{\{a\}} \leftarrow \mathbf{0}_{\{b\}} \overset{\text{D2}}{=} \delta_{\emptyset}(a) \leftarrow \delta_{\emptyset}(b) \overset{\text{D3}}{=} \delta_{\emptyset}(a \leftarrow b) \; . \end{aligned}$$

This implies

$$a \cdot b \overset{\text{S5}}{=} a \leftarrow b + a \rightarrow b = a \leftarrow b + \delta_{\emptyset}(a \leftarrow b) \overset{\text{C5}}{=} a \leftarrow b \; .$$

Generalising this to weak sequential compositions with more than two components, we can rewrite $Prot$ to $Data' \cdot Rel'$ where

$$\begin{aligned} Data' &= \mathbf{1} + dreq_A \leftarrow dind_B \\ Rel' &= rreq_A \leftarrow (rind_B \leftarrow rcnf_A) + rreq_B \leftarrow (rind_A \leftarrow rcnf_B) \end{aligned}$$

This is now the starting point for showing the equality advocated above.

$$\begin{aligned} Data' \cdot Rel' &= Data' \leftarrow Rel' + Data' \rightarrow Rel' && \text{(S5)} \\ Data' \leftarrow Rel' &= \mathbf{1} \leftarrow Rel' + (dreq_A \leftarrow dind_B) \leftarrow Rel' && \text{(LS1)} \\ &= \delta_{\emptyset}(Rel') + dreq_A \leftarrow (dind_B \cdot Rel') && \text{(LS3, LS2)} \end{aligned}$$

The first component of the choice will later be absorbed by the rest of the term with the help of axiom C5, the second component is already nearly part (ii) of our target term. Thus we now only look at the other part of our current term: $Data' \rightarrow Rel'$.

$$\begin{aligned} Data' \rightarrow Rel' &= Data' \rightarrow (rreq_A \leftarrow rind_B \leftarrow rcnf_A) \\ &\quad + Data' \rightarrow (rreq_B \leftarrow rind_A \leftarrow rcnf_B) && \text{(RS1)} \\ &= (Data' \rightarrow rreq_A) \leftarrow rind_B \leftarrow rcnf_A \\ &\quad + (Data' \rightarrow rreq_B) \leftarrow rind_A \leftarrow rcnf_B && \text{(RS2)} \\ & && (*) \end{aligned}$$

The interesting part is now $(*)$, it describes the behaviour of the protocol after an initial $rreq_B$. Thus we only take a further look at this part.

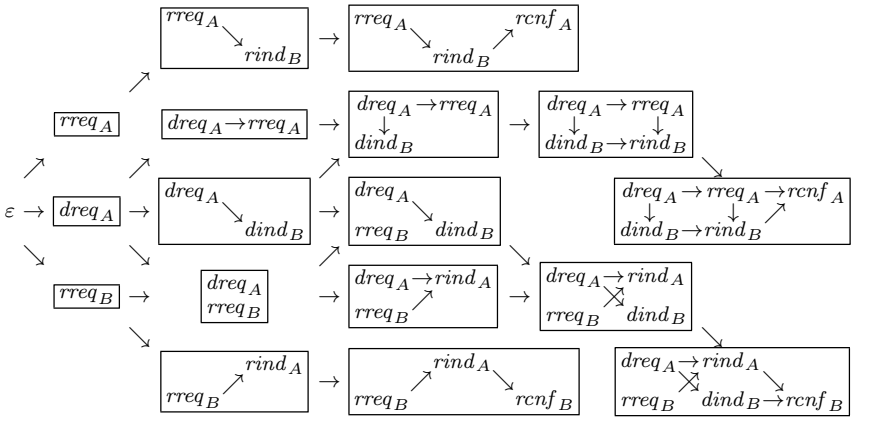
$$\begin{aligned}(*) &= \big(rreq_B{\leftarrow}(Data' \downarrow rreq_B) + \delta_{\{rreq_B\}}(Data')\big) \\ &\quad {\leftarrow} rind_A {\leftarrow} rcnf_B && \text{(RS3)} \\ Data' \downarrow rreq_B &= \mathbf{1} \downarrow rreq_B + (dreq_A{\leftarrow}dind_B) \downarrow rreq_B && \text{(RD1)} \\ &= \mathbf{1} + dreq_A{\leftarrow}(dind_B \downarrow rreq_B) && \text{(RD2, RD4)} \\ &= \mathbf{1} + dreq_A{\leftarrow}dind_B{\leftarrow}(\mathbf{1} \downarrow rreq_B) && \text{(RD2)}\end{aligned}$$

Using $\mathbf{1}\downarrow rreq_B = \mathbf{1}$ (RD4) and $\delta_{\{rreq_B\}}(Data') = \mathbf{0}_{rreq_B}$ (D1–3, C5, LS3) we obtain

$$\begin{aligned}(*) &= \big(rreq_B{\leftarrow}(\mathbf{1} + dreq_A{\leftarrow}dind_B) + \mathbf{0}_{rreq_B}\big) \\ &\quad {\leftarrow} rind_A {\leftarrow} rcnf_B \\ &= \big(rreq_B{\leftarrow}(\mathbf{1} + dreq_A{\leftarrow}dind_B)\big){\leftarrow} rind_A {\leftarrow} rcnf_B \\ & && \text{(LS3, D1–3, C5)}\end{aligned}$$

This can then be rewritten to part (iii) of our target term, which described the behaviour of $Prot$ after $rreq_B$.

Although the operational semantics is the simplest and most tractable, it is also interesting to see for once the denotational model for a small specification like $Prot$. Leaving out the event identities and termination sets for the sake of simplicity, the following graph depicts the system runs of $[\![Prot]\!]$, ordered by prefix:



It should be noted that this example did not rely on action refinement. It would be interesting to initially regard the data and release phases, $Data$ and Rel, as single actions, sequentially composed in $Prot$, and afterwards,

in the next design step, refine them into their respective definitions above, using

$$\begin{aligned} r\colon \quad & Data \mapsto \mathbf{1} + dreq_A \cdot dind_B \\ & Rel \mapsto rreq_A \cdot rind_B \cdot rcnf_A + rreq_B \cdot rind_A \cdot rcnf_B \ . \end{aligned}$$

However, this is unfortunately not compatible with the requirement of strong D-consistency we have imposed on refinement functions (see Sect. 3.1): clearly $Data\ D\ Rel$ and therefore it should be the case that $Data\ D\ a$ for any initial action a of Rel; yet $Data$ is (partially) terminated for the initial action $rreq_B$ of Rel, since $Rel \xrightarrow{rreq_B}$ and $Data \xrightarrow{\checkmark_{rreq_B}}$.

6.2 Communication closed layers

An algebraic law that has been quite successfully applied in a linear time setting is the *communication closed layers* law (CCL), originally due to Elrad and Francez [27] and advocated for instance by Zwiers et al. in [39, 74,28], also working with action dependencies. In our setting, CCL can be formulated as follows (with some side conditions, which we omit for the time being):

$$\begin{pmatrix} B_1 \parallel_{A_1} C_1 \\ \cdot \\ B_2 \parallel_{A_2} C_2 \end{pmatrix} = \begin{pmatrix} B_1 \\ \cdot \\ B_2 \end{pmatrix} \parallel_{A_1 \cup A_2} \begin{pmatrix} C_1 \\ \cdot \\ C_2 \end{pmatrix} \ . \tag{3}$$

CCL is used to facilitate the development of distributed system by enabling a *transformational* design: initially, the system or algorithms can be specified as a number of sequential phases or *layers* (which is probably close to the designers' conception of the system), each of which consists of a number of cooperating distributed entities; CCL then allows this specification to be transformed into a behaviourally equivalent parallel composition. In Eq. (3), the individual layers are given by the terms $B_i \parallel_{A_i} C_i$ on the left hand side: the B_i and C_i are the distributed entities within the layer, which cooperate through their communication over A_i and through their respective action dependencies. On the right hand side, we see that all the B_i are composed sequentially into one component of a parallel composition, and so are the C_i.

The two views are equivalent only if the entities from *different* layers cannot interfere. This is the requirement of *communication closedness*; in our setting, it is expressed by the following conditions (for all $i \neq j$):

- different entities of different layers are mutually independent, i.e., the actions in $\mathcal{A}(B_i)$ are independent of those in $\mathcal{A}(C_j)$;
- different layers do not synchronise; i.e., $\mathcal{A}(B_i) \cap A_j = \mathcal{A}(C_i) \cap A_j = \emptyset$.

Communication closedness is necessary for (3) to hold. Recall that $A_1 \; I \; A_2 \iff \forall a_1 \in A_1, a_2 \in A_2.\; a_1 \; I \; a_2$; then the formal statement of CCL is as follows.

Theorem 6.1 *If $B_i, C_i \in \mathbf{L}$ and $A_i \subseteq Act$ for $i = 1, 2$ such that for all $i \neq j$*

- *$\mathcal{A}(B_i) \; I \; \mathcal{A}(C_j)$, and*
- *$\mathcal{A}(B_i) \cap A_j = \mathcal{A}(C_i) \cap A_j = \emptyset$*

then (3) holds up to strong bisimilarity.

This theorem thus generalises the results of Zwiers et al. [39,74,28] to a branching time setting. The proof is again in Appendix A (Page 232). The theorem can easily be generalised to arbitrarily many layers:

Corollary 6.2 *If $B_i, C_i \in \mathbf{L}$ and $A_i \subseteq Act$ for $1 \leq i \leq n$ such that for all $i \neq j$*

- *$\mathcal{A}(B_i) \; I \; \mathcal{A}(C_j)$, and*
- *$\mathcal{A}(B_i) \cap A_j = \mathcal{A}(C_i) \cap A_j = \emptyset$*

then the following holds up to strong bisimilarity:

$$\begin{pmatrix} B_1 \parallel_{A_1} C_1 \\ \vdots \\ B_n \parallel_{A_n} C_n \end{pmatrix} = \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix} \parallel_{A_1 \cup \cdots \cup A_n} \begin{pmatrix} C_1 \\ \vdots \\ C_n \end{pmatrix} \quad .$$

Proof. By induction on n. The case for $n = 1$ is trivial, and the case for $n = 2$ was proved in Theorem 6.1. Now consider the case up to $n-1$ proven ($n > 2$); then

$$\begin{aligned} \begin{pmatrix} B_1 \parallel_{A_1} C_1 \\ \vdots \\ B_n \parallel_{A_n} C_n \end{pmatrix} &= \left(\begin{array}{c} \begin{pmatrix} B_1 \\ \vdots \\ B_{n-1} \end{pmatrix} \parallel_{A_1 \cup \cdots \cup A_{n-1}} \begin{pmatrix} C_1 \\ \vdots \\ C_{n-1} \end{pmatrix} \\ \cdot \\ B_n \parallel_{A_n} C_n \end{array} \right) \\ &= \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix} \parallel_{A_1 \cup \cdots \cup A_n} \begin{pmatrix} C_1 \\ \vdots \\ C_n \end{pmatrix} \end{aligned}$$

where the first equality holds by the induction hypothesis, and the second by Theorem 6.1. □

Application. We now show an application of CCL. Consider a data phase consisting of $n \geq 1$ data transfers, each initially specified by a single action $data_n$. The initial specification of the entire data phase is

$$Data = data_1 \cdot \ldots \cdot data_n \ .$$

The $data_i$ are dependent actions; i.e., $data_i \ D \ data_j$ for all $1 \leq i, j \leq n$. In a first design step, the actions are refined as follows:

$$r\colon data_i \mapsto prod_i \cdot dreq_i \cdot dind_i \cdot cons_i \quad (1 \leq i \leq n) \ .$$

Here, $prod_i$ is an action of the sending party which *produces* the ith data, the $dreq_i$ and $dind_i$ are data requests and indications as in Sect. 6.1, which convey the data over the medium from the sending to the receiving party, and $cons_i$ is an action of the receiving party which *consumes* the ith data. We assume that produce and consume actions are independent of each other, and so are all requests and indications of different layers. Moreover, production and data indication, as well as data request and consumption, which take place at different parties, are independent as well. Summarising, D is the reflexive and symmetric closure of the relation generated by

$$\begin{array}{llll} prod_i \ D \ prod_{i+1} & prod_i \ D \ dreq_i & dreq_i \ D \ dind_i \\ dind_i \ D \ cons_i & cons_i \ D \ cons_{i+1}. \end{array}$$

Note that the refinement function r is strongly D-consistent. The behaviour of the refined data phase, specified by $Data[r]$, is not strictly sequential: due to the independencies, the production and data sending of one transfer can overlap with the data consumption of previous transfers. For instance, if $n = 2$ then

$$\begin{aligned} Data[r] &\xrightarrow{prod_1} (dreq_1 \cdot dind_1 \cdot cons_1) \cdot data_2[r] \\ &\xrightarrow{dreq_1} (dind_1 \cdot cons_1) \cdot data_2[r] \\ &\xrightarrow{prod_2} (dind_1 \cdot cons_1) \cdot (dreq_2 \cdot dind_2 \cdot cons_2) \cdot \mathbf{1}[r] \\ &\xrightarrow{dreq_2} (dind_1 \cdot cons_1) \cdot (dind_2 \cdot cons_2) \cdot \mathbf{1}[r] \end{aligned}$$

after which the data indication for all send data has to take place and finally the data are consumed in the order of production. The denotational model of $Data[r]$ consists of a single maximal run and its prefixes, depicted in Fig. data3. The dashed lines indicate the refinements of the individual data transfer actions.

Now we want to transform the refined specification into one which is composed "vertically", that is, in which the roles of the sending and receiving

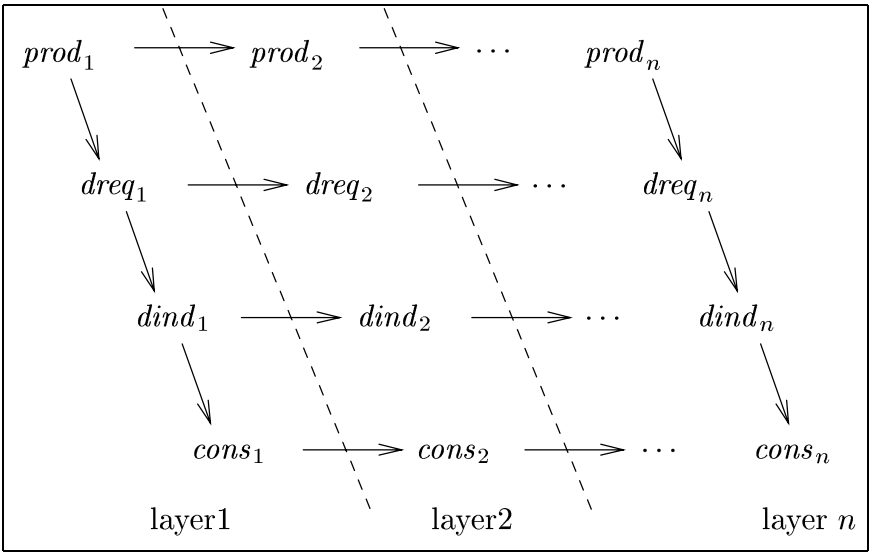


Fig. 3. Data transfer phase consisting of n layers

parties and that of the channel are distinguished. First we transform the individual data transfers. Let r' be a new refinement function given by

$$r' \colon data_i \mapsto ((prod_i \cdot dreq_i) \,|||\, (dind_i \cdot cons_i)) \,\|_{dreq_i, dind_i}\, (dreq_i \cdot dind_i) \\ \times\ (1 \leq i \leq n).$$

For clarity, we introduce auxiliary names $Send_i = prod_i \cdot dreq_i$, $Rec_i = dind_i \cdot cons_i$, $Chan_i = dreq_i \cdot dind_i$ and $A_i = \{dreq_i, dind_i\}$; this allows us to write

$$r' \colon data_i \mapsto (Send_i \,|||\, Rec_i) \,\|_{A_i}\, Chan_i \quad (1 \leq i \leq n) \ .$$

$r(data_i)$ can be rewritten to $r'(data_i)$ using the proof system developed in Sect. 5, using especially the expansion axioms for parallel composition (P, LM1–5 and CM1–6) and the axioms for choice (C1–5). It follows that $r(data_i) \sim r'(data_i)$ for all $1 \leq i \leq n$; since bisimulation is a congruence for refinement (Theorem 3.14), we may conclude

$$Data[r] \sim Data[r'] \ .$$

Repeatedly using Axiom RF3 with $\diamond = \ \cdot$ and RF1, we obtain

$$Data[r'] \sim \begin{pmatrix} (Send_1 \,|||\, Rec_1) \,\|_{A_1}\, Chan_1 \\ \vdots \\ (Send_n \,|||\, Rec_n) \,\|_{A_n}\, Chan_n \end{pmatrix} \ .$$

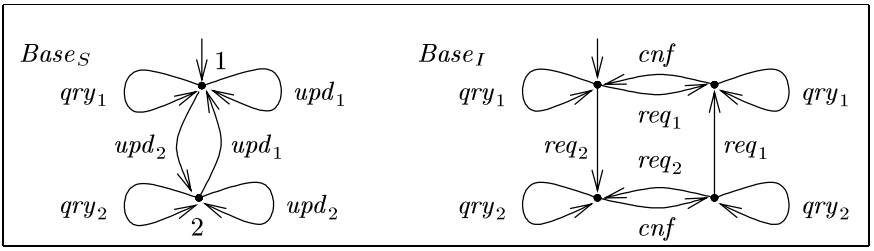


Fig. 4. Specification and desired implementation of a 2-state data base

The phases $(Send_i \;|||\; Rec_i) \parallel_{A_i} Chan_i$ are communication closed: for all $i \neq j$,

- $\mathcal{A}(Send_i \;|||\; Rec_i) = \{prod_i, dreq_i, dind_i, cons_i\} \;\; I \;\; \{dreq_j, dind_j\} = \mathcal{A}(Chan_j)$;
- $\mathcal{A}(Data_i) \cap A_j = \emptyset$.

Hence the conditions of Corollary 6.2 are fulfilled, implying

$$Data[r'] \sim \begin{pmatrix} (Send_1 \;|||\; Rec_1) \\ \vdots \\ (Send_n \;|||\; Rec_n) \end{pmatrix} \parallel_A \begin{pmatrix} Chan_1 \\ \vdots \\ Chan_n \end{pmatrix}$$

where $A = \bigcup_{1 \leq i \leq n} A_i$. The left hand side can in turn be subjected to CCL, since $\mathcal{A}(Send_i) \; I \; \mathcal{A}(Rec_j)$ for all $i \neq j$; hence we have

$$Data[r] \sim \left(\begin{pmatrix} Send_1 \\ \vdots \\ Send_n \end{pmatrix} ||| \begin{pmatrix} Rec_1 \\ \vdots \\ Rec_n \end{pmatrix} \right) \parallel_A \begin{pmatrix} Chan_1 \\ \vdots \\ Chan_n \end{pmatrix} \quad .$$

The right hand side has clearly recognisable subterms describing the behaviour of sender, receiver and channel, and can therefore be mapped directly on an implementation architecture.

6.3 Data base access

Finally, we apply our theory to a small example inspired by Brinksma, Jonsson and Orava [13]. This example shows that dependency-based specification and action refinement allows a design strategy where sequentially specified abstract actions can be implemented in an overlapping fashion, when this is consistent with their mutual dependencies. This is a clear advantage over standard action refinement.

The example concerns a distributed data base that can be queried and updated. We assume that the data base has only two possible states, which we denote 1 and 2. Querying is done using actions qry_i for $i = 1, 2$, where the index i models the return value; updating is done using action upd_i for $i = 1, 2$, where the index denotes the new state of the data base. The data base specification is modelled by the transition system $Base_S$ in Fig. 4.

The problem considered in [13] is to change the interface of the data base, so that updating consists not of a single action but of two successive stages, in which the update is *requested* (using req_i for $i = 1, 2$) and *confirmed* (using cnf), respectively. Moreover, it is required that in the meantime (between request and confirmation), querying the data base should still be possible. This behaviour is modelled by $Base_I$ in Fig. 4.

In our approach, this implementation can be obtained algebraically through an application of the refinement operator, with refinement function

$$r\colon\ upd_i \mapsto req_i; cnf \quad (i = 1, 2).$$

The overlap between qry_i and cnf is obtained by setting the dependencies appropriately: $qry_i\ D\ req_j$ but $qry_i\ I\ cnf$. Note that r is strongly D-consistent. Let D_i stand for the term describing the behaviour of the data base in state i (where $i = 1, 2$); i.e.,

$$\begin{aligned} D_1 &= qry_1{\cdot}D_1 + upd_1{\cdot}D_1 + upd_2{\cdot}D_2 \\ D_2 &= qry_2{\cdot}D_2 + upd_2{\cdot}D_2 + upd_1{\cdot}D_1 \ . \end{aligned}$$

The specification and implementation shown in Fig. 4 are then obtained modulo bisimulation as the semantics of

$$\begin{aligned} Base_S &= D_1 \\ Base_I &= Base_S[r] \ . \end{aligned}$$

More precisely, the operational behaviour of $Base_I$ is depicted (modulo Axiom S1 of Table 6, which states that $1 \cdot t = t$ for arbitrary t) by the left hand transition system in Fig. 5. The right hand system of Fig. 5 shows the case where $qry_i\ D\ cnf$ instead, in which case the next query must wait for the second phase of the updating to finish.

7 Conclusion

7.1 Summary

We briefly summarise the main achievements of this work. We have defined a process algebra with a built-in notion of *dependency* among actions, thus

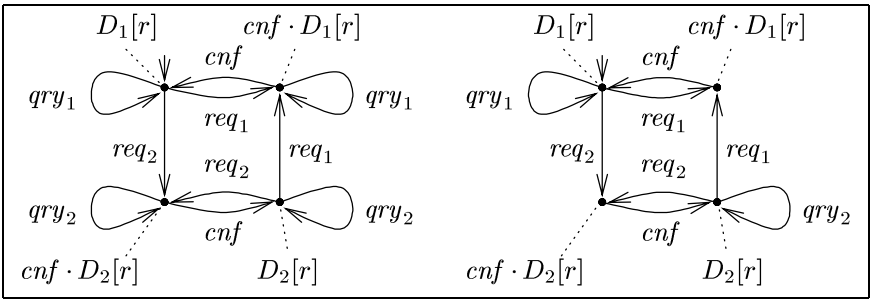


Fig. 5. Refinement of $Base_S$ with $qry_i\ I\ cnf$ (left) and $qry_i\ D\ cnf$ (right)

giving a rudimentary form of semantics to the otherwise uninterpreted actions. Dependencies influence the ordering among the actions in a process, and thereby the interpretation of the operators. In particular the combination of action refinement with dependencies turns out to be an interesting concept, which – as demonstrated in the previous section – can be useful in the hierarchical design of specifications.

For this process algebra we developed semantics using several consistent approaches: an interleaving operational semantics, a causality based denotational semantics and an axiomatisation with respect to bisimulation equivalence. These semantics on the one hand provide us with different ways of representing and verifying properties about processes and on the other hand were used to validate their correctness against each other. All three are *branching time* semantics, which faithfully reflect the moments of choice. The precise modelling of branching points was achieved by introducing a new concept of *partial termination*, with the noteworthy feature that it may resolve choices. This feature, which is a natural consequence of the concept of dependencies, nevertheless complicates the semantics quite a bit.

The (toy) examples in the previous section showed how the theory can in principle be applied in system design, for instance of telecommunication protocols or database access. The theory can however also be useful to give precise semantics to other specification methods. An example of this is [31] in which a formal semantics to Message Sequence Charts (MSCs) – a standardised language for specifying message passing systems – by means of a process algebra with action dependencies is given. Interestingly, Message Sequence Charts also allow to specify global choices, thus precisely the concept of partial termination developed here is needed for their semantics. A semantics for Interworkings also relying on our weak sequential composition can be found in [44], and similar for High-level-MSCs in [45].

To get rid of these complications and still have a consistent and intuitively correct semantics, one could restrict oneself to *local* choices, i.e.,

occurrences of the operator $t_1 + t_2$ where the dependencies of t_1 and t_2 are the same. (This is advocated in Huhn [36,37], in the dual setting of *localities* rather than dependencies.) On the other hand, it is precisely this effect of the resolution of choice by partial termination that has allowed the straightforward modelling of certain features of MSCs in [31,44,45].

7.2 Extensions

Invisible actions. Our process algebra does not incorporate a notion of invisible action. While neither the (CCS-like) choice nor the synchronisation introduce invisible actions, as soon as one adds an operator for *hiding*, invisible actions come into play. The question then arises of how to set the dependencies. In principle, there are at least three possibilities for this:

- The invisible action is dependent on all visible actions;
- The invisible action is independent of all visible actions;
- There is a *family* of invisible actions, indexed with sufficient information to reconstruct the dependencies of the original (hidden) action.

Since hiding should not alter the ordering of actions within a process, the third possibility seems the only feasible one.

Data. Several extensions to process algebras with data exist; the best know example is LOTOS (see [11] for an introduction or [38] for the full standard). It should be possible to smoothly integrate data into our process algebra. However, while so far action dependencies are a priori given, in a setting with data (actions reading or writing variables) dependencies have to be *derived* since then actions already have a semantics. A simple method could make all actions dependent that access the same resources (e.g. the same set of variables). Similar methods for computing dependencies can be found in the work on partial order reductions, which also rely on a notion of dependency leading to commutation of independent actions (see [55] for an overview). For example, the model-checking tool SPIN contains a partial-order package which automatically computes independencies.

7.3 Related work

We briefly recapitulate related work. The approach closest to ours is the one of Janssen, Poel and Zwiers [39,40], who study a process algebra with similar operators and dependencies in a linear time setting. Their process algebra contains both a sort of weak sequential composition (called layered composition) and a dependency-based refinement. While we study three branching time semantics in this paper, they just define a denotational linear

time semantics. Nevertheless, their work already shows the usefulness of dependencies, and in particular, of a dependency-based action refinement, in the design of distributed systems. Other approaches to design by refinement allowing an overlap of refinements of sequential actions, but not based on dependencies, can be found in [23, 59, 62]. A dependency-based sequential composition (in a linear time setting) can also be found in [30] (therein called D-local concatenation).

Furthermore, the notion of dependency is (of course) central in the huge amount of work around Mazurkiewicz traces (for a recent overview see [26]), which we cannot fully discuss here. In particular, however, in this context a notion of termination similar to ours has been proposed in [25].

A process algebra with a notion of *location* which naturally induces dependencies can be found in [36, 37], together with a suitable logic to reason about such specifications. (This is not to be confused with the location-based approach to semantics investigated in for instance [12, 1]: there, locations play an entirely different role, where they are derived from the process terms rather than a priori associated with actions.)

Acknowledgements. Many thanks to Michel Reniers for pointing us to an unsound axiom in the earlier version [63], and especially to Walter Vogler for many insightful comments, including a flaw in an earlier version of Theorem 6.1.

References

1. Luca Aceto. A static view of localities. Formal Aspects of Computing, 6:201–222, 1994
2. Luca Aceto, Bard Bloom, Frits W. Vaandrager. Turning SOS rules into equations. Information and Computation, 111(1):1–52, May 1994. LICS '92 Special Issue
3. Luca Aceto, Matthew C. B. Hennessy. Towards action-refinement in process algebras. Information and Computation, 103:204–269, 1993
4. Luca Aceto, Matthew C. B. Hennessy. Adding action refinement to a finite process algebra. Information and Computation, 115:179–247, 1994
5. Eric Badouel, Philippe Darondeau. On guarded recursion. Theoretical Computer Science, 82:403–408, 1991
6. Jos C. M. Baeten, Rob J. van Glabbeek. Abstraction and empty process in process algebra. Fundamenta Informaticae, XII:221–242, 1989
7. Jos C. M. Baeten, W. P. Weijland. Process Algebra. Cambridge: Cambridge University Press, 1990
8. Jan A. Bergstra, Jan Willem Klop. Algebra of communicating processes with abstraction. Theoretical Computer Science, 37(1):77–121, 1985
9. Eike Best, Raymond Devillers, Javier Esparza. General refinement and recursion operators for the Petri box calculus. In: P. Enjalbert, A. Finkel, K. W. Wagner (eds.) STACS 93, Vol. 665 of Lecture Notes in Computer Science, pp. 130–140. Berlin Heidelberg New York: Springer 1993
10. Bard Bloom, Sorin Istrail, Albert R. Meyer. Bisimulation can't be traced. Journal of the ACM, 42(1):232–268, January 1995

11. Tommaso Bolognesi, Ed Brinksma. Introduction to the ISO specification language LOTOS. Computer Networks and ISDN Systems, 14:25–59, 1987
12. Gérard Boudol, Ilaria Castellani, Matthew C. B. Hennessy, Astrid Kiehn. A theory of processes with localities. Formal Aspects of Computing, 6:165–200, 1994
13. Ed Brinksma, Bengt Jonsson, Fredrik Orava. Refining interfaces of communicating systems. In: Samson Abramsky, T. S. E. Maibaum (eds.) TAPSOFT '91, Volume 2, Vol. 494 of Lecture Notes in Computer Science, pp. 297–312. Berlin Heidelberg New York: Springer 1991
14. Stephen D. Brookes, C. A. R. Hoare, A. W. Roscoe. A theory of communicating sequential processes. Journal of the ACM, 31(3):560–599, July 1984
15. Nadia Busi, Rob J. van Glabbeek, Roberto Gorrieri. Axiomatising ST bisimulation equivalence. In: Olderog [53], pp. 169–188. Amsterdam: North-Holland 1994
16. L. Castellano, G. De Michelis, L. Pomello. Concurrency vs. interleaving: An instructive example. Bull. Eur. Ass. Theoret. Comput. Sci., 31:12–15, 1987
17. W. R. Cleaveland, editor. Concur '92, Vol. 630 of Lecture Notes in Computer Science. Berlin Heidelberg New York: Springer 1992
18. Philippe Darondeau, Pierpaolo Degano. Refinement of actions in event structures and causal trees. Theoretical Computer Science, 118:21–48, 1993
19. J. W. de Bakker, W.-P. de Roever, Grzegorz Rozenberg (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Vol. 354 of Lecture Notes in Computer Science. Berlin Heidelberg New York: Springer 1989
20. J. W. de Bakker, W.-P. de Roever, Grzegorz Rozenberg (eds.) Semantics: Foundations and Applications, Vol. 666 of Lecture Notes in Computer Science. Berlin Heidelberg New York: Springer 1992
21. J. W. de Bakker, J. I. Zucker. Processes and the denotational semantics of concurrency. Information and Computation, 54:70–120, 1982
22. Pierpaolo Degano, Roberto Gorrieri. A causal operational semantics of action refinement. Information and Computation, 122:97–119, 1995
23. Pierpaolo Degano, Roberto Gorrieri, G. Rosolini. A categorical view of process refinement. In: de Bakker et al. [20], pp. 138–153
24. Volker Diekert. On the concatenation of infinite traces. Theoretical Computer Science, 113:35–54, 1993
25. Volker Diekert, Paul Gastin. A domain for concurrent termination: A generalization of Mazurkiewicz traces. In: Z. Fülöp and F. Gécseg (eds.) Automata, Languages and Programming, Vol. 944 of Lecture Notes in Computer Science, pp. 15–26. Berlin Heidelberg New York: Springer 1995
26. Volker Diekert, Grzegorz Rozenberg (eds.) The Book of Traces. World Scientific, 1995
27. T. Elrad, N. Francez. Decomposition of distributed programs into communication closed layers. Science of Computer Programming, 2, 1982
28. Maarten Fokkinga, Mannes Poel, Job Zwiers. Modular completeness for communication closed layers. In: E. Best (ed.) Concur '93, Vol. 715 of Lecture Notes in Computer Science, pp. 50–65. Berlin Heidelberg New York: Springer 1993
29. Wan Fokkink, Chris Verhoef. A conservative look at operational semantics with variable binding. Information and Computation, 146(1):24–54, 1998
30. Haim Gaifman. Modeling concurrency in partial orders and nonlinear transition systems. In: de Bakker et al. [19], pp. 467–488
31. Thomas Gehrke, Michaela Huhn, Arend Rensink, Heike Wehrheim. An algebraic semantics for message sequence chart documents. In: S. Budkowski, A. Cavalli, E. Najm (eds.) Formal Description Techniques and Protocol Specification, Testing and Verification. Chapman-Hall, 1998. Full report version: Hildesheimer Informatik-Bericht 5/98

32. Ursula Goltz, Roberto Gorrieri, Arend Rensink. Comparing syntactic and semantic action refinement. Information and Computation, 125(2):118–143, March 1996
33. Ursula Goltz, Norbert Götz. Modelling a simple communication protocol in a language with action refinement. Draft version, 1991
34. Jan Friso Groote. Transition system specifications with negative premises. Theoretical Computer Science, 118:263–299, 1993
35. C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985
36. Michaela Huhn. Action refinement and property inheritance in systems of sequential agents. In: Montanari, Sassone [51], pp. 639–654
37. Michaela Huhn. On the Hierarchical Design of Distributed Systems. PhD thesis, University of Hildesheim, 1997
38. ISO. Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, ISO, Geneva, February 1989. 1st Edition
39. Wil Janssen, Mannes Poel, Job Zwiers. Action systems and action refinement in the development of parallel systems. In: J. C. M. Baeten, J. F. Groote (eds.) Concur '91, Vol. 527 of Lecture Notes in Computer Science, pages 298–316. Berlin Heidelberg New York: Springer 1991
40. Wil Janssen, Job Zwiers. From sequential layers to distributed processes. In: Principles of Distributed Computing, pp. 215–227. ACM, 1992
41. Wil Janssen, Job Zwiers. Protocol design by layered decomposition: A compositional approach. In: Formal Techniques in Real-Time and Fault-Tolerant Systems, Vol. 571 of Lecture Notes in Computer Science, Berlin Heidelberg New York: Sringer 1992
42. Lalita Jategaonkar, Albert R. Meyer. Testing equivalences for Petri nets with action refinement. In: Cleaveland [17], pp. 17–31. s.o Springer 1992
43. Rita Loogen, Ursula Goltz. Modelling nondeterministic concurrent processes with event structures. Fundamenta Informaticae, XIV:39–73, 1991
44. S. Mauw, M. A. Reniers. Refinements in interworkings. In: Montanari, Sassone [51], pp. 671–686. s.o. Springer 1996
45. S. Mauw, M. A. Reniers. High-level Message Sequence Charts. In: SDL' 97: Time for Testing – SDL, MSC and Trends. Amsterdam: North-Holland 1997
46. Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Report PB–78, Aarhus University, 1977
47. Antoni Mazurkiewicz. Traces, histories, graphs: instances of a process monoid. Lecutre Notes in Computer Science, 176:115–133, 1984
48. Antoni Mazurkiewicz. Basic notions of trace theory. In: de Bakker et al. [19], pp. 285–363, s.o. Springer 1989
49. Robin Milner. Communication and Concurrency. Englewood Cliffs, N.J.: Prentice-Hall 1989
50. Robin Milner, Davide Sangiorgi. The problem of "weak bisimulation up to". In: Cleaveland [17], pp. 32–46. s.o. Springer 1992
51. Ugo Montanari, Vladimiro Sassone (eds.) Concur '96: Concurrency Theory, Vol. 1119 of Lecture Notes in Computer Science. Berlin Heidelberg New York: Springer 1996
52. Mogens Nielsen, Uffe Engberg, Kim Guldstrand Larsen. Fully abstract models for a process language with refinement. In: de Bakker et al. [19], pages 523–549. s.o. Springer 1989
53. E.-R. Olderog (ed.) Programming Concepts, Methods and Calculi, Vol. A–56 of IFIP Transactions. IFIP, North-Holland Publishing Company, 1994
54. D. Park. Concurrency and automata on infinite sequences. In: P. Deussen (ed.) Proceedings 5th GI Conference, Vol. 104 of Lecture Notes in Computer Science, pp. 167–183. Berlin Heidelberg New York: Springer 1981

55. Doron A. Peled, Vaughan R. Pratt, Gerard J. Holzmann (eds.) Partial Order Methods in Verification, Vol. 29 of DIMACS series. American Mathematical Society, 1997
56. Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981
57. Arend Rensink. Posets for configurations! In: Cleaveland [17], pages 269–285. s.o. Springer 1992
58. Arend Rensink. Models and Methods for Action Refinement. PhD thesis, University of Twente, Enschede, Netherlands, August 1993
59. Arend Rensink. Methodological aspects of action refinement. In: Olderog [53], pp. 227–246. Amsterdam: North-Holland 1994
60. Arend Rensink. An event-based SOS for a language with refinement. In: Jörg Desel (ed.) Structures in Concurrency Theory, Workshops in Computing, pp. 294–309. Berlin Heidelberg New York: Springer 1995
61. Arend Rensink. Bisimilarity of open terms. Information and Computation, 156(1/2):345–385, January 2000
62. Arend Rensink, Roberto Gorrieri. Vertical implementation. Information and Computation, 2000. To appear. Extended version of "Vertical Bisimulation" (TAPSOFT '97). Full report version: Hildesheimer Informatik-Bericht 9/98, University of Hildesheim
63. Arend Rensink, Heike Wehrheim. Weak sequential composition in process algebras. In: Bengt Jonsson, Joachim Parrow (eds.) Concur '94: Concurrency Theory, Vol. 836 of Lecture Notes in Computer Science, pp. 226–241. Berlin Heidelberg New York: Springer 1994
64. Arend Rensink, Heike Wehrheim. Dependency-based action refinement. In: I. Prívara, P. Ruzicka (eds.) Mathematical Foundations of Computer Science 1997, Vol. 1295 of Lecture Notes in Computer Science, pp. 468–477. Berlin Heidelberg New York: Springer 1997
65. Vladimiro Sassone, Mogens Nielsen, Glynn Winskel. Models for concurrency: Towards a classification. Theoretical Computer Science, 170:297–348, 1996
66. Frits W. Vaandrager. Expressiveness results for process algebras. In: de Bakker et al. [20], pp. 609–638. Report version: Centrum voor Wiskunde en Informatica, Report CS–R9301
67. Rob J. van Glabbeek. The meaning of negative premises in transition system specifications II. In: F. Meyer auf der Heide, B. Monien (eds.) Automata, Languages and Programming, Vol. 1099 of Lecture Notes in Computer Science, pp. 502–513. Berlin Heidelberg New York: Springer 1996. Full report version: STAN-CS-TN-95-16, Department of Computer Science, Stanford University
68. Rob J. van Glabbeek, Ursula Goltz. Refinement of actions in causality based models. In: J. W. de Bakker, W.-P. de Roever, Grzegorz Rozenberg (eds.) Stepwise Refinement of Distributed Systems – Models, Formalisms, Correctness, Vol. 430 of Lecture Notes in Computer Science, pp. 267–300. Berlin Heidelberg New York: Springer 1990. Report version: Arbeitspapiere der GMD 428
69. Rob J. van Glabbeek, Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. Hildesheimer Informatik-Bericht 6/96, University of Hildesheim, 1998. To appear in Acta Informatica
70. Walter Vogler. Failures semantics based on interval semiwords is a congruence for refinement. Distributed Computing, 4:139–162, 1991
71. Walter Vogler. Bisimulation and action refinement. Theoretical Computer Science, 114:173–200, 1993
72. Heike Wehrheim. Parametric action refinement. In: Olderog [53], pp. 247–266. Full report version: Hildesheimer Informatik-Berichte 18/93, Institut für Informatik, University of Hildesheim, Nov. 1993

73. Glynn Winskel. An introduction to event structures. In: de Bakker et al. [19], pp. 364–397. Springer 1989
74. Job Zwiers. Layering and action refinement for timed systems. In: J. W. de Bakker, C. Huizing, W.-P. de Roever, G. Rozenberg (eds.) Real-Time: Theory in Practice, Vol. 600 of Lecture Notes in Computer Science. Berlin Heidelberg New York: Springer 1991

A Proofs of the results

A.1 Proofs of Section 3

Proposition 3.9 *Assume $B, C \in \mathbf{L}^{wf}$ such that $\mathcal{A}(C) \subseteq \mathcal{A}_X$.*

1. *If $B \xrightarrow{\alpha} B'$, then $B\langle C/X\rangle \xrightarrow{\alpha} B'\langle C/X\rangle$.*
2. *If $B\langle C/X\rangle \xrightarrow{\alpha} B'$ and X is dependently guarded in B, then $B' = B''\langle C/X\rangle$ for some B'' such that $B \xrightarrow{\alpha} B''$.*

Proof. By induction on the structure of B. Clause 1 is immediate and omitted. For Clause 2, The interesting case is $B = B_1 \cdot B_2$. Dependent guardedness of B implies that X is dependently guarded in B_1, and either X is dependently guarded in B_2 or $B_1 \xrightarrow{\surd_a}$ implies $a \; I \; \mathcal{A}(B_2)$. Consider the possible transitions of B.

- $\alpha = a$, $B_1\langle C/X\rangle \xrightarrow{a} B_1'$ and $B' = B_1' \cdot B_2\langle C/X\rangle$. By the induction hypothesis, it follows that $B_1' = B_1''\langle C/X\rangle$ such that $B_1 \xrightarrow{\alpha} B_1'$; hence $B'' = B_1'' \cdot B_2$ fulfils the proof obligations.
- $\alpha = a$, $B_1\langle C/X\rangle \xrightarrow{\surd_a} B_1'$ and $B_2\langle C/X\rangle \xrightarrow{a} B_2'$ such that $B' = B_1' \cdot B_2'$. The induction hypothesis implies that $B_1 \xrightarrow{\surd_a}$, and Proposition 3.8 implies $a \in \mathcal{A}(B_2\langle C/X\rangle) \subseteq \mathcal{A}(B_2)$; it follows that X is dependently guarded in B_2. By the induction hypothesis, then, $B_i' = B_i''\langle C/X\rangle$ for $i = 1, 2$ such that $B_1 \xrightarrow{\surd_a} B_1''$ and $B_2 \xrightarrow{a} B_2''$. It follows that $B'' = B_1'' \cdot B_2''$ fulfils the proof obligations.
- $\alpha = \surd_a$ and $B_i\langle C/X\rangle \xrightarrow{\surd_a} B_i'$ for $i = 1, 2$. If X is dependently guarded in B_2, then the proof proceeds as in the previous case. Otherwise, by the induction hypothesis it follows that $B_1' = B_1''\langle C/X\rangle$ such that $B_1 \xrightarrow{\surd_a} B_1''$. Due to dependent guardedness, it follows that $a \; I \; \mathcal{A}(B_2)$; hence according to Proposition 3.8.3, $B_2' = B_2\langle C/X\rangle$. We may conclude that $B'' = B_1'' \cdot B_2$ fulfils the proof obligations. □

Congruence. We use the GSOS format of [10] to show that bisimulation is a congruence (that bisimilar processes can be considered equal in all contexts) for the language $\mathbf{L}$.

Definition A.1 (GSOS rules) *Suppose that Σ is a signature. A* GSOS rule *ρ over Σ is a rule of the form*

$$\frac{\{X_i \xrightarrow{a_{ij}} Y_{ij} \mid 1 \leq i \leq n, j \in M_i\} \quad \{X_i \xrightarrow{b_{ik}\!\!\!\!\!\!\rightarrow} \mid 1 \leq i \leq n, k \in N_i\}}{op(X_1, \ldots, X_n) \xrightarrow{c} C[\mathbf{X}, \mathbf{Y}]}$$

where all the variables are distinct, op is an operation symbol from Σ with arity n, and $C[\mathbf{X}, \mathbf{Y}]$ is a Σ-context which may contain only the X_i and Y_{ij} as free variables. Note that the index sets M_i and N_i may be empty for any given i.

The GSOS format allows for both positive and negative premises in the rules of the operational semantics. According to this definition, all our rules are in GSOS-format, except (as usual) for the recursion rule R_{14}, which contains a "substitution harness" (in the terminology of [29]) that is not accounted for in GSOS. The refinement operator in this setting is interpreted as an $(n+1)$-ary operator, where n equals the number of actions for which r is not the identity —which is finite since we assumed refinement functions to be the identity almost everywhere. This interpretation becomes clearer if we write out r as an explicit list of substitutions $B_1/a_1, \ldots, B_n/a_n$, where $\{a_1, \ldots, a_n\} = \{a \in Act \mid r(a) \neq a\}$ and $B_i = r(a_i)$ for all $1 \leq i \leq n$. The corresponding operational rules then become:

$$\frac{x \xrightarrow{a} x' \quad a \notin \{a_1, \ldots, a_n\}}{x[y_1/a_1, \ldots, y_n/a_n] \xrightarrow{a} \mathbf{1} \cdot x'[y_1/a_1, \ldots, y_n/a_n]}$$

$$\frac{x \xrightarrow{a_i} x' \quad y_i \xrightarrow{b} y' \quad 1 \leq i \leq n}{x[y_1/a_1, \ldots, y_n/a_n] \xrightarrow{b} y' \cdot x'[y_1/a_1, \ldots, y_n/a_n]}$$

Theorem 3.14 *Bisimulation is a congruence for all operators of $\mathbf{L}^{wf}$ (including recursion).*

Proof. For all operators except recursion this follows from the fact that all operational rules (Table 2) are GSOS rules. To prove congruence of recursion, the *up-to* technique used in [49] can be applied, since our rules contain no look-ahead (see [61]). □

Theorem 3.15 *If $B \in \mathbf{L}^{wf}$ with $fv(B) \subseteq \{X\}$, then $recX.B$ is the unique solution of $X = B$ in $\mathbf{L}^{wf}$ modulo $\sim$.*

Proof. The fact that $recX.B$ solves $X = B$ modulo $\sim$ is straightforward to establish. The proof of uniqueness is along the lines set out in Milner [49].

Assume $C \sim B\langle C/X\rangle$ for some C that is a syntactic representation of a transition system to which Proposition 3.9 applies. Consider the relation

$$\rho = \{(D_1, D_2) | \exists D \in \mathbf{L}^{wf} : fv(D) \subseteq \{X\}, D_1 \sim D\langle recX.\, B/X\rangle, D_2 \sim D\langle C/X\rangle\}.$$

We prove that ρ is a bisimulation relation. It follows (taking $D = X$ in the definition of ρ) that $recX.\, B \sim C$.

- Assume $D_1 \xrightarrow{\alpha} D_1'$. Due to $recX.\, B \sim B\langle recX.\, B/X\rangle$, the congruence of $\sim$ (Theorem 3.14) and the properties of syntactic substitution, it follows that $D_1 \sim D\langle B/X\rangle\langle recX.\, B/X\rangle$. It follows that $D\langle B/X\rangle\langle recX.\, B/X\rangle \xrightarrow{\alpha} D' \sim D_1'$. Since X is dependently guarded in B it is dependently guarded in $D\langle B/X\rangle$; hence (by Proposition 3.9) $D' = D''\langle recX.\, B/X\rangle$ for some D'' such that $D\langle B/X\rangle \xrightarrow{\alpha} D''$ and hence $D\langle B\langle C/X\rangle/X\rangle = D\langle B/X\rangle\langle C/X\rangle \xrightarrow{\alpha} D''\langle C/X\rangle$.
  Since $B\langle C/X\rangle \sim C$, it follows (by Theorem 3.14) that $D_2 \sim D\langle C/X\rangle \sim D\langle B\langle C/X\rangle/X\rangle$; hence $D_2 \xrightarrow{\alpha} D_2' \sim D''\langle C/X\rangle$. We may conclude $(D_1', D_2') \in \rho$.
- The reverse direction is analogous. □

A.2 Proofs of Section 4

Proposition 4.9 $\mathbf{M}$ *is closed under the denotational constructions for choice, parallel composition, sequential composition, refinement and recursion.*

Proof. The important thing is to prove prefix-closure of the constructed models.

Choice. Straightforward.

Parallel composition. Assume $q' \preceq q \in \mathcal{P}_1 \parallel_A \mathcal{P}_2$, where q is constructed from $p_i \in \mathcal{P}_i$ for $i = 1, 2$. Let $E_i' = \pi_i(E_{q'})$ and $p_i' = (p_i \upharpoonright E_i') \cap T_{q'}$; since E_i' is clearly $\leq_{p_i}$-left-closed (otherwise $E_{q'}$ would not be $\leq_q$-left-closed) and

$$T_{p_i'} = T_{q'} \subseteq T_q \setminus [\mathcal{A}_{q-q'}]_D \subseteq T_{p_i} \setminus [\mathcal{A}_{p_i - p_i'}]_D$$

it follows that $p_i' \preceq p_i$ and hence $p_i' \in \mathcal{P}_i$ for $i = 1, 2$. Moreover, the synchronisation of p_1' and p_2' gives rise to $q' \in \mathcal{P}_1 \parallel_A \mathcal{P}_2$.

Sequential composition. Assume $q' \preceq q \in \mathcal{P}_1 \cdot \mathcal{P}_2$, where q is constructed from $p_i \in \mathcal{P}_i$ for $i = 1, 2$. Let

$$p_1' = (p_1 \upharpoonright E_{q'}) \setminus [\mathcal{A}_{p_1 - q'}]_D$$
$$p_2' = (p_2 \upharpoonright E_{q'}) \cap T_{q'} \ .$$

Since $E_{p_i} \cap E_{q'}$ is $\leq_{p_i}$-left-closed due to the fact that $E_{q'}$ is $\leq_q$-left-closed, and $T_{q'} \cap [\mathcal{A}_{p_2-q'}]_D = \emptyset$, it follows that $p'_i \preceq p_i$ and hence $p'_i \in \mathcal{P}_i$ for $i = 1, 2$. For all $d \in E_{p_1-q'}$ and $e \in E_{p'_2}$, it follows that $d \not\leq_q e$ and hence $d \; I \; e$; hence $\mathcal{A}_{p'_2} \subseteq T_{p'_1}$. Now the composition of p'_1 and p'_2 gives rise to $q' \in \mathcal{P}_1 \cdot \mathcal{P}_2$; in particular, due to

$$T_{q'} \;\subseteq\; T_q \setminus [\mathcal{A}_{q-q'}]_D \;=\; (T_{p_1} \setminus [\mathcal{A}_{p_1-q'}]_D) \cap (T_{p_2} \setminus [\mathcal{A}_{p_2-q'}]_D)$$

it follows that

$$T_{p'_1} \cap T_{p'_2} = (T_{p_1} \setminus [\mathcal{A}_{p_1-q'}]_D) \cap T_{p_2} \cap T_{q'} = T_{q'} \;\; .$$

Refinement. Assume $q' \preceq q \in \mathcal{P}[\mathcal{R}]$, where q is constructed from $p \in \mathcal{P}$ and $w \colon E_p \to \mathbf{M}$. Let $E' = \pi_1(E_{q'})$, and let

$$p' = (p \restriction E') \cap T_{q'} \;\; .$$

Moreover, for all $d \in E'$ let $E'_d = \{e \mid (d, e) \in E_{q'}\}$ and

$$w'(d) = (w(d) \restriction E'_d) \setminus [\mathcal{A}_{w(d) \setminus E'_d}]_D \;\; .$$

For an arbitrary $d'' \in E_p \setminus E'$ and $a \; D \; \ell(d'')$, consider $e'' \in \min w(d'')$; then $\langle e'' \rangle \preceq w(d'')$, hence (due to prefix closure) $\langle e'' \rangle \in \mathcal{R}(\ell(d''))$. Denotational D-consistency (Definition 4.7) then implies $a \; D \; \ell(e'')$. Due to $(d'', e'') \in E_{q-q'}$ with $\ell(e'') = \ell(d'', e'')$ it follows that $[\mathcal{A}_{p-p'}]_D \subseteq [\mathcal{A}_{q-q'}]_D$ and hence

$$T_{p'} \;=\; T_p \cap T_{q'} \;\subseteq\; T_p \cap (T_q \setminus [\mathcal{A}_{q-q'}]_D) \;\subseteq\; T_p \setminus [\mathcal{A}_{p-p'}]_D$$

This implies $p' \preceq p$ and hence $p' \in \mathcal{P}$.
Moreover, for all $d \in E_{p'}$ it follows by construction (since the $\leq_q$-left-closure of $E_{q'}$ implies that also E'_d is $\leq_{w(d)}$-left-closed) that $w'(d) \preceq w(d)$; hence $w'(d) \in \mathcal{R}(\ell(d))$.
Finally, it is straightforward to see that the refinement of p' according to w' gives rise to q'; hence $q' \in \mathcal{P}[\mathcal{R}]$.

Recursion. The union of an arbitrary set of system models is easily seen to yield a system model again (in fact, this is the same argument as for the choice operator). □

Well-definedness of the denotational semantics. In order to prove Proposition 4.11, the semantic refinement function constructed in Table 3 must be shown to be denotationally D-consistent in the sense of Definition 4.7. Since the construction of the refinement function itself uses the denotational semantics, albeit on subterms, well-definedness can only be established by induction on the term structure. For this purpose we define two auxiliary sets of terms:

- $\mathbf{L}^{cwd}$ is the set of closed well-formed terms for which $[\![-]\!]$ is well-defined.
- $\mathbf{L}^{den}$ is the largest subset of $\mathbf{L}^{wf}$ such that for all $B \in \mathbf{L}^{den}$:
  - If B is closed then $[\![B]\!]$ is well-defined;
  - $C \in \mathbf{L}^{den}$ for all syntactical subterms C of B;
  - $B\langle C/X\rangle \in \mathbf{L}^{den}$ for all $X \in fv(B)$ and $C \in \mathbf{L}^{cwd}$ with $\mathcal{A}(C) \subseteq \mathcal{A}_X$.

Proving Proposition 4.11 then comes down to showing the following:

Lemma A.2 $\mathbf{L}^{den} = \mathbf{L}^{wf}$

The proof proceeds by induction on the structure of terms in $\mathbf{L}^{wf}$. The only really interesting case is that of refinement, where induction carries through as a consequence of the following correspondence between strong and denotational D-consistency:

Lemma A.3 *If* $r\colon Act \to \mathbf{L}^{den}$ *is strongly* D*-consistent, then the function* $Act \to \mathbf{M}$ *defined by* $a \mapsto [\![r(a)]\!]$ *for all* $a \in Act$ *is denotationally* D*-consistent.*

Note that this is *implied by* Proposition 4.12 since $\mathbf{L}^{den} \subseteq \mathbf{L}^{wf}$; however, in the presence of Lemma A.2 the two statements are equivalent. The correspondence in turn relies on a certain relation between operational and denotational concepts, established by the following lemma (which stands in the same relation to Proposition 4.13 as Lemma A.3 to Proposition 4.12).

Lemma A.4 *For all closed* $B \in \mathbf{L}^{den}$*, the following holds:*

1. $a\ I\ \mathcal{A}(B)$ *implies* $a\ I\ \mathcal{A}_p$ *and* $p \cup a \in [\![B]\!]$ *for all* $p \in [\![B]\!]$.
2. $B \xrightarrow{\surd_a}$ *iff* $\varepsilon_a \in [\![B]\!]$.
3. $B \xrightarrow{a}$ *iff* $\langle e\rangle \in [\![B]\!]$ *with* $\ell(e) = a$.

Proof. In principle, this proof is also by induction on the term structure (of terms in $\mathbf{L}^{den}$). However, to cope with recursion we need to strengthen the proof obligation so that it also applies to open terms, i.e., to arbitrary $B \in \mathbf{L}^{den}$. For this purpose, we use an obvious generalisation to *simultaneous* substitution: if $\sigma\colon fv(B) \to \mathbf{L}^{cwd}$ is *compatible* in the sense that $\sigma(X)$ is closed and $\mathcal{A}(\sigma(X)) = \mathcal{A}_X$ for all $X \in fv(B)$, then $B\langle\sigma\rangle$ denotes the substitution of all $X \in fv(B)$ by their images $\sigma(X)$.

Let $B \in \mathbf{L}^{den}$, and let $\sigma\colon fv(B) \to \mathbf{L}^{cwd}$ be an arbitrary compatible substitution function. By definition of $\mathbf{L}^{den}$ it follows that $[\![B\langle\sigma\rangle]\!]$ is well-defined. We prove the following by induction on the structure of B:

1. $a\ I\ \mathcal{A}(B)$ implies $a\ I\ \mathcal{A}_p$ and $p \cup a \in [\![B\langle\sigma\rangle]\!]$ for all $p \in [\![B\langle\sigma\rangle]\!]$.
2. If all X are dependently guarded in B, then $B\langle\sigma\rangle \xrightarrow{\surd_a}$ iff $\varepsilon_a \in [\![B\langle\sigma\rangle]\!]$.
3. If all X are dependently guarded in B, then $B\langle\sigma\rangle \xrightarrow{a}$ iff $\langle e\rangle \in [\![B\langle\sigma\rangle]\!]$ with $\ell(e) = a$.

The interesting cases are weak sequential composition and recursion.

- Assume $B = B_1 \cdot B_2$. Note that (by definition of dependent guardedness) all $X \in fv(B)$ are dependently guarded in B_1. If all $X \in fv(B)$ are also dependently guarded in B_2 then the required properties follow easily by the induction hypothesis. Now assume some X is *not* dependently guarded in B_2. Using Proposition 3.9 and the fact that X is dependently guarded in B, one can then derive that for all a, $B_1\langle\sigma\rangle \xrightarrow{\checkmark_a}$ implies $a \; I \; \mathcal{A}(B_2)$.
  1. Immediate from the induction hypothesis and the construction of all $p \in [\![B\langle\sigma\rangle]\!]$ from $q_i \in [\![B_i\langle\sigma\rangle]\!]$, noting that $a \; I \; \mathcal{A}(B)$ implies $a \; I \; \mathcal{A}(B_i)$ for both $i = 1, 2$, $a \; I \; \mathcal{A}_p$ iff $a \; I \; \mathcal{A}_{q_i}$ for both $i = 1, 2$, and $p \cup a \in [\![B\langle\sigma\rangle]\!]$ iff $q_i \cup a \in [\![B_i\langle\sigma\rangle]\!]$ for both $i = 1, 2$.
  2. If $B\langle\sigma\rangle \xrightarrow{\checkmark_a}$ then $B_i\langle\sigma\rangle \xrightarrow{\checkmark_a}$ for $i = 1, 2$. For B_1 this implies $\varepsilon_a \in [\![B_1\langle\sigma\rangle]\!]$ by the induction hypothesis; furthermore, $a \; I \; \mathcal{A}(B_2)$, implying $\varepsilon_a \in [\![B_2\langle\sigma\rangle]\!]$ by Clause 1. By construction, it follows that $\varepsilon_a \in [\![B\langle\sigma\rangle]\!]$.
     If $\varepsilon_a \in [\![B\langle\sigma\rangle]\!]$ then $\varepsilon_a \in B_i\langle\sigma\rangle$ for both $i = 1, 2$. The induction hypothesis for B_1 implies $B_1\langle\sigma\rangle \xrightarrow{\checkmark_a}$; hence $a \; I \; \mathcal{A}(B_2)$. By Proposition 3.8 this implies $B_2\langle\sigma\rangle \xrightarrow{\checkmark_a}$, hence $B\langle\sigma\rangle \xrightarrow{\checkmark_a}$.
  3. If $B\langle\sigma\rangle \xrightarrow{a}$ then either $B_1\langle\sigma\rangle \xrightarrow{a}$, implying $\langle e\rangle \in [\![B_1\langle\sigma\rangle]\!]$ with $\ell(e) = a$ and hence $\langle e\rangle \in [\![B\langle\sigma\rangle]\!]$, or $B_1\langle\sigma\rangle \xrightarrow{\checkmark_a}$ and $B_2\langle\sigma\rangle \xrightarrow{a}$. However, $B_1\langle\sigma\rangle \xrightarrow{\checkmark_a}$ implies $a \; I \; \mathcal{A}(B_2)$, which by Proposition 3.8 contradicts $B_2\langle\sigma\rangle \xrightarrow{a}$.
     If $\langle e\rangle \in [\![B\langle\sigma\rangle]\!]$ with $\ell(e) = a$ then either $\langle e\rangle \in [\![B_1\langle\sigma\rangle]\!]$, or $\varepsilon_a \in [\![B_1\langle\sigma\rangle]\!]$ and $\langle e\rangle \in [\![B_1\langle\sigma\rangle]\!]$. In the former case, the induction hypothesis for B_1 implies $B_1\langle\sigma\rangle \xrightarrow{a}$; in the latter case the induction hypothesis implies $B_1\langle\sigma\rangle \xrightarrow{\checkmark_a}$ and hence $a \; I \; \mathcal{A}(B_2)$; this contradicts $\langle e\rangle \in [\![B_2\langle\sigma\rangle]\!]$ by Clause 1.
- Assume $B = recX.\, C$. Note that $B\langle\sigma\rangle = recX.\, (C\langle\sigma\rangle)$ (taking into account that $X \notin fv(B)$ and hence σ does not substitute anything for X) and $B\langle\sigma\rangle^i_X = C\langle\sigma^i_X\rangle$ for all $i > 0$, where $\sigma^i_X = \sigma \cup \{X \mapsto B\langle\sigma\rangle^{i-1}_X\}$. By definition of $\mathbf{L}^{den}$ it follows that $C \in \mathbf{L}^{den}$; moreover, by induction on i (starting with the base case observation that $B^0_X = \mathbf{0}_{\mathcal{A}_X} \in \mathbf{L}^{cwd}$) it follows that all σ^i_X map into $\mathbf{L}^{cwd}$; in particular, all $B^i_X \in \mathbf{L}^{cwd}$. Also note that all free variables of C (including X) are dependently guarded.
  1. Assume $a \; I \; \mathcal{A}(B)$; by definition, this implies $a \; I \; \mathcal{A}(C)$. If $p \in [\![B\langle\sigma\rangle]\!]$ then $p \in [\![B\langle\sigma\rangle^i_X]\!]$ for some i; hence either $p \in \bot$ (if $i = 0$) or $p \in [\![C\langle\sigma^i_X\rangle]\!]$ (if $i > 0$). In the former case, $\mathcal{A}_p = \emptyset$ and hence $a \; I \; \mathcal{A}_p$, and moreover, $p \cup a \in \bot$ ($\subseteq [\![B\langle\sigma\rangle]\!]$) by definition of $\bot$; in the latter case (by the induction hypothesis for C) $a \; I \; \mathcal{A}_p$ and $p \cup a \in C\langle\sigma^i_X\rangle$ ($\subseteq [\![B\sigma]\!]$).

2. If $B\langle\sigma\rangle \xrightarrow{\surd_a}$ then (by R_{14} and R_{15} in Table 2) either $a\ I\ \mathcal{A}(B)$ or $C\langle\sigma\rangle \xrightarrow{\surd_a}$. In the first case (by Clause 1) $\varepsilon_a \in [\![B\langle\sigma\rangle]\!]$; in the second case (by Proposition 3.9.1) $C\langle\sigma\rangle\langle\mathbf{0}_{\mathcal{A}_X}/X\rangle \xrightarrow{\surd_a}$ and hence (by the induction hypothesis for C) $\varepsilon_a \in [\![C\langle\sigma\rangle\langle\mathbf{0}_{\mathcal{A}_X}/X\rangle]\!]$. Since $C\langle\sigma\rangle\langle\mathbf{0}_{\mathcal{A}_X}/X\rangle = B\langle\sigma\rangle^1_X$ this implies $\varepsilon_a \in [\![B\langle\sigma\rangle]\!]$.
If $\varepsilon_a \in [\![B\langle\sigma\rangle]\!]$ then $\varepsilon_a \in [\![B\langle\sigma\rangle^i_X]\!]$ for some $i \in \mathbb{N}$. If $i = 0$ then $[\![B\langle\sigma\rangle^i_X]\!] = \bot$ and hence $a\ I\ \mathcal{A}(B)$, implying $B\langle\sigma\rangle \xrightarrow{\surd_a}$ by Proposition 3.8. Otherwise $B\langle\sigma\rangle^i_X = C\langle\sigma^i_X\rangle$ and hence (by the induction hypothesis on C) $C\langle\sigma^i_X\rangle \xrightarrow{\surd_a}$. But then also (by Proposition 3.9.2, since X is dependently guarded in $C\langle\sigma\rangle$) $C\langle\sigma\rangle \xrightarrow{\surd_a}$, implying $B\langle\sigma\rangle \xrightarrow{\surd_a}$.
3. If $B\langle\sigma\rangle \xrightarrow{a}$ then (by R_{14} and R_{15} in Table 2) $C\langle\sigma\rangle \xrightarrow{a}$. This implies (by Proposition 3.9.1) $C\langle\sigma\rangle\langle\mathbf{0}_{\mathcal{A}_X}/X\rangle \xrightarrow{a}$ and hence (by the induction hypothesis for C) $\langle e\rangle \in [\![C\langle\sigma\rangle\langle\mathbf{0}_{\mathcal{A}_X}/X\rangle]\!]$ with $\ell(e) = a$. Since $C\langle\sigma\rangle\langle\mathbf{0}\mathcal{A}_X/X\rangle = B\langle\sigma\rangle^1_X$ it follows that $\langle e\rangle \in [\![B\langle\sigma\rangle]\!]$.
If $\langle e\rangle \in [\![B\langle\sigma\rangle]\!]$ with $\ell(e) = a$ then $\langle e\rangle \in [\![B\langle\sigma\rangle^i_X]\!]$ for some $i \in \mathbb{N}$. $i = 0$ is ruled out since then $B\langle\sigma\rangle^i_X = \mathbf{0}_{\mathcal{A}_X}$; hence $B\langle\sigma\rangle^i_X = C\langle\sigma^i_X\rangle$, implying (by the induction hypothesis for C) $C\langle\sigma\rangle\langle B\langle\sigma\rangle^{i-1}_X/X\rangle \xrightarrow{a}$. But then also (by Proposition 3.9.2, since X is dependently guarded in $C\langle\sigma\rangle$) $C\langle\sigma\rangle \xrightarrow{a}$, implying $B\langle\sigma\rangle \xrightarrow{\surd_a}$. □

Proof of Lemma A.3. This is now immediate, since Lemma A.4 gives the necessary correspondence between the syntactic concepts in Definition 3.5 and the semantic counterparts in Definition 4.7. □

Proof of Lemma A.2. This is now easily proved by induction on the structure of terms $B \in \mathbf{L}^{wf}$. In particular, the case for refinement follows from Lemma A.3. □

With these auxiliary results established, the main results are immediate.

Proposition 4.11 $[\![B]\!] \in \mathbf{M}$ *for all* $B \in \mathbf{L}^{wf}$.

Proof. Immediate from Lemma A.2. □

Proposition 4.12 *If* $r\colon Act \to \mathbf{L}^{wf}$ *is strongly D-consistent, then the function* $Act \to \mathbf{M}$ *defined by* $a \mapsto [\![r(a)]\!]$ *for all* $a \in Act$ *is denotationally D-consistent.*

Proof. Immediate from Lemma A.3 and Lemma A.2. □

Proposition 4.13 *For all closed* $B \in \mathbf{L}^{wf}$*, the following holds:*

1. $a\ I\ \mathcal{A}(B)$ *implies* $a\ I\ \mathcal{A}_p$ *and* $p \cup a \in [\![B]\!]$ *for all* $p \in [\![B]\!]$.
2. $B \xrightarrow{\surd_a}$ *iff* $\varepsilon_a \in [\![B]\!]$.
3. $B \xrightarrow{a}$ *iff* $\langle e\rangle \in [\![B]\!]$ *with* $\ell(e) = a$.

Proof. Immediate from Lemma A.4 and Lemma A.2. □

Unique fixpoint solutions. In order to prove Proposition 4.19 (which expresses that our semantics gives rise to contracting functions in the complete metric space of denotational models), we need the following auxiliary lemma, which strengthens Proposition 4.13.2. The proof is a straightforward induction on the term structure, here omitted.

Lemma A.5 *If $B \in \mathbf{L}^{wf}_{fin}$ with $fv(B) \subseteq \{X\}$, then $B \xrightarrow{\surd_a}$ iff $\varepsilon_a \in [\![B]\!](\mathcal{P})$ for arbitrary $\mathcal{P} \in \mathbf{M}_{[\mathcal{A}_X]_I}$.*

A technical property of the functions constructed by the denotational semantics is that for all $B \in \mathbf{L}^{wf}$, $[\![B]\!](-)$ is monotonic w.r.t. $\subseteq$. This follows immediately from the fact that all constructions on $\mathbf{M}$ used in the definition above are defined pointwise on the system runs.

Lemma A.6 *Let $\mathcal{P}, \mathcal{Q} \in \mathbf{M}$ and $B \in \mathbf{L}^{wf}$ with $fv(B) \subseteq \{X\}$. If $\mathcal{P} \subseteq \mathcal{Q}$, then $[\![B]\!](\mathcal{P}) \subseteq [\![B]\!](\mathcal{Q})$.*

Again, the proof is omitted. We now come to the contraction property.

Proposition 4.19 *Let $B \in \mathbf{L}^{wf}_{fin}$ with $fv(B) \subseteq \{X\}$.*

1. *$[\![B]\!](-)$ is non-increasing;*
2. *If X is dependently guarded in B, then $[\![B]\!](-)$ is contracting.*

Proof. Let $f = [\![B]\!](-)$. We prove that for all $\mathcal{P}, \mathcal{Q} \in \mathbf{M}$ and all $n \in \mathbb{N}$, the following holds:

1. If $\mathcal{P}{\uparrow}n \subseteq \mathcal{Q}$, then $f(\mathcal{P}){\uparrow}n \subseteq f(\mathcal{Q})$;
2. If X is dependently guarded in B, then $f(\mathcal{P}){\uparrow}0 \subseteq f(\mathcal{Q})$, and if also $\mathcal{P}{\uparrow}n \subseteq \mathcal{Q}$, then $f(\mathcal{P}){\uparrow}(n+1) \subseteq f(\mathcal{Q})$.

Together with the (symmetrical) inverse properties, this implies the clauses of the lemma. For instance, item 1 implies

$$\sup\{n+1 \mid f(\mathcal{P}){\uparrow}n = f(\mathcal{Q}){\uparrow}n\} \geq \sup\{n+1 \mid \mathcal{P}{\uparrow}n = \mathcal{Q}{\uparrow}n\} \ ,$$

and hence $\delta(f(\mathcal{P}), f(\mathcal{Q})) \leq \delta(\mathcal{P}, \mathcal{Q})$; and similar for clause 2 (where the contraction constant is $\frac{1}{2}$).

1. By induction on the structure of B. We only sketch the proof. The cases where $B = \mathbf{0}_A$ or $B = a$ are trivial, since then f is a constant function. The cases where $B = B_1 \diamond B_2$ for a binary operator $\diamond \in \{+, \|_A, \cdot\}$ are all proved in a similar fashion: one shows that for a given $\mathcal{P} \in \mathbf{M}$, every $q \in f(\mathcal{P})$ is constructed from $p_i \in [\![B_i]\!](\mathcal{P})$ such that $depth(q) \geq depth(p_i)$ for $i = 1, 2$; hence $f(\mathcal{P}){\uparrow}n \subseteq [\![B_1]\!](\mathcal{P}){\uparrow}n \diamond [\![B_2]\!](\mathcal{P}){\uparrow}n$ (where the latter $\diamond$ is the semantic counterpart of the operator in B). By the induction hypothesis, it follows that $\mathcal{P}{\uparrow}n \subseteq \mathcal{Q}$ implies $[\![B_i]\!](\mathcal{P}){\uparrow}n \subseteq [\![B_i]\!](\mathcal{Q})$ for $i = 1, 2$, which by the above argument and the fact that all operators are monotonic w.r.t. $\subseteq$ (Lemma A.6) implies $f(\mathcal{P}){\uparrow}n \subseteq f(\mathcal{Q})$.
   Analogous arguments apply in the cases where $B = C[r]$. Finally, the case where $B = X$ is trivial, since then f is the identity function.

2. Again by induction on the structure of B. Except in the case where the top-level operator of B is sequential composition, B is guarded iff its operands are guarded; hence the proof is entirely analogous to the one sketched for clause 1.
   Assume $B = B_1 \cdot B_2$, and let $f_i = [\![B_i]\!](-)$ for $i = 1, 2$. X is guarded in B iff X is guarded in B_1 and *either* X is guarded in B_2 (in which case the proof is again analogous to the one for clause 1) *or* $B_1 \xrightarrow{\surd_a}$ implies $a \; I \; \mathcal{A}(B_2)$. We concentrate on the latter case.
   According to the definition of weak sequential composition in $\mathbf{M}$ (see above), an arbitrary system run $q \in f(\mathcal{P})$ is constructed as $q = p_1 \cdot p_2$ where $p_i \in \iota_i(f_i(\mathcal{P}))$ for $i = 1, 2$ such that $\mathcal{A}_{p_2} \subseteq T_{p_1}$ and

$$p_1 \cdot p_2 = \langle E_{p_1} \cup E_{p_2}, \leq_{p_1} \cup ((E_{p_1} \times E_{p_2}) \cap D) \cup \leq_{p_2}, T_{p_1} \cap T_{p_2} \rangle \ .$$

   Let $\mathcal{P}, \mathcal{Q} \in \mathbf{M}_{[\mathcal{A}_X]_I}$ be arbitrary. We first prove $f(\mathcal{P})\!\uparrow\!0 \subseteq f(\mathcal{Q})$. If $q \in f(\mathcal{P})\!\uparrow\!0$ then $q = \varepsilon_{T_q}$ and $p_1 = \varepsilon_{T_{p_1}}$, implying (due to $T_q \subseteq T_{p_1}$) $q \preceq p_1$ and hence $q \in \iota_1(f_1(\mathcal{P})\!\uparrow\!0)$. By the induction hypothesis, therefore, $q \in \iota_1(f_1(\mathcal{Q}))$. As a consequence, $\varepsilon_a \in f_1(\mathcal{Q})$ for all $a \in T_q$, implying (by Lemma A.5) $B_1 \xrightarrow{\surd_a}$ and hence $a \; I \; \mathcal{A}(B_2)$. Due to $\iota_2(f_2(\mathcal{Q})) \in \mathbf{M}_{[\mathcal{A}(B_2)]_I}$ it follows that $\varepsilon_{T_q} \in \iota_2(f_2(\mathcal{Q}))$, implying $q = q \cdot \varepsilon_{T_q} \in f(\mathcal{Q})$. Now assume that, moreover, $\mathcal{P}\!\uparrow\!n \subseteq \mathcal{Q}\!\uparrow\!n$. We prove $f(\mathcal{P})\!\uparrow\!(n+1) \subseteq f(\mathcal{Q})$. Let $q \in f(\mathcal{P})\!\uparrow\!(n+1)$ be arbitrary; assume $q = p_1 \cdot p_2$ where $p_i = \iota_i(f_i(\mathcal{P}))$ for $i = 1, 2$.
   - If $depth(p_2) = 0$ then $depth(p_1) = depth(q) \leq n + 1$, implying (by the induction hypothesis) $p_i \in \iota_i(f_i(\mathcal{Q}))$ for $i = 1, 2$ and hence $q \in f(\mathcal{Q})$.
   - Now assume $depth(p_2) > 0$, and let $e \in E_{p_2}$ be arbitrary with $a = \ell(e)$. By construction $a \in T_{p_1}$. If moreover $\mathcal{A}_{p_1} \; I \; a$ then $\varepsilon_a \preceq p_1$ and hence $\varepsilon_a \in f_1(\mathcal{P})$; by Lemma A.5 this implies $B_1 \xrightarrow{\surd_a}$ and hence $a \; I \; \mathcal{A}(B_2)$, which contradicts $p_2 \in \iota_2(f_2(\mathcal{P}))$. We may conclude that $d <_q e$ for some $d \in E_{p_1}$.
     It follows that every maximal $<_q$-chain starts with an event from p_1, and hence $depth(p_2) \leq depth(q) - 1 \leq n$. On the other hand, $depth(p_1) \leq depth(q) \leq n + 1$ is immediate. By the induction hypothesis, it follows that $p_i \in \iota_i(f_i(\mathcal{Q}))$ for $i = 1, 2$ and hence $q \in f(\mathcal{Q})$. $\square$

For the proof of Theorem 4.20 we need an auxiliary lemma — which strengthens Proposition 4.10, where it was stated that that all constructions we have considered over $\mathbf{M}$ are well-defined up to isomorphism. Recall that $Evt \rightharpoonup Evt$ denotes the space of *partial* functions over Evt.

Lemma A.7 *For all $B \in \mathbf{L}^{wf}_{fin}$ with $fv(B) \subseteq \{X\}$, there is a functional transformer $\Psi_B \colon (Evt \rightharpoonup Evt) \rightharpoonup (Evt \rightharpoonup Evt)$ such that*

1. $\phi_1 \subseteq \phi_2$ *implies* $\Psi_B(\phi_1) \subseteq \Psi_B(\phi_2)$;
2. $\phi\colon \mathcal{P} \cong \mathcal{Q}$ *implies* $\Psi_B(\phi)\colon [\![B]\!](\mathcal{P}) \cong [\![B]\!](\mathcal{Q})$.

Proof. If $\phi_i\colon Evt \rightharpoonup Evt$ for $i = 1, 2$, then $(\phi_1 \times \phi_2)\colon Evt \rightharpoonup Evt$ is defined by $(e_1, e_2) \mapsto (\phi_1(e_1), \phi_2(e_2))$. Φ_B is defined inductively on the structure of B, as follows:

$$\begin{aligned}
\Psi_{\mathbf{0}_A}(\phi) &= \emptyset \\
\Psi_a(\phi) &= \{(e, e)\} \\
\Psi_{B_1 \diamond B_2}(\phi) &= \Psi_{B_1}(\phi) \times \Psi_{B_2}(\phi) \quad \text{for } \diamond \in \{+, \|_A, \cdot\} \\
\Psi_{B[r]} &= \Psi_B(\phi) \times id_{Evt} \\
\Psi_X(\phi) &= \phi
\end{aligned}$$

The properties 1. and 2. of the lemma are now straightforward to prove by induction on the structure of B. □

Theorem 4.20 *If $B \in \mathbf{L}^{wf}$ with $fv(B) \subseteq X$, and X is dependently guarded in B, then $[\![recX.\, B]\!]$ is the unique solution of $X = B$ modulo $\cong$ in $\mathbf{M}_{[\mathcal{A}_X]_I}$.*

Proof. First we prove that $[\![recX.\, B]\!]$ is indeed a solution of $X = B$, by showing that it is the (unique) fixpoint of the function $[\![B]\!](-)\colon \mathbf{M}_T \to \mathbf{M}_T$, with $T = [\mathcal{A}_X]_I$. Above, we observed that the limit $\mathcal{P}$ of an arbitrary Cauchy sequence $(\mathcal{P}_i)_i$ in $\langle \mathbf{M}, \delta \rangle$ can be constructed according to

$$\mathcal{P} = \bigcup_{i \in \mathbb{N}} \bigcap_{j \geq i} \mathcal{P}_i \ .$$

In particular, this holds for $([\![B]\!]^i(\mathcal{P}))_i$ obtained by applying $[\![B]\!](-)$ i times to an arbitrary starting point $\mathcal{P} \in \mathbf{M}_T$ (which is a Cauchy sequence because $[\![B]\!](-)$ is contracting, see Proposition 4.19); and in even more particular, it also holds for $([\![B^i_X]\!])_i$, which equals the above sequence if we choose the starting point $\bot_T = [\![\mathbf{0}_{\mathcal{A}_X}]\!]$. The special feature of this starting point is that $\bot_T \subseteq \mathcal{P}$ for all $\mathcal{P} \in \mathbf{M}_T$, and hence $[\![B^0_X]\!] = \bot_T \subseteq [\![B^1_X]\!]$. Consequently, it can be proved by induction on i, using the monotonicity of $[\![B]\!]$ w.r.t. $\subseteq$ (Lemma A.6) that $[\![B^i_X]\!] \subseteq [\![B^{i+1}_X]\!]$ for all $i \in \mathbb{N}$. This, in turn, implies that the limit of this sequence can be constructed more simply by

$$\bigcup_{i \in \mathbb{N}} [\![B]\!]^i(\bot_X) = \bigcup_{i \in \mathbb{N}} [\![B^i_X]\!] = [\![recX.\, B]\!] \ .$$

Now we prove that all solutions of $X = B$ modulo isomorphism are isomorphic to $[\![recX.\, B]\!]$. A system model $\mathcal{P} \in \mathbf{M}_T$ is a solution of $X = B$ modulo isomorphism iff $\mathcal{P} \cong [\![B]\!](\mathcal{P})$, i.e., if $\mathcal{P} = \phi([\![B]\!](\mathcal{P}))$ for some bijective $\phi\colon Evt \to Evt$, meaning that $\mathcal{P}$ is a fixpoint of the function $\phi[\![B]\!](-)\colon \mathbf{M}_T \to \mathbf{M}_T$ defined by $\mathcal{Q} \mapsto \phi([\![B]\!](\mathcal{Q}))$ for all $\mathcal{Q} \in \mathbf{M}_T$.

$\phi[\![B]\!](-)$ is contracting since $[\![B]\!](-)$ is; therefore, its fixpoint $\mathcal{P}$ is unique and can be constructed in a similar way as $[\![recX.\ B]\!]$:

$$\mathcal{P} = \bigcup_{i \in \mathbb{N}} (\phi[\![B]\!])^i(\bot_T) \ .$$

Now consider the following sequence of bijections $\psi^i \colon Evt \to Evt$ for all $i \in \mathbb{N}$, where Ψ_B is the isomorphism transformer whose existence was shown in Lemma A.7 above:

$$\begin{aligned} \psi^0 &= \emptyset \\ \psi^{i+1} &= \phi \circ \Psi_B(\psi^i) \ . \end{aligned}$$

By Lemma A.7, it follows that $\psi^i \colon [\![B]\!](\bot_T) \cong (\phi[\![B]\!])^i(\bot_T)$ and $\psi^i \subseteq \psi^{i+1}$ for all $i \in \mathbb{N}$. It follows that $\psi \colon [\![recX.\ B]\!] \cong \mathcal{P}$ where $\psi = \bigcup_{i \in \mathbb{N}} \psi^i$. □

Correspondence of operational and denotational semantics. For the purpose of proving the correspondence result Theorem 4.22 we introduce additional constants $t_\mathcal{P}$ to $\mathbf{L}^{wf}$ for every $\mathcal{P} \in \mathbf{M}$, with alphabet given by

$$\mathcal{A}(t_\mathcal{P}) = \{a \in Act \mid \forall p \in \mathcal{P} \colon a \ I \ \mathcal{A}_p, p \cup a \in \mathcal{P}\} \ ,$$

operational semantics generated by

$$\frac{\mathcal{P} \xrightarrow{\alpha} \mathcal{P}'}{t_\mathcal{P} \xrightarrow{\alpha} t_{\mathcal{P}'}}$$

and denotational semantics determined by

$$[\![t_\mathcal{P}]\!] = \mathcal{P} \ .$$

It is immediately clear that $lts(\mathcal{P}) \sim lts(t_\mathcal{P})$; moreover, for all $a \ I \ \mathcal{A}(T_\mathcal{P})$, $t_\mathcal{P} \xrightarrow{\surd_a} t_{\mathcal{P}'}$ iff $t_\mathcal{P} = t_{\mathcal{P}'}$. As remarked in Sect. 3.4, the proof of Theorem 3.15 is not invalidated if we extend $\mathbf{L}$ in this way.

Furthermore, for the correspondence result we will use an alternative representation of the operational semantics of system models, in which the states are system models $\mathcal{P} \in \mathbf{M}$. The intuition behind this alternative representation is that from $\mathcal{P}$, an a-labelled transition may occur if there is a (causally) minimal a-labelled event in one of the system runs of $\mathcal{P}$; the resulting target model consists of all system runs which had that event in a causally minimal position, minus the event itself. A $\surd_a$-labelled transition may occur if the model contains an empty system run that is partially terminated for a; the resulting model consists of all runs that are partially terminated for a.

To formalise this, we introduce the concept of a *remainder*: if $p \in \mathcal{P}$ then $\mathcal{P} - p$ corresponds to the "difference" between $\mathcal{P}$ and p, i.e., what is left of $\mathcal{P}$ after p has been done. Formally:

$$\mathcal{P} - p = \{q - p \mid p \preceq (q \cup T_p) \in \mathcal{P}\} \ .$$

It is not difficult to see that $\mathcal{P} - p$ is a system model whenever $p \in \mathcal{P}$. Moreover, the following property holds for all $p \preceq (q \cup T_p) \in \mathcal{P}$:

$$\mathcal{P} - (q \cup T_p) = (\mathcal{P} - p) - (q - p) \ . \tag{4}$$

Now $\rightarrow \subseteq \mathbf{M} \times (Act \cup \checkmark_{Act}) \times \mathbf{M}$ is defined as the smallest relation such that:

- $\mathcal{P} \xrightarrow{\ell(e)} \mathcal{P} - \langle e \rangle$ if $\langle e \rangle \in \mathcal{P}$;
- $\mathcal{P} \xrightarrow{\checkmark_a} \mathcal{P} - \varepsilon_a$ if $\varepsilon_a \in \mathcal{P}$.

It follows that we can define a mapping $lts \colon \mathbf{M} \to \mathbf{LTS}$, as follows:

$$lts(\mathcal{P}) = \langle \mathbf{M}, \rightarrow, \mathcal{P} \rangle \ .$$

The following proposition states that the two methods for defining transitions on the denotational model give rise to bisimilar interpretations.

Proposition A.8 *For all* $\mathcal{P} \in \mathbf{M}$, $lts(\mathcal{P}) \sim \langle \mathcal{P}, \rightarrow, \varepsilon_\emptyset \rangle$.

Proof. Let $\rho = \{(\mathcal{P} - p, p) \mid p \in \mathcal{P}\}$. We prove that ρ is a bisimulation relation. Let $(\mathcal{P} - p, p) \in \rho$ be arbitrary.

- Assume $(\mathcal{P} - p) \xrightarrow{\alpha} \mathcal{P}'$.
  - If $\alpha = a$ then $\mathcal{P}' = (\mathcal{P} - p) - \langle e \rangle$ such that $\langle e \rangle \in (\mathcal{P} - p)$ and $\ell(e) = a$. It follows that $\langle e \rangle = p' - p$ where $p \preceq (p' \cup T_p) \in \mathcal{P}$. Let $p'' = p' \cup T_p$; then $e \in \max E_{p''}$, $a \ I \ T_{p''}$ and $p = p'' \setminus e$ and hence $p \xrightarrow{a} p''$. Moreover, $\mathcal{P}' = (\mathcal{P} - p) - (p' - p) = \mathcal{P} - p''$ according to (4) and hence $(\mathcal{P}', p'') \in \rho$.
  - If $\alpha = \checkmark_a$ then $\varepsilon_a \in \mathcal{P} - p$ and $\mathcal{P}' = (\mathcal{P} - p) - \varepsilon_a$. It follows that $\varepsilon_a = p' - p$ where $p \preceq (p' \cup T_p) \in \mathcal{P}$. Let $p'' = p' \cup T_p$; then $p'' = p \cup a$ and hence $p \xrightarrow{\checkmark_a} p''$. Moreover, $\mathcal{P}' = (\mathcal{P} - p) - (p' - p) = \mathcal{P} - p''$ according to (4) and hence $(\mathcal{P}', p'') \in \rho$.
- Assume $p \xrightarrow{\alpha} p'$.
  - If $\alpha = a$ then there is an $e \in \max E_{p'}$ such that $\ell(e) = a \ I \ T_p$ and $p = p' \setminus e$. Let $p'' = p' \setminus T_p$; hence $p' = p'' \cup T_p$. It follows that $p'' - p = \langle e \rangle$ and $p \preceq p' \in \mathcal{P}$; hence $p'' - p \in \mathcal{P} - p$, implying $(\mathcal{P} - p) \xrightarrow{a} (\mathcal{P} - p) - (p'' - p) = \mathcal{P} - p'$ (the last equation by (4)). Since $(\mathcal{P} - p', p') \in \rho$, we are done.
  - If $\alpha = \checkmark_a$ then $p' = p \cup a$. Let $p'' = (p \setminus T_p) \cup a$; hence $p' = p'' \cup T_p$. It follows that $p'' - p = \varepsilon_a$ and $p \preceq p' \in \mathcal{P}$; hence $p'' - p \in \mathcal{P} - p$, implying $(\mathcal{P} - p) \xrightarrow{\checkmark_a} (\mathcal{P} - p) - (p'' - p) = \mathcal{P} - p'$ (the last equation by (4)). Since $(\mathcal{P} - p', p') \in \rho$, we are done. □

It should be noted that, although the two transition systems are bisimilar, they are in no way isomorphic: the former distinguishes many states that are identified in the latter. These distinctions are sometimes based on information that is irrelevant; for instance, $[\![\mathbf{0}_A]\!]$ interpreted as a transition system has as many states as there are system runs, to the number of 2^n where n is the number of actions independent of A; all of these states, however, are bisimilar, and indeed are identified in the latter interpretation (where there is just a single state, $[\![\mathbf{0}_A]\!]$ itself).

Theorem 4.22 *For all closed $B \in \mathbf{L}^{wf}$, $B \sim lts([\![B]\!])$.*

Proof. In the sequel, we write $C \sim \mathcal{P}$ rather than $lts(C) \sim lts(\mathcal{P})$. The theorem is proved by a nested induction on the recursion depth, i.e., the number of nested recursion operators in B (outer induction) and the structure of B (inner induction). The outer induction hypothesis is that the theorem holds whenever the recursion depth of B is smaller than i (starting at $i = 0$, where the statement is vacuously true).

Auxiliary constants. Assume $B = t_{\mathcal{P}}$ for some $\mathcal{P} \in \mathbf{M}$. As we saw above, $B \sim [\![B]\!]$ by construction.

Deadlock constants. Assume $B = \mathbf{0}_A$ for some $A \subseteq Act$. Then $\rho = \{(B, [\![B]\!])\}$ is a bisimulation relation, and hence $B \sim [\![B]\!]$. We prove only the second (reverse) simulation property; the proof of the first one is analogous.

Assume $[\![B]\!] \xrightarrow{\alpha} \mathcal{P}$. Since $E_{[\![B]\!]} = \emptyset$, it follows that $\alpha = \checkmark_a$ for some $a \in Act$; by construction of $[\![\mathbf{0}_A]\!]$, it follows that $A \; I \; a$ and $\mathcal{P} = [\![B]\!]$. By the operational semantics, $B \xrightarrow{\checkmark_a} \mathbf{0}_A$; moreover, $(\mathbf{0}_A, \mathcal{P}) \in \rho$.

Single actions. Assume $B = b$ for some $b \in Act$. We prove that

$$\rho = \{(B, [\![B]\!]), (\mathbf{0}_{Act}, [\![\mathbf{0}_{Act}]\!])\}$$

is a bisimulation relation; this implies $B \sim [\![B]\!]$. We only prove reverse simulation of the first pair; the other simulation direction is analogous, and the second pair was covered by the previous case.

- Assume $[\![B]\!] \xrightarrow{a} \mathcal{P}$. By construction of $[\![B]\!]$, it follows that $a = b$ and $\mathcal{P} = [\![\mathbf{0}_{Act}]\!]$. This is matched by $B \xrightarrow{a} B'$ with $B' = \mathbf{0}_{Act}$.
- Now assume $[\![B]\!] \xrightarrow{\checkmark_a} \mathcal{P}$. By construction of $[\![B]\!]$, it follows that $a \; I \; b$ and hence $\mathcal{P} = [\![B]\!]$. This is matched by $B \xrightarrow{\checkmark_a} B'$ with $B' = B = b$.

Choice. Assume $B = B_1 + B_2$, where $B_i \sim [\![B_i]\!]$ for $i = 1, 2$ (inner induction hypothesis). We show that

$$\begin{aligned}\rho = &\{(C_1 + C_2, \iota_1(\mathcal{P}_1) + \iota_2(\mathcal{P}_2)) \mid C_1 \sim \mathcal{P}_1, C_2 \sim \mathcal{P}_2\} \\ &\cup \{(C, \mathcal{P}) \mid C \sim \mathcal{P}\}\end{aligned}$$

is a bisimulation relation; this implies $B \sim [\![B]\!]$. We prove only reverse simulation of the first component; the other simulation direction is analogous, and the other pairs are bisimilar by assumption. Let $C = C_1 + C_2$ and $\mathcal{P} = \iota_1(\mathcal{P}_1) + \iota_2(\mathcal{P}_2)$.

- Assume $\mathcal{P} \xrightarrow{a} \mathcal{P}'$. Let $e \in E_{\mathcal{P}}$ be the event that occurred; then $\pi_i(e) \in E_{\mathcal{P}_i}$ for (exclusively) $i = 1$ or $i = 2$. Assume $i = 1$; the other case is symmetrical. Then $e \in \min E_p$ for $p \in \mathcal{P}$ iff $\pi_1(e) \in \min E_{\pi_1(p)}$ for $\pi_1(p) \in \mathcal{P}_1$. It follows that $\mathcal{P}_1 \xrightarrow{a} \mathcal{P}_1'$ where $\mathcal{P}' = \iota_1(\mathcal{P}_1')$. By the inner induction hypothesis, $C_1 \xrightarrow{a} C_1'$ such that $C_1' \sim \mathcal{P}_1' \cong \mathcal{P}'$, implying $(C_1', \mathcal{P}') \in \rho$; and by the operational semantics, $C \xrightarrow{a} C_1'$.
- Assume $\mathcal{P} \xrightarrow{\surd_a} \mathcal{P}'$. It follows that $\mathcal{P}' = \mathcal{P}_1' \cup \mathcal{P}_2'$ where for $i = 1, 2$, $\mathcal{P}_i' = \{p \mid p \cup a \in \iota_i(\mathcal{P}_1), a \; I \; \mathcal{A}_p\}$. It follows that $\mathcal{P}_i' \in \mathbf{M}$ iff $\varepsilon_\emptyset \in \mathcal{P}_i'$ iff $\varepsilon_{\{a\}} \in \mathcal{P}_i$.
  If $\varepsilon_{\{a\}} \in \mathcal{P}_i$ for both $i = 1, 2$ then $\mathcal{P}_i \xrightarrow{\surd_a} \pi_i(\mathcal{P}_i')$, implying (by the inner induction hypothesis) that $C_i \xrightarrow{\surd_a} C_i'$ such that $C_i' \sim \pi_i(\mathcal{P}_i')$; hence $(C_1' + C_2', \iota_1(\pi_1(\mathcal{P}_1')) + \iota_2(\pi_2(\mathcal{P}_2'))) = (C_1' + C_2', \mathcal{P}') \in \rho$. Moreover, by the operational semantics $C \xrightarrow{\surd_a} C_1' + C_2'$.
  Otherwise, assume $\varepsilon_{\{a\}} \notin \mathcal{P}_i$ for $i = 1$ (the case $i = 2$ is symmetrical). It follows that $\iota_1(\mathcal{P}_1) \xrightarrow{\surd_a\!\!\!\!\!\!/} $ and $\iota_2(\mathcal{P}_2) \xrightarrow{\surd_a} \mathcal{P}'$, implying $\mathcal{P}_1 \xrightarrow{\surd_a\!\!\!\!\!\!/} $ and $\mathcal{P}_2 \xrightarrow{\surd_a} \pi_1(\mathcal{P}')$, hence by the inner induction hypothesis, $C_1 \xrightarrow{\surd_a\!\!\!\!\!\!/} $ and $C_2 \xrightarrow{\surd_a} C_2'$ such that $C_2' \sim \pi_1(\mathcal{P}') \cong \mathcal{P}'$; hence $(C_2', \mathcal{P}') \in \rho$. By the operational semantics, finally, $C \xrightarrow{\surd_a} C_2'$.

Parallel composition. The parallel composition of system models can be characterised alternatively as $\mathcal{P}_1 \parallel_A \mathcal{P}_2 = \bigcup\{p_1 \parallel_A p_2 \in \mathbf{P} | p_1 \in \mathcal{P}_1, p_2 \in \mathcal{P}_2\}$ where

$$p_1 \parallel_A p_2 = \{q \mid \begin{array}{l} E_q \subseteq (E_{p_1} \parallel_A E_{p_2}), \\ <_{p_i} = \{(\pi_i(d), \pi_i(e)) \mid d <_q e\} \quad \text{for } i = 1, 2, \\ T_q = T_{p_1} = T_{p_2}\} \end{array}$$

Assume $B = B_1 \parallel_A B_2$, where $B_i \sim [\![B_i]\!]$ for $i = 1, 2$ (inner induction hypothesis). We prove that

$$\rho = \{(C_1 \parallel_A C_2, \mathcal{P}_1 \parallel_A \mathcal{P}_2) \mid C_1 \sim \mathcal{P}_1, C_2 \sim \mathcal{P}_2\}$$

is a bisimulation relation; this implies $B \sim [\![B]\!]$. We prove only reverse simulation; the other simulation direction is analogous. Let $C = C_1 \parallel_A C_2$ and $\mathcal{P} = \mathcal{P}_1 \parallel_A \mathcal{P}_2$.

- Assume $\mathcal{P} \xrightarrow{a} \mathcal{P}'$. Assume $(e_1, e_2) \in E_{\mathcal{P}}$ is the a-labelled event that occurred. According to the above alternative characterisation, $q \in \mathcal{P}$ with $(e_1, e_2) \in \min E_q$ iff $q \in p_1 \parallel_A p_2$ with $p_i \in \mathcal{P}_i$ and either $e_i \in \min E_{p_i}$ or $e_i = *$ for $i = 1, 2$.

If $a \notin A$ then (by construction of $E_{p_1} \parallel_A E_{p_2}$) either $e_1 = *$ or $e_2 = *$. Assume the latter; the other case is symmetrical. It follows that

$$\{q \setminus (e_1, e_2) \mid q \in p_1 \parallel_A p_2\} = (p_1 \setminus e_1) \parallel_A p_2$$

hence $\mathcal{P}' = \mathcal{P}'_1 \parallel_A \mathcal{P}_2$ where $\mathcal{P}'_1 = \{p \setminus e_1 \mid p \in \mathcal{P}_1, e_1 \in \min E_p\}$. Hence $\mathcal{P}_1 \xrightarrow{a} \mathcal{P}'_1$, implying (by the inner induction hypothesis) $C_1 \xrightarrow{a} C'_1$ such that $C'_1 \sim \mathcal{P}'_1$; hence $(C'_1 \parallel_A C_2, \mathcal{P}') \in \rho$. By the operational semantics, it follows that $C \xrightarrow{a} C'_1 \parallel_A C_2$.
Otherwise $e_1 \neq * \neq e_2$, and hence

$$\{q \setminus (e_1, e_2) \mid q \in p_1 \parallel_A p_2\} = (p_1 \setminus e_1) \parallel_A (p_2 \setminus e_2) \ .$$

It follows that $\mathcal{P}' = \mathcal{P}'_1 \parallel_A \mathcal{P}'_2$ where $\mathcal{P}'_i = \{p \setminus e_i | p \in \mathcal{P}_i, e \in \min E_p\}$. Hence $\mathcal{P}_i \xrightarrow{a} \mathcal{P}'_i$, implying (by the inner induction hypothesis) $C_i \xrightarrow{a} C'_i$ such that $C'_i \sim \mathcal{P}'_i$; hence $(C'_1 \parallel_A C'_2, \mathcal{P}') \in \rho$. By the operational semantics, it follows that $C \xrightarrow{a} C'_1 \parallel_A C'_2$.

– Assume $\mathcal{P} \xrightarrow{\checkmark_a} \mathcal{P}'$. If $q \in p_1 \parallel_A p_2$ where $p_i \in [\![B_i]\!]$ for $i = 1, 2$, then $q \cup a \in \mathcal{P}$ and $a \ I \ \mathcal{A}_q$ iff $p_i \cup a \in \mathcal{P}_i$ and $a \ I \ \mathcal{A}_{p_i}$ for $i = 1, 2$. It follows that $\mathcal{P}' = \mathcal{P}'_1 \parallel_A \mathcal{P}'_2$ where $\mathcal{P}'_i = \{p \mid p \cup a \in \mathcal{P}_i, a \ I \ \mathcal{A}_p\}$.
Due to $\varepsilon_{\{a\}} \in \mathcal{P}$ we know that $\varepsilon_{\{a\}} \in \mathcal{P}_i$ and hence $\mathcal{P}_i \xrightarrow{\checkmark_a} \mathcal{P}'_i$ for $i = 1, 2$; hence (by the inner induction hypothesis) $C_i \xrightarrow{\checkmark_a} C'_i$ such that $C'_i \sim \mathcal{P}'_i$ for $i = 1, 2$; hence $(C'_1 \parallel_A C'_2, \mathcal{P}') \in \rho$. By the operational semantics, it follows that $C \xrightarrow{\checkmark_a} C'_1 \parallel_A C'_2$.

Sequential composition. The sequential composition of system models is given alternatively by $\mathcal{P}_1 \cdot \mathcal{P}_2 = \{p_1 \cdot p_2 | p_1 \in \mathcal{P}_1, p_2 \in \mathcal{P}_2, \mathcal{A}_{p_2} \subseteq T_{p_1}\}$ where

$$p_1 \cdot p_2 = \langle E_{p_1} \cup E_{p_2}, \leq_{p_1} \cup ((E_{p_1} \times E_{p_2}) \cap D) \cup \leq_{p_2}, T_{p_1} \cap T_{p_2} \rangle \ .$$

Assume $B = B_1 \cdot B_2$, where $B_i \sim [\![B_i]\!]$ for $i = 1, 2$ (inner induction hypothesis). We prove that

$$\rho = \{(C_1 \cdot C_2, \iota_1(\mathcal{P}_1) \cdot \iota_2(\mathcal{P}_2)) \mid C_1 \sim \mathcal{P}_1, C_2 \sim \mathcal{P}_2\}$$

is a bisimulation relation; this implies $B \sim [\![B]\!]$. We prove only reverse simulation; the other simulation direction is analogous. Let $C = C_1 \cdot C_2$ and $\mathcal{P} = \iota_1(\mathcal{P}_1) \cdot \iota_2(\mathcal{P}_2)$.

– Assume $\mathcal{P} \xrightarrow{a} \mathcal{P}'$. Assume $e \in E_{\mathcal{P}}$ is the a-labelled event that occurred. It follows that $e = (d, *)$ or $e = (*, d)$; furthermore, $q \in \mathcal{P}$ with $e \in \min E_q$ iff $q = p_1 \cdot p_2$ where $p_i \in \iota_i(\mathcal{P}_i)$ for $i = 1, 2$ with $\mathcal{A}_{p_2} \subseteq T_{p_1}$ and either $e \in \min E_{p_1}$ (if $e = (d, *)$) or $a \ I \ \mathcal{A}_{p_1}$ and $e \in \min E_{p_2}$ (if $e = (*, d)$).
In the former case, it follows that $q \setminus e = (p_1 \setminus e) \cdot p_2$; hence $\mathcal{P}' = \iota_1(\mathcal{P}'_1) \cdot \iota_2(\mathcal{P}_2)$ with $\mathcal{P}'_1 = \{p \setminus e \mid p \in \mathcal{P}_1), e \in \min E_p\}$. But then $\mathcal{P}_1 \xrightarrow{a}$

$\mathcal{P}'_1$, implying (by the inner induction hypothesis) $C_1 \xrightarrow{a} C'_1$ with $C'_1 \sim \mathcal{P}'_1$; hence $(C'_1 \cdot C_2, \mathcal{P}'_1) \in \rho$. By the operational semantics, therefore, $C \xrightarrow{a} C'_1 \cdot C_2$.
In the latter case, it follows that $q \setminus e = p_1 \cdot (p_2 \setminus e)$; hence $\mathcal{P}' = \mathcal{P}'_1 \cdot \mathcal{P}'_2$ with $\mathcal{P}'_1 = \{p \mid p \cup a \in \mathcal{P}_1, a \; I \; \mathcal{A}_p\}$ and $\mathcal{P}'_2 = \{p \setminus e | p \in \mathcal{P}_2, e \in \min E_p\}$. But then $\mathcal{P}_1 \xrightarrow{\checkmark_a} \mathcal{P}'_1$ and $\mathcal{P}_2 \xrightarrow{a} \mathcal{P}'_2$, implying (by the inner induction hypothesis) $C_1 \xrightarrow{\checkmark_a} C'_1$ and $C_2 \xrightarrow{a} C'_2$ with $C'_i \sim \mathcal{P}'_i$; hence $(C'_1 \cdot C'_2, \mathcal{P}') \in \rho$. By the operational semantics, therefore, $C \xrightarrow{a} C'_1 \cdot C'_2$.

– Assume $\mathcal{P} \xrightarrow{\checkmark_a} \mathcal{P}'$. The proof is analogous to that for $\checkmark_a$-transitions of parallel compositions (see above).

Refinement. The refinement of system models is given alternatively by

$$\mathcal{P}[\mathcal{R}] = \{w(p) \mid p \in \mathcal{P}, w \text{ is an } \mathcal{R}\text{-witness for } p\}$$

where an $\mathcal{R}$-witness for p is a function $w \colon E_p \to \mathbf{M}$ such that $w(d) \in \mathcal{R}(\ell(d))$ and $w(d) \neq \varepsilon_T$ for all $d \in E_p$ and $\mathcal{A}_{w(d)} \subseteq T_{w(d')}$ whenever $d \not\leq_p d'$. The refinement of a system run p according to a witness w is defined by

$$\begin{aligned} w(p) = \langle\, & \{(d, e) \mid d \in E_p, e \in E_{w(d)}\}, \\ & \{((d_1, e_1), (d_2, e_2)) \mid d_1 <_p d_2, e_1 \; D \; e_2 \text{ or } d_1 = d_2, e_1 <_{w(d_1)} e_2\}, \\ & T_{p_1} \cap \textstyle\bigcap_{e \in E_{p_1}} T_{w(e)} \rangle \;. \end{aligned}$$

Assume $B = B_1[r]$; due to well-formedness of B, r is strongly D-consistent. Let $\mathcal{R} \colon a \mapsto [\![r(a)]\!]$ for all $a \in Act$. Assume $B_1 \sim [\![B_1]\!]$ and $r(a) \sim \mathcal{R}(a)$ for all $a \in Act$ (inner induction hypothesis). We prove that $\rho = \bigcup_i \rho_i$ with

$$\begin{aligned} \rho_0 &= \{(C_1[r], \mathcal{P}_1[\mathcal{R}]) \mid C_1 \sim \mathcal{P}_1\} \\ \rho_{i+1} &= \{(C_2 \cdot C_1, (d \times \mathcal{P}_2) \cdot \mathcal{P}_1) \mid C_2 \sim \mathcal{P}_2, (C_1, \mathcal{P}_1) \in \rho_i, d \notin \pi_1(E_{\mathcal{P}_1})\} \end{aligned}$$

is a bisimulation relation; this implies $B \sim [\![B]\!]$. We prove only reverse simulation of pairs in ρ_0; the other simulation direction is analogous, and the proof for ρ_i with $i > 0$ is analogous to that for weak sequential composition. Let $C = C_1[r]$ and $\mathcal{P} = \mathcal{P}_1[\mathcal{R}]$.

– Assume $\mathcal{P} \xrightarrow{a} \mathcal{P}'$. Assume (d, e) is the a-labelled event responsible for this. We have $q \in \mathcal{P}$ with $(d, e) \in \min E_q$ iff $q = w(p)$ for $p \in \mathcal{P}'$ and w an $\mathcal{R}$-witness on p with $d \in \min E_p$ and $e \in \min E_{w(d)}$ (where the "only if" is due to Proposition 4.12). It follows that

$$q \setminus (d, e) = (d \times (w(d) \setminus e)) \cdot w'(p \setminus d)$$

where $w' = w \restriction (E_p \setminus d)$. Note that w' is indeed an $\mathcal{R}$-witness on $p \setminus d$. Moreover, since $d' \not\leq_p d$ for all $d' \in p \setminus d$, it follows that

$$\mathcal{A}_{w'(p \setminus d)} = \textstyle\bigcup_{d' \in E_{p \setminus d}} \mathcal{A}_{w'(d)} \subseteq T_{w(d) \setminus e}$$

and hence $d \times (w(d) \setminus e)$ and $w'(p \setminus d)$ satisfy the termination criterion for the weak sequential composition of system runs. Let $b = \ell(d)$; then $\mathcal{P}' = (d \times \mathcal{P}'_2) \cdot \mathcal{P}'_1[\mathcal{R}]$ where

$$\mathcal{P}'_1 = \{p \setminus d \mid p \in \mathcal{P}_1, d \in \min E_p\}$$
$$\mathcal{P}'_2 = \{p \setminus e \mid p \in \mathcal{R}(b), e \in \min E_p\} \ .$$

Thus, $\mathcal{P}_1 \xrightarrow{b} \mathcal{P}'_1$ and $\mathcal{R}(b) \xrightarrow{a} \mathcal{P}'_2$, implying (by the inner induction hypothesis) $C_1 \xrightarrow{b} C'_1$ and $r(b) \xrightarrow{a} C'_2$ such that $C'_1 \sim \mathcal{P}'_1$ and $C'_2 \sim \mathcal{P}'_2$; hence $(C'_2 \cdot C'_1[r], \mathcal{P}') \in \rho_1$. By the operational semantics, then, $C[r] \xrightarrow{a} C'_2 \cdot C'_1[r]$.

– Assume $\mathcal{P} \xrightarrow{\surd a} \mathcal{P}'$. Due to Proposition 4.12, it follows that $a \ I \ \mathcal{A}_{w(p)}$ and $w(p) \cup a \in \mathcal{P}[\mathcal{R}]$ for some $p \in \mathcal{P}'$ and $\mathcal{R}$-witness w on p iff $a \ I \ \mathcal{A}_p$ and $p \cup a \in \mathcal{P}$. It follows that $\mathcal{P}' = \mathcal{P}'_1[\mathcal{R}]$ where $\mathcal{P}'_1 = \{p \mid a \ I \ \mathcal{A}_p, p \cup a \in \mathcal{P}'\}$. Then $\mathcal{P}_1 \xrightarrow{\surd a} \mathcal{P}'_1$, implying (by the inner induction hypothesis) $C_1 \xrightarrow{\surd a} C'_1$ such that $C'_1 \sim \mathcal{P}'_1$, and hence $(C'_1[r], \mathcal{P}')$; the operational semantics imply $C \xrightarrow{\surd a} C'_1[r]$.

Recursion. Assume $B = recX.\, B_1$. By the outer induction hypothesis, it follows that $B_1\langle C/X\rangle \sim [\![B_1\langle C/X\rangle]\!] = [\![B_1]\!]([\![C]\!])$ for arbitrary closed $C \in \mathbf{L}_{fin}$ (since then the recursion depth of $B_1\langle C/X\rangle$ is one smaller than that of B). In particular, this also holds $C = t_{[\![B]\!]}$, implying (since $[\![t_{[\![B]\!]}]\!] = [\![B]\!]$ is a fixpoint of $[\![B_1]\!](-)$) that

$$B_1\langle t_{[\![B]\!]}/X\rangle \sim [\![B_1]\!]([\![B]\!]) = [\![B]\!] \sim t_{[\![B]\!]} \ ;$$

in other words, $t_{[\![B]\!]}$ is a solution of $X = B$ modulo $\sim$. Due to Theorem 3.15, therefore, $B \sim t_{[\![B]\!]} \sim [\![B]\!]$. □

It still remains to prove a proposition of the previous section, which is a corollary of Theorem 4.22.

Proposition 3.16 *For all $B \in \mathbf{L}^{wf}$, $lts(B)$ is partially commutative up to bisimulation.*

Proof. We prove that $\langle \mathbf{M}, \to, \mathcal{P}\rangle$ is partially commutative; i.e., if $\mathcal{P} \xrightarrow{a} \xrightarrow{b} \mathcal{P}'$ with $a \ I \ b$ then $\mathcal{P} \xrightarrow{b} \xrightarrow{a} \mathcal{P}'$. Theorem 4.22 then implies that if $B \xrightarrow{a} \xrightarrow{b} B'$ with $a \ I \ b$ then $B \xrightarrow{b} \xrightarrow{a} B''$ with $B' \sim \mathcal{P}' \sim B''$.

$\mathcal{P} \xrightarrow{a} \xrightarrow{b} \mathcal{P}'$ with $a \ I \ b$ implies $\langle d\rangle \in \mathcal{P}$ and $\langle e\rangle \in \mathcal{P} - \langle d\rangle$ with $\ell(d) = a$ and $\ell(e) = b$, and $\mathcal{P}' = (\mathcal{P} - \langle d\rangle) - \langle e\rangle$. From $\langle e\rangle \in \mathcal{P} - \langle d\rangle$ and $a \ I \ b$ it follows that $p = \left[\begin{smallmatrix} d \\ e \end{smallmatrix}\right] \in \mathcal{P}$; hence $\langle d\rangle = p - \langle e\rangle$, $\langle e\rangle = p - \langle d\rangle \in \mathcal{P}$ and $\langle d\rangle \in \mathcal{P} - \langle e\rangle$. Furthermore, due to (4) we have

$$\mathcal{P}' = (\mathcal{P} - \langle d\rangle) - \langle e\rangle = \mathcal{P} - p = (\mathcal{P} - \langle e\rangle) - \langle d\rangle \ .$$

This concludes the proof. □

A.3 Proofs of Section 5

Theorem 5.4 *The axioms of Table 6 are sound w.r.t. bisimilarity.*

Proof. We show only the soundness of some of the most interesting axioms: C5, S4, RS3, RS4, RD1, RD2 and RF1–RF3. The cases of RS3 and RS4, resp. RF1 and RF2, are proved in reverse order, since the proof of the former depends on the latter.

C5 We show that the following is a bisimulation relation:

$$\mathcal{R} = \{(B + \delta_A(B), B) \mid B \in \mathbf{L}^{wf+}\} \cup \sim$$

We show only left-to-right simulation. Assume $B + \delta_A(B) \xrightarrow{\alpha} B'$.

- If $\alpha = a$, then $B \xrightarrow{a} B'$ and $(B', B') \in \mathcal{R}$.
- If $\alpha = \checkmark_a$, then either $B \xrightarrow{\checkmark_a} B'$ and $a \notin [A]_I$ (i.e. the choice is resolved), in which case $(B', B') \in \mathcal{R}$; or $B \xrightarrow{\checkmark_a} B''$ and $a \in [A]_I$, in which case $B' = B'' + \delta_A(B'')$ and $(B', B'') \in \mathcal{R}$.

S4 The following is a bisimulation relation:

$$\mathcal{R} = \{(B_1 \cdot B_2, B_1 \parallel_\emptyset B_2) \mid \mathcal{A}(B_1)\ I\ \mathcal{A}(B_2)\}\ .$$

We show only left-to-right simulation of proper (i.e., non-termination) transitions. Assume $B_1 \cdot B_2 \xrightarrow{\alpha} B'$; there are two cases.

- $B_1 \xrightarrow{a} B_1'$ such that $B' = B_1' \cdot B_2$. It follows that $B_1 \parallel_\emptyset B_2 \xrightarrow{a} B_1' \parallel_\emptyset B_2$ such that $(B_1' \cdot B_2, B_1' \parallel_\emptyset B_2) \in \mathcal{R}$.
- $B_1 \xrightarrow{\checkmark_a} B_1'$ and $B_2 \xrightarrow{a} B_2'$ such that $B' = B_1' \cdot B_2'$ Due to Proposition 3.8, it follows that $a \in \mathcal{A}(B_2)$, and hence $a\ I\ \mathcal{A}(B_1)$; again by Proposition 3.8, it follows that $B_1' = B_1$. Furthermore, $B_1 \parallel_\emptyset B_2 \xrightarrow{a} B_1 \parallel_\emptyset B_2'$ such that $(B_1 \cdot B_2', B_1 \parallel_\emptyset B_2') \in \mathcal{R}$.

RS4 The following is a bisimulation relation:

$$\mathcal{R} = \{(B{\rightarrow}a, \delta_a(B)) \mid a \notin \mathcal{T}(B)\}$$

We show only left-to-right simulation. Assume $(B_1, B_2) \in \mathcal{R}$ and $B_1 \xrightarrow{\alpha} B_1'$; it follows that $B_1 = B{\rightarrow}a$, $B_2 = \delta_a(B)$ and $a \notin \mathcal{T}(B)$. Proposition 5.1 implies that $\alpha \notin Act$; hence $\alpha = \checkmark_b$ such that $a\ I\ b$.

We may derive that $B_1' = B'{\rightarrow}a$ where $B \xrightarrow{\checkmark_b} B'$, and $B_2 \xrightarrow{\checkmark_b} B_2' = \delta_a(B')$. Proposition 5.1 implies $B \xrightarrow{\checkmark_a\!\!\!\rightarrow}$; thus $B' \xrightarrow{\checkmark_a\!\!\!\rightarrow}$ due to Proposition 3.12 and hence $a \notin \mathcal{T}(B')$ due to Proposition 5.1. We may conclude $(B_1', B_2') \in \mathcal{R}$.

RS3 The following is a bisimulation relation *up to* bisimilarity (see [50]):

$$\mathcal{R} = \{(B{\rightarrow}a, a{\leftarrow}(B \downarrow a) + \delta_a(B)) \mid a \in \mathcal{T}(B)\} \cup \sim$$

We show only left-to-right simulation. Assume $(B_1, B_2) \in \mathcal{R}$ and $B_1 \xrightarrow{\alpha} B_1'$; hence $B_1 = B{\rightarrow}a$ and $B_2 = a{\leftarrow}(B \downarrow a) + \delta_a(B)$ such that $a \in \mathcal{T}(B)$.

- If $\alpha \in Act$, then $\alpha = a$ and $B_1' = B' \cdot \mathbf{1}$, where $B \xrightarrow{\surd_a} B'$. Due to Proposition 5.1, $B' \sim B \downarrow a$. It follows that $B_2 \xrightarrow{a} B_2' = \mathbf{1} \cdot (B \downarrow a)$. Using S1–S2, we obtain $B_1' \sim B_2'$ and hence $(B_1', B_2') \in \mathcal{R}$.
- If $\alpha = \surd_b$, then $a \; I \; b$ and $B_1' = B'{\rightarrow}a$, where $B \xrightarrow{\surd_b} B'$. If $B \downarrow a \xrightarrow{\surd_b} B''$ then $B \xrightarrow{\surd_a} \xrightarrow{\surd_b} B''$ (by Rule R_{24} of Table 4) and hence (according to Proposition 3.12) $B' \xrightarrow{\surd_a} B''$. According to Proposition 5.1, it follows that $a \in \mathcal{T}(B')$ and $B'' \sim B' \downarrow a$. It follows that $B_2 \xrightarrow{\surd_b} B_2' = a{\leftarrow}B'' + \delta_a(B')$, and (since bisimilarity is a congruence for left sequential and choice) $B_2' \sim B_2'' = a{\leftarrow}(B' \downarrow a) + \delta_A(B')$ with $(B_1', B_2'') \in \mathcal{R}$.
  If $B{\downarrow}a \xrightarrow{\surd_b}$ then (according to Proposition 3.12) $B' \xrightarrow{\surd_a}$; hence (according to Proposition 5.1) $a \notin \mathcal{T}(B')$. It follows that $B_2 \xrightarrow{\surd_b} B_2' = \delta_a(B')$, hence $B_1' \sim B_2'$ due to RS4; we may conclude $(B_1', B_2') \in \mathcal{R}$.

RD1 The following is a bisimulation relation:

$$\mathcal{R} = \{((B + C) \downarrow a, B \downarrow a + C \downarrow a) \mid B, C \in \mathbf{L}^{wf+}\} \cup \sim$$

We show only left-to-right simulation. Let $B_1 = (B + C) \downarrow a$ and $B_2 = B \downarrow a + C \downarrow a$, and assume $B_1 \xrightarrow{\alpha} B_1'$; hence $B + C \xrightarrow{\surd_a} B_1'' \xrightarrow{\alpha} B_1'$. We distinguish the following cases:

- $B \xrightarrow{\surd_a} B_1''$ and $C \xrightarrow{\surd_a}$. It follows that $B \downarrow a \xrightarrow{\alpha} B_1'$ and $C \downarrow a \xrightarrow{\alpha}$; hence $B_2 \xrightarrow{\alpha} B_1'$ with $(B_1', B_1') \in \mathcal{R}$.
- $B \xrightarrow{\surd_a}$ and $C \xrightarrow{\surd_a} B_1''$. Symmetrical to the previous case.
- $B \xrightarrow{\surd_a} B'$ and $C \xrightarrow{\surd_a} C'$ with $B_1'' = B' + C'$. If either $\alpha = b$ or $\alpha = \surd_b$ with $B' \xrightarrow{\surd_b}$ or $C' \xrightarrow{\surd_b}$, then either $B' \xrightarrow{\alpha} B_1'$ and hence $B \downarrow a \xrightarrow{\alpha} B_1'$, or $C' \xrightarrow{\alpha} B_1'$ and hence $C \downarrow a \xrightarrow{\alpha} B_1'$; in either case, $B_2 \xrightarrow{\alpha} B_1'$ and $(B_1', B_1') \in \mathcal{R}$. Otherwise $\alpha = \surd_b$ with $B' \xrightarrow{\surd_b} B''$ and $C' \xrightarrow{\surd_b} C''$ such that $B_1' = B'' + C''$; then also $B \downarrow a \xrightarrow{\surd_b} B''$ and $C \downarrow a \xrightarrow{\surd_b} C''$, again implying $B_2 \xrightarrow{\alpha} B_1'$ and $(B_1', B_1') \in \mathcal{R}$.

RD2 We show that the following is a bisimulation relation:

$$\mathcal{R} = \{(b{\leftarrow}B \downarrow a, b{\leftarrow}(B \downarrow a)) \mid a \; I \; b, a \in \mathcal{T}(B)\} \cup \sim$$

Let $B_1 = b{\leftarrow}B \downarrow a$ and $B_2 = b{\leftarrow}(B \downarrow a)$ with $a \; I \; b$ and $a \in \mathcal{T}(B)$, and assume $B_1 \xrightarrow{\alpha} B_1'$. Due to Proposition 5.1, it follows that $B \xrightarrow{\surd_a} B'$ such that $B' \sim B \downarrow a$ and $b{\leftarrow}B' \xrightarrow{\alpha} B_1'$.

- If $\alpha \in Act$ then $\alpha = b$, $B'_1 = \mathbf{1} \cdot B'$ and $B_2 \xrightarrow{b} B'_2 = \mathbf{1} \cdot (B \downarrow a)$. Using S1, we can deduce that $B'_1 \sim B'_2$; hence $(B'_1, B'_2) \in \mathcal{R}$.
- If $\alpha = \checkmark_c$ then $b \; I \; c$ and $B' \xrightarrow{\checkmark_c} B''$ such that $B'_1 = b \leftarrow B''$. Moreover, $B \downarrow a \xrightarrow{\checkmark_c} B''$ and hence $B_2 \xrightarrow{\checkmark_c} b \leftarrow B'' = B'_1$. Since $(B'_1, B'_2) \in \mathcal{R}$, we are done.

RF2 Immediate, by the termination rule for refinement.

RF1 The following is a bisimulation relation:

$$\mathcal{R} = \{(a[r], r(a))\} \cup \sim \; .$$

We prove only left-to-right simulation. Assume $a[r] \xrightarrow{\alpha} B'$. If $\alpha = b$, it follows that $r(a) \xrightarrow{b} C'$ and $B' = C' \cdot \mathbf{1}[r]$; hence $B' \sim C'$ by S2 and RF2. On the other hand, if $\alpha = \checkmark_b$ then $B' = a[r]$. Since $a \; I \; b$, by D-consistency of r it follows that $\mathcal{A}(r(a)) \; I \; b$, hence $r(a) \xrightarrow{\checkmark_b} r(a)$ by Proposition 3.8.

RF3 We only show the case of $\diamond = \cdot$. The case of $\diamond = \leftarrow$ after a single step evolves to this case; $\diamond = +$ is straightforward. Let

$$\mathcal{R}_0 = \{((B_1 \cdot B_2)[r], B_1[r] \cdot B_2[r]) \mid B_1, B_2 \in \mathbf{L}^{wf+}\}$$
$$\mathcal{R}_{i+1} = \{(B_0 \cdot B_1, B_0 \cdot B_2) \mid B_0 \in \mathbf{L}^{wf+}, (B_1, B_2) \in \mathcal{R}_i\} \; .$$

We prove that $\mathcal{R} = \bigcup_{i \in \mathbb{N}} \mathcal{R}_i$ is a bisimulation relation *up to* bisimilarity (see [50]). We show left-to-right simulation of proper (i.e., non-termination) transitions for arbitrary $(B, C) \in \mathcal{R}$, by induction on i. First let $(B, C) \in \mathcal{R}_0$, and assume $B \xrightarrow{a} B'$. There are two cases to consider.

- $B_1 \xrightarrow{b} B'_1$ and $r(b) \xrightarrow{a} B'_0$ such that $B' = B'_0 \cdot (B'_1 \cdot B_2)[r]$. We can then derive $C \xrightarrow{a} C' = (B'_0 \cdot B'_1[r]) \cdot B_2[r]$; by S3, it follows that $C' \sim C'' = B'_0 \cdot (B'_1[r] \cdot B_2[r])$ with $(B', C'') \in \mathcal{R}'$.
- $B_1 \xrightarrow{\checkmark_b} B'_1$, $B_2 \xrightarrow{b} B'_2$ and $r(b) \xrightarrow{a} B'_0$ such that $B' = B'_0 \cdot (B'_1 \cdot B'_2)[r]$. We can then derive $C \xrightarrow{C}{}' = B'_1[r] \cdot (B'_0 \cdot B'_2[r])$; by S3 and S4 (using the fact that $b \; I \; \mathcal{A}(B'_1)$ according to Proposition 3.8, and hence $\mathcal{A}(r(b)) \; I \; \mathcal{A}(B'_1)$ due to D-consistency of r, with $\mathcal{A}(B'_0) \subseteq \mathcal{A}(r(b))$ due to Proposition 3.8), it follows that $C' \sim C'' = B'_0 \cdot (B'_1[r] \cdot B_2[r])$ with $(B', C'') \in \mathcal{R}'$.

Now let $(B, C) \in \mathcal{R}_{i+1}$, and assume $B \xrightarrow{a} B'$. Again, there are two cases to consider.

- $B_0 \xrightarrow{a} B'_0$ and $B' = B'_0 \cdot B_1$. It follows that $C \xrightarrow{a} C' = B'_0 \cdot B_2$ such that $(B', C') \in \mathcal{R}_{i+1}$.
- $B_0 \xrightarrow{\checkmark_a} B'_0$, $B_1 \xrightarrow{a} B'_1$, and $B' = B'_0 \cdot B'_1$. By induction, $B_2 \xrightarrow{B}{}'_2 \sim B''_2$ such that $(B'_1, B''_2) \in \mathcal{R}$; say $(B'_1, B''_2) \in \mathcal{R}_j$. It follows that $C \xrightarrow{a} C' = B'_0 \cdot B'_2$ with $C' \sim C'' = B'_0 \cdot B''_2$ and $(B', C'') \in \mathcal{R}_{j+1}$. $\square$

Completeness for $\mathbf{L}_t$. In order to prove completeness of the equational theory for $\mathbf{L}_t$, we need a number of auxiliary results. First of all, note that Proposition 3.12 generalises to sequences of termination transitions: if $B \xrightarrow{\checkmark_{a_1}} \cdots \xrightarrow{\checkmark_{a_n}} B'$ then $B \xrightarrow{\checkmark_{b_1}} \cdots \xrightarrow{\checkmark_{b_n}} B'$ for arbitrary permutations $b_1 \cdots b_n$ of $a_1 \cdots a_n$. Therefore, we may unambiguously denote

$$B \xrightarrow{\checkmark_T} B' \quad \text{iff} \quad T = \{a_1, \ldots, a_n\} \text{ and } B \xrightarrow{\checkmark_{a_1}} \cdots \xrightarrow{\checkmark_{a_n}} B' \quad .$$

Using this notation, we can state the first auxiliary result, which expresses that deadlock constants add no options to a given behaviour if that behaviour is itself terminated for all actions independent of the deadlock alphabet.

Lemma A.9 *$B + \mathbf{0}_A \sim B$ iff $B \xrightarrow{\checkmark_T}$ for all $T \subseteq_{fin} [A]_I$.*

Proof. The "only if" is due to the fact that if $B \xrightarrow{\checkmark_T}\!\!\!\!\!\!\rightarrow$ for some $T \subseteq_{fin} [A]_I$ then B and $B + \mathbf{0}_A$ would obviously have different termination properties, hence they could not be bisimilar. As for the "if", we prove that the following is a bisimulation relation:

$$\mathcal{R} = \{(B + \mathbf{0}_A, B) \mid \forall T \subseteq_{fin} [A]_I \colon B \xrightarrow{\checkmark_T}\} \cup \sim$$

We show left-to-right simulation. Assume $B + \mathbf{0}_A \xrightarrow{\alpha} B'$. If $\alpha \in Act$ or $\alpha = \checkmark_a$ with $a \; D \; A$, it follows that $B \xrightarrow{\alpha} B'$, hence we are done. Otherwise, $\alpha = \checkmark_a$ with $a \; I \; A$, and hence $B' = B'' + \mathbf{0}_A$ with $B \xrightarrow{\checkmark_a} B''$. For arbitrary $T \subseteq_{fin} [A]_I$ we have $B \xrightarrow{\checkmark_{T \cup \{a\}}}$ and hence $B'' \xrightarrow{\checkmark_T}$; it follows that $(B'' + \mathbf{0}_A, B'') \in \mathcal{R}$. $\square$

Now we show that "$\mathbf{L}_{t,\delta} = \mathbf{L}_t$", i.e., if $B \in \mathbf{L}_t$ then the equational theory allows us to rewrite $\delta_A(B)$ to a term $\mathbf{L}_t$ (namely, to a sum of deadlock constants).

Lemma A.10 *For all $B \in \mathbf{L}_t$, $\mathsf{T} \vdash \delta_A(B) = \sum_{i \in I} \mathbf{0}_{A \cup A_i}$ for some finite nonempty family $A_i \subseteq Act$ for $i \in I$.*

The proof is straightforward (by induction on the structure of B) and hence omitted. As a consequence, the proof system can discard termination constants from choice terms if they do not contribute to the behaviour.

Lemma 5.6 *Let $B \in \mathbf{L}_t$. If $B + \mathbf{0}_A \sim B$, then $\mathsf{T} \vdash B + \mathbf{0}_A = B$.*

Proof. In fact, we show that $B + \mathbf{0}_A \sim B$ implies $\mathsf{T} \vdash \delta_A(B) = \mathbf{0}_A$; this gives rise to the required result due to C5. The proof proceeds by induction on the structure of B. Note that, by Lemma A.9, $B + \mathbf{0}_A \sim B$ implies $B \xrightarrow{\checkmark_T}$ for all $T \subseteq_{fin} [A]_I$.

- $B = \mathbf{0}_{A'}$. Then $a \; I \; A$ for arbitrary $a \in Act$ implies $\mathbf{0}_A \xrightarrow{\checkmark_a}$ and hence $B \xrightarrow{\checkmark_a}$, thus $a \; I \; A'$. It follows that $[A \cup A']_I = [A]_I$; hence we can derive

$$\mathsf{T} \vdash \delta_A(\mathbf{0}_{A'}) = \mathbf{0}_{A \cup A'} = \mathbf{0}_A \quad .$$

– $B = B_1 + B_2$. It follows that for either $i = 1$ or $i = 2$, $B_i \xrightarrow{\checkmark_T}$ for all finite $T \subseteq [A]_I$; for otherwise, $B_1 \xrightarrow{\checkmark_{T_1}}\!\!\!\!\!\!\rightarrow$ and $B_2 \xrightarrow{\checkmark_{T_2}}\!\!\!\!\!\!\rightarrow$ for some $T_1, T_2 \subseteq_{fin} [A]_I$, which contradicts $B \xrightarrow{\checkmark_{T_1 \cup T_2}}$. Hence (due to Lemma A.9) $B_i + \mathbf{0}_A \sim B_i$ for $i = 1$ or $i = 2$. W.l.o.g. assume $i = 1$; since $\mathsf{T} \vdash \delta_A(B_2) = \sum_{j \in J} \mathbf{0}_{A \cup A_j}$ by Lemma A.10, using the induction hypothesis we can derive

$$\mathsf{T} \vdash \delta(B_1 + B_2) = \delta_A(B_1) + \delta_A(B_2) = \mathbf{0}_A + (\textstyle\sum_{j \in J} \mathbf{0}_{A \cup A_j}) = \mathbf{0}_A$$

using $\mathsf{T} \vdash \mathbf{0}_A + \mathbf{0}_{A \cup A_i} = \mathbf{0}_A + \delta_{A_i}(\mathbf{0}_A) = \mathbf{0}_A$ for all $i \in I$.

– $B = a \leftarrow B_1$. Then $b \; I \; A$ implies $\mathbf{0}_A \xrightarrow{\checkmark_b}$ and hence $B \xrightarrow{\checkmark_b}$, thus $b \; I \; a$. It follows that $[A \cup \{a\}]_I = [A]_I$. Moreover, $B_1 \xrightarrow{\checkmark_T}$ for all finite $T \subseteq [A]_I$, and hence (due to Lemma A.9) $B_1 + \mathbf{0}_A \sim B_1$. Using the induction hypothesis for B_1, we can derive

$$\begin{aligned} \mathsf{T} \vdash \delta_A(B) &= \delta_A(a) \leftarrow \delta_A(B_1) = \mathbf{0}_{A \cup \{a\}} \leftarrow \mathbf{0}_A \\ &= \delta_{A \cup \{a\}}(\mathbf{0}_A) = \mathbf{0}_{A \cup \{a\}} = \mathbf{0}_A \quad . \end{aligned}$$

This concludes the proof. □

We are now ready to (re-)state and prove completeness for $\mathbf{L}_t$. Recall that $B = \sum_{i \in I} a_i \leftarrow B_i + \sum_{j \in J} \mathbf{0}_{A_j}$ implies $depth(B) = \max\{1 + depth(B_i) \mid i \in I\}$.

Proposition 5.5 *For all $B_1, B_2 \in \mathbf{L}_t$, $B_1 \sim B_2$ implies $\mathsf{T} \vdash B_1 = B_2$.*

Proof. Let $B_1, B_2 \in \mathbf{L}_t$ with $B_1 \sim B_2$ be given by

$$B_1 = \textstyle\sum_{i \in I} a_i \leftarrow B_i + \sum_{j \in J} \mathbf{0}_{A_j} \quad B_2 = \sum_{k \in K} a_k \leftarrow B_k + \sum_{l \in L} \mathbf{0}_{A_l} \quad .$$

We show $\mathsf{T} \vdash B_1 = B_2$ by induction on $\max\{depth(B_1), depth(B_2)\}$. The proof consists of showing that $\mathsf{T} \vdash B_1 + B_2 = B_2$ and thus (by symmetry) $\mathsf{T} \vdash B_1 + B_2 = B_1$; the required result immediately follows.

First we show that $\mathsf{T} \vdash a_i \leftarrow B_i + B_2 = B_2$ for all $i \in I$. Due to $B_1 \sim B_2$ and $B_1 \xrightarrow{a_i} \mathbf{1} \cdot B_i$, it follows that $B_2 \xrightarrow{a_i} B_2'$ such that $\mathbf{1} \cdot B_i \sim B_2'$; hence $a_i = a_k$ and $B_2' = \mathbf{1} \cdot B_k$ for some $k \in K$. It follows that $B_i \sim B_k$ and hence (by the induction hypothesis) $\mathsf{T} \vdash B_i = B_k$, implying $\mathsf{T} \vdash a_i \leftarrow B_i = a_k \leftarrow B_k$ and hence $\mathsf{T} \vdash a_i \leftarrow B_i + B_2 = B_2$.

Now we show that $\mathsf{T} \vdash \mathbf{0}_{A_j} + B_2 = B_2$ for all $j \in J$. Clearly, $B_1 \xrightarrow{\checkmark_T}$ and hence $B_2 \xrightarrow{\checkmark_T}$ for all finite $T \subseteq [A_j]_I$; by Lemma A.9 this implies $B_2 + \mathbf{0}_{A_j} \sim B_2$. Lemma 5.6 then implies $\mathsf{T} \vdash B_2 + \mathbf{0}_{A_j} = B_2$. □

In order to prove normalisation of $\mathbf{L}^{wf}$ to $\mathbf{L}_t$, we first need to know that similar properties hold for the auxiliary operators.

Lemma 5.8 *Let $B \in \mathbf{L}_t$ be arbitrary.*

1. *$\mathsf{T} \vdash \delta_A(B) = C$ for some $C \in \mathbf{L}_t$ with $depth(C) = 0$.*

2. $\mathsf{T} \vdash B \downarrow a = C$ *for some* $C \in \mathbf{L}_t$ *with* $depth(C) \leq depth(B)$.
3. $\mathsf{T} \vdash B{\leftarrow}a = C$ *for some* $C \in \mathbf{L}_t$ *with* $depth(C) \leq 1 + depth(B)$.

Proof.

1. Immediate from Lemma A.10.
2. Proved by structural induction on B, using Axioms RD1–RD4 (where $\mathcal{T}(x)$ in the side condition of RD2 and RD3 is defined since its argument B_1 does not contain residue operators).
3. Proved using the previous clause 2 and RS3–RS4 (where $\mathcal{T}(x)$ in the side condition of RS3 and RS4 is defined, since its argument B does not contain residue operators).

The normalisation property itself:

Proposition 5.7 *Let* $B_1, B_2 \in \mathbf{L}_t$*, and let* $r\colon Act \to \mathbf{L}_t$ *be strongly D-consistent.*

1. $\mathsf{T} \vdash a = a{\leftarrow}\mathbf{1}$.
2. $\mathsf{T} \vdash B_1 \cdot B_2 = C$ *for some* $C \in \mathbf{L}_t$.
3. $\mathsf{T} \vdash B_1 \parallel_A B_2 = C$ *for some* $C \in \mathbf{L}_t$.
4. $\mathsf{T} \vdash B_1[r] = C$ *for some* $C \in \mathbf{L}_t$.

Proof.

1. First note that using D1–D3 and LS3, we can derive

$$\mathsf{T} \vdash \delta_\emptyset(a{\leftarrow}\mathbf{1}) = \delta_\emptyset(a){\leftarrow}\delta_\emptyset(\mathbf{1}) = \mathbf{0}_{\{a\}}{\leftarrow}\mathbf{1} = \delta_{\{a\}}(\mathbf{1}) = \mathbf{0}_a = \delta_\emptyset(a) \ .$$

   Using S2, S5, RS5 and C5, we then have

$$\mathsf{T} \vdash a = a \cdot \mathbf{1} = a{\leftarrow}\mathbf{1} + a{\rightarrow}\mathbf{1} = a{\leftarrow}\mathbf{1} + \delta_\emptyset(a) = a{\leftarrow}\mathbf{1} + \delta_\emptyset(a{\leftarrow}\mathbf{1}) = a{\leftarrow}\mathbf{1} \ .$$

2. Let $\mathbf{L}_{t,\delta}$ denote the fragment of $\mathbf{L}^+$ consisting of $\mathbf{L}_t$ plus the δ_A-operators; then according to Lemma A.10, any term of $\mathbf{L}_{t,\delta}$ can be rewritten to a term of $\mathbf{L}_t$. We now prove that $\mathsf{T} \vdash B_1 \cdot B_2 = C$ for some $C \in \mathbf{L}_{t,\delta}$, by induction on $depth(B_1) + depth(B_2)$; this implies the required property. Let

$$B_1 = \textstyle\sum_{i\in I} a_i{\leftarrow}B_i + \sum_{j\in J} \mathbf{0}_{A_j} \quad B_2 = \sum_{k\in K} a_k{\leftarrow}B_k + \sum_{l\in L} \mathbf{0}_{A_l} \ .$$

   By S5, LS1–LS3, RS1–RS2 and RS5, it follows that $\mathsf{T} \vdash B_1 \cdot B_2 = C'$ with

$$\begin{aligned} C' = {} & \textstyle\sum_{i\in I} a_i{\leftarrow}(B_i{\cdot}B_2) + \sum_{j\in J} \delta_{A_j}(B_2) \\ & + \displaystyle\sum_{k\in K} (B_1{\rightarrow}a_k){\leftarrow}B_k + \sum_{l\in L} \delta_{A_l}(B_1) \end{aligned}$$

Lemma 5.8.3 states that for each $k \in K$, $\mathsf{T} \vdash B_1 \rightarrow a_k = B_{1,k}$ for some $B_{1,k} \in \mathbf{L}_t$ with $depth(B_{1,k}) \leq 1 + depth(B_1)$; say

$$B_{1,k} = \textstyle\sum_{i \in I_k} a_{i,k} \leftarrow B_{i,k} + \sum_{j \in J_k} \mathbf{0}_{A_{j,k}}$$

where $depth(B_{i,k}) \leq depth(B_1)$ for all $k \in K$ and $i \in I_k$. It follows that

$$\mathsf{T} \vdash (B_1 \rightarrow a_k) \leftarrow B_k = \textstyle\sum_{i \in I_k} a_{i,k} \leftarrow (B_{i,k} \cdot B_k) + \sum_{j \in J_k} \delta_{A_{j,k}}(B_k)$$

for all $k \in K$. By replacing the subterms of C' accordingly, we get $\mathsf{T} \vdash C' = C''$ with $C'' \in \mathbf{L}_{t,\delta}$. Since for all subterms $C_1 \cdot C_2$ of C'', $C_1, C_2 \in \mathbf{L}_t$ and $depth(C_1) + depth(C_2) < depth(B_1) + depth(B_2)$, it follows by induction that these subterms can be rewritten to terms in $\mathbf{L}_{t,\delta}$. It follows that $\mathsf{T} \vdash C'' = C$ for some $C \in \mathbf{L}_{t,\delta}$; hence we are done.

3. Analogous to the previous clause.
4. Assume $r = C_1/a_1, \ldots, C_n/a_n$ with $C_i \in \mathbf{L}_t$ for all $1 \leq i \leq n$. The proof proceeds by induction on the structure of B. The interesting case is $B = a \leftarrow B'$; then

$$\mathsf{T} \vdash (a \leftarrow B')[r] = a[r] \leftarrow B'[r] = r(a) \leftarrow B'[r] \ .$$

By the induction hypothesis, $\mathsf{T} \vdash B'[r] = C'$ for some $C' \in \mathbf{L}_t$. If $a \neq a_i$ for all $1 \leq i \leq n$, then $r(a) = a$; hence $\mathsf{T} \vdash r(a) \leftarrow B'[r] = a \leftarrow C'$ where $a \leftarrow C' \in \mathbf{L}_t$ and we are done.
Otherwise assume $a = a_i$ and assume $C_i = \sum_{j \in J} b_j \leftarrow D_j + \sum_{k \in K} \mathbf{0}_{A_k}$; then

$$\mathsf{T} \vdash r(a) \leftarrow B'[r] = \textstyle\sum_{j \in J} b_j \leftarrow (D_j \cdot C') + \sum_{k \in K} \delta_{A_k}(C')$$

which (latter) term can be rewritten to a term of $\mathbf{L}_t$ due to Proposition 5.7.1 and Clause 3 above. □

A.4 Proofs of Section 6

Theorem 6.1 *If $B_i, C_i \in \mathbf{L}$ and $A_i \subseteq Act$ for $i = 1, 2$ such that for all $i \neq j$*

- *$\mathcal{A}(B_i)\ I\ \mathcal{A}(C_j)$, and*
- *$\mathcal{A}(B_i) \cap A_j = \mathcal{A}(C_i) \cap A_j = \emptyset$*

then (3) holds up to strong bisimilarity.

Proof. We prove the theorem via the operational semantics. We show that the following is a bisimulation relation:

$$\begin{aligned} \mathcal{R} = \ \{\ & ((B_1 \parallel_{A_1} C_1) \cdot (B_2 \parallel_{A_2} C_2), (B_1 \cdot B_2) \parallel_{A_1 \cup A_2} (C_1 \cdot C_2)) \\ & \mid\ \mathcal{A}(B_i)\ I\ \mathcal{A}(C_j), \mathcal{A}(B_i) \cap A_j = \mathcal{A}(C_i) \cap A_j = \emptyset\} \end{aligned}$$

We use the following abbreviations in our proof: $D_i = B_i \|_{A_i} C_i$ for $i = 1, 2$, $D = D_1 \cdot D_2$, $B = B_1 \cdot B_2$, $C = C_1 \cdot C_2$ and $E = B \|_{A_1 \cup A_2} C$.

Assume $D \xrightarrow{\alpha} D'$. There are two cases to consider: $\alpha = \checkmark_a$ or $\alpha = a$. The first is the easier one: if $D \xrightarrow{\checkmark_a} D'$ then $B_i \xrightarrow{\checkmark_a} B_i'$ and $C_i \xrightarrow{\checkmark_a} C_i'$ for $i = 1, 2$ such that $D' = (B_1' \|_{A_1} C_1') \cdot (B_2' \|_{A_2} C_2')$. Then $E \xrightarrow{\checkmark_a} (B_1' \cdot B_2') \|_{A_1 \cup A_2} (C_1' \cdot C_2')$ and $(D', E') \in \mathcal{R}$.

Now assume $\alpha = a \in Act$. There are again various cases to consider.

- The action a comes from the first layer: $D_1 \xrightarrow{a} D_1'$ such that $D' = D_1' \cdot D_2$. The following cases have to be considered:
  - $B_1 \xrightarrow{a} B_1'$ with $a \notin A_1$ and $D_1' = B_1' \|_{A_1} C_1$. Then (by Proposition 3.8.1) $a \in \mathcal{A}(B_1)$ and hence by communication closedness $a \notin A_2$. Therefore $E \xrightarrow{a} (B_1' \cdot B_2) \|_{A_1 \cup A_2} (C_1 \cdot C_2)$ and $(D', E') \in \mathcal{R}$.
  - $C_1 \xrightarrow{a} C_1'$ with $a \notin A_1$. Analogous to the previous case.
  - $B_1 \xrightarrow{a} B_1'$ and $C_1 \xrightarrow{a} C_1'$ and $a \in A_1$ and $D_1' = B_1' \|_{A_1} C_1'$. It follows that $E \xrightarrow{a} E' = (B_1' \cdot B_2) \|_{A_1 \cup A_2} (C_1' \cdot C_2)$ and $(D', E') \in \mathcal{R}$.
- The more interesting case is when the action a comes from the second layer: $D_1 \xrightarrow{\checkmark_a} D_1'$ and $D_2 \xrightarrow{a} D_2'$ such that $D' = D_1' \cdot D_2'$. First note that it follows that $B_1 \xrightarrow{\checkmark_a} B_1'$ and $C_1 \xrightarrow{\checkmark_a} C_1'$ with $D_1' = B_1' \|_{A_1} C_1'$. For the D_2-transition, we have the cases
  - $B_2 \xrightarrow{a} B_2'$ with $a \notin A_2$ and $D_2 = B_2' \|_{A_2} C_2$. It follows (by Proposition 3.8.1) that $a \in \mathcal{A}(B_2)$ and hence by communication closedness, $a \notin A_1$ and $a \; I \; \mathcal{A}(C_1)$; thus (by Proposition 3.8.3) $C_1' = C_1$. Hence $E \xrightarrow{a} E' = (B_1' \cdot B_2') \|_{A_1 \cup A_2} (C_1 \cdot C_2)$ and $(D', E') \in \mathcal{R}$.
  - $C_2 \xrightarrow{a} C_2'$ with $a \notin A_2$. Analogous to the previous case.
  - $B_2 \xrightarrow{a} B_2'$ and $C_2 \xrightarrow{a} C_2'$ such that $a \in A_2$ and $D_2' = B_2' \|_{A_2} C_2'$. It follows that $E \xrightarrow{a} E' = (B_1' \cdot B_2') \|_{A_1 \cup A_2} (C_1' \cdot C_2')$ and $(D', E') \in \mathcal{R}$.

Now assume $E \xrightarrow{\alpha} E'$. The case that $\alpha = \checkmark_a$ is essentially the same as before. Now consider $\alpha = a$.

- $B \xrightarrow{a} B'$ with $a \notin A_1 \cup A_2$ such that $E' = B' \|_{A_1 \cup A_2} C$. We again have to consider two cases:
  - $B_1 \xrightarrow{a} B_1'$ such that $B' = B_1' \cdot B_2$. Then $D \xrightarrow{a} D' = (B_1' \|_{A_1} C_1) \cdot (B_2 \|_{A_2} C_2)$ and $(D', E') \in \mathcal{R}$.
  - $B_1 \xrightarrow{\checkmark_a} B_1'$ and $B_2 \xrightarrow{a} B_2'$ such that $B' = B_1' \cdot B_2'$. Due to $a \in \mathcal{A}(B_2)$ (see Proposition 3.8.1), by communication closedness we again get $a \; I \; \mathcal{A}(C_1)$, hence (due to Proposition 3.8.3) $C_1 \xrightarrow{\checkmark_a} C_1$; thus $D \xrightarrow{a} D' = (B_1' \|_{A_1} C_1) \cdot (B_2' \|_{A_2} C_2)$ and $(D', E') \in \mathcal{R}$.
- $C \xrightarrow{a} C'$ with $a \notin A_1 \cup A_2$. Analogous to the previous case.
- $B \xrightarrow{a} B'$ and $C \xrightarrow{a} C'$ with $a \in A_1 \cup A_2$. We recognise two further cases:

- $a \in A_1$. It follows that $a \notin \mathcal{A}(B_2) \cup \mathcal{A}(C_2)$; hence $B_1 \xrightarrow{a} B_1'$ such that $B' = B_1' \cdot B_2$ and $C_1 \xrightarrow{a} C_1'$ such that $C = C_1' \cdot C_2$, implying $D \xrightarrow{a} D' = (B_1' \parallel_{A_1} C_1') \cdot (B_2 \parallel_{A_2} C_2)$ and $(D', E') \in \mathcal{R}$.
- $a \in A_2$. It follows that $a \notin \mathcal{A}(B_1) \cup \mathcal{A}(C_1)$; hence $B_1 \xrightarrow{\checkmark_a} B_1'$ and $B_2 \xrightarrow{a} B_2'$ such that $B' = B_1' \cdot B_2'$, and $C_1 \xrightarrow{\checkmark_a} C_1'$ and $C_2 \xrightarrow{a} C_2'$ such that $C' = C_1' \cdot C_2'$. It follows that $D \xrightarrow{a} D' = (B_1' \parallel_{A_1} C_1') \cdot (B_2' \parallel_{A_2} C_2')$ and $(D', E') \in \mathcal{R}$. $\square$