Process Algebra with Action Dependencies

Arend Rensink*
Department of Computer Science, University of Twente
rensink@cs.utwente.nl

Heike Wehrheim
Fachbereich Informatik, University of Oldenburg
heike.wehrheim@informatik.uni-oldenburg.de

Abstract

Process algebras are a frequently used tool for the specification and verification of distributed reactive systems. In process algebras, *actions* are used to denote the basic entities of systems. In general, actions are just abstract names with no particular interpretation. The semantics of a system description, given in form of a process algebra term, is not influenced by the choice of names. In this paper, we equip a process algebra with a simple semantics for the actions, given in the form of *dependencies*. The action dependencies are to be 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 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 present 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.

*The research reported here was largely conducted while this author was employed at the University of Hildesheim

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 [58], TCSP [17, 43], ACP [9] or LOTOS [14].

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, 26, 38, 39, 69, 76, 77]. The action refinement operator enables a designer to decompose abstract actions that are regarded as atomic, i.e., whose execution 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 [19]), 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 [55, 56, 57] and others (see for instance [30]). 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 [47, 48]. 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 [78, 72, 73]. The language and basic definitions are given in Section 2. In Section 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 Section 4, we give a denotational counterpart, which is *not* based on an interleaving model but instead on an event-based formalism developed in [66]. 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., [26, 38, 69]. 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 [23]) 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 Section 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 Section 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 [31] and promoted (in a linear time setting) by Zwiers et al., for instance in [47]. 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* $\mathbf{L}$ is the set of all terms B generated by the following productions:

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

where $a \in Act$, $A \subseteq Act$, $r: Act \to \mathbf{L}$ 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* part of the language, and without recursion as the *finite* part; the latter 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 $B\langle C/X\rangle$.

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 $B_1 \parallel\!\!\parallel B_2$ for $B_1 \parallel_\emptyset B_2$. By $\Sigma_{i\in I}B_i = B_{i_1} + B_{i_2} + \ldots$ we denote a choice over all terms B_i out of a finite, non-empty index set I. $B[r : a \mapsto C]$ describes the refinement by a function r which in particular maps a onto C.

Instead of using the operator $rec\, X.$, 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. 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 $\mathcal{A}(B\langle C/X\rangle) \subseteq \mathcal{A}(B)$.

The language covers most of the standard process algebra operators, such as sequential composition, parallel composition, choice and recursion, and one slightly 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

$$
\begin{array}{rl}
\mathcal{A}(\mathbf{0}_A) & := A \\
\mathcal{A}(a) & := \{a\}, \quad \text{where } a \in Act \\
\mathcal{A}(B_1 \mathit{op}\, B_2) & := \mathcal{A}(B_1) \cup \mathcal{A}(B_2), \quad \text{where } \mathit{op} \in \{+, \cdot, \|_A\} \\
\mathcal{A}(B[r]) & := \bigcup_{a \in \mathcal{A}(B)} \mathcal{A}(r(a)) \\
\mathcal{A}(X) & := \mathcal{A}_X \\
\mathcal{A}(\mathit{rec}\, X.B) & := \mathcal{A}_X \quad \text{if } \mathcal{A}(B) \subseteq \mathcal{A}_X
\end{array}
$$

Table 1: Alphabet of a term

(mutual exclusion). The interpretation of all our operators have 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 & Recursion. There is not much to say concerning them. $a \in Act$ describes a process which executes the *action* a and then terminates successfully. Variables and the operator $\mathit{rec}\, X.B$ are used to model infinite behaviour of processes; the latter is interpreted by unrolling it to $B\langle \mathit{rec}\, X.B/X\rangle$. When developing the operational semantics, we will discuss recursion (and the problems it introduces in our setting) in greater detail.

Sequential composition & Termination. Instead of strong sequential composition, action dependencies give rise to a notion of *weak* sequential composition, denoted "$\cdot$". 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 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 [7]). 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. A process is terminated w.r.t. to an action a if there is one run of it which is completed independent of a. As an example, let

$$
\begin{array}{rcl}
B_1 & = & a \cdot b \\
B_2 & = & c \,|||\, d
\end{array}
$$

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 [9]), 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

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

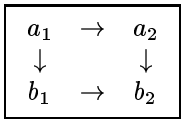
where a and c are dependent but b and c independent. The term $a + b$ is partially terminated for c. Thus B may start with c; afterwards, however, the choice should be resolved and only b should be left. Otherwise the specified order between dependent actions a and c would not be met. Thus partial termination, like global termination in [7, 9], 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, 61, 37, 12, 50, 21, 76, 12, 67] 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.

Action refinement is an operator used in the hierarchical design of systems. Therein a system is first specified on a rather abstract level and afterwards the entities of this level are given in more detail. The refinement function $r\colon Act \to \mathbf{L}$ describes the implementation of abstract actions by more complex processes. In contrast to standard action refinement, in our setting, the inheritance of abstract orderings to 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.*



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 *D-consistency* of refinement functions, which we will make precise in the next section.

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 [57] (where sequential composition is trace concatenation); in Janssen, Poel and Zwiers [80, 49, 32] both weak sequential composition (called layer composition) and dependency-based refinement are defined; and in Gaifman [34] 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 [28, 29]: Concatenation of two *infinite* traces is only possible when the alphabet of the first trace is known.

3 Operational semantics

In this section, we develop a structural operational semantics (SOS) in the style of Plotkin [65] 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 $\mathbf{L}$.

3.1 The finite, flat fragment

One of the main differences between our language and standard process algebraic languages lies in the interpretation of sequential composition. Already for strong sequential composition there are several possibilities of formalising this intuitively clear notion. They can be distinguished by the way they treat the formalisation of successful and unsuccessful termination. In a sequential composition of B_1 and B_2, B_2 may only start when B_1 has successfully terminated. Several different alternatives for rules formalising sequential composition exist (see for instance [6]). The version which matches our solution most closely is the following:

$$\frac{x \stackrel{a}{\Rightarrow} x'}{x;y \stackrel{a}{\Rightarrow} x';y} \qquad \frac{x \stackrel{\surd}{\Rightarrow} x' \quad y \stackrel{a}{\Rightarrow} y'}{x;y \stackrel{a}{\Rightarrow} y} \tag{1}$$

where $x \stackrel{\surd}{\Rightarrow} x'$ denotes that x can terminate successfully and thereby becomes x' (and ; denotes ordinary sequential composition). Besides these two rules for sequential composition, there are some rules needed to determine successful termination, i.e., to derive the relation $\stackrel{\surd}{\Rightarrow}$. Here we follow ideas of Baeten and van Glabbeek [7]: choices can be resolved by termination of a single operand.

$$\frac{x \stackrel{\surd}{\Rightarrow} x'}{x + y \stackrel{\surd}{\Rightarrow} x'} \tag{2}$$

(plus a symmetric rule for termination of the second operand). A different option is chosen by Aceto and Hennessy [3]: there, a choice only terminates if both operands do. However, the above variant (2) comes already quite close to what we need in order to correctly model the interplay between choice and weak sequential composition. In our setting, we just have to adapt the rules to our special notion of

termination. Instead of one special label $\surd$ for transitions, we use labels $\surd_a$ for every $a \in Act$; $B \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} B'$ states that B may terminate for a, after which it behaves as B'. We use $\surd_{Act} = \{\surd_a \mid a \in Act\}$ to denote the set of all such labels. The rule replacing the right hand side of (1) (the left hand rule remains the same) becomes:

$$\frac{x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad y \stackrel{a}{\Rightarrow} y'}{x \cdot y \stackrel{a}{\Rightarrow} x' \cdot y'} \tag{3}$$

The process x' is not always an empty process, thus it may — in contrast to strong sequential composition — not be discarded. As for termination, following (2), in our setting partial termination may resolve choices.

$$\frac{x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow}}{x + y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x'} \tag{4}$$

(plus a symmetric rule). In case both operands terminate w.r.t. an action, the choice is not resolved:

$$\frac{x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} y'}{x + y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' + y'} \tag{5}$$

Note the *negative* premise in (4). Negative premises are known to be potentially troublesome (see Groote [41] and Van Glabbeek [36]); indeed, when treating recursion (see below), we will have to take precautions.

The semantics of deadlock and action constants, as well as parallel composition, are as expected in the light of the discussion above. Let us use $\mathbf{L}_{ff}$ to denote the flat, finite fragment of $\mathbf{L}$ discussed so far, i.e., without refinement and recursion. Table 2 gives the SOS-rules for $\mathbf{L}_{ff}$.

deadlock	$\dfrac{a \; I \; A}{\mathbf{0}_A \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} \mathbf{0}_A}$
action	$\dfrac{}{a \stackrel{a}{\Rightarrow} \mathbf{1}} \qquad \dfrac{a \; I \; b}{b \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} b}$
choice	$\dfrac{x \stackrel{a}{\Rightarrow} x'}{x + y \stackrel{a}{\Rightarrow} x'} \qquad \dfrac{y \stackrel{a}{\Rightarrow} x'}{x + y \stackrel{a}{\Rightarrow} x'}$ $\dfrac{x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow}}{x + y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x'} \qquad \dfrac{y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow}}{x + y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x'} \qquad \dfrac{x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} y'}{x + y \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' + y'}$
sequential composition	$\dfrac{x \stackrel{a}{\Rightarrow} x'}{x \cdot y \stackrel{a}{\Rightarrow} x' \cdot y} \qquad \dfrac{x \stackrel{\surd_a}{\Rightarrow\!\!\!\!\!\Rightarrow} x' \quad y \stackrel{\alpha}{\Rightarrow} y' \quad \alpha \in \{a, \surd_a\}}{x \cdot y \stackrel{a}{\Rightarrow} x' \cdot y'}$
parallel composition	$\dfrac{x \stackrel{a}{\Rightarrow} x' \quad a \notin A}{x \parallel_A y \stackrel{a}{\Rightarrow} x' \parallel_A y} \qquad \dfrac{y \stackrel{a}{\Rightarrow} y' \quad a \notin A}{x \parallel_A y \stackrel{a}{\Rightarrow} x \parallel_A y'}$ $\dfrac{x \stackrel{\alpha}{\Rightarrow} x' \quad y \stackrel{\alpha}{\Rightarrow} y' \quad \alpha \in A \cup \surd_{Act}}{x \parallel_A y \stackrel{\alpha}{\Rightarrow} x' \parallel_A y'}$

Table 2: Operational semantics of $\mathbf{L}_{ff}$

The following proposition expresses some relations between the alphabet and the operational semantics of flat, finite terms.

Proposition 3.1 *Let $B \in \mathbf{L}_{ff}$.*

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

Let us point out another aspect of partial termination. 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 3.2 *Assume $a\ I\ b$ and let $B = a+b$. It follows that $B \stackrel{\surd_a}{\Longrightarrow} b$ and $B \stackrel{\surd_b}{\Longrightarrow} a$, hence B can terminate for* either *a or b. However, $B \stackrel{\surd_a}{\Longrightarrow}\stackrel{\surd_b}{\Longrightarrow}$ does* not *hold, hence B cannot terminate for* both *a and b.*

On the other hand, if a term can terminate for a sequence of actions in succession, e.g., $B \stackrel{\surd_a}{\Longrightarrow}\stackrel{\surd_b}{\Longrightarrow} B'$, then it can be shown that the order of termination is irrelevant, i.e., any order is possible and the target state (after termination) does not depend on the order; e.g., it follows that also $B \stackrel{\surd_b}{\Longrightarrow}\stackrel{\surd_a}{\Longrightarrow} B'$. This is formally stated in the following proposition, together with the property that termination is deterministic.

Proposition 3.3 *Let $B \in \mathbf{L}_{ff}$.*

1. *If $B \stackrel{\surd_a}{\Longrightarrow} B'$ and $B \stackrel{\surd_a}{\Longrightarrow} B''$, then $B' = B''$.*
2. *If $B \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_n}}{\Longrightarrow} B'$ then also $B \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_n}}{\Longrightarrow} B'$, where $b_1 \cdots b_n$ is an arbitrary permutation of the actions $a_1 \cdots a_n$.*

From Clause 2, it follows that we can write $B \stackrel{\surd_A}{\Longrightarrow} B'$ if $A = \{a_1, \ldots, a_n\}$ and $B \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_n}}{\Longrightarrow} B'$.

Proof of Proposition 3.3. Clause 1 is immediate from the fact that for any term, there is at most one applicable operational rule for termination. Clause 2 is shown by induction on the structure of B. We show only the most interesting case, $B = B_1 + B_2$. There are three sub-cases.

- $B_1 \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_n}}{\Longrightarrow} B'$ and $B_2 \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_{j-1}}}{\Longrightarrow}\stackrel{\surd_{a_j}}{\longrightarrow}$ for some $1 \leq i \leq n$. By the induction hypothesis, $B_1 \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_{j-1}}}{\Longrightarrow} B_1' \stackrel{\surd_{b_j}}{\longrightarrow} B_1'' \stackrel{\surd_{b_{j+1}}}{\Longrightarrow} \cdots \stackrel{\surd_{b_n}}{\Longrightarrow} B'$ and $B_2 \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_{j-1}}}{\Longrightarrow} B_2' \stackrel{\surd_{b_j}}{\longrightarrow}$ for some $1 \leq j \leq n$; hence

$$B \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_{j-1}}}{\Longrightarrow} B_1' + B_2' \stackrel{\surd_{b_j}}{\longrightarrow} B_1'' \stackrel{\surd_{b_{j+1}}}{\Longrightarrow} \cdots \stackrel{\surd_{b_n}}{\Longrightarrow} B'$$

- $B_1 \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_{i-1}}}{\Longrightarrow}\stackrel{\surd_{a_i}}{\longrightarrow}$ and $B_2 \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_n}}{\Longrightarrow} B'$ for some $1 \leq i \leq n$. Symmetrical to the previous case.
- $B_i \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_n}}{\Longrightarrow} B_i'$ for $i = 1, 2$ such that $B' = B_1' + B_2'$. By the induction hypothesis, $B_i \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_n}}{\Longrightarrow} B_i'$ for $i = 1, 2$, and hence $B \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_n}}{\Longrightarrow} B'$. □

3.2 Action refinement

In contrast to standard action refinement, in our setting the inheritance of abstract orderings to concrete actions of the refinement is driven by the dependencies between the latter. Dependency-based action refinement allows the overlap (concurrent execution) of *independent* parts of sequential abstract actions, as shown before in Example 2.1. However, as announced before, the allowed overlap is subject to some limitations due to the intuition that the abstract and concrete view of an activity should still describe the same entity:

- If an abstract action is dependent on another action, this dependency should still exist on the concrete level: in particular, the refinement of the first should not be (partially) terminated for the second. That is, $a\ D\ b$ should imply $r(a) \not\Leftrightarrow$.
- 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 $r(a) \not\Leftrightarrow r(a)$.

A refinement function is called *D-consistent* if it satisfies both of these criteria. A similar property was defined in [47]. Note that it follows that for all $a \in Act$, $[a]_D = [\mathcal{A}(r(a))]_D$ and $r(a) \not\Leftrightarrow B$ implies $B = r(a)$; indeed, these two properties form a sufficient condition for D-consistency.

Example 3.4

- *If $a\ D\ b$, $a_1\ D\ b$ and $a_2\ I\ b$ such that $r{:}\, a \mapsto a_1 + a_2$, then $r(a) \not\Leftrightarrow$; 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 $r{:}\, a \mapsto a_1 + a_2$, then $r(a) \_ bB$ 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 $a, b \in Act$.*

For the structural operational treatment of refinement, it is clear that we need at least a rule of the following form:

$$\frac{x \overset{a}{\Leftrightarrow} x' \quad r(a) \overset{b}{\Leftrightarrow} y}{x[r] \overset{b}{\Leftrightarrow} B}$$

The interesting part is the choice of B. It should capture the following points:

- The refinement of a should be able to proceed, i.e., B should contain y;
- B should resolve all the choices in x in the same way as x';
- B should be able to start the refinements of all a-independent initial actions of x';
- B should still contain the function r.

A rule which captures all these aspects is obtained by setting $B = y \cdot x'[r]$; i.e.,

$$\frac{x \overset{a}{\Leftrightarrow} x' \quad r(a) \overset{b}{\Leftrightarrow} y}{x[r] \overset{b}{\Leftrightarrow} y \cdot x'[r]} \tag{6}$$

To come back to Example 2.1, the overlapping execution of the refinement of a and b is thus derivable:

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

The rule for partial termination 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.

$$\frac{x \not\overset{a}{\Leftrightarrow} x'}{x[r] \not\overset{a}{\Leftrightarrow} x'[r]} \tag{7}$$

The operational semantics of $\mathbf{L}_{fin}$, the finite fragment of $\mathbf{L}$ (i.e., without recursion) is therefore determined by Table 2 augmented by Equations (6) and (7). Propositions 3.1 and 3.3 continue to hold in $\mathbf{L}_{fin}$.

It is worth noting that the operational refinement rules are simpler by far than the ones obtained in other approaches. For instance, we no 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 [26, 69, 18] for modelling standard action refinement.

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 $B \stackrel{a}{\Rightarrow}\stackrel{b}{\Rightarrow}$ for $a\ I\ b$, we also expect $B \stackrel{b}{\Rightarrow}\stackrel{a}{\Rightarrow}$ (see [57]). As the following example shows, this property can be destroyed by certain refinement functions.

Example 3.5 *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:*

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

However, $B[r] \stackrel{b_1}{\Rightarrow}$; 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. We will call r *strongly D-consistent* if it is D-consistent and in addition

$$a\ D\ b \wedge r(a) \stackrel{a'}{\Rightarrow} \implies a'\ D\ b\ .$$

The necessity of strong D-consistency follows from the fact that an example along the lines of Example 3.5 can always be constructed for a refinement function that is not strongly D-consistent. The sufficiency is formulated in the following proposition, which is a consequence of a stronger property stated below (Proposition 3.15).

Proposition 3.6 *If all refinement functions in $B \in \mathbf{L}$ are strongly D-consistent, then $B \stackrel{a}{\Rightarrow}\stackrel{b}{\Rightarrow}$ implies $B \stackrel{b}{\Rightarrow}\stackrel{a}{\Rightarrow}$ whenever $a\ I\ b$.*

In the remainder of this paper, we will always implicitly assume refinement functions to be strongly D-consistent, unless stated otherwise.

3.3 Recursion

The operator $rec\,X.\ \_$ looks quite standard and at first sight one does not expect any surprises from it. Unfortunately, the situation is complicated by the negative premise in the termination rule of the choice operator. The standard operational rule for recursion is the following (with $\alpha \in Act \cup \checkmark_{Act}$):

$$\frac{y\langle rec\,X.\,y/X\rangle \stackrel{\alpha}{\Rightarrow} y'}{rec\,X.\,y \stackrel{\alpha}{\Rightarrow} y'} \tag{8}$$

However, for the derivation of the partial terminations, this is not satisfactory, as can be seen by the following example:

Example 3.7 *Assume $\mathcal{A}_X = \{a, b\}$ and consider $B = rec\,X.\,a \cdot X + b$.*

1. *First consider $a\ I\ c$ and $b\ I\ c$. Let us consider the partial c-termination for B. Assume that B is not terminated for action c. With this assumption, we can derive $a \cdot (rec\, X.\, a \cdot X + b) + b \not\xrightarrow{c} b$ and therefore $rec\, X.\, a \cdot X + b \not\xrightarrow{c} b$, which leads to a contradiction. It follows that $B \xrightarrow{c} B'$ for some B'. But unfortunately we cannot derive a transition of this form. This is due to the fact that the partial termination rules for weak sequential composition test* both *arguments for termination; that is, to derive that $a \cdot B$ is terminated for c, we have to test whether $a \not\xrightarrow{c}$* and *$B \not\xrightarrow{c}$. This leads to 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 $B \xrightarrow{c} B'$; it is certainly not equal to B, since the action b should not be possible anymore. Again, any attempt to derive a transition of the form $a \cdot B \not\xrightarrow{c}$ leads to an infinite unfolding of the recursive term. Since infinite proofs are not allowable, it follows that $a \cdot B \not\xrightarrow{c}$.*

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

The problem is essentially due to the negative premise in the termination rule of choice (see Table 2), in combination with 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. 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 cyclical testing cannot occur. A standard way to achieve this is to replace (8) by

$$\frac{y \xrightarrow{a} y'}{rec\, X.\, y \xrightarrow{a} y' \langle rec\, X.\, y / X \rangle} \tag{9}$$

(see for instance [5]). This new rule 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 [41], a stratification trivially exists.)

Naturally, however, this new rule potentially gives rise to a different transition relation; in order to avoid this, one simultaneously restricts recursion to *guarded* terms. In general, a term may be called guarded if all variables occur in so-called *sleeping positions* (see Vaandrager [74]), where a sleeping position of a given operator is one which is not tested by the rules of the operational semantics. For instance, even though X would appear to be guarded in the sub-term $a \cdot X$ of B in Example 3.7, actually it is not, since the second position of a weak sequential composition is not a sleeping position (it is tested by the termination rule).

In fact, no operator of our language has a sleeping position, so the usual notion of guardedness is not applicable. Instead, we may try to isolate a set of "weakly guarded" terms $C \cdot X$ of which we can predict the termination properties without actually having to test X. For instance, an obvious way to do this is to require that $C \not\xrightarrow{a}$ for all $a \in Act$; then certainly $C \cdot X \not\xrightarrow{a}$ irregardless of X. However, we find this much too restrictive; for instance, Proposition 3.1.3 tells us that (if $C \in \mathbf{L}_{fin}$ then) $C \not\xrightarrow{a}$ whenever $a\ I\ \mathcal{A}(C)$, so the above condition can only be met if $[\mathcal{A}(C)]_D = Act$; and even that may not be sufficient.

A second attempt is to take Proposition 3.1 as inspiration. Intuitively, although we have proved this proposition only for the finite part of $\mathbf{L}$, it should hold for

the entire language. In particular, $rec\,X.\,B \stackrel{\surd}{\Longrightarrow} rec\,X.\,B$ should be true whenever $a\ I\ \mathcal{A}(rec\,X.\,B)$, where $\mathcal{A}(rec\,X.\,B) = \mathcal{A}_X$. But then we might as well introduce an operational rule for this purpose:

$$\frac{a\ I\ \mathcal{A}_X}{rec\,X.\,B \stackrel{\surd}{\Longrightarrow} rec\,X.\,B} \tag{10}$$

We then call $C \cdot X$ weakly guarded if $C \stackrel{\surd}{\Longrightarrow}$ implies $a\ I\ \mathcal{A}_X$. Since we only consider a term of the form $C \cdot X$ in the body B of a recursive term $rec\,X.\,B$, it follows that we never actually have to test $C \cdot X$ for partial a-termination (it is already taken care of on the level of $rec\,X.\,B \stackrel{\surd}{\Longrightarrow} rec\,X.\,B$). This gives rise to the following formal definition of weak guardedness:

Definition 3.8 *Let $B \in \mathbf{L}$.*

- *X is called weakly 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 $B_1\langle \mathbf{0}_{\mathcal{A}_X}/X\rangle \stackrel{\surd}{\Longrightarrow}$ implies $a\ I\ \mathcal{A}(B_2)$.*
- *B is called weakly guarded if for all subterms $rec\,X.\,C$ of B, X is weakly guarded in C.*

For instance, X is weakly guarded in $(a \cdot X) \cdot X$ iff $[a]_I = [\mathcal{A}_X]_I$. The rule (9) is still needed for ordinary transitions ($a \in Act$) and non-global termination actions ($\surd_a$ where $a \in [\mathcal{A}_X]_D$). For instance, in Example 3.7.3, we can only derive $B \stackrel{\surd}{\Longrightarrow} b$ by applying this rule. On the other hand, allowing (9) to derive $\stackrel{\surd}{\Longrightarrow}$-transitions for $a\ I\ \mathcal{A}_X$ would give rise to erroneous transitions since $X \stackrel{\surd}{\Longrightarrow}$ for arbitrary a (there simply is no rule to derive transitions from a variable); for instance if $\mathcal{A}_X = \{a, b\}$ and $c\ I\ \{a, b\}$ then the following would be a correct derivation:

$$\dfrac{\dfrac{\dfrac{\overline{X \stackrel{\surd}{\Longrightarrow}}}{a \cdot X \stackrel{\surd}{\Longrightarrow}} \qquad \dfrac{a\ I\ b}{b \stackrel{\surd}{\Longrightarrow} b}}{a \cdot X + b \stackrel{\surd}{\Longrightarrow} b}}{rec\,X.\,a \cdot X + b \stackrel{\surd}{\Longrightarrow} b}$$

which contrasts with the intended $rec\,X.\,a \cdot X + b \stackrel{\surd}{\Longrightarrow} rec\,X.\,a \cdot X + b$. If we try to solve this by adding a rule to derive $X \stackrel{\surd}{\Longrightarrow} X$ for all $a\ I\ \mathcal{A}_X$ then we obtain non-finite-state models even for quite harmless processes; e.g., if $\mathcal{A}_X = \{a\}$ and $a\ I\ b$ then

$$rec\,X.\,a \cdot X \stackrel{\surd}{\Longrightarrow} a \cdot\ rec\,X.\,a \cdot X \stackrel{\surd}{\Longrightarrow} a \cdot a \cdot\ rec\,X.\,a \cdot X \stackrel{\surd}{\Longrightarrow} \cdots$$

To avoid these difficulties, we restrict the use of (10) to those cases where α is not of the form $\surd_a$ for some $a\ I\ \mathcal{A}_X$. Table 3 summarises the additional rules for refinement and recursion. We can now extend Propositions 3.1 and 3.3 to the full language.

Proposition 3.9 *Let $B \in \mathbf{L}$.*

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

refinement	$\dfrac{x \stackrel{a}{\Longrightarrow} x' \quad r(a) \stackrel{a}{\Longrightarrow} y}{x[r] \stackrel{a}{\Longrightarrow} y \cdot x'[r]}$	$\dfrac{x \stackrel{\surd_a}{\Longrightarrow} x'}{x[r] \stackrel{\surd_a}{\Longrightarrow} x'[r]}$
recursion	$\dfrac{y \stackrel{\alpha}{\Longrightarrow} y' \quad \alpha \notin \surd_{[\mathcal{A}_X]_I}}{rec\ X.\ y \stackrel{\alpha}{\Longrightarrow} y'\langle rec\ X.\ y/X\rangle}$	$\dfrac{a\ I\ \mathcal{A}_X}{rec\ X.\ y \stackrel{\surd_a}{\Longrightarrow} rec\ X.\ y}$

Table 3: Transition and termination rules for refinement and recursion

Proposition 3.10 *Let $B \in \mathbf{L}$.*

1. *If $B \stackrel{\surd_a}{\Longrightarrow} B'$ and $B \stackrel{\surd_a}{\Longrightarrow} B''$, then $B' = B''$.*
2. *If $B \stackrel{\surd_{a_1}}{\Longrightarrow} \cdots \stackrel{\surd_{a_n}}{\Longrightarrow} B'$ then also $B \stackrel{\surd_{b_1}}{\Longrightarrow} \cdots \stackrel{\surd_{b_n}}{\Longrightarrow} B'$, where $b_1 \cdots b_n$ is an arbitrary permutation of the actions $a_1 \cdots a_n$.*

In the remainder of this paper, we implicitly assume all terms to be weakly guarded. As an added bonus, weak guardedness of X in B guarantees that $X = B$ has a unique solution (up to bisimulation); see Theorem 3.19 below.

3.4 Transition systems and bisimulation

The operational semantics in Tables 2 and 3 naturally gives rise to a transition relation over $\mathbf{L}$. Interpreting the terms of $\mathbf{L}$ as system states, we obtain a labelled transition system where the labels are taken from $Act \cup \surd_{Act}$. The general definition is as follows:

Definition 3.11 *A* labelled transition system *(lts) is a tuple $\langle S, \Leftrightarrow, s_0\rangle$ such that*

- *S is a set of* states*,*
- *$\Leftrightarrow \subseteq S \times (Act \cup \surd_{Act}) \times S$ is a* transition relation *and*
- *$s_0 \in S$ is the* initial state*.*

The class of all labelled transition systems will be denoted **LTS**. In particular, the operational semantics gives rise to an lts.

Definition 3.12 *The operational semantics for a term $B \in b$ is the transition system $lts(B) = \langle \mathbf{L}, \Leftrightarrow, B\rangle$, where $\Leftrightarrow$ is the smallest set of transitions agreeing with the rules in Tables 2 and 3.*

Note that, 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 [30]); e.g., for all $a\ I\ b$

1. if $s \stackrel{a}{\Longrightarrow} s' \stackrel{b}{\Longrightarrow} s''$, then $s \stackrel{b}{\Longrightarrow} s'_0 \stackrel{a}{\Longrightarrow} s''$ for some s'_0;
2. if $s \stackrel{a}{\Longrightarrow} s'$ and $s \stackrel{b}{\Longrightarrow} s''$, then $s' \stackrel{b}{\Longrightarrow} s''_0$ and $s'' \stackrel{a}{\Longrightarrow} s''_0$ for some s''_0.

The transition systems generated by our operational semantics satisfy neither of the above properties; the first, however, can be recaptured as soon as we interpret the transition system modulo bisimulation.

Example 3.13

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

$$\begin{array}{lllll} B[r] & \stackrel{a_1}{\Longrightarrow} & a_2 \cdot (\mathbf{1} \,|||\, b)[r] & \stackrel{b_1}{\Longrightarrow} & a_1 \cdot b_2 \cdot (\mathbf{1} \,|||\, \mathbf{1})[r] \\ B[r] & \stackrel{b_1}{\Longrightarrow} & b_2 \cdot (a \,|||\, \mathbf{1})[r] & \stackrel{a_1}{\Longrightarrow} & b_2 \cdot a_2 \cdot (\mathbf{1} \,|||\, \mathbf{1})[r] \end{array}$$

   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 valid term such that* $B \stackrel{a}{\Longrightarrow} \mathbf{1}$ *and* $B \stackrel{b}{\Longrightarrow} \mathbf{1}$, *but* $\mathbf{1} \stackrel{a}{\not\Longrightarrow}$ *and* $\mathbf{1} \stackrel{b}{\not\Longrightarrow}$.

Transition systems as a semantics for process terms are usually too informative: they distinguish processes that one would normally consider as being equal, such as, for instance, $rec\, X.a; X$ and $rec\, X.a; a; X$. 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* [63, 58].

Definition 3.14 *Let* $T_i = \langle S_i, \Longrightarrow, s_{0i} \rangle$, $i = 1, 2$, *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* $(s_{01}, s_{02}) \in \rho$ *and whenever* (s_1, s_2) *is in* ρ, *then for all* $\alpha \in Act \cup \surd_{Act}$

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

As usual, two terms are bisimilar if their corresponding transition systems are. We can now formulate a weaker form of the first diamond closure property: a transition system is called *partially commutative up to bisimulation* if for all $s \stackrel{a}{\Longrightarrow} s' \stackrel{b}{\Longrightarrow} s''$ with $a \; I \; b$, there are s_0' and s_0'' such that $s \stackrel{b}{\Longrightarrow} s_0' \stackrel{a}{\Longrightarrow} s_0''$ and $s'' \sim s_0''$. Note that partial commutativity up to bisimulation is strictly stronger than partial trace commutativity in the sense of Example 3.5, and hence the following property implies Proposition 3.6 above.

Proposition 3.15 *For all* $B \in \mathbf{L}$, $lts(B)$ *is partially commutative up to bisimulation.*

This follows directly from the denotational characterisation in the next section (Theorem 4.22).

Recursion revisited. Just as in the standard case, (weak) guardedness implies that recursive equations have a unique solution up to bisimilarity, given by the corresponding recursive term. That is, for all B in which X is weakly guarded, $rec\, X.\, B$ solves the equation $X = B$, and all other solutions are bisimilar to it. The following is the key lemma from which uniqueness follows.

Lemma 3.16 *Assume* $B \in \mathbf{L}$ *with* $fv(B) \subseteq \{X\}$ *such that* X *is weakly guarded in* B, *and let* $C_1, C_2 \in \mathbf{L}$ *be arbitrary such that* $\mathcal{A}(C_1) \cup \mathcal{A}(C_2) \subseteq \mathcal{A}_X$. *If* $B\langle C_1/X \rangle \stackrel{a}{\Longrightarrow} B'$, *then* $B' = B''\langle C_1/X \rangle$ *such that* $B\langle C_2/X \rangle \stackrel{a}{\Longrightarrow} B''\langle C_2/X \rangle$.

Proof. By induction on the structure of B. The interesting case is if $B = B_1 \cdot B_2$. Weak guardedness of B implies that X is weakly guarded in B_1, and either X is weakly guarded in B_2 or $B_1\langle \mathbf{0}_{\mathcal{A}_X}/X \rangle \stackrel{\surd}{\Longrightarrow}$ implies $a \; I \; \mathcal{A}(B_2)$. Consider the possible transitions of B.

- $\alpha = a$, $B_1\langle C_1/X\rangle \stackrel{a}{\Rightarrow} B_1'$ and $B' = B_1' \cdot B_2\langle C_1/X\rangle$. By the induction hypothesis, it follows that $B_1' = B_1''\langle C_1/X\rangle$ such that $B_1\langle C_2/X\rangle \stackrel{a}{\Rightarrow} B_1'\langle C_2/X\rangle$; hence $B'' = B_1'' \cdot B_2$ fulfils the proof obligations.

- $\alpha = a$, $B_1\langle C_1/X\rangle \stackrel{\checkmark a}{\Rightarrow} B_1'$ and $B_2\langle C_1/X\rangle \stackrel{a}{\Rightarrow} B_2'$ such that $B' = B_1' \cdot B_2'$. It follows that X is weakly guarded in B_2; hence (by the induction hypothesis) $B_i' = B_i''\langle C_1/X\rangle$ for $i = 1, 2$ such that $B_1\langle C_2/X\rangle \stackrel{\checkmark a}{\Rightarrow} B_1'\langle C_2/X\rangle$ and $B_2\langle C_2/X\rangle \stackrel{a}{\Rightarrow} B_2'\langle C_2/X\rangle$. It follows that $B'' = B_1'' \cdot B_2''$ fulfils the proof obligations.

- $\alpha = \checkmark_a$ and $B_i\langle C_1/X\rangle \stackrel{\checkmark a}{\Rightarrow} B_i'$ for $i = 1, 2$. If X is weakly guarded in B_2, then the proof proceeds as in the previous case. Otherwise, by the induction hypothesis it follows that $B_1' = B_1''\langle C_1/X\rangle$ such that $B_1\langle \mathbf{0}_{\mathcal{A}_X}/X\rangle \stackrel{\checkmark a}{\Rightarrow} B_1''\langle \mathbf{0}_{\mathcal{A}_X}/X\rangle$ and $B_1\langle C_2/X\rangle \stackrel{\checkmark a}{\Rightarrow} B_1''\langle C_2/X\rangle$. Due to guardedness, it follows that $a\ I\ \mathcal{A}(B_2)$; hence according to Proposition 3.9.3, $B_2' = B_2\langle C_1/X\rangle$. We may conclude that $B'' = B_1'' \cdot B_2$ fulfils the proof obligations. □

In fact, the above lemma still holds if C_1 and C_2 are not terms of $\mathbf{L}$, but rather syntactical representations of some transition system in which for all reachable states s and all $a\ I\ \mathcal{A}(C_i)$, $s \stackrel{\checkmark a}{\Rightarrow} s'$ iff $s' = s$.

Congruence of bisimilarity In the last years, there has been extensive work on *formats* for SOS-rules; see for instance [24, 13, 42, 8, 75, 33]. Specific formats lead to certain properties of the resulting operational semantics. One important question investigated is whether certain equivalence relations, in particular bisimilarity, are congruences for the operators of the language.[1] We use here the GSOS-format of [13] to show that bisimulation is a congruence (that bisimilar processes can be considered equal in all contexts) for the language $\mathbf{L}$.

Definition 3.17 *Suppose that Σ is a signature. A* GSOS rule *ρ over Σ is a rule of the form*

$$\frac{\{X_i \stackrel{a_{ij}}{\not\Rightarrow} Y_{ij} \mid 1 \le i \le n, j \in M_i\} \quad \{X_i \stackrel{b_{ik}}{\not\Rightarrow} \mid 1 \le i \le n, k \in N_i\}}{op(X_1, \ldots, X_n) \stackrel{c}{\Rightarrow} C[\vec{X}, \vec{Y}]}$$

where all the variables are distinct, op is an operation symbol from Σ with arity n, and $C[\vec{X}, \vec{Y}]$ 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 (9), which contains a "substitution harness" (in the terminology of [33]) 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 $1 \le i \le n$. The corresponding operational rules then become:

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

[1] Another aspect, which is particularly interesting in the presence of rules with negative premises [41, 36], is the question whether the semantics generates a *unique* transition relation. However, this has already been answered by the fact that in all our rules, the (source terms of the) premises are strictly simpler than the (source term of the) conclusion.

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

We now have the following:

Theorem 3.18 *Bisimulation is a congruence for all the operators of weakly guarded* **L** *including recursion.*

To prove congruence of recursion, the *up-to* technique used in [58] can be applied, since our rules contain no look-ahead (see [70]).

In particular, the result that bisimulation is a congruence for (dependency-based) refinement is remarkable, since the fact that it does *not* hold for standard refinement (see [19]) has been the starting point of almost all papers on action refinement cited before.

Unique fixpoints. Using the congruence result, we can now also establish that, at least for weakly guarded terms, 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 3.19 *If $B \in \mathbf{L}$ with $fv(B) \subseteq \{X\}$, and X is weakly guarded in B, then $rec\, X.\, B$ is the unique solution of $X = B$ in* **LTS** *modulo $\sim$.*

Proof. The fact that $rec\, X.\, B$ solves $X = B$ modulo $\sim$ is straightforward to establish. The proof of uniqueness is along the lines set out in Milner [58]. Assume $C \sim B\langle C/X\rangle$ for some C that is a syntactic representation of a transition system, of the aforementioned kind (i.e., to which Lemma 3.16 applies). Consider the relation

$$\rho = \{(D_1, D_2) \mid \exists D \in \mathbf{L}\colon fv(D) \subseteq \{X\}, D_1 \sim D\langle rec\, X.\, B/X\rangle, D_2 \sim D\langle C/X\rangle\} \ .$$

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

- Assume $D_1 \stackrel{a}{\Longrightarrow} D_1'$. Due to $rec\, X.\, B \sim B\langle rec\, X.\, B/X\rangle$, the congruence of $\sim$ (Theorem 3.18) and the properties of syntactic substitution, it follows that $D_1 \sim D\langle B/X\rangle\langle rec\, X B/X\rangle$. It follows that $D\langle B/X\rangle\langle rec\, X.\, B/X\rangle \stackrel{a}{\Longrightarrow} D' \sim D_1'$. Since X is weakly guarded in B it is weakly guarded in $D\langle B/X\rangle$; hence (by Lemma 3.16) $D' = D''\langle rec\, X.\, B/X\rangle$ such that $D\langle B\langle C/X\rangle/X\rangle = D\langle B/X\rangle\langle C/X\rangle \stackrel{a}{\Longrightarrow} D''\langle C/X\rangle$. Since $B\langle C/X\rangle \sim C$, it follows (Theorem 3.18) that $D_2 \sim D\langle C/X\rangle \sim D\langle B\langle C/X\rangle/X\rangle$; hence $D_2 \stackrel{a}{\Longrightarrow} D_2' \sim D''\langle C/X\rangle$. We may conclude $(D_1', D_2') \in \rho$.
- The reverse direction is analogous. □

4 Denotational semantics

In this section, we develop a denotational semantics for the language **L**. 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 operational semantics, on the other hand, will be a very rich *event-based model*; and the constructions defined for the operators of **L** are variations on known constructions on, for instance, prime event structures [52, 25], stable event structures [79] and families of posets [66, 69]. 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 be compositional.

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 $Evt_* \times Evt_* \subseteq Evt$.

Definition 4.1

- *A* system run *is a tuple* $p = \langle E, \ell, \leq, T \rangle$ *where*
  - $E \subseteq Evt$ *is a finite set of events.*
  - $\ell : E \to Act$ *is a labelling function, mapping each event in the run to the action of which it models an occurrence. We extend the dependency relation* $D \subseteq Act \times Act$ *to* E *by writing* $d\ D\ e$ *iff* $\ell(d)\ D\ \ell(e)$.
  - $\leq \subseteq E \times E$ *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 off 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, ℓ_p *etc. to denote the components of a system run* p; *i.e.,* $p = \langle E_p, \ell_p, \leq_p, T_p \rangle$. *The class of all system runs is denoted* $\mathbf{P}$.

- *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., less 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.
  - $\ell_p = \ell_q \restriction E_p$; *i.e., the events of* p *are labelled the same as in* q.
  - $T_p \subseteq T_q \setminus [\ell(E_q \setminus E_p)]_D$; *i.e., an action is terminated in* p *only if it is terminated in* q *and independent of all actions that occur in between* p *and* q.

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 [67] for a discussion); we will see below that this richness is actually necessary to model some of the features of **L**. 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. 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.

We use the following additional notations for system runs:

$$\begin{aligned} \varepsilon_T &= \langle \emptyset, \emptyset, \emptyset, T \rangle \\ p \restriction E &= \langle E_p \cap E, \ell \restriction E, \leq_p \cap (E \times E), T_p \rangle \quad \text{for } E \subseteq Evt \end{aligned}$$

$$
\begin{array}{lcll}
p \setminus E & = & p \restriction (E_p \setminus E) & \text{for } E \subseteq \mathit{Evt} \\
p \cup T & = & \langle E_p, \ell_p, \leq_p, T_p \cup T \rangle & \text{for } T \subseteq \mathit{Act} \\
p \cap T & = & \langle E_p, \ell_p, \leq_p, T_p \cap T \rangle & \text{for } T \subseteq \mathit{Act} \\
p \setminus T & = & \langle E_p, \ell_p, \leq_p, T_p \setminus T \rangle & \text{for } T \subseteq \mathit{Act}
\end{array}
$$

Furthermore, system runs can be depicted in the form $\boxed{F}T$, where F is a graphical representation of the first three components of the run, in the form of nodes ${}_e a$ (depicting an a-labelled event e) 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). We often take $\mathbb{N} \subseteq \mathit{Evt}$ in such figures; sometimes we omit the event identities altogether.

Example 4.2 *Assume $\mathit{Act} = \{a, b, c, d\}$ with a D b D c D d (and all other actions independent).*

- *System runs:* $\boxed{\begin{array}{c} {}_1a \to {}_3c \\ \nearrow \\ {}_2b \to {}_4d \end{array}}\{a, c\}$ *and* $\boxed{\begin{array}{c} b \to b \to a \\ d \end{array}}\{b, c\}$.
- *Prefixes:* $\boxed{\begin{array}{c} {}_1a \\ \searrow \\ {}_3c \to {}_2b \end{array}}\{a\} \preceq \boxed{\begin{array}{c} {}_1a \to {}_2b \\ \nearrow \\ {}_3c \to {}_4d \end{array}}\{a, c\}$ *and* $\boxed{\begin{array}{c} {}_1a \\ {}_3c \to {}_4d \end{array}}\emptyset \preceq \boxed{\begin{array}{c} {}_1a \to {}_2b \\ \nearrow \\ {}_3c \to {}_4d \end{array}}\{a, c\}$.
- *Generally, $p \preceq p \cup A$ and $\varepsilon_\emptyset \preceq p$ for all $p \in \mathbf{P}$ and $A \subseteq \mathit{Act}$.*

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 4.3 *A* system model *is a nonempty set of runs $\mathcal{P}$, such that*

- *$p \preceq q \in \mathcal{P}$ implies $p \in \mathcal{P}$; i.e., $\mathcal{P}$ is prefix closed.*
- *$\ell_p \restriction E_q = \ell_q \restriction E_p$ for all $p, q \in \mathcal{P}$; i.e., the labelling of all system runs is consistent.*

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 $\phi : E_\mathcal{P} \to E_\mathcal{Q}$ is an isomorphism between $\mathcal{P}$ and $\mathcal{Q}$ (denoted $\phi : \mathcal{P} \cong \mathcal{Q}$ or simply $\mathcal{P} \cong \mathcal{Q}$ if ϕ is irrelevant) if 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 $\mathbf{L}$.

Example 4.4 *The following graph represents a system model, where the arrows between system runs represent the prefix ordering (and we have omitted some prefixes):*

$$
\begin{array}{ccccccc}
 & & \varepsilon_{\{d\}} & \to & \boxed{a}\{d\} & \to & \boxed{a \to b}\{a, b, c, d\} \\
 & \nearrow & & \nearrow & & & \\
\varepsilon_\emptyset & \to & \boxed{a}\emptyset & & & & \\
 & \searrow & & \searrow & & & \\
 & & \boxed{c}\{c, d\} & \to & \boxed{\begin{array}{c} a \\ c \end{array}}\{c, d\} & &
\end{array}
$$

Note that $\boxed{a}\{d\} \not\preceq \boxed{\begin{array}{c} a \\ c \end{array}}\{c, d\}$.

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 **L**, and afterwards establish their formal properties, such as the fact that all constructions stay within **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 $\boxed{e^a}T$ for a given event $e \in \mathit{Evt}$ and arbitrary $T \subseteq \mathit{Act}$.

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 partial labelling functions $\ell_i\colon \mathit{Evt} \rightharpoonup \mathit{Act}$ for $i = 1, 2$:

$$\begin{aligned} \ell_1 \parallel_A \ell_2 = \ & \{((e, *), a) \mid a = \ell_1(e) \notin A\} \\ & \cup \{((*, e), a) \mid a = \ell_2(e) \notin A\} \\ & \cup \{((e_1, e_2), a) \mid a = \ell_1(e_1) = \ell_2(e_2) \in A\} \end{aligned}$$

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 4.5 *Assume* $\mathit{Act} = \{a, b, c\}$ *with* $a\ I\ \{b, c\}$. *The synchronisation of the system runs* $\boxed{{}_1a \to {}_2a \atop {}_3b}$ *and* $\boxed{{}_4a \to {}_5c \to {}_6a}$ *on the action* a *(omitting termination sets) yields one of the runs* $\boxed{{}_{14}a \atop {}_{3*}b \to {}_{*5}c \to {}_{26}a}$ *and* $\boxed{{}_{14}a \to {}_{*5}c \to {}_{26}a \atop {}_{3*}b}$ *(where* ij *denotes the pair* (i, j)*).*

For $i = 1, 2$ let $\pi_i\colon \mathit{Evt} \rightharpoonup \mathit{Evt}$ 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:

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

Note that the condition $\ell_q \subseteq (\ell_{p_1} \parallel_A \ell_{p_2})$ on the labelling function implies a corresponding condition on E_q, which, after all, can be retrieved as the domain of ℓ_q. Also 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 4.5.

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 $E_{\mathcal{P}_1} \cap E_{\mathcal{P}_2} = \emptyset$:

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

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. 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 $\mathcal{R}$. However, we have to take care that the images of w have the appropriate termination properties: especially, any event d' of p that is not a causal successor of d should be refined to a system run $w(d')$ that is terminated w.r.t. all actions of $w(d)$. (This is a similar requirement to the one used above to determine which runs are compatible for weak sequential composition.)

$$\begin{aligned}\mathcal{P}[\mathcal{R}] = \{q \mid\ & p \in \mathcal{P},\, \forall d \in E_p\colon w(d) \in \mathcal{R}(\ell_p(d)),\, w(d) \neq \varepsilon_T,\\ & \forall d \not\leq_p d'\colon \ell_{w(d)}(E_{w(d)}) \subseteq T_{w(d')},\\ & \ell_q = \{((d, e), a) \mid d \in E_p, e \in E_{w(d)}, a = \ell_{w(d)}(e)\},\\ & \leq_q = \{((d_1, e_1), (d_2, e_2)) \mid d_1 <_p d_2, e_1\ D\ e_2 \text{ or } d_1 = d_2, e_1 <_{w(d_1)} e_2\},\\ & T_q = T_p \cap \textstyle\bigcap_{d \in E_p} T_{w(d)}\}\end{aligned}$$

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).

Example 4.6 *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* $p = \boxed{\begin{array}{c} b \\ \nearrow \\ a \to c \end{array}} Act$ *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 $q = \boxed{\begin{array}{ccc} & \nearrow & b_1 \to b_2 \\ a_1 \to a_2 & & \\ & \searrow & c_1 \to c_2 \end{array}} Act$. *The prefix* $\boxed{\begin{array}{ccc} & \nearrow & b_1 \to b_2 \\ a_1 & & \\ & \searrow & c_1 \end{array}} \{b_1, b_2\}$ *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 is obtained as the limit of approximants. In our case, the approximants give rise to a $\subseteq$-chain of models $(\mathcal{P}_i)_{i \in \mathbb{N}}$, whose limit is given by $\bigcup_{i \in \mathbb{N}} \mathcal{P}_i$. We discuss the construction of the approximants and the semantic rationale behind the limit construction later in this section.

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

Proposition 4.7 $\mathbf{M}$ *is closed under the constructions defined above.*

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 $p_i' = (p_i \restriction E_i') \cap T_{q'}$; since E_i' is clearly $\leq_{p_i}$-left-closed (otherwise $E_{q'}$ would not be $\leq_q$-left-closed) and

$$T_{p_i'} = T_{q'} \subseteq T_q \setminus [\ell_q(E_q \setminus E_{q'})]_D \subseteq T_{p_i} \setminus [\ell_{p_i}(E_{p_i} \setminus E_i')]_D$$

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

$$\begin{aligned} p_1' &= (p_1 \restriction E_{q'}) \setminus [\ell_{p_1}(E_{p_1} \setminus E_{q'})]_D \\ p_2' &= (p_2 \restriction E_{q'}) \cap T_{q'} \ . \end{aligned}$$

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 $T_{q'} \cap [\ell_{p_2}(E_{p_2} \setminus E_{q'})]_D = \emptyset$, it follows that $p_i' \preceq p_i$ and hence $p_i' \in \mathcal{P}_i$ for $i = 1, 2$. For all $d \in E_{p_1} \setminus E_{q'}$ and $e \in E_{p_2'}$, it follows that $d \not\leq_q e$ and hence $d \mathrel{I} e$; hence $\ell_{p_2'}(E_{p_2'}) \subseteq T_{p_1'}$. 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

$$T_{q'} \subseteq T_q \setminus [\ell_q(E_q \setminus E_{q'})]_D = (T_{p_1} \setminus [\ell_{p_1}(E_{p_1} \setminus E_{q'})]_D) \cap (T_{p_2} \setminus [\ell_{p_2}(E_{p_2} \setminus E_{q'})]_D)$$

it follows that

$$T_{p_1'} \cap T_{p_2'} = (T_{p_1} \setminus [\ell_{p_1}(E_{p_1} \setminus E_{q'})]_D) \cap T_{p_2} \cap T_{q'} = T_{q'} \ .$$

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

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

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

Table 4: Denotational semantics for closed terms of $\mathbf{L}$

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

$$w'(d) = (w(d) \upharpoonright E'_d) \setminus [\ell_{w(d)}(E_{w(d)} \setminus E'_d)]_D \ .$$

Since $[\ell_p(E_p \setminus E')]_D \cap T_{q'} = [\ell_q(E_q \setminus E_{q'})]_D \cap T_{q'} = \emptyset$ (the first equality due to D-consistency) it follows that $p' \preceq p$ and hence $p' \in \mathcal{P}$; moreover, for all $d \in E_{p'}$ it follows by construction (since the $\leq_p$-left-closure of $E_{q'}$ implies that also E'_d is $\leq_{w(d)}$-left-closed) that $w'(d) \preceq w(d)$ and hence $w'(d) \in \mathcal{R}(\ell_{p'}(d))$.

Furthermore, $E_{w'(d)} \neq \emptyset$ and for all $d \not\leq_p d'$, if $e\ D\ e'$ for some $e \in E_{w'(d)}$ and $e' \in E_{w(d')}$ then (due to D-consistency of r) $d\ D\ d'$, implying $d' <_p d$; hence (due to the $\leq_q$-left-closure of $E_{q'}$) $(d', e') \in E_{q'}$, implying $e' \in E'_{d'}$. We may conclude $\ell_{w'(d)}(E_{w'(d)})\ I\ \ell_{w(d')}(E_{w(d')}) \setminus E'_{d'}$, and hence $\ell_{w'(d)}(E_{w'(d)}) \subseteq T_{w(d')} \setminus [\ell_{w(d')}(E_{w(d')}) \setminus E'_{d'}]_D = T_{w'(d')}$.

Finally, it is straightforward to see that the refinement of p' according to w' gives rise to $q' \in \mathcal{P}[\mathcal{R}]$.

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). □

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. Hence we have

Proposition 4.8 *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 $\mathbf{L}$, in the form of a function $[\![\Leftrightarrow]\!] \colon \mathbf{L} \to \mathbf{M}$. It is given in Table 4.

One property of the denotational semantics that will be used below is the following correspondence between (operational) termination of a term and the presence of empty system runs in its denotational model:

Lemma 4.9 *For all closed* $B \in \mathbf{L}$ *and* $a \in Act$, $B \stackrel{a}{\Leftrightarrow}$ *iff* $\varepsilon_a \in [\![B]\!]$.

This is straightforward to prove by induction on the structure of B.

4.2 Recursion

Denotationally, the semantics of a (closed) recursive term $rec\, X.\, B$ (i.e., such that $fv(B) \subseteq \{X\}$) 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 $[\![B]\!](\leftrightarrows): \mathbf{M} \to \mathbf{M}$ is a function derived from B in the standard way: the equations in Table 4 are extended with a parameter, i.e., such that

$$\begin{aligned}
[\![\mathbf{0}_A]\!](\mathcal{P}) &= \{\varepsilon_T \mid T \ I \ A\} \\
[\![a]\!](\mathcal{P}) &= [\![\mathbf{0}_a]\!] \cup \{\boxed{\epsilon^a}T \mid T \subseteq Act\} \\
[\![B_1 + B_2]\!](\mathcal{P}) &= \iota_1([\![B_1]\!](\mathcal{P})) + \iota_2([\![B_2]\!](\mathcal{P})) \\
[\![B_1 \|_A B_2]\!](\mathcal{P}) &= [\![B_1]\!](\mathcal{P}) \|_A [\![B_2]\!](\mathcal{P}) \\
[\![B_1 \cdot B_2]\!](\mathcal{P}) &= \iota_1([\![B_1]\!](\mathcal{P})) \cdot \iota_2([\![B_2]\!](\mathcal{P})) \\
[\![B[r]]\!](\mathcal{P}) &= [\![B]\!](\mathcal{P})[a \mapsto [\![r(a)]\!] \mid a \in Act] \\
[\![rec\, Y.\, B]\!](\mathcal{P}) &= \bigcup_{i \in \mathbb{N}} [\![B^i_Y]\!](\mathcal{P}) \\
[\![X]\!](\mathcal{P}) &= \mathcal{P}
\end{aligned}$$

(Note that this effectively defines a semantic counterpart to syntactic substitution; in particular, $[\![B\langle C/X\rangle]\!] = [\![B]\!]([\![C]\!])$.) We first show that $[\![B]\!](\leftrightarrows)$ itself has a unique fixpoint, which equals $[\![rec\, X.\, B]\!]$; this is certainly a solution of $X = B$. We then show that all solutions are isomorphic.

A technical property of the functions thus constructed, which will be useful in the sequel, is that for all $B \in \mathbf{L}$, $[\![B]\!](\leftrightarrows)$ 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 4.10 *Let $\mathcal{P}, \mathcal{Q} \in \mathbf{M}$ and $B \in \mathbf{L}$ with $fv(B) \subseteq \{X\}$. If $\mathcal{P} \subseteq \mathcal{Q}$, then $[\![B]\!](\mathcal{P}) \subseteq [\![B]\!](\mathcal{Q})$.*

We have to be slightly careful, however, about the desired termination properties. Namely, as we have seen in the operational semantics (Proposition 3.9), 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 the desired uniqueness of fixpoints does not hold.

Example 4.11 *Consider $rec\, X.\, a \cdot X$ with $\mathcal{A}_X = \{a\}$. The semantics of this term according to Table 4 is given by $[\![B]\!] \cong \{\langle a^n, T\rangle \mid n \in \mathbb{N}, T \subseteq [a]_I\}$ (where $\langle a^n, T\rangle$ denotes a system run consisting of n consecutive occurrences of the action a and termination set T); taking just the runs with maximal termination sets, this can be depicted by*

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

This system model satisfies the equation $\mathcal{P} \cong [\![a \cdot X]\!](\mathcal{P})$, but not uniquely: for instance, $\mathcal{Q} = \{\langle a^n, \emptyset\rangle \mid n \in \mathbb{N}\}$ is another solution.

We may call $\mathcal{P}$ *globally terminated* w.r.t. $T \subseteq Act$ if $T \ I \ \ell_p(E_p)$ and $p \cup T \in \mathcal{P}$ for all $p \in \mathcal{P}$; the class of all such system models will be denoted $\mathbf{M}_T$. Note that $\mathbf{M}_T \subseteq \mathbf{M}_{T'}$ if $T \supseteq T'$. (for instance, $[\![b]\!]$ in the above example is globally terminated w.r.t. $[a]_I$, whereas $\mathcal{Q}$ is not.) In the semantics defined by Table 4, the system model of arbitrary $B \in \mathbf{L}$ is globally terminated w.r.t. $[\mathcal{A}(B)]_I$.

Proposition 4.12 *For all closed $B \in \mathbf{L}$, $[\![B]\!] \in \mathbf{M}_{[\mathcal{A}(B)]_I}$.*

Proof. This follows from Proposition 4.13 just below, since any closed term B gives rise to a constant function $[\![B]\!](\leftrightarrows)$, namely such that $[\![B]\!](\mathcal{P}) = [\![B]\!]$ for all $\mathcal{P} \in \mathbf{M}$. □

Accordingly, we will in fact interpret $[\![B]\!](\leftrightarrows)$ 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 implies that this interpretation is valid.

Proposition 4.13 *Let $B \in \mathbf{L}$ with $fv(B) \subseteq \{X\}$. If $\mathcal{P} \in \mathbf{M}_{[\mathcal{A}_X]_I}$, then $[\![B]\!](\mathcal{P}) \in \mathbf{M}_{[\mathcal{A}(B)]_I}$.*

It follows that any closed recursive term $rec\, X.\, B \in \mathbf{L}$ (which is well-formed only if $\mathcal{A}(B) \subseteq \mathcal{A}_X$, see Table 1) gives rise to a function $[\![B]\!](\Leftrightarrow)\colon \mathbf{M}_T \to \mathbf{M}_T$ with $T = [\mathcal{A}_X]_I$. We now show that if X is in addition weakly guarded in B, then $[\![rec\, X.\, B]\!]$ as defined in Table 4 is the unique fixpoint of $[\![B]\!](\Leftrightarrow)$ in $\mathbf{M}_T$ —even if it is not unique in $\mathbf{M}$, as Example 4.11 shows.

A complete metric space of system models. In order to achieve this, we use the theory of metric spaces; cf. [23] 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 at least the termination properties of $\mathcal{P}$ and $\mathcal{Q}$ coincide, then $\delta(\mathcal{P}, \mathcal{Q}) = \frac{1}{2}$ and note $\delta(\mathcal{P}, \mathcal{Q}) = 1$. 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 constructions are standard —for event-based models, very similar ones can be found in [52]— and so is the (proof of the) following theorem.

Theorem 4.14 *$\langle \mathbf{M}, \delta \rangle$ 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 $\bigcup_{i \in \mathbb{N}} \bigcap_{j \ge i} \mathcal{P}_i$. Note that the concepts of "contracting" and "non-increasing" apply to partial as well as total functions over $\mathbf{M}$. 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 [23]). We now prove that functions of the form $[\![B]\!](\Leftrightarrow)$ are contracting if X is weakly guarded in B. (Note that this is not true for arbitrary B; as an extreme case, $[\![X]\!](\Leftrightarrow)$ is clearly not contracting.)

Proposition 4.15 *Let $B \in \mathbf{L}$ with $fv(B) \subseteq \{X\}$.*

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

Proof. Let $f = [\![B]\!](\Leftrightarrow)$. 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 weakly 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

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

and hence $\delta(f(\mathcal{P}), f(\mathcal{Q})) \le \delta(\mathcal{P}, \mathcal{Q})$; 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 4.10) implies $f(\mathcal{P})\!\uparrow\! n \subseteq f(\mathcal{Q})$.

   Analogous arguments apply in the cases where $B = C[r]$ or $B = rec\, Y.\, C$. 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 $f_i = [\![B_i]\!](\Leftrightarrow)$ 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* $B_1\langle \mathbf{0}_{\mathcal{A}_X} / X\rangle \Leftrightarrow$ implies $a\ I\ \mathcal{A}(B_2)$. We concentrate on the latter case. As we have seen above, $[\![B_1\langle \mathbf{0}_{\mathcal{A}_X} / X\rangle]\!] = f_1([\![\mathbf{0}_{\mathcal{A}_X}]\!])$, hence (due to Lemma 4.9) $B_1\langle \mathbf{0}_{\mathcal{A}_X} / X\rangle \Leftrightarrow$ iff $\varepsilon_a \in f_1([\![\mathbf{0}_{\mathcal{A}_X}]\!])$. Since by the induction hypothesis $f_1([\![\mathbf{0}_{\mathcal{A}_X}]\!])\!\uparrow\! 0 = f_1(\mathcal{P})\!\uparrow\! 0$, the weak guardedness condition has the consequence that $\varepsilon_T \in f_1(\mathcal{P})$ implies $T\ I\ \mathcal{A}(B_2)$, which in turn implies that $f_2(\mathcal{P})$ is globally terminated for T, i.e., all $p \in f_2(\mathcal{P})$ satisfy $T\ I\ \ell_p(E_p)$ and $p \cup T \in f_2(\mathcal{P})$.

   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 $\ell_{p_2}(E_{p_2}) \subseteq T_{p_1}$ and

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

   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, $q \in \iota_1(f_1(\mathcal{Q}))$. By weak guardedness, it follows that $\varepsilon_{T_q} \in \iota_2(f_2(\mathcal{Q}))$, and hence $q \in f(\mathcal{Q})$.

   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. It follows that $d\ D\ e$ for some $d \in E_{p_1}$, since otherwise $\varepsilon_a \preceq p_1$ with $a = \ell_{p_2}(e)$, which would imply $\varepsilon_a \in f_1(\mathcal{P})$ and $a\ D\ \mathcal{A}(B_2)$, which contradicts the weak guardedness condition. It follows that every maximal $<_q$-chain starts with an event from p_1, and hence $depth(p_2) \leq depth(q) \Leftrightarrow 1 \leq n$. 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 $q \in f(\mathcal{Q})$. □

This gives rise to the following theorem.

Theorem 4.16 *If $B \in \mathbf{L}$ with $fv(B) \subseteq X$, and X is weakly guarded in B, then $[\![rec\, X.\, B]\!]$ is the unique solution of $X = B$ in $\mathbf{M}_{[\mathcal{A}_X]_I}$ modulo $\cong$.*

Proof. First we prove that $[\![rec\, X.\, B]\!]$ is indeed a solution of $X = B$, by showing that it is the (unique) fixpoint of the function $[\![B]\!](\Leftrightarrow): \mathbf{M}_T \to \mathbf{M}_T$, 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 $[\![B]\!](\Leftrightarrow)$ i times to an arbitrary starting point $\mathcal{P} \in \mathbf{M}_T$ (which is a Cauchy sequence because $[\![B]\!](\Leftrightarrow)$ is contracting, see Proposition 4.15); 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 4.10) 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

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

Now we prove that all solutions of $X = B$ modulo isomorphism are isomorphic to $[\![rec\, X.\, B]\!]$. 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 $\phi: Evt \to Evt$, meaning that $\mathcal{P}$ is a fixpoint of the function $\phi[\![B]\!](\Leftrightarrow): \mathbf{M}_T \to \mathbf{M}_T$ defined by $\mathcal{Q} \mapsto \phi([\![B]\!](\mathcal{Q}))$ for all $\mathcal{Q} \in \mathbf{M}_T$. $\phi[\![B]\!](\Leftrightarrow)$ is contracting since $[\![B]\!](\Leftrightarrow)$ is; therefore, its fixpoint $\mathcal{P}$ is unique and can be constructed in a similar way as $[\![rec\, X.\, B]\!]$:

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

We now show that there exists a sequence of isomorphisms $\psi^i: [\![B^i_X]\!] \cong (\phi[\![B]\!])^i(\bot_T)$ such that $\psi^i \subseteq \psi^{i+1}$ for all $i \in \mathbb{N}$, where the ψ^i depend on ϕ and B. It follows that $\psi = \bigcup_{i \in \mathbb{N}} \psi^i$ is an isomorphism proving $[\![rec\, X.\, B]\!] \cong \mathcal{P}$.

The ψ^i are constructed using a *transformer of functions* $\Phi_B: (Evt \rightharpoonup Evt) \to (Evt \rightharpoonup Evt)$ mapping partial function to partial functions, in particular partial injections to partial injections: namely, $\psi^0 = \emptyset$ (the empty function) and $\psi^{i+1} = (\phi \circ \Phi_B(\psi^i)) \restriction E_{[\![B^i_X]\!]}$. Φ_B is defined inductively on the structure of B, as follows:

$$\begin{array}{rcl}
\Phi_{\mathbf{0}_A}(\psi) & = & \emptyset \\
\Phi_{a,\phi}(\psi) & = & \{e \mapsto e\} \\
\Phi_{B_1 \diamond B_2}(\psi) & = & \Phi_{B_1}(\psi) \times \Phi_{B_1}(\psi) \quad \text{for } \diamond \in \{+, \|_A, \cdot\} \\
\Phi_{B[r]} & = & \Phi_B(\psi) \times id_{Evt} \\
\Phi_{rec\, Y.\, B}(\psi) & = & \bigcup_{i \in \mathbb{N}} \Phi_{B^i_Y}(\psi) \\
\Phi_X(\psi) & = & \psi
\end{array}$$

where, if $\psi_i: Evt \rightharpoonup Evt$ are partial injections for $i = 1, 2$, then $(\psi_i \times \psi_2): Evt \rightharpoonup Evt$ is a partial injection defined by $(e_1, e_2) \mapsto (\psi_1(e_1), \psi_2(e_2))$, with the understanding that $\psi_i(*) = *$ for $i = 1, 2$.

Each Φ_B is $\subseteq$-monotonic over the space of partial injections; since ψ^0 is the empty function, it follows that $\psi^i \subseteq \psi^{i+1}$ for all $i \in \mathbb{N}$. By induction on the structure of B, it is straightforward to prove $\phi([\![B]\!](\psi([\![C]\!]))) = \phi(\Phi_B(\psi)([\![B]\!]([\![C]\!])))$ for arbitrary closed $C \in \mathbf{L}$ with $[\![C]\!] \in \mathbf{M}_T$ and arbitrary injections $\psi: E_{[\![C]\!]} \to Evt$. By induction on i, it follows that $\psi^i([\![B^i_X]\!]) = (\phi[\![B]\!])^i(\bot_T)$ for all $i \in \mathbb{N}$. Hence the proof obligations for the ψ^i are met. $\square$

4.3 Transitions

To show the correspondence between operational and denotational semantics, we turn $\mathbf{M}$ into an $Act \cup \checkmark_{Act}$-labelled transition system. An a-labelled transition may occur if there is a (causally) minimal a-labelled event in one of the system runs; the resulting model consists of all system runs which had that event in a causally minimal position, minus the event itself. A $\checkmark_a$-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, and their prefixes. Thus, $\Leftrightarrow \subseteq \mathbf{M} \times (Act \cup \checkmark_{Act}) \times \mathbf{2^P}$ is the smallest relation such that:

- If $p \in \mathcal{P}$ with $E_p = \{e\}$, then $\mathcal{P} \overset{\ell_p(e)}{\Leftrightarrow} \{p \setminus e \mid p \in \mathcal{P}, e \in \min_{\leq_p} E_p\}$;
- If $\varepsilon_{\{a\}} \in \mathcal{P}$, then $\mathcal{P} \overset{\checkmark_a}{\Leftrightarrow} \{p \mid p \cup a \in \mathcal{P}, a \; I \; \ell_p(E_p)\}$.

The following lemma (the proof of which is left to the reader) states that the target of transitions is always a system model; i.e., $\mathbf{M}$ is closed under transitions.

Lemma 4.17 *If $\mathcal{P} \in \mathbf{M}$ and $\mathcal{P} \overset{\alpha}{\Leftrightarrow} \mathcal{P}'$, then $\mathcal{P}' \in \mathbf{M}$.*

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

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

The next observation is that these transitions are well-defined up to isomorphism, in the following sense:

Lemma 4.18 *If $\mathcal{P} \cong \mathcal{Q}$ and $\mathcal{P} \overset{\alpha}{\Leftrightarrow} \mathcal{P}'$ for $\alpha \in Act \cup \checkmark_{Act}$, then $\mathcal{Q} \overset{\alpha}{\Leftrightarrow} \mathcal{Q}'$ such that $\mathcal{P}' \cong \mathcal{Q}'$.*

Since $\cong$ is a symmetrical relation, the above lemma can also be read to say that $\cong$ is a bisimulation relation over $\mathbf{M}$. This gives rise to the following property:

Proposition 4.19 *If $\mathcal{P} \cong \mathcal{Q}$, then $lts(\mathcal{P}) \sim lts(\mathcal{Q})$.*

Note that transitions can be defined alternatively between the system runs of a given system model $\mathcal{P}$, as the smallest relation $\Leftrightarrow \subseteq \mathcal{P} \times (Act \cup \checkmark_{Act}) \times \mathcal{P}$ such that for all $p \in \mathcal{P}$:

- If $e \in \max_{\leq_p} E_p$ and $\ell_p(e) \notin T_p$, then $p \setminus e \overset{\ell_p(e)}{\Leftrightarrow} p$;
- If $p \cup a \in \mathcal{P}$, then $p \overset{\checkmark_a}{\Leftrightarrow} p \cup a$.

The first item states that every system run p can be reached via a transition corresponding to a maximal event of p. Note that $p \setminus e \preceq p$ if $e \in \max E_p$ and $\ell_p(e) \; I \; T_p$, and hence $p \setminus e \in \mathcal{P}$. The following proposition states that the two methods for defining transitions on the denotational model give rise to bisimilar interpretations.

Proposition 4.20 *For all $\mathcal{P} \in \mathbf{M}$, $lts(\mathcal{P}) \sim \langle \mathcal{P}, \Leftrightarrow, \varepsilon_\emptyset \rangle$.*

Proof. The bisimulation relation is given by $\rho = \{(\mathcal{P} \Leftrightarrow p, p) \mid p \in \mathcal{P}\}$, where for all $p \in \mathcal{P}$ the "residual" $\mathcal{P} \Leftrightarrow p$ is given by

$$\mathcal{P} \Leftrightarrow p = \{q \setminus E_p \mid p \preceq (q \cup T_p) \in \mathcal{P}\}$$

It is not difficult to see that the following holds for arbitrary $p \preceq q \in \mathcal{P}$:

$$\mathcal{P} \Leftrightarrow q = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow (q \setminus E_p)$$

Moreover, the transition relation over $\mathbf{M}$ can be characterised as follows:

$$\begin{aligned} \mathcal{P} \overset{a}{\Leftrightarrow} \mathcal{P}' \quad &\Longleftrightarrow \quad \exists p \in \mathcal{P}\colon E_p = \{e\},\ \ell_p(e) = a,\ \mathcal{P}' = \mathcal{P} \Leftrightarrow p \\ \mathcal{P} \overset{\checkmark_a}{\Leftrightarrow} \mathcal{P}' \quad &\Longleftrightarrow \quad \varepsilon_a \in \mathcal{P},\ \mathcal{P}' = \mathcal{P} \Leftrightarrow \varepsilon_a \ . \end{aligned}$$

Now we prove that ρ above is a bisimulation relation. Let $(\mathcal{P} \Leftrightarrow p, p) \in \rho$ be arbitrary.

- Assume $(\mathcal{P} \Leftrightarrow p) \stackrel{a}{\Leftrightarrow} \mathcal{P}'$.
  - Assume $\alpha = a$, and let e be the event modelling this a-occurrence; i.e., $\mathcal{P}' = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow q$ for some $q \in \mathcal{P} \Leftrightarrow p$ with $E_q = \{e\}$ and $\ell_q(e) = a$. It follows that there must be some $p' \in \mathcal{P}$ such that $p \preceq p'$ and $q = p' \setminus E_p$; hence $E_{p'} = E_p \cup e$ and $T_{p'} = T_p$. This implies $a \notin T_p$, and hence $p \stackrel{a}{\Leftrightarrow} p'$; moreover, $\mathcal{P} \Leftrightarrow p' = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow p' = \mathcal{P}'$, hence $(\mathcal{P}', p') \in \rho$.
  - Assume $\alpha = \checkmark_a$. It follows that $\varepsilon_a \in \mathcal{P} \Leftrightarrow p$; hence there is some $p' \cup T_p \in \mathcal{P}$ such that $p \preceq p'' = p' \cup T_p$ and $\varepsilon_a = p' \setminus E_p$, implying $p'' = p \cup a$. It follows that $p \stackrel{\checkmark_a}{\Rightarrow} p''$. Moreover, since $q \cup T_p \in \mathcal{P} \Leftrightarrow p$ iff $q \in \mathcal{P} \Leftrightarrow p$ we have $\mathcal{P} \Leftrightarrow p'' = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow (p'' \setminus E_p) = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow \varepsilon_{T_p \cup a} = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow \varepsilon_a = \mathcal{P}'$; hence $(\mathcal{P}', p'') \in \rho$.
- Assume $p \stackrel{a}{\Leftrightarrow} p'$.
  - Assume $\alpha = a$; hence there is an $e \in \max E_{p'}$ such that $\ell_{p'} = a$ and $p = p' \setminus e$. It follows that $p' \setminus E_p \in \mathcal{P} \Leftrightarrow p$, and hence $\mathcal{P} \Leftrightarrow p \stackrel{a}{\Leftrightarrow} (\mathcal{P} \Leftrightarrow p) \Leftrightarrow (p' \setminus E_p) = \mathcal{P} \Leftrightarrow p'$ with $(\mathcal{P} \Leftrightarrow p', p') \in \rho$.
  - Assume $\alpha = \checkmark_a$; hence $p' = p \cup a$. It follows that $p' \setminus E_p = \varepsilon_{T_{p'}} \in \mathcal{P} \Leftrightarrow p$ and hence $\varepsilon_a \in \mathcal{P} \Leftrightarrow p$. Using the fact that $q \cup T_p \in \mathcal{P} \Leftrightarrow p$ iff $q \in \mathcal{P} \Leftrightarrow p$, we can derive $\mathcal{P} \stackrel{\checkmark_a}{\Leftrightarrow} (\mathcal{P} \Leftrightarrow p) \Leftrightarrow \varepsilon_a = (\mathcal{P} \Leftrightarrow p) \Leftrightarrow (p' \setminus E_p) = \mathcal{P} \Leftrightarrow p'$ with $(\mathcal{P} \Leftrightarrow p', p') \in \rho$. □

It should be noted that, although the two transition systems are bisimilar, they are in no way isomorphic: the former identifies many states that are distinguished 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 former interpretation (where there is just a single state, $[\![\mathbf{0}_A]\!]$ itself).

We will now prove a correspondence between transitions derived using the operational rules and transitions of the denotational model. For the refinement operator, the correspondence only holds for strongly D-consistent refinement functions. The denotational consequence of strong D-consistency is formulated in the following proposition.

Proposition 4.21 *If r is strongly D-consistent, then $a\ D\ b$ iff for all $p \in [\![r(a)]\!]$, $q \in [\![r(b)]\!]$ and $e \in \min E_q$, either $e\ D\ d$ for some $d \in E_p$ or $\ell_q(e) \notin T_p$.*

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 $B \in \mathbf{L}$, $lts(B) \sim lts([\![B]\!])$.*

The proof uses induction on the structure of B. For the case of recursion, we use the uniqueness of fixpoints modulo $\sim$, stated in Theorem 3.19: essentially, because the denotational semantics of $[\![rec\, X.B]\!]$ yields a solution of $X = B$, it must be bisimilar to (the operational interpretation of) $rec\, X.B$. However, a technical problem in this argument is that we have proved Theorem 3.19 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 $[\![rec\, X.B]\!]$. To circumvent this, for the purpose of the following proof, we introduce additional constants $t_{\mathcal{P}}$ to $\mathbf{L}$ for every $\mathcal{P} \in \mathbf{M}$, with operational semantics generated by

$$\frac{\mathcal{P} \stackrel{a}{\Leftrightarrow} \mathcal{P}'}{t_{\mathcal{P}} \stackrel{a}{\Leftrightarrow} t_{\mathcal{P}'}}$$

and denotational semantics determined by $[\![t_{\mathcal{P}}]\!] = \mathcal{P}$. It is immediately clear that $lts(\mathcal{P}) \sim lts(t_{\mathcal{P}})$. As remarked in Section 3.4, the proof of Theorem 3.19 is not invalidated if we extend $\mathbf{L}$ in this way.

Proof of Theorem 4.22. 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 $[\![B]\!] \stackrel{a}{\Longrightarrow} \mathcal{P}$. Since $E_{[\![B]\!]} = \emptyset$, it follows that $\alpha = \surd_a$ 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, $B \stackrel{\surd a}{\Longrightarrow} \mathbf{0}_A$; 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 $[\![B]\!] \stackrel{a}{\Longrightarrow} \mathcal{P}$. By construction of $[\![B]\!]$, it follows that $a = b$ and $\mathcal{P} = [\![\mathbf{0}_{Act}]\!]$. This is matched by $B \stackrel{a}{\Longrightarrow} B'$ with $B' = \mathbf{0}_{Act}$.
- Now assume $[\![B]\!] \stackrel{\surd a}{\Longrightarrow} \mathcal{P}$. By construction of $[\![B]\!]$, it follows that $a\ I\ b$ and hence $\mathcal{P} = [\![B]\!]$. This is matched by $B \stackrel{\surd a}{\Longrightarrow} B'$ 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

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

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 $\mathcal{P} \stackrel{a}{\Longrightarrow} \mathcal{P}'$. 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 $\pi_1(e) \in E_{\pi_1(p)}$ for $\pi_1(p) \in \mathcal{P}_1$. It follows that $\mathcal{P}_1 \stackrel{a}{\Longrightarrow} \mathcal{P}_1'$ where $\mathcal{P}' = \iota_1(\mathcal{P}_1')$. By the inner induction hypothesis, $C_1 \stackrel{a}{\Longrightarrow} C_1'$ such that $C_1' \sim \mathcal{P}_1' \cong \mathcal{P}'$, implying $(C_1', \mathcal{P}') \in \rho$; and by the operational semantics, $C \stackrel{a}{\Longrightarrow} C_1'$.
- Assume $\mathcal{P} \stackrel{\surd a}{\Longrightarrow} \mathcal{P}'$. It follows that $\mathcal{P}' = \mathcal{P}_1' \cup \mathcal{P}_2'$ where for $i = 1, 2$, $\mathcal{P}_i' = \{p \mid p \cup a \in \iota_i(\mathcal{P}_1), a\ I\ \ell_p(E_p)\}$. 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 $\mathcal{P}_i \stackrel{\surd a}{\Longrightarrow} \pi_i(\mathcal{P}_i')$, implying (by the inner induction hypothesis) that $C_i \stackrel{\surd a}{\Longrightarrow} C_i'$ 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 $C \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} C_1' + C_2'$.

Otherwise, assume $\varepsilon_{\{a\}} \notin \mathcal{P}_i$ for $i = 1$ (the case $i = 2$ is symmetrical). It follows that $\iota_1(\mathcal{P}_1) \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow}$ and $\iota_2(\mathcal{P}_2) \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} \mathcal{P}'$, implying $\mathcal{P}_1 \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow}$ and $\mathcal{P}_2 \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} \pi_1(\mathcal{P}')$, hence by the inner induction hypothesis, $C_1 \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow}$ and $C_2 \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} C_2'$ such that $C_2' \sim \pi_1(\mathcal{P}') \cong \mathcal{P}'$; hence $(C_2', \mathcal{P}') \in \rho$. By the operational semantics, finally, $C \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} C_2'$.

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

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

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 $\mathcal{P} \stackrel{a}{\Longrightarrow} \mathcal{P}'$. 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 $\ell_{p_1} \parallel_A \ell_{p_2}$) 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 $\mathcal{P}_1 \stackrel{a}{\Longrightarrow} \mathcal{P}_1'$, implying (by the inner induction hypothesis) $C_1 \stackrel{a}{\Longrightarrow} C_1'$ 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 $C \stackrel{a}{\Longrightarrow} C_1' \parallel_A C_2$.

  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 $\mathcal{P}_i' = \{p \setminus e_i \mid p \in \mathcal{P}_i, e \in \min E_p\}$. Hence $\mathcal{P}_i \stackrel{a}{\Longrightarrow} \mathcal{P}_i'$, implying (by the inner induction hypothesis) $C_i \stackrel{a}{\Longrightarrow} C_i'$ 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 $C \stackrel{a}{\Longrightarrow} C_1' \parallel_A C_2'$.

- Assume $\mathcal{P} \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} \mathcal{P}'$. 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 $a \ I \ \ell_q(E_q)$ iff $p_i \cup a \in \mathcal{P}_i$ and $a \ I \ \ell_{p_i}(E_{p_i})$ for $i = 1, 2$. It follows that $\mathcal{P}' = \mathcal{P}_1' \parallel_A \mathcal{P}_2'$ where $\mathcal{P}_i' = \{p \mid p \cup a \in \mathcal{P}_i, a \ I \ \ell_p(E_p)\}$.

  Due to $\varepsilon_{\{a\}} \in \mathcal{P}$ we know that $\varepsilon_{\{a\}} \in \mathcal{P}_i$ and hence $\mathcal{P}_i \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} \mathcal{P}_i'$ for $i = 1, 2$; hence (by the inner induction hypothesis) $C_i \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} C_i'$ 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 $C \stackrel{\varepsilon_{\{a\}}}{\Longrightarrow} C_1' \parallel_A C_2'$.

Sequential composition. The sequential composition of system models is given alternatively by $\mathcal{P}_1 \cdot \mathcal{P}_2 = \{p_1 \cdot p_2 \mid p_1 \in \mathcal{P}_1, p_2 \in \mathcal{P}_2, \ell_{p_2}(E_{p_2}) \subseteq T_{p_1}\}$ where

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

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 $\mathcal{P} \stackrel{a}{\Leftarrow\!\!\Rightarrow} \mathcal{P}'$. 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 $\ell_{p_2}(E_{p_2}) \subseteq T_{p_1}$ and either $e \in \min E_{p_1}$ (if $e = (d, *)$) or $a \ I \ \ell_{p_1}(E_{p_1})$ 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 $\mathcal{P}_1 \stackrel{a}{\Leftarrow\!\!\Rightarrow} \mathcal{P}_1'$, implying (by the inner induction hypothesis) $C_1 \stackrel{a}{\Leftarrow\!\!\Rightarrow} C_1'$ with $C_1' \sim \mathcal{P}_1'$; hence $(C_1' \cdot C_2, \mathcal{P}_1') \in \rho$. By the operational semantics, therefore, $C \stackrel{a}{\Leftarrow\!\!\Rightarrow} C_1' \cdot C_2$.

  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 $\mathcal{P}_1' = \{p \mid p \cup a \in \mathcal{P}_1, a \ I \ \ell_p(E_p)\}$ and $\mathcal{P}_2' = \{p \setminus e \mid p \in \mathcal{P}_2, e \in \min E_p\}$. But then $\mathcal{P}_1 \stackrel{\surd_a}{\Leftarrow\!\!\Rightarrow} \mathcal{P}_1'$ and $\mathcal{P}_2 \stackrel{a}{\Leftarrow\!\!\Rightarrow} \mathcal{P}_2'$, implying (by the inner induction hypothesis) $C_1 \stackrel{\surd_a}{\Leftarrow\!\!\Rightarrow} C_1'$ and $C_2 \stackrel{a}{\Leftarrow\!\!\Rightarrow} C_2'$ with $C_i' \sim \mathcal{P}_i'$; hence $(C_1' \cdot C_2', \mathcal{P}') \in \rho$. By the operational semantics, therefore, $C \stackrel{a}{\Leftarrow\!\!\Rightarrow} C_1' \cdot C_2'$.

- Assume $\mathcal{P} \stackrel{\surd_a}{\Leftarrow\!\!\Rightarrow} \mathcal{P}'$. The proof is analogous to that for $\surd_a$-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 $w(d) \in \mathcal{R}(\ell_p(d))$ and $w(d) \neq \varepsilon_T$ for all $d \in E_p$ and $\ell_{w(d)}(E_{w(d)}) \subseteq T_{w(d')}$ whenever $d \not\leq_p d'$. The refinement of a system run p according to a witness w is defined by

$$\begin{aligned} w(p) = \langle\ & \{(d, e) \mid d \in E_p, e \in E_{w(d)}\}, \\ & \{((d, e), a) \mid a = \ell_{w(d)}(e)\}, \\ & \{((d_1, e_1), (d_2, e_2)) \mid d_1 <_p d_2, e_1 \ D \ e_2 \text{ or } d_1 = d_2, e_1 <_{w(d_1)} e_2\}, \\ & T_{p_1} \cap T_{p_2} \rangle \ . \end{aligned}$$

Assume $B = B_1[r]$, where 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 $\mathcal{P} \stackrel{a}{\Leftrightarrow} \mathcal{P}'$. 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 4.21). 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

$$\ell_{w'(p\setminus d)}(E_{w'(p\setminus d)}) = \bigcup_{d' \in E_{p\setminus d}} \ell_{w'(d)}(E_{w'(d)}) \subseteq T_{w(d)\setminus e}$$

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 $b = \ell_p(d)$; then $\mathcal{P}' = (d \times \mathcal{P}_2') \cdot \mathcal{P}_1'[\mathcal{R}]$ where

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

Thus, $\mathcal{P}_1 \stackrel{b}{\Leftrightarrow} \mathcal{P}_1'$ and $\mathcal{R}(b) \stackrel{a}{\Leftrightarrow} \mathcal{P}_2'$, implying (by the inner induction hypothesis) $C_1 \stackrel{b}{\Leftrightarrow} C_1'$ and $r(b) \stackrel{a}{\Leftrightarrow} C_2'$ 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, $C[r] \stackrel{a}{\Leftrightarrow} C_2' \cdot C_1'[r]$.

- Assume $\mathcal{P} \stackrel{\surd a}{\Leftrightarrow} \mathcal{P}'$. Due to Proposition 4.21, it follows that $a \ I \ \ell_{w(p)}(E_{w(p)})$ and $w(p) \cup a \in \mathcal{P}[\mathcal{R}]$ for some $p \in \mathcal{P}'$ and $\mathcal{R}$-witness w on p iff $a \ I \ \ell_p(E_p)$ and $p \cup a \in \mathcal{P}$. It follows that $\mathcal{P}' = \mathcal{P}_1'[\mathcal{R}]$ where $\mathcal{P}_1' = \{p \mid a \ I \ \ell_p(E_p), p \cup a \in \mathcal{P}'\}$. Then $\mathcal{P}_1 \stackrel{\surd a}{\Leftrightarrow} \mathcal{P}_1'$, implying (by the inner induction hypothesis) $C_1 \stackrel{\surd a}{\Leftrightarrow} C_1'$ such that $C_1' \sim \mathcal{P}_1'$, and hence $(C_1'[r], \mathcal{P}')$; the operational semantics imply $C \stackrel{\surd a}{\Leftrightarrow} C_1'[r]$.

Recursion. Assume $B = rec\, X.\, B_1$. 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 C with recursion depth 0; in particular, also for $C = t_{[B]}$. But this implies (since $[\![t_{[\![B]\!]}]\!] = [\![B]\!]$ is a fixpoint of $[\![B_1]\!](\Leftrightarrow)$) 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 3.19, therefore, $B \sim t_{[\![B]\!]} \sim [\![B]\!]$. □

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 $\mathbf{L}_{fin}$,

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

that is, T will be sound and complete for bisimilarity in $\mathbf{L}_{fin}$. 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

For the sake of simplicity, we first deal with the language without refinement. As might be expected, the unusual behaviour of weak sequential composition forces some modifications to the standard axioms for bisimilarity. The axiomatisation

left merge	$\dfrac{x \stackrel{a}{\Longrightarrow} x' \quad a \notin A}{x \between_A y \stackrel{a}{\Longrightarrow} x' \parallel_A y}$ $\qquad$ $\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \quad y \stackrel{\surd a}{\Longrightarrow} y'}{x \between_A y \stackrel{\surd a}{\Longrightarrow} x' \between_A y'}$
communication merge	$\dfrac{x \stackrel{a}{\Longrightarrow} x' \quad y \stackrel{a}{\Longrightarrow} y' \quad a \in A}{x \mid_A y \stackrel{a}{\Longrightarrow} x' \parallel_A y'}$ $\qquad$ $\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \quad y \stackrel{\surd a}{\Longrightarrow} y'}{x \mid_A y \stackrel{\surd a}{\Longrightarrow} x' \mid_A y'}$
left sequential	$\dfrac{x \stackrel{a}{\Longrightarrow} x'}{x \leftarrow y \stackrel{a}{\Longrightarrow} x' \cdot y}$ $\qquad$ $\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \quad y \stackrel{\surd a}{\Longrightarrow} y'}{x \leftarrow y \stackrel{\surd a}{\Longrightarrow} x' \leftarrow y'}$
right sequential	$\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \quad y \stackrel{a}{\Longrightarrow} y'}{x \rightarrow y \stackrel{a}{\Longrightarrow} x' \cdot y'}$ $\qquad$ $\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \quad y \stackrel{\surd a}{\Longrightarrow} y'}{x \rightarrow y \stackrel{\surd a}{\Longrightarrow} x' \rightarrow y'}$
residue	$\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \stackrel{\beta}{\Longrightarrow} x''}{x \downarrow a \stackrel{\beta}{\Longrightarrow} x''}$
deadlock	$\dfrac{x \stackrel{\surd a}{\Longrightarrow} x' \quad a \in [A]_I}{\delta_A(x) \stackrel{\surd a}{\Longrightarrow} \delta_A(x')}$

Table 5: Operational semantics of auxiliary operators

of ACP [10] 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 $(a + b) \cdot c \Leftrightarrow (a + b) \cdot \mathbf{1}$ which can still do both a and b; however, if $a \cdot c + b \cdot c \Leftrightarrow B$, 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 $(a + b) \cdot c \not\leftrightarrow a \cdot c + b \cdot c$.

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 $b \stackrel{\surd a}{\Longrightarrow} b \stackrel{\surd a}{\Longrightarrow} \cdots$. 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 5. The language including all auxiliary operators is denoted $\mathbf{L}^+$; as before, $\mathbf{L}^+_{fin}$ and $\mathbf{L}^+_{ff}$ are the finite and the finite flat fragments of $\mathbf{L}^+$, respectively. The same argument before shows that bisimilarity is still a congruence over the extended language.

Left merge and communication merge. $\between_A$ is the standard *left merge* from ACP, adapted to our notion of synchronisation and extended to deal with termination. $\mid_A$ is the relevant version of the *communication merge*. The idea is that $B \between_A C$ 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:

$$x \parallel_A y = x \between_A y + y \between_A x + x \mid_A y \;\; .$$

$$
\begin{array}{rcll}
\checkmark(\mathbf{0}_A) & = & [A]_I & \\
\checkmark(a) & = & [a]_I & \\
\checkmark(B_1 + B_2) & = & \checkmark(B_1) \cup \checkmark(B_2) & \\
\checkmark(B_1 \diamond B_2) & = & \checkmark(B_1) \cap \checkmark(B_2) & \text{if } \diamond \in \{\|_A, \between A, |A, \cdot, \leftarrow, \rightarrow\} \\
\checkmark(\delta_A(B)) & = & [A]_I \cap \checkmark(B) & \\
\checkmark(B[r]) & = & \checkmark(B) &
\end{array}
$$

Table 6: Partial termination set

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. Sequential composition can then be split up as follows:

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

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] and Section 5.3 below). For right sequential, on the other hand, the situation is more complicated.

The following distributivity and associativity properties are straightforward to establish:

$$
\begin{array}{rcl}
(x + y) \leftarrow z & = & x \leftarrow z + y \leftarrow z \\
x \rightarrow (y + z) & = & x \rightarrow y + x \rightarrow z \\
\\
(x \leftarrow y) \leftarrow z & = & x \leftarrow (y \cdot z) \\
x \rightarrow (y \leftarrow z) & = & (x \rightarrow y) \leftarrow z
\end{array}
$$

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 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 5 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 6 defines a partial function $\checkmark : \mathbf{L}^+_{fin} \rightharpoonup \mathbf{2}^{Act}$ that returns the set of actions for which a term, which may itself not contain the residue operator, is partially terminated. Note that $\checkmark(B)$ is partially dependent on the alphabet of B, in that $[\mathcal{A}(B)]_I \subseteq \checkmark(B)$ for all B. The following proposition states the crucial property of the partial termination set and the residue operator.

Proposition 5.1 *Let $B \in \mathbf{L}$ and $a \in Act$. $a \in \checkmark(B)$ if and only if $B \stackrel{\checkmark a}{\Rightarrow} B'$ 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 $B \stackrel{\checkmark a}{\Rightarrow}$ whereas $C \stackrel{\checkmark a}{\Rightarrow}$; 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 5). The deadlock operator serves a purpose similar to the encapsulation operator of ACP [10]; however, the use of the index set is different. In general, δ_A transforms a term B into a deadlocked term (i.e., $\delta_A(B) \stackrel{a}{\not\Rightarrow}$ 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. $\forall a \in Act: B \stackrel{a}{\not\Rightarrow}$), then $\delta_A(B) \sim \mathbf{0}_A \cdot B$.*

Using this forced deadlock operator, we can formulate the following axiom in place of the unsound $x + \mathbf{0}_{\mathcal{A}(x)} = x$:

$$x + \delta_A(x) = x \ .$$

5.2 The equational theory

The set of all relevant axioms is collected in Table 7. Note that Axiom S4 contains a side condition referring to the *alphabet* of a term; this is as yet undefined for the auxiliary operators introduced above. We extend Table 1 with the following rules:

$$\begin{aligned} \mathcal{A}(B_1 \diamond B_2) &= \mathcal{A}(B_1) \cup \mathcal{A}(B_2) \quad \text{where } \diamond \in \{|_A, \between_A, \leftarrow, \rightarrow\} \\ \mathcal{A}(\delta_A(B)) &= \mathcal{A}(B) \cup A \end{aligned}$$

Just as the termination set $\checkmark$ (see Table 6), the alphabet of terms containing a residue operator remains undefined. We can now extend Proposition 3.9 to $\mathbf{L}^+$:

Proposition 5.4 *Let $B \in \mathbf{L}^+$ such that $\mathcal{A}(B)$ is defined.*

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

Apart from the axioms already discussed at the introduction of the auxiliary operators, Table 7 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 [9]). 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.

We now come to the soundness proof of the axioms. To prove RS3 sound, we first need to extend Proposition 3.10 to $\mathbf{L}^+$:

$\mathbf{0}_A = \mathbf{0}_{A'}$	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
$x \parallel_A y = x \between_A y + y \between_A x + x \mid_A y$		P	$x\rightarrow a = a\leftarrow(x\downarrow a) + \delta_{\{a\}}(x)$	
$(x+y) \between_A z = (x \between_A z) + (y \between_A z)$		LM1		if $a \in \checkmark(x)$ RS3
$a\leftarrow x \between_{A\setminus\{a\}} y = a\leftarrow(x \parallel_{A\setminus\{a\}} y)$		LM2	$x\rightarrow a = \delta_{\{a\}}(x)$	
$a\leftarrow x \between_{A\cup\{a\}} y = \delta_{\{a\}}(x \parallel_{A\cup\{a\}} y)$		LM3		if $a \notin \checkmark(x)$ RS4
$\mathbf{0}_{A'} \between_A x = \delta_{A'}(x)$		LM4	$x\rightarrow\mathbf{0}_A = \delta_A(x)$	RS5
$x \mid_A y = y \mid_A x$		CM1	$(x+y)\downarrow a = x\downarrow a + y\downarrow a$	RD1
$(x+y) \mid_A z = x \mid_A z + y \mid_A z$		CM2	$b\leftarrow x\downarrow a = b\leftarrow(x\downarrow a)$	
$a\leftarrow x \mid_{A\cup\{a\}} a\leftarrow y = a\leftarrow(x \parallel_{A\cup\{a\}} y)$		CM3		if $a\ I\ b$ and $a \in \checkmark(x)$ RD2
$a\leftarrow x \mid_{A\cup\{a\}} b\leftarrow y = \delta_{\{a,b\}}(x \parallel_{A\cup\{a\}} y)$			$b\leftarrow x\downarrow a = \mathbf{0}$	
	if $a \neq b$	CM4		if $a\ D\ b$ or $a \notin \checkmark(x)$ RD3
$a\leftarrow x \mid_{A\setminus\{a\}} y = \delta_{\{a\}}(x \parallel_{A\setminus\{a\}} y)$		CM5	$\mathbf{0}_A\downarrow a = \mathbf{0}_A$	if $a\ I\ A$ RD4
$\mathbf{0}_{A'} \mid_A x = \delta_{A'}(x)$		CM6	$\mathbf{0}_A\downarrow a = \mathbf{0}$	if $a\ D\ A$ RD4
$\mathbf{1}\cdot x = x$		S1	$\delta_A(\mathbf{0}_{A'}) = \mathbf{0}_{A\cup A'}$	D1
$x\cdot\mathbf{1} = x$		S2	$\delta_A(a) = \mathbf{0}_{A\cup\{a\}}$	D2
$(x\cdot y)\cdot z = x\cdot(y\cdot z)$		S3	$\delta_A(x\diamond y) = \delta_A(x)\diamond\delta_A(y)$	
$x\cdot y = x \parallel_\emptyset y$	if $\mathcal{A}(x)\ I\ \mathcal{A}(y)$	S4		where $\diamond \in \{+, \leftarrow\}$ D3
$x\cdot y = x\leftarrow y + x\rightarrow y$		S5		

Table 7: The equational theory T

Proposition 5.5 *Let $B \in \mathbf{L}^+$.*

1. *If $B \stackrel{\checkmark_a}{\Longrightarrow} B'$ and $B \stackrel{\checkmark_a}{\Longrightarrow} B''$, then $B' = B''$.*
2. *If $B \stackrel{\checkmark_{a_1}}{\Longrightarrow} \cdots \stackrel{\checkmark_{a_n}}{\Longrightarrow} B'$ then also $B \stackrel{\checkmark_{b_1}}{\Longrightarrow} \cdots \stackrel{\checkmark_{b_n}}{\Longrightarrow} B'$, where $b_1 \cdots b_n$ is an arbitrary permutation of the actions $a_1 \cdots a_n$.*

Proof. The proof of Clause 2 extends that of Proposition 3.10 with cases for the auxiliary operators. We show only the most interesting case, $B = B_1 \downarrow a$.

It follows that $B_1 \stackrel{\checkmark_a}{\Longrightarrow} B_1' \stackrel{\checkmark_{a_1}}{\Longrightarrow} \cdots \stackrel{\checkmark_{a_n}}{\Longrightarrow} B'$. Since $a\, b_1 \cdots b_n$ is a permutation of $a\, a_1 \cdots a_n$ iff $b_1 \cdots b_n$ is a permutation of $a_1 \cdots a_n$, the induction hypothesis implies $B_1 \stackrel{\checkmark_a}{\Longrightarrow} B_1' \stackrel{\checkmark_{b_1}}{\Longrightarrow} \cdots \stackrel{\checkmark_{b_n}}{\Longrightarrow} B'$; hence $B \stackrel{\checkmark_{b_1}}{\Longrightarrow} \cdots \stackrel{\checkmark_{b_n}}{\Longrightarrow} B'$. □

The main soundness result is then formulated in the following theorem. It implies that we may use the equational theory T generated by Table 7 together with the usual equational proof rules to derive bisimilarity of terms in $\mathbf{L}^+$.

Theorem 5.6 *The axioms of Table 7 are sound w.r.t. bisimilarity.*

Proof. We show only the soundness of some of the most interesting axioms: C5, S4, RS3, RS4, RD1 and RD2.

C5 We show that the following is a bisimulation relation:

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

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

- If $\alpha = a$, then $B \stackrel{a}{\Longrightarrow} B'$ and $(B', B') \in \mathcal{R}$.
- If $\alpha = \checkmark_a$, then either $B \stackrel{\checkmark_a}{\Longrightarrow} B'$ and $a \notin [A]_I$ (i.e. the choice is resolved), in which case $(B', B') \in \mathcal{R}$; or $B \stackrel{\checkmark_a}{\Longrightarrow} B''$ 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 $B_1 \cdot B_2 \stackrel{a}{\Longrightarrow} B'$; there are two cases.

- $B_1 \stackrel{a}{\Longrightarrow} B_1'$ such that $B' = B_1' \cdot B_2$. It follows that $B_1 \parallel_\emptyset B_2 \stackrel{a}{\Longrightarrow} B_1' \parallel_\emptyset B_2$ such that $(B_1' \cdot B_2, B_1' \parallel_\emptyset B_2) \in \mathcal{R}$.
- $B_1 \stackrel{\checkmark_a}{\Longrightarrow} B_1'$ and $B_2 \stackrel{a}{\Longrightarrow} B_2'$ such that $B' = B_1' \cdot B_2'$ Due to Proposition 5.4, it follows that $a \in \mathcal{A}(B_2)$, and hence $a\ I\ \mathcal{A}(B_1)$; again by Proposition 5.4, it follows that $B_1' = B_1$. Furthermore, $B_1 \parallel_\emptyset B_2 \stackrel{a}{\Longrightarrow} B_1 \parallel_\emptyset B_2'$ such that $(B_1 \cdot B_2', B_1 \parallel_\emptyset B_2') \in \mathcal{R}$.

RS4 The following is a bisimulation relation:

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

We show only left-to-right simulation. Assume $(B_1, B_2) \in \mathcal{R}$ and $B_1 \stackrel{a}{\Longrightarrow} B_1'$; it follows that $B_1 = B \rightarrow a$, $B_2 = \delta_a(B)$ and $a \notin \checkmark(B)$. 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 $B \stackrel{\checkmark_b}{\Longrightarrow} B'$, and $B_2 \stackrel{\checkmark_b}{\Longrightarrow} B_2' = \delta_a(B')$. Proposition 5.1 implies $B \stackrel{\checkmark_b}{\Longrightarrow}$; thus $B' \stackrel{\checkmark_b}{\Longrightarrow}$ due to Proposition 5.5 and hence $a \notin \checkmark(B')$ 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 [59]):

$$\mathcal{R} = \{(B \triangleright a, a \triangleleft (B \downarrow a) + \delta_a(B)) \mid a \in \checkmark(B)\} \cup \sim$$

We show only left-to-right simulation. Assume $(B_1, B_2) \in \mathcal{R}$ and $B_1 \stackrel{\alpha}{\Longrightarrow} B_1'$; hence $B_1 = B \triangleright a$ and $B_2 = a \triangleleft (B \downarrow a) + \delta_a(B)$.

- If $\alpha \in Act$, then $\alpha = a$ and $B_1' = B' \cdot \mathbf{1}$, where $B \stackrel{\checkmark a}{\Longrightarrow} B'$. Due to Proposition 5.1, $B' \sim B \downarrow a$. It follows that $B_2 \stackrel{a}{\Longrightarrow} B_2' = \mathbf{1} \cdot (B \downarrow a)$. Using S1–S2, we obtain $B_1' \sim B_2'$ and hence $(B_1', B_2') \in \mathcal{R}$.
- If $\alpha = \checkmark_b$, then $a \; I \; b$ and $B_1' = B' \triangleright a$, where $B \stackrel{\checkmark b}{\Longrightarrow} B'$. If $B \downarrow a \stackrel{\checkmark a}{\Longrightarrow} B''$ then $B \stackrel{\checkmark b}{\Longrightarrow} \stackrel{\checkmark a}{\Longrightarrow} B''$ (by the operational semantics of residue) and hence (according to Proposition 5.5) $B' \stackrel{\checkmark a}{\Longrightarrow} B''$. According to Proposition 5.1, it follows that $a \in \checkmark(B')$ and $B'' \sim B' \downarrow a$. It follows that $B_2 \stackrel{\checkmark a}{\Longrightarrow} B_2' = a \triangleleft B'' + \delta_a(B')$, and (since bisimilarity is a congruence for left sequential and choice) $B_2' \sim B_2'' = a \triangleleft (B' \downarrow a) + \delta_A(B')$ with $(B_1', B_2'') \in \mathcal{R}$.

  If $B \downarrow a \stackrel{\checkmark b}{\Longrightarrow}$ then (according to Proposition 5.5) $B' \stackrel{\checkmark a}{\Longrightarrow}$; hence (according to Proposition 5.1) $a \notin \checkmark(B')$. It follows that $B_2 \stackrel{\checkmark a}{\Longrightarrow} B_2' = \delta_a(B')$, 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:

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

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 $B_1 \stackrel{a}{\Longrightarrow} B_1'$; hence $B + C \stackrel{\checkmark a}{\Longrightarrow} B_1'' \stackrel{a}{\Longrightarrow} B_1'$. We distinguish the following cases:

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

RD2 We show that the following is a bisimulation relation:

$$\mathcal{R} = \{(b \triangleleft B \downarrow a, b \triangleleft (B \downarrow a)) \mid a \; I \; b, a \notin \checkmark(B)\} \cup \sim$$

Let $B_1 = b \triangleleft B \downarrow a$ and $B_2 = b \triangleleft (B \downarrow a)$ with $a \; I \; b$ and $a \in \checkmark(B)$, and assume $B_1 \stackrel{a}{\Longrightarrow} B_1'$. Due to Proposition 5.1, it follows that $B \stackrel{\checkmark a}{\Longrightarrow} B'$ such that $B' \sim B \downarrow a$ and $b \triangleleft B' \stackrel{a}{\Longrightarrow} B_1'$.

- If $\alpha \in Act$ then $\alpha = b$, $B_1' = \mathbf{1} \cdot B'$ and $B_2 \stackrel{b}{\Longrightarrow} B_2' = \mathbf{1} \cdot (B \downarrow a)$. 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 $B' \stackrel{\checkmark b}{\Longrightarrow} B''$ such that $B_1' = b \triangleleft B''$. Moreover, $B \downarrow a \stackrel{\checkmark b}{\Longrightarrow} B''$ and hence $B_2 \stackrel{\checkmark b}{\Longrightarrow} b \triangleleft B'' = B_1'$. Since $(B_1', B_2') \in \mathcal{R}$, we are done.

5.3 Completeness

Our axiom system is 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). Completeness of the core language then comes down to the following property.

Proposition 5.7 *For all $B_1, B_2 \in \mathbf{L}_t$, $B_1 \sim B_1$ implies $T \vdash B_1 = B_2$.*

In order to prove this, we need a number of auxiliary results, the first of which is the following lemma:

Lemma 5.8 *$B + \mathbf{0}_A \sim B$ iff $B \stackrel{\surd_T}{\Rightarrow}$ for all $T \subseteq_{fin} [A]_I$.*

Proof. The following is a bisimulation relation:

$$\mathcal{R} = \{(B + \mathbf{0}_A, B) \mid \forall T \subseteq_{fin} [A]_I : B \stackrel{\surd_T}{\Rightarrow}\} \cup \sim$$

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

Another partial result is the following lemma, which states that the equational theory allows to discard termination constants from choice terms if they do not contribute to the behaviour.

Lemma 5.9 *Let $B \in \mathbf{L}_t$. If $B + \mathbf{0}_A \sim B$, then $T \vdash B + \mathbf{0}_A = B$.*

Proof. In fact, we show that $B + \mathbf{0}_A \sim B$ implies $T \vdash \delta_A(B) = \mathbf{0}_A$; this gives rise to the required result due to C5. First note that for arbitrary $B \in \mathbf{L}_t$, $T \vdash \delta_A(B) = \sum_{i \in I} \mathbf{0}_{A \cup A_i}$ (this is easily proved by induction on the structure of B). The lemma is proved by induction on the structure of B.

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

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

- $B = B_1 + B_2$. It follows that for either $i = 1$ or $i = 2$, $B_i \stackrel{\surd_T}{\Rightarrow}$ for all finite $T \subseteq [A]_I$; for otherwise, $B_1 \stackrel{\surd_{T_1}}{\Rightarrow}$ and $B_2 \stackrel{\surd_{T_2}}{\Rightarrow}$ for some $T_1, T_2 \subseteq_{fin} [A]_I$, which contradicts $B \stackrel{\surd_{T_1 \cup T_2}}{\Rightarrow}$. Hence (due to Lemma 5.8) $B_i + \mathbf{0}_A \sim B_i$ for $i = 1$ or $i = 2$. W.l.o.g. assume $i = 1$; since $T \vdash \delta_A(B_2) = \sum_{j \in J} \mathbf{0}_{A_j}$, using the induction hypothesis we can derive

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

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

- $B = a \leftarrow B_1$. Then $b \; I \; A$ implies $\mathbf{0}_A \stackrel{\surd_b}{\Rightarrow}$ and hence $B \stackrel{\surd_b}{\Rightarrow}$, thus $b \; I \; a$. It follows that $[A \cup \{a\}]_I = [A]_I$. Moreover, $B_1 \stackrel{\surd_T}{\Rightarrow}$ for all finite $T \subseteq [A]_I$, and hence (due to Lemma 5.8) $B_1 + \mathbf{0}_A \sim B_1$. Using the induction hypothesis, we can derive

  $$T \vdash \delta_A(B) = \delta_A(a) \leftarrow \delta_A(B_1) = \mathbf{0}_{A \cup \{a\}} \leftarrow \mathbf{0}_A = \delta_{A \cup \mathrm{det}\, a}(\mathbf{0}_A) = \mathbf{0}_{A \cup \{a\}} = \mathbf{0}_A \quad .$$

  This concludes the proof. □

Using D1–D3, it is not difficult to show that for all $B \in \mathbf{L}_t$, $T \vdash \delta_{Act}(B) = \mathbf{0}_{Act}$; 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 $I = \emptyset$. Thus, each term $B \in \mathbf{L}_t$ can be written in the form $\sum_{i \in I} a_i \leftarrow B_i + \sum_{j \in J} \mathbf{0}_{A_j}$, where $B_i \in \mathbf{L}_t$ for all $i \in I$. The completeness proof over $\mathbf{L}_t$ uses the *depth* of tree terms, which is defined by

$$depth(B) = \max\{1 + depth(B_i) \mid i \in I\} \ .$$

Proof of Proposition 5.7. Let $B_1, B_2 \in \mathbf{L}_t$ with $B_1 \sim B_2$ be given by

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

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

First we show that $T \vdash a_i \leftarrow B_i + B_2 = B_2$ for all $i \in I$. Due to $B_1 \sim B_2$ and $B_1 \stackrel{a_i}{\Rightarrow} \mathbf{1} \cdot B_i$, it follows that $B_2 \stackrel{a_i}{\Rightarrow} B_2'$ 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) $T \vdash B_i = B_k$, implying $T \vdash a_i \leftarrow B_i = a_k \leftarrow B_k$ and hence $T \vdash a_i \leftarrow B_i + B_2 = B_2$.

Now we show that $T \vdash \mathbf{0}_{A_j} + B_2 = B_2$ for all $j \in J$. Clearly, $B_1 \stackrel{\sqrt{T}}{\Rightarrow}$ and hence $B_2 \stackrel{\sqrt{T}}{\Rightarrow}$ for all finite $T \subseteq [A_j]_I$; by Lemma 5.8 this implies $B_2 + \mathbf{0}_{A_j} \sim B_2$. Lemma 5.9 then implies $T \vdash B_2 + \mathbf{0}_{A_j} = B_2$. □

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

Proposition 5.10 *Let $B_1, B_2 \in \mathbf{L}_t$ be arbitrary.*

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

Proof.

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

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

   Using S1, S5, RS5 and C5, we then have

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

Before proving Clause 2, we first provide two auxiliary properties:

4. $T \vdash \delta_A(B_1) = C$ for some $C \in \mathbf{L}_t$. This is proved by structural induction on B_1, using Axioms D1–D3. Note that $depth(C) = 0$.
5. $T \vdash B_1 \downarrow a = C$ for some $C \in \mathbf{L}_t$. This is proved by structural induction on B_1, using Axioms RD1–RD4 (where $\checkmark(x)$ in the side condition of RD2 and RD3 is defined since its argument B_1 does not contain residue operators). Note that $depth(C) \leq depth(B_1)$.

We are now ready to continue with Clause 2.

2. First note that with RS3 and RS4 and the auxiliary clauses above, we can rewrite every term $B_1 \triangleright a$ to a term $C \in \mathbf{L}_t$ with $depth(C) \leq 1 + depth(B_1)$ (where $\checkmark(x)$ in the side condition of RS3 and RS4 is defined, since its argument B_1 does not contain residue operators).

   Let $\mathbf{L}_{t,\delta}$ denote the fragment of $\mathbf{L}^+$ consisting of $\mathbf{L}_t$ plus the δ_A-operators; then according to Clause 4, any term of $\mathbf{L}_{t,\delta}$ can be rewritten to a term of $\mathbf{L}_t$. We now prove that $T \vdash B_1 \cdot B_2 = C$ for some $C_1 \in \mathbf{L}_{t,\delta}$, by induction on $depth(B_1) + depth(B_2)$; this implies the required property. Let

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

   By S5, LS1–LS3, RS1–RS2 and RS4–RS5, it follows that $T \vdash B_1 \cdot B_2 = C'$ with

$$C' = \sum_{i \in I} a_i \triangleleft (B_i \cdot B_2) + \sum_{j \in J} \delta_{A_j}(B_2) + \sum_{k \in K} (B_1 \triangleright a_k) \triangleleft B_k + \sum_{l \in L} \delta_{A_l}(B_1) \ .$$

   We saw above that for each $k \in K$, $T \vdash B_1 \triangleright a_k = B_{1,k}$ for some $B_{1,k} \in \mathbf{L}_t$ with $depth(B_{1,k}) \leq 1 + depth(B_1)$; say

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

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

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

   for all $k \in K$. By replacing the subterms of C' accordingly, we get $T \vdash C' = C''$ 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 $T \vdash C'' = C$ for some $C \in \mathbf{L}_{t,\delta}$; hence we are done.

3. Analogous to the previous clause. □

The main completeness result is now straightforward to prove.

Theorem 5.11 *For all $B_1, B_2 \in \mathbf{L}_{ff}$, $B_1 \sim B_2$ implies $T \vdash B_1 = B_2$.*

Proof. According to Proposition 5.10, there are terms $B_1', B_2' \in \mathbf{L}_t$ such that $T \vdash B_1 = B_1'$ and $T \vdash B_2 = B_2'$. Since all axioms of T are sound (Theorem 5.6), it follows that $B_1' \sim B_2'$; hence $T \vdash B_1' = B_2'$ according to Proposition 5.7. Combining these equalities, we obtain $T \vdash B_1 = B_2$. □

5.4 Axiomatising refinement

Next we proceed with the axiomatisation of refinement. What we need are equations which help us to get rid of all refinement operators in terms, so that every term can be rewritten into a normal form. Fortunately, the axioms for refinement are rather simple: see Table 8.

$a[r] = r(a)$		RF1
$\mathbf{0}_A[r] = \mathbf{0}_A$		RF2
$(x_1 \diamond x_2)[r] = x[r] \diamond y[r]$	where $\diamond \in \{+, \cdot, \triangleleft\}$	RF3

Table 8: Axioms for refinement

Theorem 5.12 *The axioms of Table 8 are sound w.r.t. bisimilarity.*

Proof.

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 $a[r] \stackrel{a}{\Longrightarrow} B'$. If $\alpha = b$, it follows that $r(a) \stackrel{b}{\Longrightarrow} C'$ and $B' = C' \cdot \mathbf{1}[r]$; hence $B' \sim C'$ by S2 and RF2. On the other hand, if $\alpha = \surd_b$ then $B' = a[r]$. Since $a \ I \ b$, by D-consistency of r it follows that $\mathcal{A}(r(a)) \ I \ b$, hence $r(a) \stackrel{\surd_b}{\Longrightarrow} r(a)$ by Proposition 5.4.

RF3 We only show the case for $\diamond = \cdot$; the others are straightforward. Let

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

$\mathcal{R} = \bigcup_{i \in \mathbb{N}} \mathcal{R}_i$ is is a bisimulation relation *up to* bisimilarity (see [59]). 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 $B \stackrel{a}{\Longrightarrow} B'$. There are two cases to consider.

- $B_1 \stackrel{b}{\Longrightarrow} B_1'$ and $r(b) \stackrel{a}{\Longrightarrow} B_0'$ such that $B' = B_0' \cdot (B_1' \cdot B_2)[r]$. We can then derive $C \stackrel{a}{\Longrightarrow} C' = (B_0' \cdot B_1'[r]) \cdot B_2[r]$; by S3, it follows that $C' \sim C'' = B_0' \cdot (B_1'[r] \cdot B_2[r])$ with $(B', C'') \in \mathcal{R}'$.
- $B_1 \stackrel{\surd_b}{\Longrightarrow} B_1'$, $B_2 \stackrel{b}{\Longrightarrow} B_2'$ and $r(b) \stackrel{a}{\Longrightarrow} B_0'$ such that $B' = B_0' \cdot (B_1' \cdot B_2')[r]$. We can then derive $C \stackrel{C'}{\Longrightarrow} = B_1'[r] \cdot (B_0' \cdot B_2'[r])$; by S3 and S4 (using the fact that $b \ I \ \mathcal{A}(B_1')$ according to Proposition 5.4, 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 5.4), 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 $B \stackrel{a}{\Longrightarrow} B'$. Again, there are two cases to consider.

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

We reuse the symbol T to denote the extended equational theory, generated by the axioms in Tables 7 and 8. The following theorem generalises Theorem 5.11 from $\mathbf{L}_{\mathit{ff}}$ to $\mathbf{L}_{\mathit{fin}}$.

Theorem 5.13 *For all $B_1, B_2 \in \mathbf{L}_{\mathit{fin}}$, $B_1 \sim B_2$ implies $T \vdash B_1 = B_2$.*

Proof. We merely have to show that for all $B \in \mathbf{L}_t$ and all $r = B_1/a_1, \ldots, B_n/a_n$ with $B_i \in \mathbf{L}_t$ for all $1 \leq i \leq n$, there is a $C \in \mathbf{L}_t$ such that $T \vdash B[r] = C$. This can be proved by induction on $\mathit{depth}(B) + \sum_{1 \leq i \leq n} \mathit{depth}(B_i)$. The interesting case is where $B = a \leftarrow B'$; then

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

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

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

which (latter) term can be rewritten to a term of $\mathbf{L}_t$ due to Proposition 5.10. □

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 [31], (advocated in [49]) 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 [16].

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 [40]. The protocol consists of three phases: *connection establishment*, *data transfer* and *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 Figure 1, 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 Figure 1. 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 Figure 1. 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

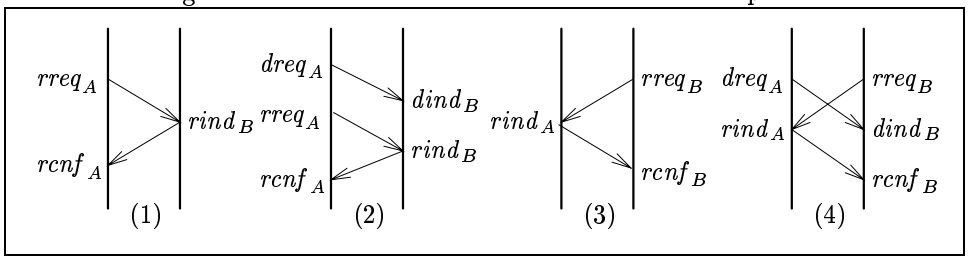
$$Prot' = Rel + dreq_A \cdot dind_A \cdot Rel \ .$$

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:

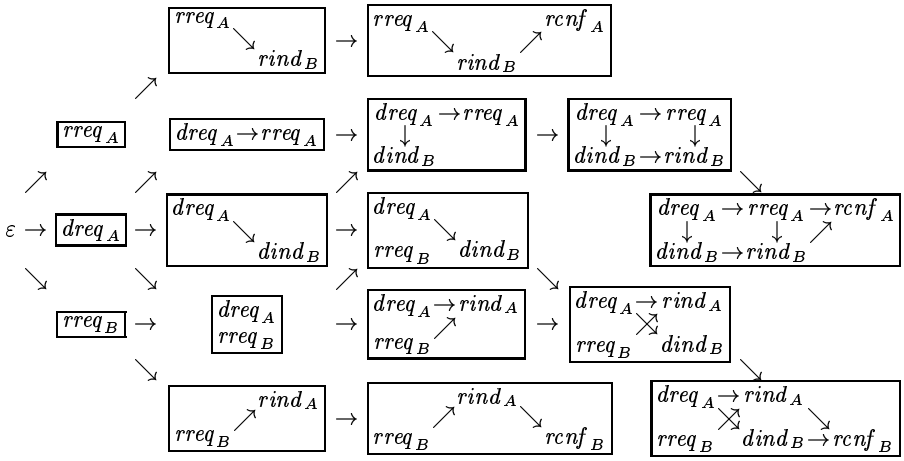
$$Prot' \xrightarrow{rreq_B} \mathbf{1} \cdot rind_A \cdot rcnf_B \ .$$

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 $Prot$.

Figure 1: Possible interactions of data and release phase



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

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

However, this is unfortunately not compatible with the requirement of strong D-consistency we have imposed on refinement functions (see Section 3.2): 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 $Rel \overset{rreq_B}{\not\Longrightarrow}$ and $Data \overset{\checkmark_{rreq_B}}{\Longrightarrow}$.

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 [31] and advocated for instance by Zwiers et al. in [47, 80, 32], 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):

$$\begin{pmatrix} B_1 \parallel_{A_1} C_1 \\ \cdot \\ B_2 \parallel_{A_2} C_2 \end{pmatrix} = \begin{pmatrix} B_1 \\ \cdot \\ B_2 \end{pmatrix} \parallel_{A_1 \cup A_2} \begin{pmatrix} C_1 \\ \cdot \\ C_2 \end{pmatrix} \ . \tag{11}$$

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 equation (11), 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, insofar they do not synchronise; i.e., the actions in $\mathcal{A}(B_i) \setminus A_i$ are independent of those in $\mathcal{A}(C_j) \setminus A_j$;
- different layers do not synchronise; i.e., $\mathcal{A}(B_i) \cap A_j = \mathcal{A}(C_i) \cap A_j = \emptyset$.

These conditions constitute the requirement of *communication closedness*, which is necessary for (11) 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 $i \neq j$*

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

then (11) holds up to strong bisimilarity.

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

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

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

Assume $D \stackrel{a}{\Longrightarrow} D'$. There are two cases to consider: $\alpha = \checkmark_a$ or $\alpha = a$. The first is the easier one: if $D \stackrel{\checkmark_a}{\Longrightarrow} D'$ then $B_i \stackrel{\checkmark_a}{\Longrightarrow} B_i'$ and $C_i \stackrel{\checkmark_a}{\Longrightarrow} C_i'$ for $i = 1, 2$ such that $D' = (B_1' \parallel_{A_1} C_1') \cdot (B_2' \parallel_{A_2} C_2')$. Then $E \stackrel{\checkmark_a}{\Longrightarrow} (B_1' \cdot B_2') \parallel_{A_1 \cup A_2} (C_1' \cdot C_2')$ 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: $D_1 \stackrel{a}{\Longrightarrow} D_1'$ such that $D' = D_1' \cdot D_2$. The following cases have to be considered:
  - $B_1 \stackrel{a}{\Longrightarrow} B_1'$ with $a \notin A_1$ and $D_1' = B_1' \parallel_{A_1} C_1$. Then (by Proposition 3.9.1) $a \in \mathcal{A}(B_1)$ and hence by communication closedness $a \notin A_2$. Therefore $E \stackrel{a}{\Longrightarrow} (B_1' \cdot B_2) \parallel_{A_1 \cup A_2} (C_1 \cdot C_2)$ and $(D', E') \in \mathcal{R}$.
  - $C_1 \stackrel{a}{\Longrightarrow} C_1'$ with $a \notin A_1$. Analogous to the previous case.
  - $B_1 \stackrel{a}{\Longrightarrow} B_1'$ and $C_1 \stackrel{a}{\Longrightarrow} C_1'$ and $a \in A_1$ and $D_1' = B_1' \parallel_{A_1} C_1'$. It follows that $E \stackrel{a}{\Longrightarrow} E' = (B_1' \cdot B_2) \parallel_{A_1 \cup A_2} (C_1' \cdot C_2)$ and $(D', E') \in \mathcal{R}$.
- The more interesting case is when the action a comes from the second layer: $D_1 \stackrel{\checkmark_a}{\Longrightarrow} D_1'$ and $D_2 \stackrel{a}{\Longrightarrow} D_2'$ such that $D' = D_1' \cdot D_2'$. First note that it follows that $B_1 \stackrel{\checkmark_a}{\Longrightarrow} B_1'$ and $C_1 \stackrel{\checkmark_a}{\Longrightarrow} C_1'$ with $D_1' = B_1' \parallel_{A_1} C_1'$. For the B_2-transition, we have the cases
  - $B_2 \stackrel{a}{\Longrightarrow} B_2'$ with $a \notin A_2$ and $D_2 = B_2' \parallel_{A_2} C_2$. It follows (by Proposition 3.9.1) 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 3.9.3) $C_1' = C_1$. Hence $E \stackrel{a}{\Longrightarrow} E' = (B_1' \cdot B_2') \parallel_{A_1 \cup A_2} (C_1' \cdot C_2)$ and $(D', E') \in \mathcal{R}$.
  - $C_2 \stackrel{a}{\Longrightarrow} C_2'$ with $a \notin A_2$. Analogous to the previous case.
  - $B_1 \stackrel{a}{\Longrightarrow} B_1'$ and $B_2 \stackrel{a}{\Longrightarrow} B_2'$ such that $a \in A$ and $D_2' = B_2' \parallel_{A_2} C_2'$. It follows that $E \stackrel{a}{\Longrightarrow} E' = (B_1' \cdot B_2') \parallel_{A_1 \cup A_2} (C_1' \cdot C_2')$ and $(D', E') \in \mathcal{R}$.

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

- $B \stackrel{a}{\Longrightarrow} B'$ with $a \notin A_1 \cup A_2$ such that $E' = B' \parallel_{A_1 \cup A_2} C$. We again must consider two cases:
  - $B_1 \stackrel{a}{\Longrightarrow} B_1'$ such that $B' = B_1' \cdot B_2$. Then $D \stackrel{a}{\Longrightarrow} D' = (B_1' \parallel_{A_1} C_1) \cdot (B_2 \parallel_{A_2} C_2)$ and $(D', E') \in \mathcal{R}$.
  - $B_1 \stackrel{\surd_a}{\Longrightarrow} B_1'$ and $B_2 \stackrel{a}{\Longrightarrow} B_2'$ such that $B' = B_1' \cdot B_2'$. Due to $a \in \mathcal{A}(B_2)$ (see Proposition 3.9.1), by communication closedness we again get $a\ I\ \mathcal{A}(C_1)$, hence (due to Proposition 3.9.3) $C_1 \stackrel{\surd_a}{\Longrightarrow} C_1$; thus $D \stackrel{a}{\Longrightarrow} D' = (B_1' \parallel_{A_1} C_1) \cdot (B_2' \parallel_{A_2} C_2)$ and $(D', E') \in \mathcal{R}$.
- $C \stackrel{a}{\Longrightarrow} C'$ with $a \notin A_1 \cup A_2$. Analogous to the previous case.
- $B \stackrel{a}{\Longrightarrow} B'$ and $C \stackrel{a}{\Longrightarrow} C'$ 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 $B_1 \stackrel{a}{\Longrightarrow} B_1'$ such that $B' = B_1' \cdot B_2$ and $C_1 \stackrel{a}{\Longrightarrow} C_1'$ such that $C = C_1' \cdot C_2$, implying $D \stackrel{a}{\Longrightarrow} D' = (B_1' \parallel_{A_1} C_1') \cdot (B_2 \parallel_{A_2} C_2)$ 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 $B_1 \stackrel{\surd_a}{\Longrightarrow} B_1'$ and $B_2 \stackrel{a}{\Longrightarrow} B_2'$ such that $B' = B_1' \cdot B_2'$, and $C_1 \stackrel{\surd_a}{\Longrightarrow} C_1'$ and $C_2 \stackrel{a}{\Longrightarrow} C_2'$ such that $C' = C_1' \cdot C_2'$. It follows that $D \stackrel{a}{\Longrightarrow} D' = (B_1' \parallel_{A_1} C_1') \cdot (B_2' \parallel_{A_2} C_2')$ and $(D', E') \in \mathcal{R}$. □

This 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 $i \neq j$*

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

then the following holds up to strong bisimilarity:

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

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

$$\begin{pmatrix} B_1 \parallel_{A_1} C_1 \\ \vdots \\ B_n \parallel_{A_n} C_n \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} B_1 \\ \vdots \\ B_{n-1} \end{pmatrix} \parallel_{A_1 \cup \ldots \cup A_{n-1}} \begin{pmatrix} C_1 \\ \vdots \\ C_{n-1} \end{pmatrix} \\ \cdot \\ B_n \parallel_{A_n} C_n \end{pmatrix} = \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix} \parallel_{A_1 \cup \ldots \cup A_n} \begin{pmatrix} C_1 \\ \vdots \\ C_n \end{pmatrix}$$

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:

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

Here, *prod* is an action of the sending party which *produces* data , the $dreq_i$ and $dind_i$ are data requests and indications as in Section 6.1, which convey the data over the medium from the sending to the receiving party, and *cons* an action of the receiving party which *consumes* the data. We assume that the produce and consume actions are independent of each other, and so are all requests and indications of different layers; that is, $prod\ I\ cons$ and $dreq_j\ I\ dreq_i\ I\ dind_j\ I\ dind_i$ for all $i \neq j$. Moreover, production and data indication, as well as data request and consumption, which take place at different parties, are independent as well: $prod\ I\ dind_i$ and $dreq_i\ I\ cn$ for all $1 \leq i \leq n$. Summarising, D is the reflexive and symmetric closure of the relation generated by

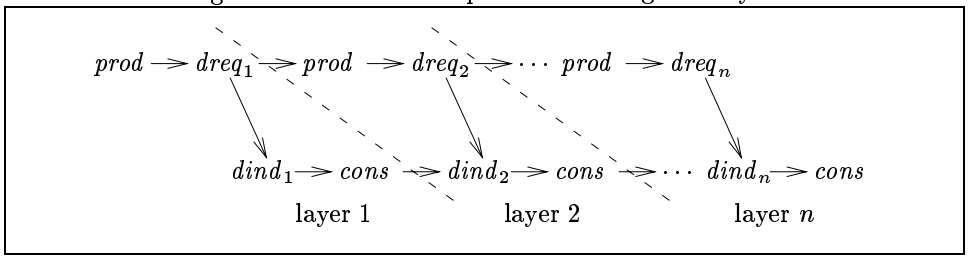
$$prod\ D\ dreq_i \qquad dreq_i\ D\ dind_i \qquad dind_i\ D\ cons\ .$$

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 indication of transfer $i+1$ can overlap with the data indication and consumption of transfer i. For instance, if $n = 2$ then

$$\begin{aligned} Data[r] &\xrightarrow{prod} (dreq_1 \cdot dind_1 \cdot cons) \cdot data_2[r] \\ &\xrightarrow{dreq_1} (dind_1 \cdot cons) \cdot data_2[r] \\ &\xrightarrow{prod} (dind_1 \cdot cons) \cdot (dreq_2 \cdot dind_2 \cdot cons) \cdot \mathbf{1}[r] \\ &\xrightarrow{dreq_2} (dind_1 \cdot cons) \cdot (dind_2 \cdot cons) \cdot \mathbf{1}[r] \end{aligned}$$

after which the first data transfer must proceed with $dind_1$. The denotational model of $Data[r]$ consists of a single maximal run and its prefixed, depicted in Figure 2. The dashed lines indicate the refinements of the individual data transfer actions.

Figure 2: Data transfer phase consisting of n layers



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

$$r'\colon data_i \mapsto ((prod \cdot dreq_i) \,|||\, (dind_i \cdot cons)) \,\|_{dreq_i, dind_i}\, (dreq_i \cdot dind_i) \quad (1 \leq i \leq n)\ .$$

$r(data_i)$ can be rewritten to $r'(data_i)$ using the axioms in Table 7, especially the expansion axioms for parallel composition (P, LM1–5 and CM1–6) and the axioms for choice (C1–5). For clarity, we introduce auxiliary names $Send_i = prod \cdot dreq_i$, $Rec_i = dind_i \cdot cons$, $Chan_i = dreq_i \cdot dind_i$ and $A_i = \{dreq_i, dind_i\}$; this allows us to write

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

Since $r(data_i) \sim r'(data_i)$ for all $1 \leq i \leq n$ and bisimulation is a congruence for refinement (Theorem 3.18), we may conclude

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

The refined phases $r'(data_i)$ are communication closed: for all $i \neq j$,

- $\mathcal{A}(Send_i \,|||\, Rec_i) \setminus A_i = \{prod, cons\}\ I\ \emptyset = \mathcal{A}(Chan_j) \setminus A_j$;
- $\mathcal{A}(Data_i) \cap A_j = \emptyset$.

Hence the conditions of Corollary 6.2 are fulfilled, implying

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

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

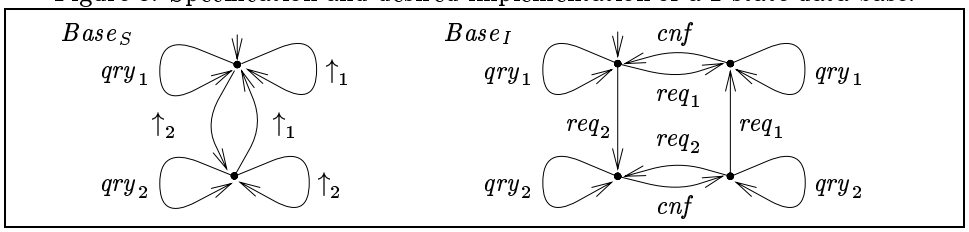
$$Data[r] \sim \left(\begin{pmatrix} Send_1 \\ \vdots \\ Send_n \end{pmatrix} ||| \begin{pmatrix} Rec_1 \\ \vdots \\ Rec_n \end{pmatrix} \right) \|_A \begin{pmatrix} Chan_1 \\ \vdots \\ Chan_n \end{pmatrix} \ .$$

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 [16]. 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. The data base specification is modelled by the transition system $Base_S$ in Figure 3. The problem considered in [16] is to change the interface of the

Figure 3: Specification and desired implementation of a 2-state data base.



data base, so that updating consists not of a single action but of two successive stages, in which the update is *requested* and *confirmed*, respectively. In our setting, this can be expressed by a refinement function

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

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 Figure 3.

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

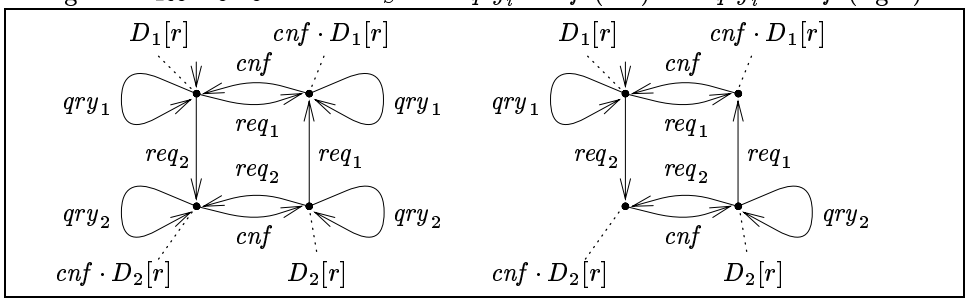
$$\begin{aligned} D_1 &= qry_1 \cdot D_1 + upd_1 \cdot D_1 + upd_2 \cdot D_2 \\ D_2 &= qry_2 \cdot D_2 + upd_2 \cdot D_2 + upd_1 \cdot D_1 \ . \end{aligned}$$

(Note that we have not explicitly introduced such recursive definitions in our language; however, it is well known that there is an equivalent term using the recursion operator for each specification using finitely many recursively defined variables.) The specification and implementation shown in Figure 3 is then obtained as the behaviour of

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

The operational behaviour of $Base_I$ is depicted (modulo Axiom S1 of Table 7, which states that $1 \cdot t = t$ for arbitrary t) by the left hand transition system in Figure 4. The right hand system of Figure 4 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.

Figure 4: Refinement of $Base_S$ with $qry_i\ I\ cnf$ (left) and $qry_i\ D\ cnf$ (right).



This examples shows, 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.

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 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 [35] 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 [53], and similar for High-level-MSCs in [54].

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 [44, 45], 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 [35, 53, 54].

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 [14] for an introduction or [46] 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 [64] 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 [47, 48], 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 [27, 68, 71]. A dependency-based sequential composition (in a linear time setting) can also be found in [34] (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 [30]), which we cannot fully discuss here. In particular, however, in this context a notion of termination similar to ours has been proposed in [29].

A process algebra with a notion of *location* which naturally induces dependencies can be found in [44, 45], 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 [15, 1]: there, locations play an entirely different role, where they are derived from the process terms rather than a priori associated with actions.)

Acknowledgement. Many thanks to Michel Reniers who pointed us to an unsound axiom in the earlier version [72].

References

[1] L. Aceto. A static view of localities. *Formal Aspects of Computing*, 6:201–222, 1994.

[2] L. Aceto, B. Bloom, and F. W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, May 1994. LICS '92 Special Issue.

[3] L. Aceto and M. C. B. Hennessy. Towards action-refinement in process algebras. *Information and Computation*, 103:204–269, 1993.

[4] L. Aceto and M. C. B. Hennessy. Adding action refinement to a finite process algebra. *Information and Computation*, 115:179–247, 1994.

[5] E. Badouel and P. Darondeau. On guarded recursion. *Theoretical Comput. Sci.*, 82:403–408, 1991.

[6] J. C. M. Baeten and F. W. Vaandrager. An Algebra for Process Creation. *Acta Inf.*, 29(4):303–334, 1992. Report version: CS-R8907, CWI, Amsterdam; also available in "J.W. de Bakker, 25 Jaar Semantiek — Liber Amicorum", Stichting Mathematisch Centrum, Amsterdam, 1989.

[7] J. C. M. Baeten and R. J. van Glabbeek. Abstraction and empty process in process algebra. *Fundamenta Informaticae*, XII:221–242, 1989.

[8] J. C. M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In Best [11], pages 477–492.

[9] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[10] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Comput. Sci.*, 37(1):77–121, 1985.

[11] E. Best, editor. *Concur '93*, volume 715 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[12] E. Best, R. Devillers, and J. Esparza. General refinement and recursion operators for the Petri box calculus. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS 93*, volume 665 of *Lecture Notes in Computer Science*, pages 130–140. Springer-Verlag, 1993.

[13] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, Jan. 1995.

[14] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[15] G. Boudol, I. Castellani, M. C. B. Hennessy, and A. Kiehn. A theory of processes with localities. *Formal Aspects of Computing*, 6:165–200, 1994.

[16] E. Brinksma, B. Jonsson, and F. Orava. Refining interfaces of communicating systems. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT '91, Volume 2*, volume 494 of *Lecture Notes in Computer Science*, pages 297–312. Springer-Verlag, 1991. Also available as Memoranda Informatica 91–19, TIOS 91/003, University of Twente, The Netherlands.

[17] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, July 1984.

[18] N. Busi, R. J. van Glabbeek, and R. Gorrieri. Axiomatising ST bisimulation equivalence. In Olderog [62], pages 169–188.

[19] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs. interleaving: An instructive example. *Bull. Eur. Ass. Theoret. Comput. Sci.*, 31:12–15, 1987. Note.

[20] W. R. Cleaveland, editor. *Concur '92*, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[21] P. Darondeau and P. Degano. Refinement of actions in event structures and causal trees. *Theoretical Comput. Sci.*, 118:21–48, 1993.

[22] J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[23] J. W. de Bakker and J. I. Zucker. Processes and the denotational semantics of concurrency. *Information and Computation*, 54:70–120, 1982.

[24] R. De Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Comput. Sci.*, 37:245–267, 1985.

[25] P. Degano, R. De Nicola, and U. Montanari. A partial ordering semantics for CCS. *Theoretical Comput. Sci.*, 75:223–262, 1991.

[26] P. Degano and R. Gorrieri. A causal operational semantics of action refinement. *Information and Computation*, 122:97–119, 1995.

[27] P. Degano, R. Gorrieri, and G. Rosolini. A categorical view of process refinement. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 138–153. Springer-Verlag, 1992.

[28] V. Diekert. On the concatenation of infinite traces. *Theoretical Comput. Sci.*, 113:35–54, 1993.

[29] V. Diekert and P. Gastin. A domain for concurrent termination: A generalization of Mazurkiewicz traces. In Z. Fülöp and F. Gécseg, editors, *Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 15–26. Springer-Verlag, 1995.

[30] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.

[31] T. Elrad and N. Francez. Decomposition of distributed programs into communication closed layers. *Science of Computer Programming*, 2, 1982.

[32] M. Fokkinga, M. Poel, and J. Zwiers. Modular completeness for communication closed layers. In Best [11], pages 50–65.

[33] W. Fokkink and C. Verhoef. A conservative look at operational semantics with variable binding. *Information and Computation*, 146(1):24–54, 1998.

[34] H. Gaifman. Modeling concurrency in partial orders and nonlinear transition systems. In de Bakker et al. [22], pages 467–488.

[35] T. Gehrke, M. Huhn, A. Rensink, and H. Wehrheim. An algebraic semantics for message sequence chart documents. In *Formal Description Techniques*. Chapman-Hall, 1998. Full report version: Hildesheimer Informatik-Bericht 5/98.

[36] R. J. van Glabbeek. The meaning of negative premises in transition system specifications II. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 502–513. Springer-Verlag, 1995. Full report version: STAN-CS-TN-95-16, Department of Computer Science, Stanford University.

[37] R. J. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems — Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer-Verlag, 1990. Report version: Arbeitspapiere der GMD 428.

[38] R. J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. Hildesheimer Informatik-Bericht 6/96, University of Hildesheim, 1998.

[39] U. Goltz, R. Gorrieri, and A. Rensink. Comparing syntactic and semantic action refinement. *Information and Computation*, 125(2):118–143, Mar. 1996.

[40] U. Goltz and N. Götz. Modelling a simple communication protocol in a language with action refinement. Draft version, 1991.

[41] J. F. Groote. Transition system specifications with negative premises. *Theoretical Comput. Sci.*, 118:263–299, 1993.

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

[43] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[44] M. Huhn. Action refinement and property inheritance in systems of sequential agents. In Montanari and Sassone [60], pages 639–654.

[45] M. Huhn. *On the Hierarchical Design of Distributed Systems.* PhD thesis, University of Hildesheim, 1997.

[46] 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, Feb. 1989. 1st Edition.

[47] W. Janssen, M. Poel, and J. Zwiers. Actions systems and action refinement in the development of parallel systems. In J. C. M. Baeten and J. F. Groote, editors, *Concur '91*, volume 527 of *Lecture Notes in Computer Science*, pages 298–316. Springer-Verlag, 1991.

[48] W. Janssen and J. Zwiers. From sequential layers to distributed processes. In *Principles of Distributed Computing.* ACM, 1992.

[49] W. Janssen and J. Zwiers. Protocol design by layered decomposition: A compositional approach. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, Berlin etc., 1992. Springer-Verlag.

[50] L. Jategaonkar and A. R. Meyer. Testing equivalences for Petri nets with action refinement. In Cleaveland [20], pages 17–31.

[51] B. Jonsson and J. Parrow, editors. *Concur '94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.

[52] R. Loogen and U. Goltz. Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informaticae*, XIV:39–73, 1991.

[53] S. Mauw and M. A. Reniers. Refinements in interworkings. In Montanari and Sassone [60], pages 671–686.

[54] S. Mauw and M. A. Reniers. High-level Message Sequence Charts. In *SDL' 97: Time for Testing — SDL, MSC and Trends.* North-Holland, 1997.

[55] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Report PB–78, Aarhus University, 1977.

[56] A. Mazurkiewicz. Traces, histories, graphs: instances of a process monoid. *Lecutre Notes in Computer Science*, 176:115–133, 1984.

[57] A. Mazurkiewicz. Basic notions of trace theory. In de Bakker et al. [22], pages 285–363.

[58] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[59] R. Milner and D. Sangiorgi. The problem of "weak bisimulation up to". In Cleaveland [20], pages 32–46.

[60] U. Montanari and V. Sassone, editors. *Concur '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[61] M. Nielsen, U. Engberg, and K. G. Larsen. Fully abstract models for a process language with refinement. In de Bakker et al. [22], pages 523–549.

[62] E.-R. Olderog, editor. *Programming Concepts, Methods and Calculi*, volume A–56 of *IFIP Transactions*. IFIP, 1994.

[63] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[64] D. A. Peled, V. R. Pratt, and G. J. Holzmann, editors. *Partial Order Methods in Verification*, volume 29 of *DIMACS series*. American Mathematical Society, 1997.

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

[66] A. Rensink. Posets for configurations! In Cleaveland [20], pages 269–285.

[67] A. Rensink. *Models and Methods for Action Refinement*. PhD thesis, University of Twente, Enschede, Netherlands, Aug. 1993.

[68] A. Rensink. Methodological aspects of action refinement. In Olderog [62], pages 227–246.

[69] A. Rensink. An event-based SOS for a language with refinement. In J. Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing, pages 294–309. Springer-Verlag, 1995.

[70] A. Rensink. Bisimilarity of open terms. In C. Palamidessi and J. Parrow, editors, *Expressiveness in Concurrency*, volume 7 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1997. Full report version: Hildesheimer Informatik-Bericht 5/97, University of Hildesheim, May 1997.

[71] A. Rensink and R. Gorrieri. Vertical bisimulation. Hildesheimer Informatik-Bericht 9/98, University of Hildesheim, June 1998. Improved version of the TAPSOFT '97 paper "Action Refinement as an Implementation Relation".

[72] A. Rensink and H. Wehrheim. Weak sequential composition in process algebras. In Jonsson and Parrow [51], pages 226–241.

[73] A. Rensink and H. Wehrheim. Dependency-based action refinement. In I. Prívara and P. Ruzicka, editors, *Mathematical Foundations of Computer Science 1997*, volume 1295 of *Lecture Notes in Computer Science*, pages 468–477. Springer-Verlag, 1997.

[74] F. W. Vaandrager. Expressiveness results for process algebras. Report CS–R9301, Centre for Mathematics and Computer Science, 1993.

[75] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In Jonsson and Parrow [51], pages 433–448.

[76] W. Vogler. Failures semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.

[77] W. Vogler. Bisimulation and action refinement. *Theoretical Comput. Sci.*, 114:173–200, 1993.

[78] H. Wehrheim. Parametric action refinement. In Olderog [62], pages 247–266. Full report version: Hildesheimer Informatik-Berichte 18/93, Institut für Informatik, University of Hildesheim, Nov. 1993.

[79] G. Winskel. An introduction to event structures. In de Bakker et al. [22], pages 364–397.

[80] J. Zwiers. Layering and action refinement for timed systems. In J. W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.