Weak Sequential Composition in Process Algebras

Arend Rensink and Heike Wehrheim*

Institut für Informatik, University of Hildesheim
Postfach 101363, D–31113 Hildesheim, Germany
{rensink,wehrheim}@informatik.uni-hildesheim.de

Appeared in: B. Jonsson and J. Parrow (Eds.), Concur '94: Concurrency Theory, LNCS 836, Springer-Verlag, 1994, pp. 226–241

Abstract. In this paper we study a special operator for sequential composition, which is defined relative to a *dependency relation* over the actions of a given system. The idea is that actions which are *not* dependent (intuitively because they share no common resources) do not have to wait for one another to proceed, even if they are composed sequentially. Such a notion has been studied before in a linear-time setting, but until recently there has been no systematic investigation in the context of process algebras.
We give a structural operational semantics for a process algebraic language containing such a sequential composition operator, which shows some interesting interplay with choice. We give a complete axiomatisation of strong bisimilarity and we show consistency of the operational semantics with an event-based denotational semantics developed recently by the second author. The axiom system allows to derive the *communication closed layers law*, which in the linear time setting has been shown to be a very useful instrument in correctness preserving transformations. We conclude with a couple of examples.

1 Introduction

We are interested in the subject of sequential versus concurrent behaviour in process algebra. In the usual interleaving semantics, two actions that are specified as occurring in parallel will be modelled as occurring in either of the two possible orders; the parallelism is deemed unobservable and hence not explicitly modelled. On the other hand, if an ordering is specified between two actions then it is usually assumed that this ordering will actually be realised in practice, in other words the actions will indeed occur in the specified order. The first assumption has been the subject of much debate, and in fact a whole branch of computer science dealing with non-standard, *partial order* semantics has been developed as a result of dropping this assumption and modelling parallelism more faithfully. The second assumption, however, has hardly been questioned. Yet there are actually some arguments against it. If one postulates an inherent notion of *dependency* among the actions performed by a system, then one can imagine that only *dependent* actions will actually be executed in the specified order, whereas *independent* actions can be performed in either order

* Research partially supported by the HCM Cooperation Network "EXPRESS" (Expressiveness of Languages for Concurrency), the Esprit Basic Research Working Group 6067 (CALIBAN) and a Graduiertenförderungsstipendium of the University of Hildesheim

even if they are actually composed in sequence. For instance, in compiler optimisation, if neither of two sequentially composed assignment statements depends on the other then a compiler is free to reorder them. There is a similar connection to *serialisability* in data bases.

The idea of a dependency relation over the actions of a system can already be found in *trace theory* as developed by Mazurkiewicz [14]. Zwiers et al. have also exploited this idea in [13, 21, 10]. In both cases however, the models used are *linear-time*, which is to say that the points in time at which choices are made are not represented in the model. We aim at extending this idea to branching-time semantics, in particular to strong bisimulation. In this effort we are guided by an existing partial-order denotational model developed by one of the authors (Wehrheim [20]). Other partial-order models in which an explicit notion of (in)dependency plays a role are e.g. Shields, [17], Bednarczyk [5], Stark [18], but there dependency is defined on the level of *events*, i.e., *occurrences* of actions, rather than actions themselves. The resulting concept is much more concrete than the one we present here.

We postulate a dependency relation over the actions and develop an operational semantics (Section 2) based on a *weak* notion of sequential composition, which takes dependency into account. The resulting semantics has some surprising features. In particular, the occurrence of an action may resolve choices that are in some sense in the "future" of the system. In Section 3, our semantics is shown to adhere to a well-studied format for SOS rules, the so-called *GSOS format* (cf. [7]). As a consequence we can apply existing (meta-level) theory to derive that strong bisimilarity is a congruence. We also develop a complete axiomatisation for bisimilarity. In Section 4, we show consistency of our operational semantics with the partial-order denotational semantics of [20] mentioned above. (Historically we *started out* with the denotational model, and the operational semantics was developed as a justification of it.) Most of the features that give rise to complications in the operational semantics are completely natural in the denotational model. In Section 5 we discuss some examples where the notion of weak sequential composition is used to good advantage. Among others, we recapture the *communication closed layers law* advocated in the work of Zwiers et al., extended to take synchronisation into account. Finally, Section 6 contains some concluding remarks.

For lack of space, all proofs have been omitted.

2 Language and Operational Semantics

We assume a global set of actions Act with a reflexive and symmetric relation $D \subseteq Act \times Act$ called *dependency*. The inverse notion of *independency* is defined by $a\ I\ b$ iff $\neg(a\ D\ b)$. The *dependency class* of a given action is denoted $[a]_D := \{\, b \mid b\ D\ a \,\}$, extended to $[A]_D := \bigcup_{a \in A} [a]_D$; similarly $[a]_I := \{\, b \mid b\ I\ a \,\}$ and $[A]_I := \bigcap_{a \in A} [a]_I$. The language $\mathbf{L}$ studied in this paper is generated by the following grammar:

$$B ::= \mathbf{0}_P \mid a \mid B + B \mid B \cdot B \mid B \parallel_A B$$

where $a \in Act$ and $P, A \subseteq Act$. We will also use the *alphabet* $\alpha(B)$ of a term B, recursively defined as follows:

$$\alpha(\mathbf{0}_P) := \varnothing$$

$$\begin{aligned} \alpha(a) &:= \{a\} \quad \text{where } a \in Act \\ \alpha(B * C) &:= \alpha(B) \cup \alpha(C) \quad \text{where } * \in \{+, \cdot, \|_A\} \end{aligned}$$

The operators of **L** have been taken from existing languages (CCS [15], CSP [12], ACP [4]) but some of them will get a non-standard interpretation. The basic new idea is the effect of dependency on sequential composition: in our semantics, independent actions never have to wait for one another to proceed even if they are sequentially composed. Actions from the second operand C of a term $B \cdot C$ are able to "overtake" B if they are independent of B. We will call such actions *permissible according to* B. Even if B is "deadlocked" in the sense of not being able to perform any action itself, it may still permit actions of C. Note that if all actions are dependent, our notion of sequential composition reduces to the standard one. The index P in the deadlock constants $\mathbf{0}_P$ explicitly represents the permissible actions, i.e. the actions for which $\mathbf{0}_P$ acts like successful termination rather than proper deadlock. We use auxiliary notations $\mathbf{0} = \mathbf{0}_\varnothing$ (no actions are permitted; complete deadlock) and $\mathbf{1} = \mathbf{0}_{Act}$ (all actions are permitted; complete termination).

As regards the rest of the language: the term a executes a and then terminates successfully. $B + C$ denotes the *choice* between B and C, which can not only be resolved in the usual way, by the first action of B or C, but also by actions of processes that (sequentially) follow the choice. For instance, if $a \ D \ c$ and $a \ I \ b$ then $(a + b) \cdot c$ denotes a process that either executes a or b and afterwards c (as usual) or can start with c after which the choice between a and b is resolved and only b is left to be performed. The family of operators $\{\|_A\}_{A \subseteq Act}$ stand for TCSP-like *parallel composition* with synchronisation on actions of A, with the additional requirement (important in the partial order semantics) that dependent actions of parallel components have to be executed in a nondeterministically chosen order rather than (truly) concurrently.

We now formalise these intuitions operationally. First consider sequential composition. Examples of operational rules for normal sequential composition are the following from Baeten and Weijland [4] (for a detailed discussion see [3]):

$$\frac{B \xrightarrow{a} B'}{B \cdot C \xrightarrow{a} B' \cdot C} \qquad \frac{B \xrightarrow{a} \surd}{B \cdot C \xrightarrow{a} C}$$

where $B \xrightarrow{a} \surd$ denotes that B can terminate successfully by executing a. The rules state that either B has not terminated yet, in which case the sequential composition can only execute actions of B, or B terminates and afterwards C starts. In contrast to this, our weak sequential composition allows execution of actions of C if these actions are independent of B. In a first attempt to capture this operationally, instead of the second rule above we propose

$$\frac{C \xrightarrow{a} C' \quad a \in [\alpha(B)]_I}{B \cdot C \xrightarrow{a} B \cdot C'}$$

This works satisfactorily with terms like $a \cdot b$ where $a \ I \ b$: we can derive $a \cdot b \xrightarrow{b} a \cdot \mathbf{1}$. However if $a \ D \ c$ and $b \ I \ c$ then $c \in [\alpha(a+b)]_D$, hence the above rule would not allow to derive the desired transition $(a+b) \cdot c \xrightarrow{c} b$. We see that the effect of dependencies is more subtle than allowed for by the above rule. In particular, the first operand

may actually change as a consequence of "permitting" actions. To capture this effect we define a transition-like *permission* relation $B \stackrel{a}{\cdots\!\!\rightarrow} B'$, expressing that B permits a and changes into B'. The rules for this relation are given in Table 1.

Table 1. Permission relation

action	$\dfrac{a \; I \; b}{b \stackrel{a}{\cdots\!\!\rightarrow} b}$	deadlock $\dfrac{a \in P}{\mathbf{0}_P \stackrel{a}{\cdots\!\!\rightarrow} \mathbf{0}_P}$
choice	$\dfrac{B \stackrel{a}{\cdots\!\!\rightarrow} B' \quad C \stackrel{a}{\cdots\!\!\not\rightarrow}}{B + C \stackrel{a}{\cdots\!\!\rightarrow} B'}$ $\dfrac{C \stackrel{a}{\cdots\!\!\rightarrow} C' \quad B \stackrel{a}{\cdots\!\!\not\rightarrow}}{B + C \stackrel{a}{\cdots\!\!\rightarrow} C'}$ $\dfrac{B \stackrel{a}{\cdots\!\!\rightarrow} B' \quad C \stackrel{a}{\cdots\!\!\rightarrow} C'}{B + C \stackrel{a}{\cdots\!\!\rightarrow} B' + C'}$	
sequential composition	$\dfrac{B \stackrel{a}{\cdots\!\!\rightarrow} B' \quad C \stackrel{a}{\cdots\!\!\rightarrow} C'}{B \cdot C \stackrel{a}{\cdots\!\!\rightarrow} B' \cdot C'}$	parallel composition $\dfrac{B \stackrel{a}{\cdots\!\!\rightarrow} B' \quad C \stackrel{a}{\cdots\!\!\rightarrow} C'}{B \parallel_A C \stackrel{a}{\cdots\!\!\rightarrow} B' \parallel_A C'}$

The most interesting permission rules are those for choice: if only one of the operands permits an action a then the choice can thereby be resolved. The sequential and parallel composition of processes only permit actions if both components do. The following property reflects some of the intuitions behind the permission relation:

Proposition 1. $B \stackrel{a}{\cdots\!\!\rightarrow} B' \Rightarrow a \in [\alpha(B')]_I$.

Ordinary transitions are defined in Table 2. Note especially the second rule of sequential composition, which states that $B \cdot C$ can execute actions from C if they are permitted by B. It is now straightforward to derive the transition $(a + b) \cdot c \stackrel{c}{\rightarrow} b \cdot \mathbf{1}$ discussed above (where $a \; D \; c \; I \; b$). We define *transition-permission systems* as the natural extension of labelled transition systems to our setting.

Table 2. Transition relation

action $\dfrac{}{a \stackrel{a}{\rightarrow} \mathbf{1}}$	choice $\dfrac{B \stackrel{a}{\rightarrow} B'}{B + C \stackrel{a}{\rightarrow} B'}$ $\dfrac{C \stackrel{a}{\rightarrow} B'}{B + C \stackrel{a}{\rightarrow} B'}$	
sequential composition $\dfrac{B \stackrel{a}{\rightarrow} B'}{B \cdot C \stackrel{a}{\rightarrow} B' \cdot C}$	$\dfrac{B \stackrel{a}{\cdots\!\!\rightarrow} B' \quad C \stackrel{a}{\rightarrow} C'}{B \cdot C \stackrel{a}{\rightarrow} B' \cdot C'}$	
parallel composition $\dfrac{B \stackrel{a}{\rightarrow} B' \quad a \notin A}{B \parallel_A C \stackrel{a}{\rightarrow} B' \parallel_A C}$	$\dfrac{C \stackrel{a}{\rightarrow} C' \quad a \notin A}{B \parallel_A C \stackrel{a}{\rightarrow} B \parallel_A C'}$	$\dfrac{B \stackrel{a}{\rightarrow} B' \; C \stackrel{a}{\rightarrow} C' \quad a \in A}{B \parallel_A C \stackrel{a}{\rightarrow} B' \parallel_A C'}$

Definition 2 (transition-permission system). For a term $B \in \mathbf{L}$, the *transition-permission system* of B, $tps(B)$, is defined by $\langle Act, \mathbf{L}, \rightarrow, \cdots\!\!\rightarrow, B \rangle$.

3 Axiomatisation of Bisimilarity

To interpret the operational semantics we define an equivalence relation over tps's. Two terms are then regarded to describe the same behaviour if the tps's generated by the operational semantics are equivalent. The equivalence relation we choose for this purpose is the standard (strong) *bisimilarity*.

Definition 3 (bisimilarity). Let $T_i = \langle Act, S_i, \rightarrow_i, \cdot\!\cdot\!\rightarrow_i, q_i \rangle$ be tps's for $i = 1, 2$. A *bisimulation relation* is a binary relation $\rho \subseteq S_1 \times S_2$ such that $q_1 \ \rho \ q_2$ and whenever $s_1 \ \rho \ s_2$ [resp. $s_2 \ \rho \ s_1$] then

1. $s_1 \xrightarrow{a} s_1'$ implies $s_2 \xrightarrow{a} s_2'$ for some $s_2' \ \rho^{-1} \ s_1'$ [resp. $s_2' \ \rho \ s_1'$];
2. $s_1 \stackrel{a}{\cdots\!\rightarrow} s_1'$ implies $s_2 \stackrel{a}{\cdots\!\rightarrow} s_2'$ for some $s_2' \ \rho^{-1} \ s_1'$ [resp. $s_2' \ \rho \ s_1'$].

If a bisimulation relation exists, we call T_1 and T_2 *bisimilar*, denoted $T_1 \sim T_2$.

This notion is lifted to terms as usual: $B \sim C$ iff $tps(B) \sim tps(C)$. We establish that bisimilarity is a congruence with respect to the operators of our language. For this purpose, rather than giving a direct proof we derive the result from existing meta-theory. In the past few years we have seen the development of theory relating the *format* of SOS rules to properties of the resulting operational semantics. Typically, the kind of property proved in this way is the congruence of certain equivalence relations with respect to operations defined by the SOS rules; in particular, this is done for strong bisimilarity in the seminal paper by Bloom, Israel and Meyer [7]. In order to apply this general theory, we have to reinterpret permissions $\stackrel{a}{\cdots\!\rightarrow}$ as transitions with special labels, e.g., $\xrightarrow{\hat{a}}$, where for all $a \in Act$, $\hat{a}$ is a new label not in Act. Hence $\xrightarrow{\lambda}$ denotes a "proper" transition if $\lambda \in Act$ and a permission if $\lambda = \hat{a}$ for some $a \in Act$. We state without proof that with this modification, the SOS rules in Tables 1 and 2 all satisfy the GSOS format defined in [7]. Hence the following is a direct consequence of [7, Theorem 5].

Theorem 4 (congruence). $\sim$ *is a congruence over* $\mathbf{L}$.

Now for an axiomatisation of bisimilarity. The unusual behaviour of weak sequential composition forces some modifications to the standard axioms for bisimilarity. For instance, the axiomatisation of ACP [6] contains the rule $(x + y)z = xz + yz$ (where juxtaposition is sequential composition). For weak sequential composition however, this is not valid: for instance, if $a \ I \ c \ I \ b$ then $(a + b) \cdot c \xrightarrow{c} (a + b) \cdot \mathbf{1}$ which can still do both a and b; however, if $a \cdot c + b \cdot c \xrightarrow{c} B$ then either $B = a \cdot \mathbf{1}$ or $B = b \cdot \mathbf{1}$, neither of which can do both a and b. It follows that $(a + b) \cdot c \not\sim a \cdot c + b \cdot c$.

Aceto, Bloom and Vaandrager [1] have developed a general method for deriving complete axiomatisations for strong bisimilarity directly from GSOS rules. Unfortunately, it turns out that this part of the existing SOS meta-theory is not directly applicable to our system in its current form. One problem lies in the fact that although our language can only describe finite behaviour, still in a technical sense it allows infinite computations to be specified: for instance, if $a \ I \ b$ then $b \xrightarrow{\hat{a}} b \xrightarrow{\hat{a}} \cdots$. This means that the technique of [1] fails to induce a normal form. Nevertheless, a complete axiomatisation does exist, as we show below. Unfortunately, to obtain normal forms we need auxiliary operators not in $\mathbf{L}$. In particular, we introduce a new family of *prefix operators* for all $a \in Act$, which we denote by juxtaposition, i.e. aB prefixes B with $a \in Act$. The operational semantics of prefix is given in Table 3. Note in particular the difference between the prefix aB and the sequential composition $a \cdot B$, for instance if $B \xrightarrow{b} B'$ where $a \ I \ b$: the latter then allows $a \cdot B \xrightarrow{b} a \cdot B'$ which cannot be matched by aB.

Apart from prefix, Table 3 defines auxiliary operators needed to axiomatise sequential composition and synchronisation. Unlike prefix, these other auxiliaries appear only temporarily and can always be removed by rewriting. $\between_A$ is the standard

Table 3. Operational semantics of auxiliary operators

prefix	$\dfrac{}{aB \xrightarrow{a} B}$	$\dfrac{a\ I\ b \quad B \overset{b}{\cdots\!\!\rightarrow} B'}{aB \overset{b}{\cdots\!\!\rightarrow} aB'}$
left sequential	$\dfrac{B \xrightarrow{a} B'}{B \leftharpoondown C \xrightarrow{a} B' \cdot C}$	$\dfrac{B \overset{a}{\cdots\!\!\rightarrow} B' \quad C \overset{a}{\cdots\!\!\rightarrow} C'}{B \leftharpoondown C \overset{a}{\cdots\!\!\rightarrow} B' \leftharpoondown C'}$
right sequential	$\dfrac{B \overset{a}{\cdots\!\!\rightarrow} B' \quad C \xrightarrow{a} C'}{B \rightharpoondown C \xrightarrow{a} B' \cdot C'}$	$\dfrac{B \overset{a}{\cdots\!\!\rightarrow} B' \quad C \overset{a}{\cdots\!\!\rightarrow} C'}{B \rightharpoondown C \overset{a}{\cdots\!\!\rightarrow} B' \rightharpoondown C'}$
left merge	$\dfrac{B \xrightarrow{a} B' \quad a \notin A}{B \between_A C \xrightarrow{a} B' \between_A C}$	$\dfrac{B \overset{a}{\cdots\!\!\rightarrow} B' \quad C \overset{a}{\cdots\!\!\rightarrow} C'}{B \between_A C \overset{a}{\cdots\!\!\rightarrow} B' \between_A C'}$
communication merge	$\dfrac{B \xrightarrow{a} B' \quad C \xrightarrow{a} C' \quad a \in A}{B \mid_A C \xrightarrow{a} B' \between_A C'}$	$\dfrac{B \overset{a}{\cdots\!\!\rightarrow} B' \quad C \overset{a}{\cdots\!\!\rightarrow} C'}{B \mid_A C \overset{a}{\cdots\!\!\rightarrow} B' \mid_A C'}$

left merge from ACP, adapted to our notion of synchronisation and extended to deal with permissions. $|_A$ is the relevant version of the *communication merge*. $\leftharpoondown$ and $\rightharpoondown$, called *left sequential* and *right sequential*, serve a similar purpose with respect to sequential composition as left and communication merge with respect to synchronisation: in $B \leftharpoondown C$, intuitively the first action comes from B whereas in $B \rightharpoondown C$ it should come from C.

The language obtained by extending $\mathbf{L}$ with the above auxiliary operators is denoted $\mathbf{L}^+$. Before we can give the axiomatisation of $\mathbf{L}^+$ we still need some more machinery in the form of functions over $\mathbf{L}^+$. First of all, $\pi\colon \mathbf{L}^+ \to \mathbf{2}^{Act}$ returns the *permissible actions* of a term, intuitively those actions for which a permission relation can be deduced. Second, we define the *residue* of a term after permitting a

Table 4. Permissible actions and residue

$\pi(\mathbf{0}_P) := P$	$res_a(\mathbf{0}_P) := \mathbf{0}_P$
$\pi(a) := [a]_I$	$res_a(b) := b$
$\pi(B + C) := \pi(B) \cup \pi(C)$	$res_a(B + C) := \begin{cases} res_a(B) + res_a(C) & \text{if } a \in \pi(B) \cap \pi(C) \\ res_a(B) & \text{if } a \in \pi(B) \setminus \pi(C) \\ res_a(C) & \text{if } a \in \pi(C) \setminus \pi(B) \end{cases}$
$\pi(B * C) := \pi(B) \cap \pi(C)$	$res_a(B * C) := res_a(B) * res_a(C) \quad (* \in \{\cdot, \leftharpoondown, \rightharpoondown, \between_A, \mathbin{\mathbb{L}}_A, \mid_A\})$
$\pi(aB) := [a]_I \cap \pi(B)$	$res_a(bB) := b\ res_a(B)$

given action, as a family of functions $res_a\colon \mathbf{L}^+ \to \mathbf{L}^+$ for every $a \in Act$. Intuitively, if a is permitted by B then $res_a(B)$ corresponds to the remainder of B after that permission, i.e., $B \overset{a}{\cdots\!\!\rightarrow} res_a(B)$. The residue can unfortunately not be dealt with by adding it as yet another auxiliary operator to $\mathbf{L}^+$, basically because it cannot be captured operationally. The following lemma formalises the intuitions underlying π and res. It is proved by induction on the structure of B.

Lemma 5. *For all $B \in \mathbf{L}^+$, $B \overset{a}{\cdots\!\!\rightarrow} B'$ if and only if $a \in \pi(B)$ and $B' = res_a(B)$.*

Finally, we come to the axiomatisation of $\mathbf{L}^+$. It is given in Table 5.

The equations for choice, C1–4, form an important part of our equational theory, in that over the sublanguage consisting of just deadlock, prefix and choice, all

Table 5. Axioms for bisimulation

$x + y = y + x$	C1	$x \parallel_A y = x \between_A y + y \between_A x + x \mid_A y$	P
$x + (y + z) = (x + y) + z$	C2	$(x + y) \between_A z = (x \between_A z) + (y \between_A z)$	PL1
$x + x = x$	C3	$ax \between_{A \setminus \{a\}} y = a(x \parallel_A y)$	PL2
$x + \mathbf{0}_{\pi(x) \cap P} = x$	C4	$ax \between_{A \cup \{a\}} y = \mathbf{0}_{\pi(ax) \cap \pi(y)}$	PL3
$a = a\mathbf{1}$	A	$\mathbf{0}_P \between_A x = \mathbf{0}_{P \cap \pi(x)}$	PL4
$x \cdot y = x \leftharpoonup y + x \rightharpoonup y$	S	$x \mid_A y = y \mid_A x$	PC1
$(x + y) \leftharpoonup z = (x \leftharpoonup z) + (y \leftharpoonup z)$	SL1	$(x + y) \mid_A z = x \mid_A z + y \mid_A z$	PC2
$ax \leftharpoonup y = a(x \cdot y)$	SL2	$ax \mid_{A \cup \{a\}} ay = a(x \parallel_A y)$	PC3
$\mathbf{0}_P \leftharpoonup x = \mathbf{0}_{P \cap \pi(x)}$	SL3	$ax \mid_{A \cup \{a\}} by = \mathbf{0}_{\pi(ax) \cap \pi(by)}$ if $a \neq b$	PC4
$x \rightharpoonup (y + z) = x \rightharpoonup y + x \rightharpoonup z$	SR1	$ax \mid_{A \setminus \{a\}} y = \mathbf{0}_{\pi(ax) \cap \pi(y)}$	PC5
$x \rightharpoonup ay = a(res_a(x) \cdot y)$ if $a \in \pi(x)$	SR2	$\mathbf{0}_P \mid_A x = \mathbf{0}_{P \cap \pi(x)}$	PC6
$x \rightharpoonup ay = \mathbf{0}_{\pi(x) \cap \pi(ay)}$ if $a \notin \pi(x)$	SR3		
$x \rightharpoonup \mathbf{0}_P = \mathbf{0}_{\pi(x) \cap P}$	SR4		

bisimilarities can be proved using these equations only. This is stated in the following theorem, which slightly extends the standard result in that we have a family of deadlock constants $\mathbf{0}_P$ instead of just a single one (corresponding to our $\mathbf{0}_\varnothing$—note that if we restrict $\mathbf{L}$ to $\mathbf{0}_\varnothing$ then indeed we cannot derive any permissions any more).

Theorem 6. *If $\mathbf{L}_t^+$ (for* tree language *denotes the fragment of $\mathbf{L}^+$ consisting of deadlock, prefix and choice, then C1–4 are complete for bisimilarity in $\mathbf{L}_t^+$.*

The other equations in our system basically allow to reduce every term to this fragment $\mathbf{L}_t^+$; the above completeness result then carries over to the entire language. The interesting operator is once more weak sequential composition, axiomatised in S–SR4. We follow the standard technique (cf. [1]) of splitting the operator into the two auxiliary ones $\leftharpoonup$ and $\rightharpoonup$ introduced above. The former is relatively easy to capture equationally. Note in particular that here we do have the distributivity over choice discussed at the beginning of this section. Right sequential is more complex: especially, if the second operand is a prefix term then we have to distinguish whether or not the first operand *permits* the prefixed action (SR2 and 3); and as the first operand may change as a consequence of this permission, we also need the *residue*. It is here, therefore, that we need the functions π and *res*.

Let T^+ denote the theory in Table 5. We first need to show that all the equations of T^+ are sound modulo bisimilarity, and then that they induce normal forms which are terms of $\mathbf{L}_t^+$. Together with Theorem 6, this establishes completeness. For the soundness proof we need that π and *res* are well-defined modulo the given equations.

Proposition 7. *If $B = C$ is an instance of one of the equations in Table 5, then on the one hand, $\pi(B) = \pi(C)$, and on the other, $res_a(B) = res_a(C)$ is an instance of the same equation.*

Now we state the required soundness property.

Theorem 8 (soundness). *If $T^+ \vdash B = C$ then $B \sim C$.*

Next, we show that all terms of $\mathbf{L}^+$ can be rewritten to *normal forms* in $\mathbf{L}_t^+$.

Theorem 9 (normalisation). *If $C \in \mathbf{L}^+$ then $T^+ \vdash B = C$ for some $B \in \mathbf{L}_t^+$.*

The final completeness result just collects the previous theorems.

Corollary 10 (completeness). *For all $B, C \in \mathbf{L}^+$, $B \sim C$ iff $T^+ \vdash B = C$.*

In order to work with the proof system in practise, one would need to prove a lot of auxiliary equations first. At this point we merely mention that sequential composition is associative and that $\mathbf{1} \cdot B = B$ for all $B \in \mathbf{L}$. This allows us, respectively, to write series of sequential compositions without parentheses and to get rid of spurious left-over $\mathbf{1}$'s in many derivations; for instance, if $a \; D \; b \; D \; c$ then $a \cdot b \cdot c \xrightarrow{a} \mathbf{1} \cdot b \cdot c = b \cdot c \xrightarrow{b} \mathbf{1} \cdot c = c$, instead of arriving at either $(\mathbf{1} \cdot \mathbf{1}) \cdot c$ or $\mathbf{1} \cdot (\mathbf{1} \cdot c)$.

4 Denotational Semantics

Besides the operational semantics we will also define a denotational semantics for the language $\mathbf{L}$ which is shown to be consistent with the operational semantics. More precisely, we will show that the transition-permission system of a term B derived via the operational semantics is bisimilar to the transition-permission system obtained from the denotational semantics of B. The model we use for this purpose was introduced in Wehrheim [20], where it is discussed and motivated in detail.

The models are sets of partial runs, implicitly ordered by prefix. We will use a global set of events $\mathbf{E}$, assumed to be closed under pairing: $(\mathbf{E} \cup \{*\}) \times (\mathbf{E} \cup \{*\}) \subseteq \mathbf{E}$, where $* \notin \mathbf{E}$ is a special symbol. A *directed acyclic graph* is an ordered subset of $\mathbf{E}$ where the (reflexive and cycle-free) ordering represents the *causal relation* between events. Runs are represented by *labelled dags with permission sets*, or *P-dags* for short, where the dag part supplies information about the *past* behaviour up to a certain point, and the permission set consists of all actions that are independent of the *future* behaviour.

Definition 11 (labelled P-dag). A *labelled P-dag* is a tuple $u = \langle E, \leq, l, P \rangle$ where
- $E \subseteq \mathbf{E}$ is a set of *events*;
- $\leq \subseteq E \times E$ is a reflexive and cycle-free *flow relation*;
- $l : E \to Act$ is a *labelling function*, and
- $P \subseteq Act$ is a *permission set* (P-set).

We use u, v, w to range over P-dags, and E_u, $\leq_u$ etc. to denote the components of a P-dag u. The labelling function extends the dependency relation to events: $e \; D_u \; e'$ iff $l_u(e) \; D \; l_u(e')$. u is called *D-compatible* if precisely all dependent events are ordered, i.e., $\leq_u \cup \geq_u = D_u$. The set of all D-compatible P-dags is denoted $\mathbf{PD}_D$. The P-dags we use in our semantics will always be D-compatible. If $f : E_u \to E$ is an injective function, then $f(u) := \langle f(E_u), f(\leq_u), l_u \circ f^{-1}, P \rangle$ defines the image of u under f. Two dags u and v are said to be *dag-equal* (denoted $u =_{dg} v$) if $\langle E_u, \leq_u, l_u \rangle = \langle E_v, \leq_v, l_v \rangle$, and *compatibly labelled* iff $l_u|_{E_u \cap E_v} = l_v|_{E_u \cap E_v}$.

Now we define some P-dag constants and operators. The following constants will be used to represent the maximal runs of the processes $\mathbf{0}_P$ and a:

$$\begin{aligned} \varepsilon_P &:= \langle \varnothing, \varnothing, \varnothing, P \rangle \\ e_a &:= \langle \{e\}, \{(e,e)\}, \{(e,a)\}, Act \rangle \ . \end{aligned}$$

The operators to be considered are union and weak sequential composition of P-dags. Union is only defined for compatibly labelled P-dags and sequential composition is only defined if the event sets are disjoint and the first operand permits the events of the second to happen. The latter idea is captured by the notion of *enabling*: we say that u *enables* v (denoted $u \vdash v$) if $E_u \cap E_v = \varnothing$ and $P_u \supseteq l_v(E_v)$.

$$\begin{aligned} u \cup v &:= \langle E_u \cup E_v, \leq_u \cup \leq_v, l_u \cup l_v, P_u \cap P_v \rangle \quad \text{if } l_u|_{E_u \cap E_v} = l_v|_{E_u \cap E_v} \\ u \cdot v &:= \langle E_u \cup E_v, \leq_u \cup \leq_v \cup ((E_u \times E_v) \cap D), l_u \cup l_v, P_u \cap P_v \rangle \quad \text{if } u \vdash v. \end{aligned}$$

Finally, we define a *smoothening* and a *prefix relation* over P-dags, as follows:

$$\begin{aligned} u \sqsubseteq v &:\Leftrightarrow E_u = E_v \wedge \leq_u \subseteq \leq_v \wedge l_u = l_v \wedge P_u = P_v \\ u \preccurlyeq v &:\Leftrightarrow E_u \subseteq E_v \wedge \leq_u = \leq_v \cap (E_v \times E_u) \wedge \ell_u = \ell_v|_{E_u} \wedge P_u = P_v \setminus [l(E_v \setminus E_u)]_D \end{aligned}$$

Intuitively, $u \sqsubseteq v$ states that v augments u with some additional ordering (for instance to make it D-compatible), whereas $u \preccurlyeq v$ states that u is a sub-behaviour of v, that is, the computation can be carried on after u and may evolve into v.

Definition 12 (P-dag structure). A *P-dag structure* is a non-empty prefix closed set of compatibly labelled P-dags. The set of P-dag structures is denoted **PDS**.

The above operators are now lifted to P-dag structures, and used to model the operators of **L**. To model the behaviour of terms $B \parallel_A C$, we rely on *label-preserving bijections* from the A-labelled events of the left hand operand to the A-labelled events from the right hand operand. Such bijections establish which events synchronise with one another. If u and v are P-dags then $[u \rightarrowtail_A v]$ denotes the space of label-preserving bijections from $\{ e \in E_u \mid l_u(e) \in A \}$ to $\{ e \in E_v \mid l_v(e) \in A \}$; if $f : u \rightarrowtail_A v$ then the following functions map the events from u resp. v to their synchronisations:

$$\hat{f}_1 : e \mapsto \begin{cases} (e, f(e)) & \text{if } f(e) \text{ is defined} \\ (e, *) & \text{otherwise} \end{cases} \qquad \hat{f}_2 : e \mapsto \begin{cases} (f^{-1}(e), e) & \text{if } f^{-1}(e) \text{ is defined} \\ (*, e) & \text{otherwise.} \end{cases}$$

As a consequence, synchronising u and v is a matter of finding an $f : u \rightarrowtail_A v$ and constructing $\hat{f}_1(u) \cup \hat{f}_2(v)$. Unfortunately, in general this is not D-compatible; the ordering has to be augmented. In the end therefore, we model the synchronisation of u and v according to f by all $w \in \mathbf{PD}_D$ such that $\hat{f}_1(u) \cup \hat{f}_2(v) \sqsubseteq w$.

The denotational semantics of **L** is now given by the function $[\![\cdot]\!] : \mathbf{L} \times \mathbf{E} \to \mathbf{PDS}$ inductively defined in Table 6. The second argument of the function is only used to ensure disjointness of sets of events.

Consider again $B = (a+b) \cdot c$ where $a \ I \ b \ I \ c$ and $a \ D \ c$. This yields the following P-dag structure (where $\rightarrow$ denotes prefix, and we have left out the events, which in this case does not matter since there is only one occurrence of every action):

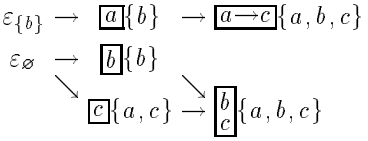


Table 6. P-dag structure semantics for **L**

$$
\begin{array}{rcl}
[\![\mathbf{0}_P]\!]_e & := & \{\varepsilon_P\} \\
[\![a]\!]_e & := & \{\varepsilon_{[a]_I}, e_a\} \\
[\![B + C]\!]_e & := & [\![B]\!]_{(e,*)} \cup [\![C]\!]_{(*,e)} \\
[\![B \cdot C]\!]_e & := & \{\, u \cdot v \mid u \in [\![B]\!]_{(e,*)}, v \in [\![C]\!]_{(*,e)}, u \vdash v \,\} \\
[\![B \parallel_A C]\!]_e & := & \{\, w \in \mathbf{PD}_D \mid u \in [\![B]\!]_e, v \in [\![C]\!]_e, f\colon u \rightharpoonup_A v, \hat{f}_1(u) \cup \hat{f}_2(v) \sqsubseteq w \,\}
\end{array}
$$

To relate this to the operational semantics, we have to define a notion of *state* over the denotational model, and introduce *transitions* and *permissions* over them. Here we encounter the interesting situation that the P-dags themselves are not suitable to represent states. For instance, in the example above we see that the P-dag structure does not have a smallest element, and there is no natural notion of initial state. Instead, we use *nonempty sets of dag-equal P-dags* as states: if $\mathcal{P} \in \mathbf{PDS}$ then

$$
\begin{array}{rcl}
s \xrightarrow{a}_{\mathcal{P}} s' & :\Leftrightarrow & \exists e \in \mathbf{E}.\, s' = \{\, u \in \mathcal{P} \mid \exists v \in s.\, v \preccurlyeq u \wedge E_u = E_v \cup \{e\} \wedge l_u(e) = a \,\} \\
s \cdots\!\!\rightarrow^{a}_{\mathcal{P}} s' & :\Leftrightarrow & s' = \{\, u \in s \mid a \in P_u \,\}
\end{array}
$$

For the above example, this yields the following transitions (permissions are ignored):

$$
\begin{array}{rcccl}
\{\varepsilon_{\{b\}}\} & \rightarrow & \{\boxed{a}\{b\}\} & \rightarrow & \{\boxed{a \rightarrow c}\{a,b,c\}\} \\
\{\varepsilon_{\{b\}}, \varepsilon_{\varnothing}\} & \rightarrow & \{\boxed{b}\{b\}\} & & \\
\{\varepsilon_{\varnothing}\} & \rightarrow & \{\boxed{c}\{a,c\}\} & \rightarrow & \{\boxed{\begin{array}{c} b \\ c \end{array}}\{a,b,c\}\}
\end{array}
$$

In general, therefore, the states of $\mathcal{P}$ are subsets of $[u]_{=_{dg}} \cap \mathcal{P}$; a natural initial state of a given P-dag structure $\mathcal{P}$ is then $[\varepsilon_{\varnothing}]_{=_{dg}} \cap \mathcal{P}$. Hence for a P-dag structure $\mathcal{P}$, the transition-permission system of $\mathcal{P}$ is defined by

$$
tps(\mathcal{P}) := \langle Act, \{\, s \mid \exists u.\, \varnothing \subset s \subseteq [u]_{=_{dg}} \cap \mathcal{P} \,\}, \rightarrow_{\mathcal{P}}, \cdots\!\!\rightarrow_{\mathcal{P}}, [\varepsilon_{\varnothing}]_{=_{dg}} \cap \mathcal{P} \rangle \ .
$$

The next theorem states the consistency of operational and denotational semantics.

Theorem 13. *For all $B \in \mathbf{L}$ and $e \in \mathbf{E}$, $tps(B) \sim tps([\![B]\!]_e)$.*

5 Examples

We discuss some small examples from the world of protocols, in which our notion of weak sequential composition and the interaction with choice are essential.

5.1 Connection Release Phase

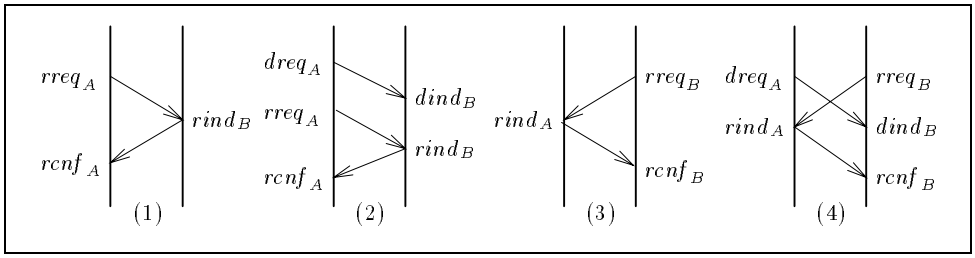
We consider a small protocol for connection-oriented data transfer between two parties. The example is inspired by Goltz and Götz [11]. The protocol consists of three phases: *connection establishment*, *data transfer* and *connection release*. Here we concern ourselves only with the interaction between the data transfer and release

phases. We start by a specification of the form $Prot = Data \cdot Rel$, which reflects the idea that after connection release, no data can be transferred any more. However, it can in general not be ruled out that *some* actions from the data phase take place only after the release phase has started. 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. The release phase, on the other hand, can be initiated by either A or B by a *release request* $rreq_A$ or $rreq_B$, is indicated at the other end by a *release indication* $rind_B$ or $rind_A$, and confirmed by a *release confirm* $rcnf_A$ or $rcnf_B$. The corresponding processes are specified as follows:

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

(We have modelled just one possible data transfer; in the next example we will see a somewhat more involved data phase.) The four possible interactions are depicted in Fig. 1 below. 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.

Fig. 1. Possible interactions of data and release phase



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 a data indication coming 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

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

which corresponds to scenario (2) or (4) of Fig. 1. Which of these two is chosen depends on who initiates the release. Note that both $rreq_A$ and $rreq_B$ are already enabled in $dind_B \cdot Rel$. In fact, $dind_B$ can be delayed even further:

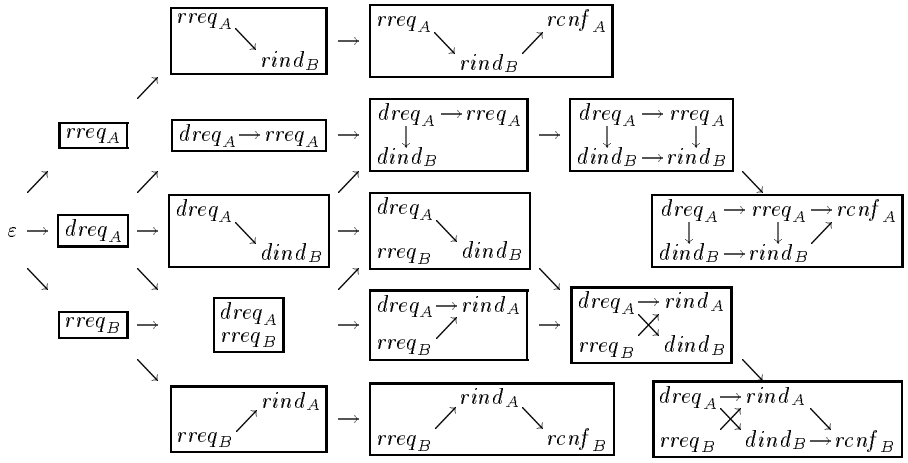
$$dind_B \cdot Rel \xrightarrow{rreq_B} dind_B \cdot rind_A \cdot rcnf_B \xrightarrow{rind_A} dind_B \cdot rcnf_B$$

At this stage, however, the data indication must take place. On the other hand, if the first action of $Prot$ is the release request from B, then (because $res_a(Data) = Data$) the choice in the data phase is not resolved by this and up to bisimilarity we get

$$Prot \xrightarrow{rreq_B} Data \cdot rind_A$$

corresponding to scenario (3) or (4) in Fig. 1. The next action will decide between these two possibilities: it is either $dreq_A$ or $rind_A$, the latter of which *does* decide the choice in $Data$, i.e., $Data \cdot rind_A \xrightarrow{rind_A} \mathbf{1}$.

Note in particular that the non-right-distributivity of choice over (weak) sequential composition is important here: the alternative protocol $Rel + dreq_A \cdot dind_A \cdot Rel$, obtained by distributing Rel over the choice in $Data$, is different from $Prot$, since in this new protocol, an initial $rreq_B$-action resolves the choice and $dreq_A$ may be refused afterwards. The denotational model of $Prot$ looks as follows (where we have left out the permission sets and hence the states collapse to their underlying dags):



5.2 Communication Closed Layers

An algebraic law that has been applied quite successfully in a linear time setting is the *communication closed layers* law (CCL), advocated for instance by Zwiers et al. in [13, 21, 10]. As mentioned in the introduction, this law has actually been one of the motivations for the present work. In our setting, CCL can be formulated as follows:

$$\begin{pmatrix} (B_1 \parallel_{A_1} C_1) \\ \cdot \\ (B_2 \parallel_{A_2} C_2) \end{pmatrix} = \begin{pmatrix} B_1 \\ \cdot \\ B_2 \end{pmatrix} \parallel_{A_1 \cup A_2} \begin{pmatrix} C_1 \\ \cdot \\ C_2 \end{pmatrix} \quad . \tag{1}$$

Each pair B_i, C_i is thought to form a *layer* of an ongoing algorithm or protocol, in which information is exchanged between the components B_i and C_i using two different mechanisms: interference due to dependencies between actions, and synchronisation over A_i. Both kinds of interference are however ruled out between components of *different* layers: if $i \neq j$ then

- different components of different layers are mutually independent; i.e., $\alpha(B_i) \setminus A_i$ is independent of $\alpha(C_j) \setminus A_j$;
- different layers do not synchronise; i.e., $\alpha(B_i) \cap A_j = \varnothing$ and $\alpha(C_i) \cap A_j = \varnothing$.

These conditions constitute the requirement of *communication closedness*, which is necessary for (1) 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.

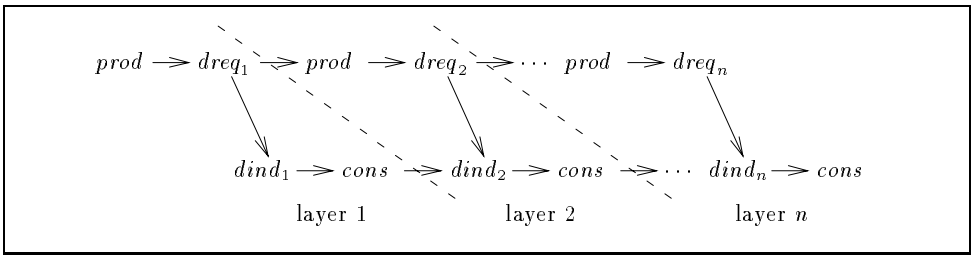
Theorem 14 (CCL). *If $B_i, C_i \in \mathbf{L}$ and $A_i \subseteq Act$ for $i = 1, 2$ and for all $i \neq j$, $(\alpha(B_i) \setminus A_i)\ I\ (\alpha(C_j) \setminus A_j)$ and $\alpha(B_i) \cap A_j = \alpha(C_i) \cap A_j = \varnothing$, then*

$$(B_1 \parallel_{A_1} C_1) \cdot (B_2 \parallel_{A_2} C_2) = (B_1 \cdot B_2) \parallel_{A_1 \cup A_2} (C_1 \cdot C_2)$$

An interesting special case is $B \cdot C = B \parallel_\varnothing C = C \cdot B$, which holds if $\alpha(B)\ I\ \alpha(C)$ (obtained by setting $B_1 = B$, $B_2 = C_1 = \mathbf{1}$, $C_2 = C$ and $A_1 = A_2 = \varnothing$). By induction one can extend Theorem 14 to $m > 1$ layers of $n > 1$ components each.

Here we show an application of CCL. Consider a data phase consisting of $n \geq 1$ data transfers, each specified in a "logical layer" $Data_n$. The specification of the data phase is $Data = Data_1 \cdots Data_n$, where $Data_i = prod \cdot dreq_i \cdot dind_i \cdot cons$ for $1 \leq i \leq n$. The $dreq_i$ and $dind_i$ are data transfer requests and indications as before (which are always in the same direction), *prod* is an action at the sending party which *produces* data and *cons* an action at the receiving party which *consumes* them. The produce and consume actions and data actions of different layers are independent; that is, $prod\ I\ cons$ and $dreq_j\ I\ dreq_i\ I\ dind_j\ I\ dind_i$ for all $i \neq j$. The overall behaviour of *Data* is depicted in Fig. 2. Note that without the *prod*- and *cons*-actions, the different data phases would be completely independent, which is not the kind of behaviour we want to specify.

Fig. 2. Data transfer phase consisting of n layers



Now we want to transform this specification to 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 do this to the separate data layers:

$$Data'_i = ((prod \cdot dreq_i) \parallel_\varnothing (dind_i \cdot cons)) \parallel_{dreq_i,\, dind_i} (dreq_i \cdot dind_i) \ .$$

It is easy to see that $Data_i$ can be rewritten to $Data'_i$ using the equations P–PC6 and C4. Now we introduce auxiliary names $Send_i = prod \cdot dreq_i$, $Rec_i = dind_i \cdot cons$ and $Chan_i = dreq_i \cdot dind_i$, allowing us to write

$$Data = ((Send_1 \parallel_\varnothing Rec_1) \parallel_{dreq_1,\, dind_1} Chan_1) \cdots ((Send_n \parallel_\varnothing Rec_n) \parallel_{dreq_n,\, dind_n} Chan_n) \ .$$

If $i \neq j$ then one the one hand, $\alpha(Send_i \parallel_\varnothing Rec_i) \setminus \{dreq_i, dind_i\} = \{prod, cons\}\ I\ \varnothing = \alpha(Chan_j) \setminus \{dreq_j, dind_j\}$, and on the other, $\alpha(Data_i) \cap \{dreq_j, dind_j\} = \varnothing$. Hence the conditions of CCL are fulfilled, implying

$$Data = \begin{pmatrix} (Send_1 \parallel_\varnothing Rec_1) \\ \vdots \\ (Send_n \parallel_\varnothing Rec_n) \end{pmatrix} \parallel_A \begin{pmatrix} Chan_1 \\ \vdots \\ Chan_n \end{pmatrix}$$

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

$$Data = \left(\begin{pmatrix} Send_1 \\ \vdots \\ Send_n \end{pmatrix} \|_{\varnothing} \begin{pmatrix} Rec_1 \\ \vdots \\ Rec_n \end{pmatrix} \right) \|_{A} \begin{pmatrix} Chan_1 \\ \vdots \\ Chan_n \end{pmatrix}$$

This is indeed the structure we were aiming at: there are now clearly recognisable subterms describing the behaviour of sender, receiver and channel.

6 Conclusions

The work reported here is part of an ongoing project in which the connection between such concepts as causality and action refinement is investigated. In this paper we have succeeded in "downgrading" the notion of causality to the operational idea that an action may "overtake" any term of which it is independent, even if it is specified as taking place only after that term, resulting in a weakened notion of sequential composition, the consequences of which we have discussed in some detail.

In the literature, this idea has so far been considered primarily in a linear time setting; see especially Mazurkiewicz [14]. In the process algebraic setting of this paper it has some surprising consequences, especially for the combination of sequential composition and choice. In this respect, we have investigated only one of a number of possible alternatives, driven by considerations based on the existing partial-order model of Wehrheim [20]. In a sense, the operational semantics forms the "projection" of a partial-order denotational semantics to an interleaving setting, retaining, however, some independence information that is more commonly associated with non-interleaving models. When we set out, it was not clear that this could be done at all; and in fact, many of the aspects that are unusual in the operational, interleaving point of view, such as the moment at which choices are resolved, are completely natural in the denotational, partial order model.

Note that the linear time model of Janssen, Poel and Zwiers [13] and Fokkinga, Poel and Zwiers [10], which is used as denotational model for a language similar to ours (especially also including weak sequential composition), does not lend itself easily to a compositional operational characterisation, since deadlocking runs are simply thrown out (for instance, they have the equivalent of $B \cdot \mathbf{0}_{\varnothing} = \mathbf{0}_{\varnothing}$).

Since the rules of our operational semantics fit into the GSOS format known from the literature, we could immediately use existing SOS theory to show that the operational semantics defines a unique transition relation—which is not self-evident, given the fact that it features negative premises—and that (strong) bisimilarity is a congruence for all the operators of our language. Since we had in no way worked towards that result, we regard this as an interesting "proof of the pudding," both for the SOS theory and for our own semantics. We have moreover given a complete axiomatisation of the language with respect to bisimilarity. In this respect however, the existing SOS theory, although yielding useful hints, was not directly applicable.

The usefulness of the notion of weak sequential composition has been demonstrated by two small examples from the area of protocol design. The first of these

shows that in some situations, the interplay with choice we have specified is exactly what one wants. As part of the second example, we have extended the *communication closed layers law*, known from the linear time setting, to our language. This example does not involve the choice operator (although it does contain synchronisation and in that sense goes beyond the usual linear time applications); we conjecture that to put CCL to maximal benefit in the context of process algebra, it should be generalised somehow to include choices between layers.

The language we have considered in this paper may be changed or extended in several ways. The interplay between weak sequential composition and choice, which is the central issue of this paper, can conceivably be simplified – although we stress once more that our approach is very natural in the denotational model. In fact the peculiarities (if they should be called such) of the operational characterisation might as well be attributed to the interleaving nature of this characterisation, raising the immediate question if some partial-order operational semantics would not be more appropriate. Another suggestion (thanks to one of the referees) is that the problems we envisage to solve with weak sequential composition could alternatively be tackled with *prioritised parallel composition*. This is an interesting subject for study, although it is perhaps questionable if any simplification could be obtained this way, the interplay of priority and choice itself being a very nontrivial issue.

Possible language extensions to be investigated are: ordinary ("strong") sequential composition, renaming, recursion and action refinement. Adding strong sequential composition would have the advantage that action prefix is no longer an auxiliary operator. (Note that we can simulate strong sequential composition using a total dependency relation, but since this relation is global we would then lose the weak version.) Adding renaming is straightforward. In the denotational semantics, adding recursion is also straightforward (using standard fixpoint techniques). Hence once more we have a measuring stick for the operational case. The operational characterisation however turns out to be problematic, mainly due to the fact that we cannot define *guardedness* in the usual way. Usually process variables are said to be guarded if they are in the scope of a prefixing operator or, more generally, if they only occur on *sleeping positions* (Vaandrager [19]), corresponding to operands which are not tested by the rules of the operational semantics. However, our rules for sequential composition test both arguments and in fact there are no operators at all in our language which have a sleeping position. This will be the subject of further research. Finally, we plan to investigate the consistent extension of dependencies to action refinement. This has already been done in the denotational model (see [20]). The operational characterisation will probably not be easier than for the "ordinary" case without dependencies; it remains to be seen if the existing techniques (see e.g. Aceto and Hennessy [2], Degano and Gorrieri [9], Rensink [16]) are applicable.

Acknowledgement. Thanks are due to Frits Vaandrager for some very helpful suggestions regarding the operational semantics.

References

1. L. Aceto, B. Bloom, and F. Vaandrager. Turning SOS rules into equations. In *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 113–124. IEEE, Com-

puter Society Press, 1992. Full version available as CWI Report CS-R9218, June 1992, Amsterdam. To appear in the LICS 92 Special Issue of Information and Computation.
2. L. Aceto and M. Hennessy. Towards action-refinement in process algebras. *Information and Computation*, 103:204–269, 1993.
3. J. C. M. Baeten and F. W. Vaandrager. An algebra for process creation. In *J.W. de Bakker, 25 Jaar Semantiek — Liber Amicorum*. Stichting Mathematisch Centrum, Amsterdam, Apr. 1989. Also availabe as: Report CS-R8907, CWI, Amsterdam.
4. J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.
5. M. A. Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, University of Sussex, Oct. 1987. Available as Report 1/88, School of Cognitive and Computing Sciences, University of Sussex.
6. J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Comput. Sci.*, 37(1):77–121, 1985.
7. B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. In *Fifteenth Annual Symposium on the Principles of Programming Languages*, pages 229–239. ACM, 1988. Preliminary Report.
8. 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.
9. P. Degano and R. Gorrieri. An operational definition of action refinement. Technical Report TR–28/92, Università di Pisa, 1992. To appear in Information and Computation.
10. M. Fokkinga, M. Poel, and J. Zwiers. Modular completeness for communication closed layers. In E. Best, editor, *Concur '93*, volume 715 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 1992.
11. U. Goltz and N. Götz. Modelling a simple communication protocol in a language with action refinement. Draft version, 1991.
12. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
13. 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.
14. A. Mazurkiewicz. Basic notions of trace theory. In de Bakker et al. [8], pages 285–363.
15. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
16. A. Rensink. *Models and Methods for Action Refinement*. PhD thesis, University of Twente, Enschede, Netherlands, Aug. 1993.
17. M. W. Shields. Concurrent machines. *The Computer Journal*, 28(5):449–465, 1985.
18. E. W. Stark. Concurrent transition systems. *Theoretical Comput. Sci.*, 64:221–269, 1989.
19. F. W. Vaandrager. Expressiveness results for process algebras. Report CS–R9301, Centre for Mathematics and Computer Science, 1993. Available by ftp: ftp.cwi.nl, pub/CWIreports/AP.
20. H. Wehrheim. Parametric action refinement. Hildesheimer Informatik-Berichte 18/93, Institut für Informatik, Universität Hildesheim, Nov. 1993. To be presented at PROCOMET '94, San Miniato, June 1994.
21. 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.

This article was processed using the LaTeX macro package with LLNCS style