The Expressive Power of Parallelism*

Joachim Parrow

Swedish Institute of Computer Science
Box 1263, S-164 28 Kista, Sweden

December 11, 1990

Abstract

We explore an algebraic language for networks consisting of a fixed number of reactive units, communicating synchronously over a fixed linking structure. The language has only two operators: *disjoint parallelism*, where two networks are composed in parallel without any interconnections, and *linking*, where an interconnection is formed between two ports. The intention is that these operators correspond to the primitive steps when constructing networks, and that they therefore are conceptually simpler than the operators in existing process algebras. We investigate the expressive power of our language. The results are: (1) Definability of behaviours: with only three simple processing units, every finite-state behaviour can be constructed. (2) Definability of operators: we characterise the network operators which are definable within the language; these turn out to include most operators previously suggested for describing parallelism. Our results hold for any congruence between trace equivalence and observation equivalence.

1 Introduction

In this paper we will investigate an algebraic language for networks of synchronously communicating units. Several such algebras have been developed in recent years; examples are CCS [14], SCCS [12], CSP [4], MEIJE [1], ACP [2], CIRCAL [10]. These algebras give a semantic account of different ways to combine networks; consequently they contain a variety of operators such as nondeterministic choice and sequential and parallel composition. In contrast, in the language in the present paper the only operators are parallel composition and interlinking of processes. We contend that these operators form a sufficient basis for the study of many fundamental properties of synchronising parallel

*This is a revised and extended version of the paper "The expressive power of simple parallelism" in the proceedings of PARLE '89 Vol II (Eindhoven, June 1989), published as Springer Verlag LNCS 366 pages 389–405. The present paper, minus the appendix, has been accepted for publication in *Future Generation Computer Systems*.

processes; in particular we will here explore the expressive power as measured by the definable terms and operators in order to gain insight in what phenomena can be derived from parallelism and linking alone. This insight is relevant for an understanding and comparison of other semantic accounts of parallelism and may prove useful in situations such as hardware design, where parallel composition and linking are the only ways to combine networks.

An inspiration for our language comes from data flow networks. In a data flow network, each functional unit has a set of input ports and a set of output ports. The units receive data on the input ports, perform computations on these data, and transmit the results on the output ports. Communication between units is through *links*; each link connects one output port with one input port. The links remain unchanged when the units execute. Typically, a data flow network realises some complex function, using only a set of standard (predefined) functional units. The semantics of data flow has been the subject of many papers [8, 15, 9, 20, 7]. The networks in the present paper will be reminiscent of data flow nets, but the links will be used for synchronisations of rendezvous type (two units must participate in events on a link simultaneously). Any buffering on a link must be modelled explicitly as an intermediary unit acting as a buffer. This synchronous communication is in accordance with the usual semantics of process algebras.

In an algebraic language for description of networks the operators correspond to ways in which larger networks can be built from smaller networks. In our language there are two types of operators. One is *disjoint parallelism*. With this operator, two existing networks can be put in parallel without any internetwork links, thus the networks will execute independently of each other. This corresponds to "disjoint union" [20] and "aggregation" [9]. The other operator type is *linking*: two ports of a network can be linked, thus enforcing synchronisation between the events on these ports. Each port can be attached to at most one link; more complicated structures (such as broadcasting or multi way communication) must be accomplished through intermediate units. The linking operators are inspired by the "linking" [20] and "loop" [9]; to our knowledge the present paper is the first attempt to give them an operational semantics.

Other algebras for description of synchronously communicating networks, for example CCS and CSP, use other families of operators in order to describe parallelism and linking between units. One idea common to these algebras is that the parallel operator implicitly achieves linking. For example, in CCS two ports with complementary names will automatically be linked in a parallel composition, and in CSP two ports with the same name will be linked. Moreover, both these algebras allow a port to be linked to more than one other port, but they give different semantics to such multiple linking: in CCS a synchronisation event always involves events on exactly two linked ports, while in CSP an event on a port must involve events on all linked ports. This makes it impossible to directly define the parallel operators in CCS and CSP in terms of each other. Similarly, in SCCS and ACP the linking between units is partially determined by a function on port names; this function is not an operator but rather a parameter which determines the semantics of the parallel operator. Our position is that our algebraic language, where parallelism and linking are directly reflected as different operators, provides a more natural basis for the study of fundamental properties of parallelism.

Our main results are that with only three simple types of predefined units, any finite-state behaviour can be defined, and the operators normally used to describe parallel structures in many other process algebras are definable. We also determine that some operators related to nondeterminism and sequential composition are not definable.

Related work includes Milner's investigation [13] of the definable behaviours in a language containing only prefixing and choice operators, and de Simone's investigation [5] of the definable operators in the synchronous process algebras MEIJE and SCCS. One main difference between the present work and de Simone's is that de Simone considers a synchronous form of composition as a primitive operator. This operator forces its arguments to execute in lock-step, as if synchronised by a global clock. In contrast our primitive parallel operator is asynchronous; this implies that fewer operators are definable. Another difference is that the results in the present paper are valid for a wide range of behaviour equivalences, and not only for observation equivalence based on bisimulation.

The work presented here is an elaboration and continuation of [16, 17] by the same author, and a related paper [18] examines different models and axiomatisations of the algebraic language.

In Sections 2 and 3 below we present the syntax and semantics of the algebraic language. Section 4 is devoted to examples of networks and their behaviours. In Section 5 we define notions of behaviour equivalence. Section 6 contains our first expressiveness result: with only three simple behaviours, any finite-state behaviour can be constructed. In Section 7 we present a class of operators which are definable within the language, and discuss to what extent operators from other algebras are definable. Section 8 contains some final remarks and open problems. We collect the full proofs of all results in an appendix.

2 Syntax

The purpose of this section is to define an algebraic language where the terms correspond to networks of interconnected units. Figure 1 illustrates an example of such a network. There are three units, each with a set of *ports*. Each port is assigned an internal identification number which is unique within the unit. Two ports in a network may be connected by a *link*; a port so connected is called *internal* to the network. A port not connected by a link is called *external*, and has a unique (for the whole network) *name*. In Figure 1 the network has four internal ports, and three external ports named a, b and c.

A network may contain several units which are instances of the same *module*. A module can be thought of as a schematic unit or a template for a unit; intuitively, we expect different instances of a module to exhibit the same "behaviour" but on different ports. In Figure 1 there are two instances of a module M and one instance of a module M'. Each module has a nonnegative *arity* which is the number of ports of the module. In the example, M has arity 2 and M' has arity 3.

So, in the following we assume a fixed potentially infinite set Λ, called the set of *port names*, and use $a, b, c, \ldots$ to range over Λ. We let $\mathbf{M}$ range over sets of modules, and use M, M' etc. to range over modules.

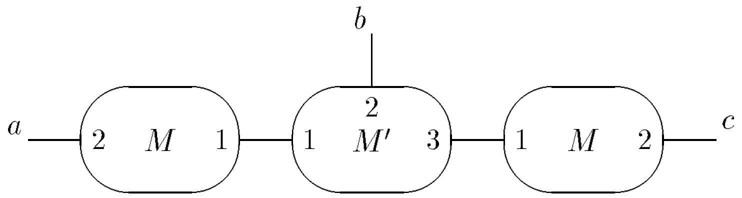


Figure 1: An example network.

Definition 1 $\mathcal{T}_{\mathbf{M}}$ (the set of *terms* over $\mathbf{M}$), in the following ranged over by $A, B, \ldots$, is the least set satisfying the following clauses:

1. If $M \in \mathbf{M}$ and M has arity n, and $a_1, \ldots, a_n$ are port names, then $M(a_1, \ldots, a_n) \in \mathcal{T}_{\mathbf{M}}$.

2. If $A \in \mathcal{T}_{\mathbf{M}}$ and $B \in \mathcal{T}_{\mathbf{M}}$ then $(A|B) \in \mathcal{T}_{\mathbf{M}}$.

3. If $A \in \mathcal{T}_{\mathbf{M}}$ and a and b are distinct port names then $(A\langle a \frown b \rangle) \in \mathcal{T}_{\mathbf{M}}$.

□

We adopt the convention that $|$ associates to the left and that $\langle a \frown b \rangle$ binds tighter than $|$; this allows us to drop some of the parentheses. For example, $A|B\langle a \frown b \rangle|C$ means $((A|(B\langle a \frown b \rangle))|C)$. A term of type $M(a_1, \ldots, a_n)$ corresponds to an atomic network consisting of only one unit obtained from the module M; the external port names of this network are given by $a_1, \ldots, a_n$. The operator $|$ is called the *parallel* operator; the intention is that $A|B$ corresponds to a network obtained as the union of the networks of A and B without any internetwork links. The operators of type $\langle a \frown b \rangle$ are called *linking* operators; a term $A\langle a \frown b \rangle$ denotes the network A where the ports named a and b have been joined by a link. As an example, the network in Figure 1 can be described by the term

$$((M(d, a) \,|\, M'(e, b, f)) \,|\, M(g, c))\langle d \frown e \rangle \langle f \frown g \rangle$$

The *sort* of a term is the set of external port names of the corresponding network:

Definition 2 The *sort* $L(A)$ of a term A is defined inductively as follows:

1. $L(M(a_1, \ldots, a_n)) = \{a_1, \ldots, a_n\}$

2. $L(A|B) = L(A) \cup L(B)$

3. $L(A\langle a \frown b \rangle) = L(A) - \{a, b\}$

□

Term	Condition on formation	Sort
$M(a_1,\ldots,a_n)$	n is the arity of M, $\ i \neq j \Rightarrow a_i \neq a_j$	$\{a_1,\ldots,a_n\}$
$A\vert B$	$L(A) \cap L(B) = \emptyset$	$L(A) \cup L(B)$
$A\langle a\frown b\rangle$	$a \neq b, \ \ a,b \in L(A)$	$L(A) - \{a,b\}$

Table 1: Summary of the formation rules of the algebraic language, and the definition of sort.

We will use $L, L', \ldots$ to range over sorts (i.e. finite subsets of Λ). We can now formalise the requirement that external port names are unique in a network, and that each internal port can be attached to at most one link: we call such terms *well-formed.*

Definition 3 (Well-formed terms)

1. $M(a_1,\ldots,a_n)$ is well-formed if all a_i are pairwise distinct.
2. $A|B$ is well-formed if A and B are well formed and $L(A) \cap L(B) = \emptyset$.
3. $A\langle a\frown b\rangle$ is well-formed if A is well formed and $a, b \in L(A)$.

□

In the rest of this paper we will exclude non-well-formed terms, so we let $\mathcal{T}_{\mathbf{M}}$ stand for the well-formed terms and call those just "terms". A summary of the definitions appears in Table 1.

3 Operational Semantics

We will present the operational semantics for the language in a way that has become standard for process algebras: through a family of labelled binary relations $\xrightarrow{\alpha}$, so called transition relations, on terms. Our definition will be in the form of a Plotkin-style induction on the structure of terms. If desired, from this definition a formal interpretation of terms into transition diagrams can be obtained in a completely standard way.

The label α of a transition $A \xrightarrow{\alpha} B$ is called the *action* of the transition. An action is a set of port names[1] and we use $\alpha, \beta, \gamma, \ldots$ to range over actions. The intended meaning of the transition $A \xrightarrow{\alpha} B$ is that the network corresponding to A can evolve to the network

[1]Actions as sets of port names are also used in CIRCAL [10]; other process algebras tend to use either single port names or multisets.

$$\frac{M \xrightarrow{K} M' \qquad \alpha = \{a_i : i \in K\}}{M(a_1, \ldots, a_n) \xrightarrow{\alpha} M'(a_1, \ldots, a_n)}$$

$$\frac{A \xrightarrow{\alpha} A'}{A|B \xrightarrow{\alpha} A'|B} \qquad \frac{B \xrightarrow{\alpha} B'}{A|B \xrightarrow{\alpha} A|B'} \qquad \frac{A \xrightarrow{\alpha} A' \quad B \xrightarrow{\beta} B'}{A|B \xrightarrow{\alpha \cup \beta} A'|B'}$$

$$\frac{A \xrightarrow{\alpha} A' \qquad a, b \notin \alpha}{A\langle a \frown b\rangle \xrightarrow{\alpha} A'\langle a \frown b\rangle} \qquad \frac{A \xrightarrow{\alpha} A' \quad a, b \in \alpha}{A\langle a \frown b\rangle \xrightarrow{\alpha - \{a,b\}} A'\langle a \frown b\rangle}$$

Table 2: The operational semantics of terms.

corresponding to B by participating in synchronisations on all ports in α. There may be several different transitions from a term with the same action; this amounts to nondeterminism in the term. Note that the empty set $\emptyset$ is also an action, this corresponds to an internal action (cf. τ in CCS).

We take the view that the operational semantics of terms is dependent on an *operational interpretation* of the modules. An operational interpretation is a family of schematic transition relations where the "actions" refer to the internal port numbers within the module:

Definition 4 An *operational interpretation* on a set of modules $\mathbf{M}$ is a set of *schematic transitions* of the form

$$M \xrightarrow{K} M'$$

where $M, M' \in \mathbf{M}$ are of the same arity n, and K is a subset of $\{1, \ldots, n\}$. □

The intention is that $M \xrightarrow{K} M'$ means that the module M can evolve to the module M' by participating in synchronisations on the ports in K. Note that M' must have the same arity as M: a module cannot change its number of ports when it executes.

In the following we assume that every set of modules is associated with an operational interpretation. We now define the transition relations $\xrightarrow{\alpha}$ on terms:

Definition 5 The family of labelled transition relations on $\mathcal{T}_{\mathbf{M}}$ consists of the least relations $\xrightarrow{\alpha}$ satisfying the rules in Table 2. □

The rule for units says that $M(a_1, \ldots, a_n)$ can do exactly what is decreed by M if the ports are named $a_1, \ldots, a_n$. The first and second rules for parallel composition say that

$A|B$ can do whatever A or B can do in isolation. The third rule says that if both A and B can do something, then $A|B$ can do the union of the actions. The parallel operator expresses a form of independent parallelism — A and B execute asynchronously side by side without affecting each other.

For the linking operator, the intuition is that if A can do an action involving neither a nor b, then $A\langle a \frown b\rangle$ can do the same action (the link has no effect at all). If A can do an action involving both a and b, then $A\langle a \frown b\rangle$ can do the same action, now with a and b removed (intuitively this action involves a synchronisation over the link). As a consequence, actions of A involving exactly one of a and b are disallowed in $A\langle a \frown b\rangle$ (such actions would correspond to synchronisations where only one endpoint of the link is involved). While the parallel operator is independent parallelism, the linking operator is used to explicitly express dependencies: actions involving different endpoints of the link must be synchronised.

4 Examples

In this section we present some examples of terms, their corresponding networks, and their operational semantics in the form of *behaviours*. A behaviour is a graph where nodes are terms and labelled edges correspond to transitions between terms; the *behaviour of a term* is simply a behaviour containing that term as well as all transitions from the terms in the graph. Networks will be displayed as in Figure 1, and the internal port numbers of modules will be omitted whenever they are unimportant or can be inferred from the shape of the graphic symbol of the module. Note that behaviours and networks serve only to informally illustrate terms and their operational semantics. In a companion paper [18] we define behaviours and networks formally and explore algebras over them.

To begin we introduce four modules $\textsc{sy}$, $\textsc{ar}$, $\textsc{al}$, and $\textsc{al}'$; the first two of arity three and the last two of arity two. We assume that the schematic transitions of these modules are

$$\begin{array}{ll} \mathrm{SY} \xrightarrow{\{1,2,3\}} \mathrm{SY} & \\ \mathrm{AR} \xrightarrow{\{1,2\}} \mathrm{AR} & \mathrm{AR} \xrightarrow{\{1,3\}} \mathrm{AR} \\ \mathrm{AL} \xrightarrow{\{1\}} \mathrm{AL}' & \mathrm{AL}' \xrightarrow{\{2\}} \mathrm{AL} \end{array}$$

Instances of these modules are depicted in Figure 2; for each unit we give its behaviour (top), and a symbol to be used in graphic descriptions of networks (bottom).

The first unit $\textsc{sy}(a, b, c)$ is a three-way synchroniser. It has three ports; its behaviour is to repeatedly perform an action involving all three ports. A three-way synchroniser can thus be used as an intermediate unit to link three ports. In the same way, we could assume modules for n-way synchronisers $\textsc{sy}_n$ of arity n for arbitrary n: the schematic transitions would be

$$\mathrm{SY}_n \xrightarrow{\{1,\dots,n\}} \mathrm{SY}_n$$

However, n-way synchronisers are definable by three-way synchronisers in the following sense: for each instance of an n-way synchroniser there exists a term with the same

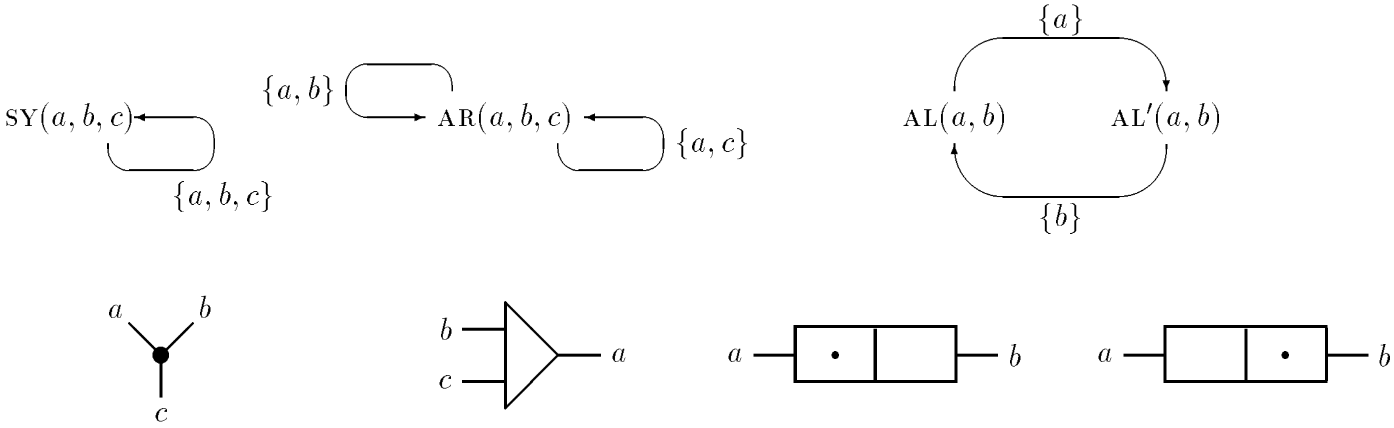


Figure 2: Four units: A three-way synchroniser, an arbiter, and two alternators.

behaviour (this notion will be made precise in Section 5), containing only three-way synchronisers. For example, the term

$$(\textsc{sy}(a,b,f) \,|\, \textsc{sy}(g,c,d))\langle f^\frown g\rangle$$

has the same behaviour as $\textsc{sy}_4(a,b,c,d)$.

The second unit $\textsc{ar}(a,b,c)$ is an arbiter. It repeatedly synchronises its right-hand port a with exactly one of its left-hand ports b and c. This represents another way to interconnect three ports: imagine the left-hand ports to be competing for the privilege of synchronising with the right-hand port. In analogy with the n-way synchronisers it is possible to define n-way arbiters, with n left-hand ports, as a combination of arbiters. For example, a three-way arbiter $\textsc{ar}_3$ would have three schematic transitions labelled $\{1,2\}, \{1,3\}$ and $\{1,4\}$. Again, arbiters of higher arity turn out to be definable by ordinary arbiters. For example, the term

$$(\textsc{ar}(a,b,f) \,|\, \textsc{ar}(g,c,d))\langle f^\frown g\rangle$$

has the same behaviour as $\textsc{ar}_3(a,b,c,d)$.

The third unit $\textsc{al}(a,b)$ in Figure 2 is an alternator: it alternatingly synchronises on its two ports. We use two different modules ($\textsc{al}$ and $\textsc{al}'$) to represent the two states. In the network symbol we use a small filled circle to indicate the state of the alternator: when this circle is to the left (module $\textsc{al}$), the next synchronisation will be on the left-hand port, and if it is to the right (module $\textsc{al}'$) it will be on the right-hand port. Since $\textsc{al}'(a,b)$ has the same behaviour as $\textsc{al}(b,a)$ we will only make use of the $\textsc{al}$ symbol.

A three-way alternator $\textsc{al}_3$, i.e. a module which repeatedly synchronises on its three ports in sequence, is depicted in Figure 3. Formally the schematic transitions of $\textsc{al}_3$ are

$$\textsc{al}_3 \xrightarrow{\{1\}} \textsc{al}_3' \quad \textsc{al}_3' \xrightarrow{\{2\}} \textsc{al}_3'' \quad \textsc{al}_3'' \xrightarrow{\{3\}} \textsc{al}_3$$

In fact, the three-way alternator is definable by using three ordinary alternators and three synchronisers: the term

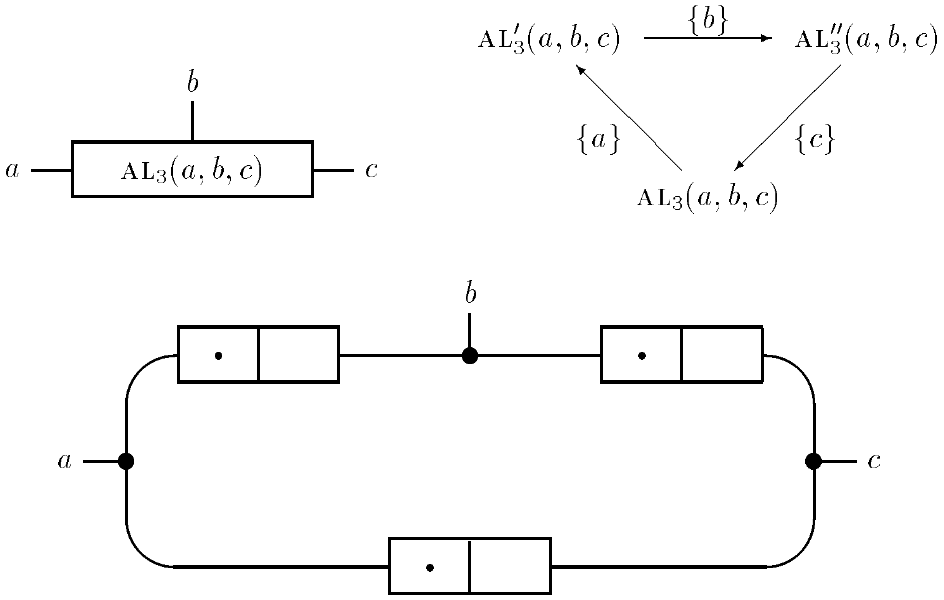


Figure 3: A three-way alternator: Behaviour and equivalent network.

$$(\textsc{sy}(a,f,g)\,|\,\textsc{al}(h,i)\,|\,\textsc{sy}(j,b,k)\,|\,\textsc{al}(l,m)\,|\,\textsc{sy}(n,c,o)\,|\,\textsc{al}(q,p))$$
$$\langle f^\frown h\rangle\langle i^\frown j\rangle\langle k^\frown l\rangle\langle m^\frown n\rangle\langle o^\frown p\rangle\langle q^\frown g\rangle$$

has a behaviour which is precisely the transition graph in Figure 3. The network corresponding to this term is also presented in Figure 3 (bottom).

As a more complicated example, assume that we wish to construct a Controller $CONT$ for a critical resource. This Controller should have the behaviour as indicated in Figure 4 (left). It has four ports: use_1, rel_1, use_2, and rel_2. The intention is that use_1 and rel_1 are connected to one process, and use_2 and rel_2 to another process. A process signals the use of the critical resource with an action on its use port and the release of the resource with an action on its rel port. The Controller makes sure that at most one process has the resource at a time, and that only the process which currently has the resource may release it. A construction of such a Controller is also given in Figure 4 (right)[2]. It uses two alternators to remember which process last acquired the resource, and one alternator to remember whether the resource is available; it also uses two arbiters and four synchronisers to connect these alternators with the external ports. The term corresponding to this network consists of nine units in parallel and ten linking operators — we will not exhibit this term here.

These examples make clear that interesting behaviours can be defined using only a small set of modules; we will return to this topic in Section 6. Our language can also be used to illuminate the relationships between other formalisms for concurrency. Recall for example the parallel operators in CCS and CSP (Figure 5). In CCS, two parallel agents which share a port name a will compete for the use of a in synchronisations.

[2]I am grateful to Tom Verhoeff for this example.

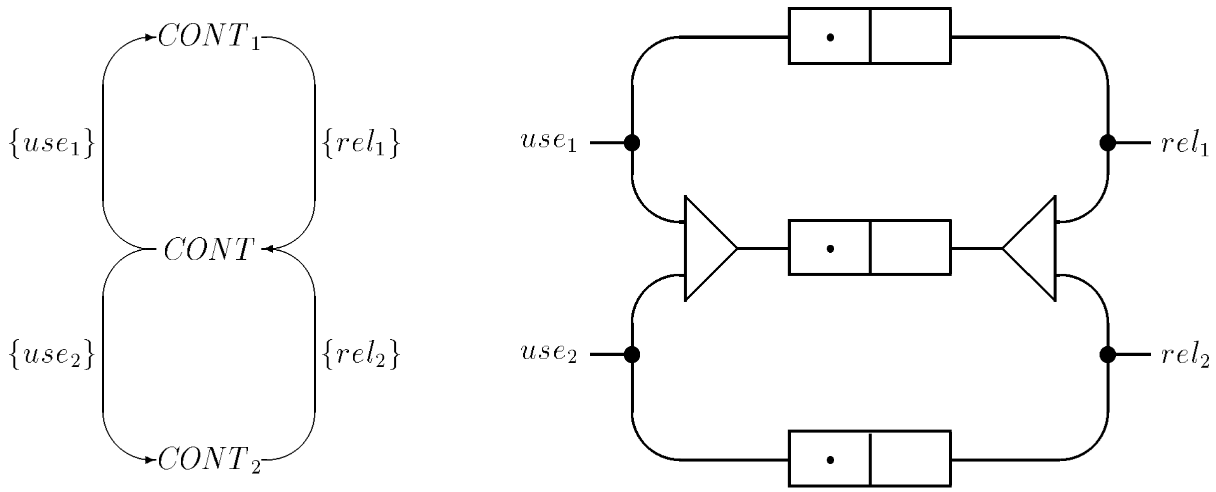
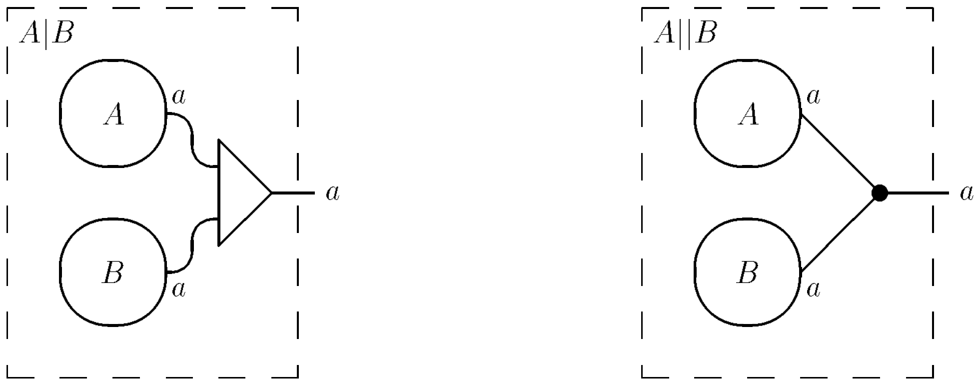


Figure 4: A Controller $CONT$ for a critical resource: Behaviour and network.



CCS-style parallel composition:
Implicit use of an arbiter.

CSP-style parallel composition:
Implicit use of a synchroniser.

Figure 5: Parallel compositions in CCS and CSP.

When the environment of these agents perceives an a action, then either of the agents, but not both, may be the source of the a. Thus the parallel composition in CCS can be considered to implicitly use arbiters to resolve the use of shared port names. On the other hand, in CSP two parallel processes sharing a port a will synchronise on that port: when the environment perceives an a action, then both processes must contribute on their respective ports. Thus parallel composition in CSP implicitly makes use of synchronisers to resolve shared port names.

As a final example, it is straightforward to encode a simple version of place/transition nets [19]. In such a net there are two types of modules: transitions and places. A transition with indegree m and outdegree n is an instance of a module $\mathrm{TR}_{m,n}$ of arity $m+n$; this module has the only schematic transition

$$\mathrm{TR}_{m,n} \xrightarrow{\{1,\ldots,m+n\}} \mathrm{TR}_{m,n}$$

In fact, $\mathrm{TR}_{m,n}$ is just an $(m+n)$-way synchroniser: it requires interactions on all its ports. The intuition is that for a transition to fire, tokens must be received on all incoming arcs, and tokens will be produced on all outgoing arcs.

A place with indegree m and outdegree n currently holding k tokens is an instance of a module $\mathrm{PL}_{m,n,k}$ of arity $m+n$; the schematic transitions for this module are all transitions

$$\mathrm{PL}_{m,n,k} \xrightarrow{K} \mathrm{PL}_{m,n,k'}$$

satisfying

$$k \geq out(K) \quad \text{and} \quad k' = k + in(K) - out(K)$$

where $in(K)$ is the number of integers between 1 and m in K, and $out(K)$ is the number of integers between $m+1$ and $m+k$ in K. The intuition is that a place can emit tokens on its outgoing arcs, but it cannot emit more tokens than it currently holds, and the number of tokens will be modified according to tokens received and emitted. Note that the "behaviour" of such a place is not finite-state (since k can grow unboundedly), so it cannot be described in terms of the other modules in this section.

In the encoding of a place/transition net, a link will will always connect an "in"-port of a unit (a port with a number less than or equal to m in an instance of either of $\mathrm{TR}_{m,n}$ or $\mathrm{PL}_{m,n,k}$) with an "out"-port (a port which is not an in-port) of a unit of the other type. It is trivial to verify that with our encoding the firing rule for place/transition nets is enforced, and that a synchronisation on a link occurs when a token "travels" along that link (either from a place to a transition or vice versa). Note that our encoding allows several transitions to fire in the same action. If a strict interleaving of transitions is desired then the transitions can additionally be connected to a multi way arbiter which only allows one transition to fire at a time.

5 Behaviour Equivalences

Intuitively, two terms can be regarded as equivalent if their respective behaviours are sufficiently similar. Many such equivalences have been proposed in the literature (observation equivalence, failure equivalence, testing equivalence, trace equivalence etc.). Our results will hold for any equivalence which lies between *observation equivalence* ($\approx$) and *trace equivalence* ($=_T$) as defined below. We call such an equivalence a *behaviour equivalence*. We require equivalent terms to have the same sort; without this requirement the congruence result (Theorem 1) below would not hold.

We use σ to range over sequences of non-empty actions, and write $\langle\rangle$ for the empty sequence.

Definition 6

1. $$A \stackrel{\alpha}{\Longrightarrow} A' \text{ if } \begin{cases} \exists n \geq 0 : \ A \underbrace{\stackrel{\emptyset}{\longrightarrow} \cdots \stackrel{\emptyset}{\longrightarrow}}_{n \text{ times}} A' & \text{if } \alpha = \emptyset \\ \exists n, m \geq 0 : \ A \underbrace{\stackrel{\emptyset}{\longrightarrow} \cdots \stackrel{\emptyset}{\longrightarrow}}_{n \text{ times}} \stackrel{\alpha}{\longrightarrow} \underbrace{\stackrel{\emptyset}{\longrightarrow} \cdots \stackrel{\emptyset}{\longrightarrow}}_{m \text{ times}} A' & \text{if } \alpha \neq \emptyset \end{cases}$$

2. A binary relation $\mathcal{R}$ on terms is a *simulation* if $A\mathcal{R}B$ implies

   For all $\alpha, A' : A \stackrel{\alpha}{\longrightarrow} A'$ implies that there exists $B' : B \stackrel{\alpha}{\Longrightarrow} B'$ and $A'\mathcal{R}B'$

3. $\mathcal{R}$ is a *bisimulation* if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations.

4. $A \approx B$ if $L(A) = L(B)$ and there exists a bisimulation $\mathcal{R}$ such that $A\mathcal{R}B$.

5. $A \stackrel{\langle\rangle}{\Longrightarrow} A'$ if $A \stackrel{\emptyset}{\Longrightarrow} A'$, and $A \stackrel{\alpha_1 \cdots \alpha_n}{\Longrightarrow} A'$ if $A \stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow} A'$.

6. $A \stackrel{\sigma}{\Longrightarrow}$ (σ is a *trace* of A) if for some A' : $A \stackrel{\sigma}{\Longrightarrow} A'$.

7. $A =_T B$ if for all σ : $(A \stackrel{\sigma}{\Longrightarrow}$ iff $B \stackrel{\sigma}{\Longrightarrow})$ and $L(A) = L(B)$.

8. An equivalence $\simeq$ on terms is a *behaviour equivalence* if $\approx \ \subseteq \ \simeq \ \subseteq \ =_T$.

$\square$

Definition 7 An equivalence $\simeq$ is a *congruence* if it is preserved by the operators, i.e. if $A \simeq A'$ and $B \simeq B'$ then

1. $A|B$ is a term iff $A'|B'$ is a term, and if so $A|B \simeq A'|B'$.

2. $A\langle a \frown b\rangle$ is a term iff $A'\langle a \frown b\rangle$ is a term, and if so $A\langle a \frown b\rangle \simeq A'\langle a \frown b\rangle$.

$\square$

Our first theorem is that observation equivalence and trace equivalence are congruences:

Theorem 1 $\approx$ and $=_T$ are congruences. $\square$

We omit the proof since it is similar to proofs of similar known results. In the following we restrict attention to behaviour equivalences which are congruences.

6 A Basis for Finite-State Terms

In a practical design situation, it will be useful to know what modules are necessary to build a certain set of behaviours (up to some behaviour equivalence). Conversely, it will be useful to know what behaviours can be constructed given a certain set of modules. Such results also have intrinsic theoretical interest: they illuminate the expressive power of the operators. The main result in this section is that the three modules SY, AR, and AL from Section 4 suffice to build any finite-state behaviour. This indicates that our operators exhibit a considerable expressive power.

In the following $\mathcal{T}$ ranges over sets of terms, and we assume a fixed behaviour equivalence $\simeq$ on terms.

Definition 8

1. $\mathbf{M}$ is a *basis* for $\mathcal{T}$ if each term in $\mathcal{T}$ is equivalent with a term in $\mathcal{T}_{\mathbf{M}}$.
2. $\mathbf{M}$ is a *proper basis* for $\mathcal{T}$ if it is a basis, and each term in $\mathcal{T}_{\mathbf{M}}$ is equivalent with a term in $\mathcal{T}$.
3. M is *independent* of $\mathbf{M}$ if some instance of M is not equivalent with any term in $\mathcal{T}_{\mathbf{M}}$.
4. $\mathbf{M}$ is *independent* if, for all $M \in \mathbf{M}$, the module M is independent of $\mathbf{M} - \{M\}$.

$\square$

Intuitively, "$\mathbf{M}$ is a basis for $\mathcal{T}$" means that the set of behaviours represented by $\mathcal{T}$ can be constructed (up to equivalence) with modules in $\mathbf{M}$. If in addition $\mathbf{M}$ is proper, then $\mathcal{T}$ represents all behaviours that can be constructed, and if $\mathbf{M}$ is independent, then all modules in $\mathbf{M}$ are actually necessary (no module can be removed).

We can now ask many intriguing questions. For example, given an interesting set of terms $\mathcal{T}$, what is an independent proper basis? Conversely, given an interesting set of modules $\mathbf{M}$, is it independent, and is it a proper basis for an interesting set of terms? We will answer only one such question here, and leave the rest as topics for further research.

Definition 9 The *state space* of a term A is the set $\{B : \text{ for some } \sigma : A \stackrel{\sigma}{\Longrightarrow} B\}$. A term is *finite-state* if its state space is finite. $\square$

Consider the modules SY, AR, and AL with associated schematic transitions as defined in Section 4. The main theorem of this section is (note that it holds regardless of the choice of $\simeq$):

Theorem 2 $\{\text{SY}, \text{AR}, \text{AL}\}$ is an independent proper basis for the set of finite-state terms. $\square$

Proof sketch (the full proof can be found in the appendix): To show that this set of modules is a basis, we must show that for every finite-state term A, there is an equivalent term B in $\mathcal{T}_{\{\text{SY},\text{AR},\text{AL}\}}$. The proof is by direct construction of a network from the behaviour of A. The idea is that every state in A corresponds to one alternator. Transitions between states in A correspond to links between the alternators; each alternator has one port for incoming transitions and one port for outgoing transitions. If a state has several outgoing or incoming transitions then arbiters are used: all links corresponding to incoming transitions are routed through arbiters to one port of the alternator, and all links corresponding to outgoing transitions are routed to the other port. If a transition has a non-empty action, then a three-way synchroniser is interposed on the corresponding link: the third port of this synchroniser is used to generate the action. Examples of such constructions can be found in Section 4, Figure 3 and Figure 4. The desired term B can be defined as a term corresponding to this network; we can then prove $A \approx B$, which implies $A \simeq B$.

To prove that the basis is proper, it suffices to observe that instances of $\textsc{sy}$, $\textsc{ar}$ and $\textsc{al}$ are all finite-state, and that the operators (parallelism and linking) preserve the property of being finite-state.

Finally, to prove that $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ is independent, we establish for each $M \in \{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ a property φ_M of terms satisfying the following conditions: (1) Instances of the two modules in $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ which are not M all have property φ_M; (2) φ_M is preserved by the operators (parallel and linking); (3) φ_M is preserved by trace equivalence; (4) Some instance of M lacks property φ_M. Thus, (1) and (2) guarantee that all terms over $\{\textsc{sy}, \textsc{ar}, \textsc{al}\} - \{M\}$ have property φ_M, and (3) and (4) guarantee that an instance of M is not trace equivalent, and hence not $\simeq$, with any such term.

The properties φ_M are defined as follows. φ_{AL} is defined to hold for a term if it is trace equivalent with a term which has exactly one state. The property φ_{AR} is *intersection closure*: a term is intersection closed if whenever it can do actions α and β it can also do $\alpha \cap \beta$. Finally the property φ_{SY} is *partition closure*: this holds for a term A if whenever A can do an action containing more than two ports, then this action can be partitioned into smaller actions with at most two ports each such that A can do any union of these smaller actions. □

It is interesting to note that this result holds also for equivalences stronger than observation equivalence. In fact, the proof carries over to "strong equivalence" (bisimulation equivalence defined in terms of $\xrightarrow{\alpha}$ rather than $\xRightarrow{\alpha}$, i.e. internal actions are significant).

An alternative way to phrase the ideas in this section is to define a formal interpretation of terms into behaviours and define the equivalences directly on behaviours (rather than terms), and say that a behaviour is *definable* if there is a term with an equivalent behaviour. Theorem 2 then implies that any finite-state behaviour is definable even if no other modules than $\textsc{sy}$, $\textsc{ar}$ and $\textsc{al}$ are used, i.e. that any finite-state behaviour can be built from three-way synchronisers, arbiters, and alternators.

One other result is that $\{\textsc{sy}, \textsc{ar}\}$ is a proper independent basis for the set of "one state" terms. We do not know any interesting basis for any other set of terms. Note that some sets of terms which have traditionally been regarded as interesting, viz. the "deterministic" terms and the "confluent" terms [14] are not closed under the linking operator, and hence cannot have a proper basis.

7 Definability of Operators

In this section we examine to what extent other operators are definable within the language. Briefly stated, an operator is definable if there is an equivalent combination of parallel and linking operators. As an example, assume that we desire a three-way linking operator $\langle a^\frown b^\frown c\rangle$: the intention is that in the term $A\langle a^\frown b^\frown c\rangle$, the three ports a, b, c in A should be linked. We thus extend the language with a new class of operators $\langle a^\frown b^\frown c\rangle$, and give these operators operational definitions (in the same way as $\langle a^\frown b\rangle$):

$$\frac{A \xrightarrow{\alpha} A' \quad a,b,c \notin \alpha}{A\langle a^\frown b^\frown c\rangle \xrightarrow{\alpha} A'\langle a^\frown b^\frown c\rangle} \qquad \frac{A \xrightarrow{\alpha} A' \quad a,b,c \in \alpha}{A\langle a^\frown b^\frown c\rangle \xrightarrow{\alpha-\{a,b,c\}} A'\langle a^\frown b^\frown c\rangle}$$

However, such an extension would in one sense be redundant: the effect of $\langle a^\frown b^\frown c\rangle$ can be achieved with a three-way synchroniser (cf. Figure 2) as follows. Let A be a term with a, b, c in its sort, and choose port names d, e, f not in the sort of A. We can then prove that

$$(A|\mathrm{SY}(d,e,f))\langle a^\frown d\rangle\langle b^\frown e\rangle\langle c^\frown f\rangle \;\approx\; A\langle a^\frown b^\frown c\rangle$$

Thus we say that if $\mathrm{SY}(d,e,f)$ (or an equivalent term) is definable, then three-way linking is definable in the language: a designer can confidently use three-way links when constructing networks, and later expand these links according to the definition. We will investigate which operators are definable in this sense. We will give a general definability result, and discuss some examples of operators from other process algebras. As might be expected, whether an operator is definable depends critically on which equivalence is used. It turns out that more operators are definable up to trace equivalence than up to observation equivalence.

In the following we write $\widetilde{A}$ for the sequence of terms $A_1,\ldots,A_n$ (here $n \geq 0$), similarly $\widetilde{A'}$ will mean the sequence $A'_1,\ldots,A'_n$. We write $\widetilde{L}$ for the sequence of sorts $L_1,\ldots,L_n$, and say that $\widetilde{A}$ has sort $\widetilde{L}$ if $L(A_k) = L_k$ for all k $(1 \leq k \leq n)$. We will introduce new operators ranged over by *op*, *op'* etc. and assume that each such operator has an *arity* $n \geq 0$ and a *type* $\widetilde{L} \rightarrow L$.

Definition 10 The set $\mathcal{T}_{\mathbf{op},\mathbf{M}}$ of terms over a set $\mathbf{op}$ of operators and a set $\mathbf{M}$ of modules is defined by generalising definitions 1–3 in the obvious way: if $A_i \in \mathcal{T}_{\mathbf{op},\mathbf{M}}$ for all $i = 1,\ldots,n$ and $\widetilde{A}$ has sort $\widetilde{L}$ and $op \in \mathbf{op}$ has type $\widetilde{L} \rightarrow L$, then $op(\widetilde{A})$ is in $\mathcal{T}_{\mathbf{op},\mathbf{M}}$ and has sort L. $\square$

For example, a three-way linking operator $\langle a^\frown b^\frown c\rangle$ would be of type $L \rightarrow (L - \{a,b,c\})$ for a sort L containing a, b, c. Notice that each operator has a fixed type, so there will be one three-way linking operator for each such L. Strictly speaking our original operators $|$ and $\langle a^\frown b\rangle$ are families of operators for the same reason, but for convenience we will continue to refer to them as "operators". We let $|$ and $\langle a^\frown b\rangle$ be tacitly present in any set $\mathbf{op}$ under consideration; with this convention $\mathcal{T}_{\emptyset,\mathbf{M}}$ is just $\mathcal{T}_{\mathbf{M}}$.

In what follows we assume that whenever **op** is a set of operators, a fixed definition of the operational semantics of **op** determines the labelled transition relations on terms in $\mathcal{T}_{\mathbf{op},\mathbf{M}}$. For example, a family of three-way linking operators can be given an operational semantics as in the beginning of this section. We do not require that the definition of the operational semantics is presented in a particular format; formally an operational semantics is just a set of transitions of type $A \xrightarrow{\alpha} B$ which agree with the rules in Table 2. Definitions 6 and 7 generalise directly to terms in $\mathcal{T}_{\mathbf{op},\mathbf{M}}$, so it is possible to talk about behaviour equivalences on $\mathcal{T}_{\mathbf{op},\mathbf{M}}$. This is essential for our definition of definability below.

Definition 11 An *n-ary context* $\mathcal{C}$ (here $n \geq 0$) over the modules $\mathbf{M}$ is a term in $\mathcal{T}_{\mathbf{M}}$ with n numbered holes in it, and we write $\mathcal{C}(\widetilde{A})$ for the term obtained by inserting $A_1, \ldots, A_n$ in the holes in $\mathcal{C}$. □

To make this definition precise we could introduce variables in the language and talk about terms with variables, and substitution of terms for variables. We trust the reader to accept our more simplistic definition.

Definition 12 An operator *op* of type $\widetilde{L} \to L$ is *$\simeq$-definable over* $\mathbf{M}$ (or simply *definable* if $\mathbf{M}$ and the equivalence $\simeq$ are implicit) if there exists a context $\mathcal{C}$ over $\mathbf{M}$ such that

$$\text{for all } \widetilde{A} \text{ of sort } \widetilde{L}, \quad \mathcal{C}(\widetilde{A}) \simeq op(\widetilde{A})$$

We call such a $\mathcal{C}$ a *$\simeq$-defining context* for *op*. □

As an example, take a unary three-way linking operator $\langle a^\frown b^\frown c\rangle$ of sort $L \to L - \{a, b, c\}$ for an L not containing d, e or f. This operator is $\approx$-definable over $\{\textsc{sy}\}$:

$$\mathcal{C}(\bullet) = (\bullet | \textsc{sy}(d, e, f))\langle a^\frown d\rangle\langle b^\frown e\rangle\langle c^\frown f\rangle$$

is a defining context. In the rest of this section we aim to show that an operator is definable precisely if its operational semantics can be presented in a particular format. This will require some preliminary definitions.

Definition 13 A *de Simone rule* is a tuple $\langle op, \mu_1, \ldots, \mu_n, \alpha, op'\rangle$ where *op* and *op'* are (possibly the same) n-ary operators of the same type $\widetilde{L} \to L$, each μ_i is either a subset of L_i or the special symbol $*$, and α is a subset of L. □

A de Simone rule $\langle op, \mu_1, \ldots, \mu_n, \alpha, op'\rangle$ will be considered as an inference rule

$$\frac{\left[A_i = A_i' \text{ if } \mu_i = *, \quad A_i \xrightarrow{\mu_i} A_i' \text{ otherwise}\right]_{i=1,\ldots,n}}{op(\widetilde{A}) \xrightarrow{\alpha} op'(\widetilde{A'})}$$

The rule has n premises, and premise number i is "$A_i = A_i'$" if $\mu_i = *$, otherwise the premise is "$A_i \xrightarrow{\mu_i} A_i'$". The rule should be read: "If the premises hold for the terms $\widetilde{A}, \widetilde{A'}$, then the conclusion also holds."

Definition 14 A set **op** of operators is called a *de Simone set* if the transitions on terms in $\mathcal{T}_{\mathbf{op},\mathbf{M}}$ are exactly the transitions which can be proven from a set of de Simone rules. An operator is a *de Simone operator* if it is in a de Simone set. □

The class of de Simone operators was first suggested by Robert de Simone [5] who proved that these operators are exactly the definable operators in the synchronous algebras MEIJE and SCCS. We use the same definition, only slightly adapted to our framework. Practically all operators studied in process algebras are de Simone operators (in two recent papers [6, 3] more general rule formats are suggested). As an example, the singleton set containing only a three-way linking operator constitutes a de Simone set, so three-way linking is a de Simone operator. Also, from Table 2 it is evident that parallel and linking are de Simone operators. Intuitively, an operator op is a de Simone operator if the transitions from a term $op(\widetilde{A})$ can be inferred from the transitions of $\widetilde{A}$ alone, and if the derivative $op'(\widetilde{A'})$ contains each derivative A'_i from the premises exactly once.

Since our parallel operator $|$ is asynchronous, it turns out that not all the de Simone operators are definable. As a simple example consider the unary operator ext of type $\emptyset \to \{a\}$ with the only rule

$$\frac{A \xrightarrow{\emptyset} A'}{ext(A) \xrightarrow{\{a\}} ext(A')}$$

(in the more compact rendering as a tuple this rule is $\langle ext, \emptyset, \{a\}, ext\rangle$). Intuitively, ext transforms internal actions $\emptyset$ into external actions $\{a\}$. This operator is not definable in our language. A simple proof of this is to observe that ext does not respect trace equivalence: if A and B are two terms of sort $\emptyset$ which differ in the number of internal actions they can perform, then $A =_T B$ but $ext(A) \neq_T ext(B)$. Since parallel and linking respect $=_T$ it follows that $\mathcal{C}(A) =_T \mathcal{C}(B)$ for all contexts $\mathcal{C}$. Hence no context can be a defining context for ext.

In essence, the definable operators in our algebraic language turn out to be the de Simone operators that cannot distinguish between internal actions and absence of actions in their arguments:

Definition 15 A set of de Simone rules is called an *asynchronous rule set* if it satisfies the following requirements:

1. $\emptyset$ and $*$ occur interchangeably in the set, i.e. a rule $r = \langle op, \mu_1, \ldots, \mu_n, \alpha, op'\rangle$ with $\mu_i = \emptyset$ is in the set if and only if a rule which differs from r only in that $\mu_i = *$ also is in the set. There is one exception from this requirement: rules of type $\langle op, *, \ldots, *, \emptyset, op\rangle$, where the operators are the same, all μ_i are $*$, and $\alpha = \emptyset$ are not required to be in the set.

2. The set contains an *idling* rule $\langle op, \emptyset, \ldots, \emptyset, \emptyset, op\rangle$ for each operator op occurring in a rule in the set.

□

The first requirement above ensures that an operator treats the absence of an action in an argument $(*)$ in the same way as an internal action $(\emptyset)$. An idling rule in the second requirement can be rendered

$$\frac{A_1 \xrightarrow{\emptyset} A_1' \cdots A_n \xrightarrow{\emptyset} A_n'}{op(\widetilde{A}) \xrightarrow{\emptyset} op(\widetilde{A'})}$$

The idling rule ensures that if all arguments A_i of op idle (i.e. do an internal action) then $op(\widetilde{A})$ must also have the possibility to idle. In other words, op cannot prevent its arguments from doing internal actions.

The exception in the first requirement of Definition 15 is a technical convenience only. Without this exception the requirements would imply that an asynchronous set would always contain a rule

$$\frac{-}{op(\widetilde{A}) \xrightarrow{\emptyset} op(\widetilde{A})}$$

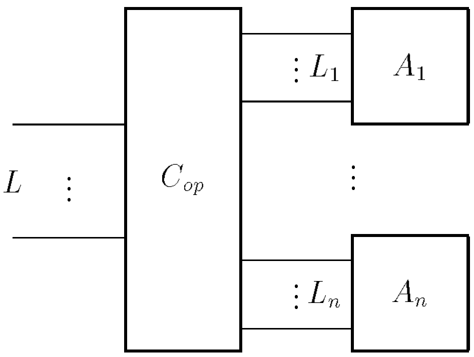
for each operator op. Such a rule would mean that each term of type $op(\widetilde{A})$ has a transition with an internal action leading back to itself. Clearly, the presence or absence of such transitions is unimportant for the purpose of determining observation equivalence, and hence for any behaviour equivalence.

In analogy with Definition 14 we say that a set of operators **op** is *asynchronous* if the transitions over $\mathcal{T}_{\mathbf{op},\mathbf{M}}$ can be determined by an asynchronous set of rules, and that an operator is asynchronous if it is a member of an asynchronous set. One way to think about an asynchronous operator is as a large network where the arguments represent subnetworks. This large network can control the subnetworks only through the ports which are external to the subnetworks; in particular the large network cannot distinguish between internal actions and absence of actions in the subnetworks. Also, the the larger network must always be able to idle when all subnetworks idle. For example, parallel and linking are asynchronous as is the three-way linking suggested in the beginning of this section. This fact can be seen immediately since the rules for these operators form asynchronous sets. In contrast, the singleton set containing the only rule for the operator ext above does not form an asynchronous set: it satisfies neither of the two conditions.

The main theorem in this section can now be stated. It holds regardless of the choice of behaviour equivalence.

Theorem 3 For any asynchronous operator op there is a set of modules **M** such that op is $\simeq$-definable over **M**. $\square$

Proof sketch (the full proof can be found in the appendix): Let op be an asynchronous operator of type $\widetilde{L} \rightarrow L$. The main idea is to derive a defining context for op from the set of rules for op. This context consists of a "controller" C_{op} interlinked with the operands $\widetilde{A}$ of op in the following way:



Intuitively, C_{op} encodes the rules of *op*: it controls the transitions of the operands and generates the appropriate external actions. By choosing an appropriate $\mathbf{M}$ (with appropriate schematic transitions) we can use an instance of a module in $\mathbf{M}$ for C_{op}. Unfortunately the linking can not be achieved directly: L_k is not necessarily disjoint from L, so the parallel composition of A_k and C_{op} may be undefined. We can however make the linking in an indirect way by enclosing the A_k:s in "relabelling contexts"; these contexts effectively act as port name relabellings and make sure that all involved port names are unique. $\square$

A variant of the main result can be obtained by restricting attention to finite-state operators:

Definition 16 An asynchronous operator is *finite-state* if it is in a finite asynchronous set of operators. $\square$

In other words, an operator is finite state if only finitely many auxiliary operators are needed in order to define its semantics. Notice that a de Simone set of rules for a finite set of operators is necessarily finite (since each operator has a fixed type), hence the semantics of a finite-state operator can be defined with a finite set of rules. We also say that a module M is *finite-state* if its associated schematic transitions imply that any instance of it is finite-state.

Theorem 4 For any finite-state asynchronous operator *op* there is a set of finite-state modules $\mathbf{M}$ such that *op* is $\simeq$-definable over $\mathbf{M}$. $\square$

Proof: In the proof of Theorem 3, we only use modules in the definitions of the relabelling contexts (these will always be finite-state) and in C_{op}; and C_{op} is finite-state if *op* is finite-state. $\square$

This relates nicely with the result in Section 6: the modules SY, AR, and AL are sufficient to define any finite-state term. Thus, if $\mathbf{M}$ consists of finite-state modules, then any context over $\mathbf{M}$ has an equivalent context over $\{\text{SY}, \text{AR}, \text{AL}\}$.

Corollary 5 Any finite-state operator is $\simeq$-definable over $\{\text{SY}, \text{AR}, \text{AL}\}$. $\square$

Say that two operators *op* and *op'* are *$\simeq$-equivalent*, written $op \simeq op'$, if they are of the same type $\widetilde{L} \to L$, and for all $\widetilde{A}$ of sort $\widetilde{L}$ it holds $op(\widetilde{A}) \simeq op'(\widetilde{A})$. In essence the converses of Theorems 3 and 4 hold up to $\simeq$-equivalence of operators:

Operator	Usual notation	$\approx$	$=_T$
CCS parallel	$A \mid B$	Yes	Yes
CCS relabelling	$A[f]$	Yes	Yes
CCS restriction	$A\backslash a$	Yes	Yes
CCS nondeterministic choice	$A + B$	No	Yes
CCS prefixing	$a.A$	No	Yes
SCCS synchronous parallel	$A \times B$	No	No
CSP parallel	$A \parallel B$	Yes	Yes
CSP hiding	$A\backslash a$	Yes	Yes
CSP external nondeterminism	$A \Box B$	Yes	Yes
CSP internal nondeterminism	$A \sqcap B$	No	Yes
CSP sequential composition	$A; B$	No	Yes
LOTOS interrupt	$A \rhd B$	No	Yes
MEIJE pilot	$a * A$	No	No

Table 3: Definability of some operators.

Theorem 6 If an operator op is $\simeq$-definable over $\mathbf{M}$ then there is an asynchronous op^* such that $op \simeq op^*$. If $\mathbf{M}$ additionally contains only finite-state modules then there is such an op^* which is finite-state. $\Box$

Proof sketch (the full proof can be found in the appendix): Assume that op is $\simeq$-definable over $\mathbf{M}$. Then there is a context $\mathcal{C}_{op}$ over $\mathbf{M}$ such that $\mathcal{C}_{op}(\widetilde{A}) \simeq op(\widetilde{A})$ for all $\widetilde{A}$ of the appropriate sort. We can prove by structural induction on contexts that for each context $\mathcal{C}$ there is an asynchronous operator $op^{\mathcal{C}}$ such that $\mathcal{C}$ is a $\approx$-defining context for $op^{\mathcal{C}}$. If the modules in $\mathcal{C}$ are finite state then $op^{\mathcal{C}}$ will also be finite state. Put $op^* = op^{\mathcal{C}_{op}}$; we then have that $op^*(\widetilde{A}) \approx \mathcal{C}_{op}(\widetilde{A}) \simeq op(\widetilde{A})$ holds for all $\widetilde{A}$ of the appropriate sort; this proves $op^* \simeq op$. $\Box$

Notice that if op is $\simeq$-equivalent with an asynchronous operator op^* then op is $\simeq$-definable even if op itself is not asynchronous. This is demonstrates that $\simeq$-definability varies with the choice of $\simeq$.

It is interesting to consider some popular operators from other process algebras here. Table 3 summarises the $\approx$-definability and $=_T$-definability. In this table "Yes" means that the operator is definable over any basis for finite-state terms, while "No" means that the operator is not definable over any basis for finite-state terms. The proofs of these results can be found in the appendix.

As can be seen, the operators fall into three groups. The first group consists of operators which are $\approx$-definable, and hence also $=_T$-definable. The proof that these operators are definable follows from Theorem 4 by providing $\approx$-equivalent finite-state asynchronous operators.

The second group consists of $=_T$-definable but not $\approx$-definable operators. Again, the proofs of definability amount to giving $=_T$-equivalent finite-state asynchronous operators.

The undefinability proofs are perhaps more interesting. As an example consider the CCS prefixing operator. This operator has the only rule

$$\frac{-}{a.A \xrightarrow{\{a\}} A}$$

The proof that prefixing is not $\approx$-definable over any basis $\mathbf{M}$ for finite-state terms uses Theorem 6 and is a proof by contradiction as follows. Assume that there is an asynchronous operator op^* such that $op^*(A) \approx a.A$ for all A. Choose A such that $A \stackrel{\emptyset}{\Longrightarrow} A'$ with $a.A \not\approx a.A'$. Since all finite-state behaviours are definable such a term A must exist in $\mathcal{T}_{\mathbf{M}}$. Now since op^* is asynchronous and $A \stackrel{\emptyset}{\Longrightarrow} A'$ the idling rule of op^* yields $op^*(A) \stackrel{\emptyset}{\Longrightarrow} op^*(A')$. Then because $op^*(A) \approx a.A$ there must be a term B such that $a.A \stackrel{\emptyset}{\Longrightarrow} B$ and $op^*(A') \approx B$. We deduce from $a.A \stackrel{\emptyset}{\Longrightarrow} B$ and the only rule for prefixing that $a.A = B$. Hence A and A' must satisfy $a.A \approx op^*(A') \approx a.A'$. But this contradicts $a.A \not\approx a.A'$. Hence no such op^* can exist.

Finally, the third group consists of two operators which are not $=_T$-definable and hence not $\approx$-definable. For example, the MEIJE pilot operator is not definable because internal actions ($\emptyset$) in its operand are significant for determining external actions of the operator (cf. the *ext* operator in this section). Thus, two terms which are trace equivalent but differ in their capability to perform internal actions may, when used as operands to the pilot operator, yield nonequivalent terms. So this operator can by Theorem 1 not have a defining context. The situation for SCCS synchronous parallel is similar.

8 Conclusion

We have explored an algebraic language with an operational semantics for description of networks of processes. The aim has been to keep the primitives as simple as possible, and then explore the expressive power as measured by the definable terms and operators. It has turned out that this language is expresses the static parts, i.e. the operators normally used to combine processes in parallel, of many existing process algebras. Our conclusion is that the language is simple and expressive enough to throw light on fundamental properties of parallelism, and on other formalisms which describe parallelism.

There are many interesting questions left unanswered:

- Are there other interesting expressiveness results for terms? For example, is there a notion of "computable behaviour", and do the terms corresponding to computable behaviours have a proper basis?
- What is the definability of the operators in Table 3 with respect to some other equivalence, e.g. failure equivalence or testing equivalence?
- Is there a natural extension of our results to an algebra where the communications carry data values from one unit to another?

Much work in related process algebras have focused on complete axiomatisations of particular behaviour equivalences. Say that a statement $A \simeq B$ is *valid* if it is true in *all* operational interpretations of module symbols. In our companion paper [18] we formulate a complete axiomatisation for validity in this sense, and compare it with an axiomatisation of equality of networks. It turns out that validity coincides for a large class of behaviour equivalences, and that the complete axiomatisation is similar to the flow graph axiomatisation [11].

Acknowledgments

I am deeply grateful to Robin Milner, David Walker, Bengt Jonsson, Frits Vaandrager, and Tom Verhoeff for inspiring discussions on this and related subjects, and for critical readings of draft versions of this paper. Many illuminating comments, in particular related to Section 7, have been received from Kim Larsen. A large part of this research was done while the author was on leave at the University of Edinburgh, supported by a grant from the British Science and Engineering Research Council.

References

[1] D. Austry and G. Boudol. Algèbre de processus et synchronisation. *Theoretical Computer Science*, 30(1):91–131, 1984.

[2] J. Bergstra and J. Tucker. Top-down design and the algebra of communicating processes. *Science of Computer Programming*, 5(2):171–199, 1985.

[3] B. Bloom, S. Istrail, and A. Meyer. Bisimulation can't be traced. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages*, pages 229–239, 1988.

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

[5] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37(3):245–267, 1985.

[6] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 1989 or 1990. To appear.

[7] B. Jonsson. A model and proof system for asynchronous networks. In *Proceedings of the 4:th ACM Symposium on Principles of Distributed Computing*, pages 49–58, 1985.

[8] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74*, pages 471–475, 1974.

[9] R. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In Brookes, Roscoe, and Winskel, editors, *Seminar on Concurrency 1984, LNCS 197*, pages 479–496, 1985.

[10] G. Milne. CIRCAL and the representation of communication, concurrency and time. *ACM Transactions on Programming Languages and Systems*, 7(2):270–298, 1985.

[11] R. Milner. Flowgraphs and flow algebras. *J. ACM*, 26(4):794–818, 1979.

[12] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[13] R. Milner. A complete inference system for a class of regular behaviours. *J. of Computer System Science*, 28:439–466, 1984.

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

[15] D. Park. The 'fairness' problem and nondeterministic computing networks. In de Bakker and van Leeuwen, editors, *Foundations of Computer Science IV, Part 2*, pages 133–161. Amsterdam, 1983. Mathematical Centre Tracts 159.

[16] J. Parrow. Synchronisation flow algebra. Technical Report LFCS-87-35, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1987.

[17] J. Parrow. The expressive power of simple parallelism. In *Proceedings of PARLE '89, Vol. II; LNCS 366*, pages 389–405, 1989.

[18] J. Parrow. Structural and behavioural equivalences of networks. In *Proceedings of ICALP '90, LNCS 443*, pages 540–552, 1990.

[19] W. Reisig. *Petri Nets*. Springer Verlag, 1985. EATCS Monographs on Theoretical Computer Science Vol. 4.

[20] J. Staples and V. L. Nguyen. A fixpoint semantics for nondeterministic data flow. *J. ACM*, 32(2):411–444, 1985.

Appendix

Proof of Theorem 2

The proof of Theorem 2 stretches over several lemmas and auxiliary definitions. The idea of the proof is presented in Section 6; the proof is not complicated but rather long.

In the following we will use networks as graphical representations of terms; in particular we will use the symbols for the modules in Figure 2. We assume an operational semantics according to that figure; the semantics of other modules is unimportant.

Let A be a term. The *transition graph corresponding to* A, written $T(A)$, is the rooted labelled graph where the set of nodes is the state space of A, the initial node (root) is A, and there is an edge from B to B' labelled α exactly if there is a transition $B \xrightarrow{\alpha} B'$ between the terms B and B'. In the following we will write such edges in the same way as transitions; it will be clear from the context whether "$B \xrightarrow{\alpha} B'$" refers to an edge in a graph or to a transition. With the *sort* $L(T)$ of a transition graph T we will mean the set of port names occurring in actions (transition labels) in T. Thus, $a \in L(T)$ iff $\exists A, A', \alpha$ such that $a \in \alpha$ and $A \xrightarrow{\alpha} A'$ is an edge in T. Note that it is not necessarily the case that $L(T(A)) = L(A)$. The reason is that there might be ports in $L(A)$ which are "useless" in the sense that they do not occur in any action of any transition in $T(A)$. However, it is always the case that $L(T(A)) \subseteq L(A)$.

In the following we extend the notion of bisimulation to transition graphs in the obvious way. If G and G' are transition graphs or terms, and there exists a bisimulation between G and G', we write $G \approx_b G'$ and say that G and G' are *bisimilar*. Thus, if A and B are terms, then $A \approx B$ iff $A \approx_b B$ and $L(A) = L(B)$.

Say that a transition graph is *self-loop free* iff it has no edges of type $A \xrightarrow{\alpha} A$, i.e. no edges where the source state is identical with the destination state.

Lemma 7 Let T be a transition graph. Then there exists a bisimilar self-loop free transition graph T'. □

Proof: For each node A in T, assume a new unique node A' not in T. Let the nodes of T' be

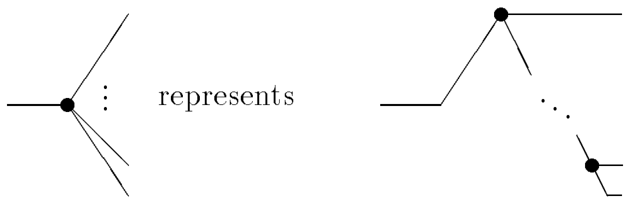
$$\{A : A \text{ is a node in } T\} \cup \{A' : A \text{ is a node in } T\}$$

Let the edges in T' be

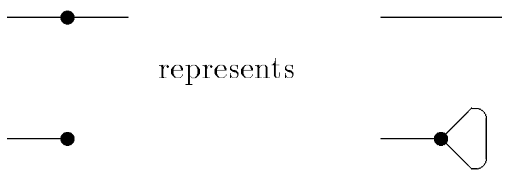
$$\begin{array}{ll} & \{A \xrightarrow{\alpha} B : A \neq B \text{ and } A \xrightarrow{\alpha} B \text{ is an edge in } T\} \\ \cup & \{A' \xrightarrow{\alpha} B : A \xrightarrow{\alpha} B \text{ is an edge in } T\} \\ \cup & \{A \xrightarrow{\alpha} A' : A \xrightarrow{\alpha} A \text{ is an edge in } T\} \end{array}$$

It is straightforward to demonstrate a bisimulation between T and T': a node A in T bisimulates the nodes A and A' in T'. □

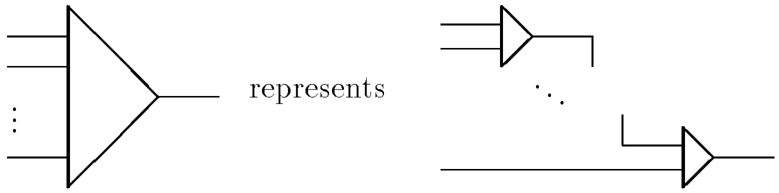
We next introduce network symbols for multi way synchronisers (constructed from three way synchronisers), and multi way arbiters (constructed from ordinary arbiters). A *multi way synchroniser* is depicted as a filled circle with arbitrarily many external ports, and represents a combination of three way synchronisers:



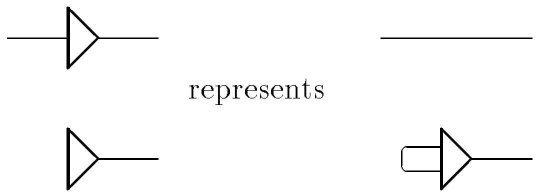
Thus, a multi way synchroniser requires that all ports are involved in a transition. In case there are only two ports, the multi way synchroniser represents a link. In case there is exactly one port, the multi way synchroniser represents a three way synchroniser with two ports interlinked: it can always perform an action involving its only port. Graphically:



A *multi way arbiter*, depicted as a big triangle, represents a combination of arbiters:

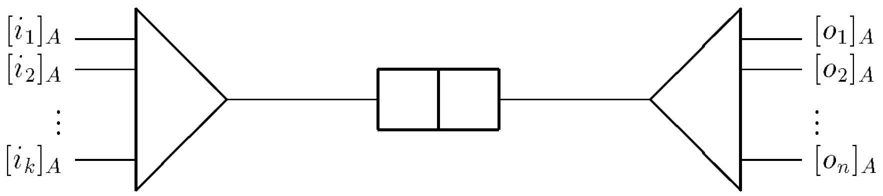


Thus, the multi way arbiter requires that the right-hand port and exactly one of the left-hand ports are involved in a transition. In case there is only one port to the left, the multi way arbiter represents as a link. In case there are no ports to the left, the multi way arbiter represents an arbiter with two ports interlinked: it can never perform an action involving the right-hand port. Graphically:



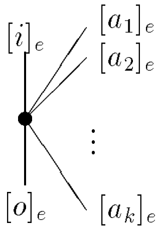
Let T be a finite self-loop free transition graph. A *canonic term* corresponding to T, written $CT(T)$, is a term corresponding to the network constructed as follows:

1. For each node A in T there is a subnetwork N_A as follows. Assume that the node A has incoming edges $i_1, \ldots, i_k$ and outgoing edges $o_1, \ldots, o_n$. Since T is self-loop free, all these edges are distinct. In the following we use port names enclosed in square brackets to represent fresh and unique port names. The subnetwork N_A has ports $[i_1]_A, \ldots, [i_k]_A$ and $[o_1]_A, \ldots, [o_n]_A$, and is defined as:



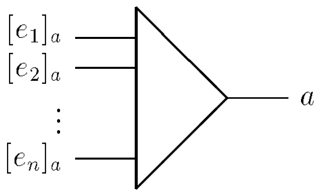
Thus, N_A consists of one alternator and (if $n, k \geq 1$) $n - k - 2$ arbiters. If A is the initial node of the transition graph, then the alternator is originally in its right-hand state (module AL′). This means that N_A may do a transition involving exactly one of its right-hand ports $[o_1]_A, \ldots, [o_n]_A$; after this transition the alternator will be in the left-hand state. If A is not the initial node, then the alternator is originally in its left-hand state (module AL). This means that N_A may do a transition involving exactly one of its left-hand ports $[i_1]_A, \ldots, [i_k]_A$; after this transition the alternator will be in the right-hand state.

2. For each edge $e : A \stackrel{\{a_1,\ldots,a_n\}}{\longrightarrow} A'$ in T there is a subnetwork N_e with $n+2$ ports $[i]_e$, $[o]_e$, and $[a_1]_e, \ldots, [a_n]_e$ defined with a multi way synchroniser:



Thus, an action of N_e must involve synchronisations on all ports.

3. For each $a \in L(T)$ there is a subnetwork N_a as follows. Let $e_1, \ldots, e_n$ be the edges in T where the action contains a. The subnetwork N_a has one port a and n ports $[e_1]_a, \ldots, [e_n]_a$ and is defined as:



Thus, if $n \geq 1$ then N_a contains $n-1$ arbiters. An action in N_a will involve the port a and exactly one of the left-hand ports.

4. The network corresponding to the canonic term $CT(T)$ is obtained by composing in parallel all subnetworks N_A, N_e, and N_a from steps 1–3 above, and linking the ports as follows: For each edge $e = A \stackrel{\alpha}{\longrightarrow} A'$ in T:

   (a) Link $[i]_e$ to $[e]_A$.

   (b) Link $[o]_e$ to $[e]_{A'}$.

   (c) For each $a \in \alpha$, link $[a]_e$ to $[e]_a$.

   Thus, the only non linked ports of $CT(T)$ are the ports in $L(T)$ in step 3 above. The only modules necessary in a canonic term are SY, AR, and AL (all occurrences of AL′ can be replaced by AL by reversing the order of the port names).

This completes the definition of canonic term. Actually, we have described how to build a "canonic network" rather than term, but the translation from this network to a term is trivial. The important property of a canonic term is the following:

Lemma 8 Let T be a finite self-loop free transition graph. Then

$$T \approx_b CT(T)$$

$\square$

Proof: Write T_B for T with initial state B. The alternators in $CT(T)$ all come from subterms corresponding to N_B defined in step 1 in the definition of canonic term. Thus, each node B in T corresponds to exactly one alternator, call it $\textsc{al}_B$, in $CT(T)$. By the definition of canonic term, it is easily seen that in $CT(T_B)$ all alternators except $\textsc{al}_B$ are in their left-hand states (i.e. they are instances of $\textsc{al}$), and $\textsc{al}_B$ is in its right-hand state (i.e. it is an instance of $\textsc{al}'$). Thus, $CT(T_B)$ and $CT(T_{B'})$ differ only in that $\textsc{al}_B$ and $\textsc{al}_{B'}$ are in different states.

Define the relation $\mathcal{R}$ by

$$\mathcal{R} = \{\langle T_B, CT(T_B)\rangle : B \in \text{the state space of } T\}$$

We first prove that $\mathcal{R}$ is a simulation. So assume $T_B \xrightarrow{\alpha} T_{B'}$. We will prove that $CT(T_B) \stackrel{\alpha}{\Longrightarrow} CT(T_{B'})$; in fact we will even prove the stronger $CT(T_B) \xrightarrow{\alpha} CT(T_{B'})$. Let e be the edge $T_B \xrightarrow{\alpha} T_{B'}$ in T, and let $\alpha = \{a_1, \ldots, a_n\}$. Consider the following transitions of the subterms of $CT(T_B)$, presented here in terms of the transitions from the subnetworks in the definition of canonic term:

1. In N_B the transition with action $\{[e]_B\}$.
2. In $N_{B'}$ the transition with action $\{[e]_{B'}\}$.
3. In N_e the transition with action $\{[i]_e, [o]_e, [a_1]_e, \ldots, [a_n]_e\}$.
4. if $n \geq 0$, then for all k $(1 \leq k \leq n)$: In N_{a_k} the transition with action $\{[e]_{a_k}, a_k\}$

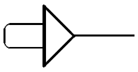
It is clear from the definition of canonic term, step 4, that all ports in these transitions are linked, with the exception $a_1, \ldots, a_n$. Also, in $CT(T_B)$ all subnetworks are in states where the transitions mentioned above are possible (here it is important that T is self-loop free, i.e. $B \neq B'$). Hence, $CT(T_B)$ can do a corresponding transition with action α. As a result of this transition, the alternators $\textsc{al}_B$ and $\textsc{al}_{B'}$ will change state, i.e. the resulting term is $CT(T_{B'})$. This proves $CT(T_B) \xrightarrow{\alpha} CT(T_{B'})$.

We next prove that $\mathcal{R}^{-1}$ is a simulation. So assume $CT(T_B) \xrightarrow{\alpha} C$. We must prove that for some B', $C = CT(T_{B'})$ and $T_B \stackrel{\alpha}{\Longrightarrow} T_{B'}$. First observe that at least one subnetwork N_e from step 2 in the construction of canonic term must take part in the transition $CT(T_B) \xrightarrow{\alpha} C$: if not, then because of the linking structure under step 4 no subnetwork has any transition. Let the ports of such an active N_e be $\{[i]_e, [o]_e, [a_1]_e, \ldots [a_n]_e\}$. Then $[i]_e$ must be linked with a right-hand port of N_B and $[o]_e$ with a left-hand port of $N_{B'}$ for some $B' \neq B$. From the definition of N_B, at most one such subnetwork N_e can be involved in this transition. The effect of the transition is that $\textsc{al}_B$ and $\textsc{al}_{B'}$ change states, i.e. the resulting term C is $CT(T_{B'})$. Furthermore, the action α of the transition must be $\{a_1, \ldots, a_n\}$. The reason for the existence of N_e and its links with N_B and $N_{B'}$ must be that T has an edge $B \xrightarrow{\alpha} B'$. This, of course, implies $T_B \xrightarrow{\alpha} T_{B'}$ which implies $T_B \stackrel{\alpha}{\Longrightarrow} T_{B'}$.

This concludes the proof that $\mathcal{R}$ is a bisimulation, and hence the proof of Lemma 8. An attentive reader will have noted that we have proved the relation $\mathcal{R}$ to be a "strong" bisimulation in the sense of [14]; in fact Theorem 2 holds also for strong bisimulation equivalence. $\square$

Lemma 9 $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ is a proper basis for the set of finite-state terms. $\square$

To prove $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ a basis for the finite-state terms, we construct for any given finite-state term A an equivalent term in the closure of $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ as follows. Let T be the transition graph corresponding to A. Clearly, $A \approx_b T$. By Lemma 7 there exists a self-loop free transition graph T' such that $T \approx_b T'$. Note that if A is finite-state, then T and T' are finite-state. Let CT be a canonic term corresponding to T'. Note that CT is in the closure of $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$. By Lemma 8, $T' \approx_b CT$, thus $A \approx_b CT$. Also, from the constructions it is obvious that $L(CT) = L(T') = L(T) \subseteq L(A)$. Let $L^+ = L(A) - L(CT)$. For each element a in L^+ define a term corresponding to a network N_a consisting of one arbiter:



Let CT^+ be CT in parallel with all terms corresponding to N_a for $a \in L^+$. Then $CT^+ \approx_b CT$, so $CT^+ \approx_b A$. Also,

$$L(CT^+) = L(CT) \cup L^+ = L(A)$$

This proves $CT^+ \approx A$.

To prove $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ a proper basis, it suffices to observe that $\textsc{sy}(a, b, c)$, $\textsc{ar}(a, b, c)$, and $\textsc{al}(a, b)$ are all finite-state for any a, b, c and that the operators preserve the property of being finite-state. $\square$

It remains to prove that $\{\textsc{sy}, \textsc{ar}, \textsc{al}\}$ is independent. This follows from the next three lemmas.

Lemma 10 $\textsc{al}$ is independent of $\{\textsc{sy}, \textsc{ar}\}$. $\square$

Proof: We prove that no term in the closure of $\{\textsc{sy}, \textsc{ar}\}$ is equivalent with $\textsc{al}(a, b)$. Say that a term A is *single state* if for some term C such that $A =_T C$ it holds

$$\text{for all } \alpha\text{: } (C \xrightarrow{\alpha} C' \text{ implies } C = C')$$

The lemma then follows from the following observations:

1. The units $\textsc{sy}(a, b, c)$ and $\textsc{ar}(a, b, c)$ are single state for any a, b, c.

2. If A and B are single state and $L(A) \cap L(B) = \emptyset$ then $A|B$ is single state.

3. If A is single state and $a, b \in L(A)$ and $a \neq b$ then $A\langle a^\frown b\rangle$ is single state.

4. If A is single state and $A =_T B$ then B is single state.

5. The unit $\textsc{al}(a,b)$ is not single state.

The proofs of these observations are all simple. By 1-4 above, if a term is equivalent with a term in the closure of $\{\textsc{sy},\textsc{ar}\}$, then it is single state. But by 5, $\textsc{al}(a,b)$ is not single state. $\square$

For the next lemmas we introduce some additional notation. We use σ and ρ to range over sequences of nonempty actions, the concatenation of σ and ρ is written $\sigma\rho$, and the sequence consisting of only one action α is also written α. If L is a set of port names we write $\sigma \setminus L$ for the sequence obtained by removing all names in L from all actions in σ, where actions which thus become empty are removed. Note that $\setminus$ distributes over concatenation. The set of sequences $\sigma||\alpha$ is defined to contain exactly the sequences which can be obtained from σ by conjoining α to some of the actions, and also possibly interposing actions α between actions in σ. Note that if $\sigma' \in \sigma||\alpha$ then $\sigma' \setminus \alpha = \sigma$.

We will use the following fact, which is an easy consequence of the rules for the parallel operator, remembering that A and B must have disjoint sorts in $A|B$:

$$A|B \stackrel{\sigma}{\Longrightarrow} A'|B' \text{ iff } A \stackrel{\sigma\setminus L(B)}{\Longrightarrow} A' \text{ and } B \stackrel{\sigma\setminus L(A)}{\Longrightarrow} B'$$

We will also use the following property of the linking operators:

$$A\langle a^\frown b\rangle \stackrel{\sigma}{\Longrightarrow} A'\langle a^\frown b\rangle \text{ iff } \exists \sigma' \in \sigma||\{a,b\} : A \stackrel{\sigma'}{\Longrightarrow} A'$$

Lemma 11 $\textsc{ar}$ is independent of $\{\textsc{sy},\textsc{al}\}$. $\square$

Proof: We prove that no term in the closure of $\{\textsc{sy},\textsc{al}\}$ is equivalent with $\textsc{ar}(a,b,c)$.

Say that a term A is *intersection closed* if for all $\alpha,\beta \subseteq L(A)$ and all sequences of actions σ,ρ it holds that

$$[A \stackrel{\sigma\alpha}{\Longrightarrow} \text{ and } A \stackrel{\rho\beta}{\Longrightarrow}] \text{ implies } [A \stackrel{\sigma(\alpha\cap\beta)}{\Longrightarrow} \text{ and } A \stackrel{\rho(\alpha\cap\beta)}{\Longrightarrow}]$$

Here, if $\alpha \cap \beta = \emptyset$ then $\sigma(\alpha\cap\beta)$ is just σ, and similarly for ρ. Intuitively, if a term is intersection closed and can do an action α (after a sequence σ) and an action β (after ρ), then it can also do the intersection of α and β (after both σ and ρ). The lemma follows from the following observations:

1. The units $\textsc{sy}(a,b,c)$ and $\textsc{al}(a,b)$ are intersection closed for all a,b,c.

2. If A and B are intersection closed and $L(A)\cap L(B)=\emptyset$ then $A|B$ is intersection closed.

3. If A is intersection closed and $a,b \in L(A)$ and $a \neq b$ then $A\langle a^\frown b\rangle$ is intersection closed.

4. If A is intersection closed and $A =_T B$ then B is intersection closed.

5. The unit $\textsc{ar}(a,b,c)$ is not intersection closed.

The proofs of the nontrivial observations are:

2. Let A and B be intersection closed with $L(A) \cap L(B) = \emptyset$. Assume $A|B \stackrel{\sigma\alpha}{\Longrightarrow}$ and $A|B \stackrel{\rho\beta}{\Longrightarrow}$. We then have that

$$A \stackrel{(\sigma\alpha)\setminus L(B)}{\Longrightarrow} \text{ and } A \stackrel{(\rho\beta)\setminus L(B)}{\Longrightarrow} \text{ and } B \stackrel{(\sigma\alpha)\setminus L(A)}{\Longrightarrow} \text{ and } B \stackrel{(\rho\beta)\setminus L(A)}{\Longrightarrow}$$

From the fact that A and B are intersection closed, and using the distributivity of $\setminus$, we get that

$$A \stackrel{(\sigma(\alpha\cap\beta))\setminus L(B)}{\Longrightarrow} \text{ and } A \stackrel{(\rho(\alpha\cap\beta))\setminus L(B)}{\Longrightarrow} \text{ and } B \stackrel{(\sigma(\alpha\cap\beta))\setminus L(A)}{\Longrightarrow} \text{ and } B \stackrel{(\rho(\alpha\cap\beta))\setminus L(A)}{\Longrightarrow}$$

This implies as required that $A|B \stackrel{\sigma(\alpha\cap\beta)}{\Longrightarrow}$ and $A|B \stackrel{\rho(\alpha\cap\beta)}{\Longrightarrow}$

3. Let A be intersection closed with $a, b \in L(A)$. Assume that $A\langle a^\frown b\rangle \stackrel{\sigma\alpha}{\Longrightarrow}$ and $A\langle a^\frown b\rangle \stackrel{\rho\beta}{\Longrightarrow}$. Note that neither a nor b can then occur in any of $\sigma, \alpha, \rho, \beta$. The transitions from $A\langle a^\frown b\rangle$ imply that for some $\sigma', \alpha', \rho', \beta'$

$$A \stackrel{\sigma'\alpha'}{\Longrightarrow} \text{ and } A \stackrel{\rho'\beta'}{\Longrightarrow}$$

where $\sigma' \in \sigma||\{a, b\}$, and $\rho' \in \rho||\{a, b\}$, and α' is α or $\alpha \cup \{a, b\}$, and β' is β or $\beta \cup \{a, b\}$. Since A is intersection closed we get that

$$A \stackrel{\sigma'(\alpha'\cap\beta')}{\Longrightarrow} \text{ and } A \stackrel{\rho'(\alpha'\cap\beta')}{\Longrightarrow}$$

Since for all actions in these transitions it holds that a is in that action if and only if b is in it, we get that

$$A\langle a^\frown b\rangle \stackrel{(\sigma'(\alpha'\cap\beta'))\setminus\{a,b\}}{\Longrightarrow} \text{ and } A\langle a^\frown b\rangle \stackrel{(\rho'(\alpha'\cap\beta'))\setminus\{a,b\}}{\Longrightarrow}$$

Finally from the construction of $\sigma', \alpha', \rho', \beta'$ we infer that

$$\begin{aligned} \sigma' \setminus \{a, b\} &= \sigma \\ \rho' \setminus \{a, b\} &= \rho \\ (\alpha' \cap \beta') \setminus \{a, b\} &= \alpha \cap \beta \end{aligned}$$

So this proves $A\langle a^\frown b\rangle \stackrel{\sigma(\alpha\cap\beta)}{\Longrightarrow}$ and $A\langle a^\frown b\rangle \stackrel{\rho(\alpha\cap\beta)}{\Longrightarrow}$ as required.

By 1-4 above, if a term is trace equivalent with a term in the closure of $\{\textsc{sy}, \textsc{al}\}$, then it is intersection closed. But by 5, $\textsc{ar}(a, b, c)$ is not intersection closed. This concludes the proof of Lemma 11. □

Lemma 12 $\textsc{sy}$ is independent of $\{\textsc{ar}, \textsc{al}\}$. □

Proof: We prove that no term in the closure of $\{\text{AR}, \text{AL}\}$ is equivalent with $\text{SY}(a, b, c)$. Let A be a term and $\alpha \subseteq L(A)$. Let Γ be a partition of α into nonempty disjoint sets. Say that (A, σ, Γ) has the *arbitrary union* property, written $AU(A, \sigma, \Gamma)$, if

$$\text{For all nonempty } \Delta \subseteq \Gamma : \quad \gamma = \bigcup_{\delta \in \Delta} \delta \quad \text{implies} \quad A \stackrel{\sigma\gamma}{\Longrightarrow}$$

This means that after doing σ, A may do actions involving arbitrary unions of elements in the partition Γ. Say that A is *partition closed* if for all σ, α such that $|\alpha| \geq 3$ (here $|\alpha|$ is the cardinality of α) and $A \stackrel{\sigma\alpha}{\Longrightarrow}$ there exists a partition Γ of α with $|\delta| \leq 2$ for all $\delta \in \Gamma$, such that $AU(A, \sigma, \Gamma)$. This means that if after σ, A can do an action with more than two ports, then this action can be partitioned into actions with at most two ports satisfying the arbitrary union property.

The lemma follows from the following observations:

1. The units $\text{AR}(a, b, c)$ and $\text{AL}(a, b)$ are partition closed for all a, b, c.

2. If A and B are partition closed and $L(A) \cap L(B) = \emptyset$ then $A|B$ is partition closed.

3. If A is partition closed and $a, b \in L(A)$ and $a \neq b$ then $A\langle a \frown b\rangle$ is partition closed.

4. If A is partition closed and $A =_T B$ then B is partition closed.

5. The unit $\text{SY}(a, b, c)$ is not partition closed.

The proofs of the nontrivial observations are:

2. Let A and B be partition closed, $L(A) \cap L(B) = \emptyset$, $A|B \stackrel{\sigma\alpha}{\Longrightarrow}$, and $|\alpha| \geq 3$. We first get that

   $$A \stackrel{(\sigma\alpha)\setminus L(B)}{\Longrightarrow} \quad \text{and} \quad B \stackrel{(\sigma\alpha)\setminus L(A)}{\Longrightarrow}$$

   Put $\alpha_1 = \alpha \cap L(A)$ and $\alpha_2 = \alpha \cap L(B)$. Then $\alpha_1 \cap \alpha_2 = \emptyset$ and $\alpha_1 \cup \alpha_2 = \alpha$, so $|\alpha_1| + |\alpha_2| = |\alpha| \geq 3$. Putting $\sigma_1 = \sigma \setminus L(B)$ and $\sigma_2 = \sigma \setminus L(A)$ we can write the transitions above as

   $$A \stackrel{\sigma_1\alpha_1}{\Longrightarrow} \quad \text{and} \quad B \stackrel{\sigma_2\alpha_2}{\Longrightarrow}$$

   There are four cases:

   (a) $1 \leq |\alpha_1|, |\alpha_2| \leq 2$. Then the partition

   $$\Gamma = \{\alpha_1, \alpha_2\}$$

   of α satisfies $AU(A|B, \sigma, \Gamma)$: if γ is an arbitrary union of actions in Γ then γ is either α or α_1 or α_2, and it is clear from the transitions above that $A|B \stackrel{\sigma\gamma}{\Longrightarrow}$ for those γ.

   (b) $|\alpha_1| \geq 3, |\alpha_2| \leq 2$. Since A is partition closed, there exists a partition Γ_1 of α_1 satisfying $AU(A, \sigma_1, \Gamma_1)$. Then, the partition

$$\Gamma = \begin{cases} \Gamma_1 \cup \{\alpha_2\} & \text{if } \alpha_2 \neq \emptyset \\ \Gamma_1 & \text{if } \alpha_2 = \emptyset \end{cases}$$

of α satisfies $AU(A|B, \sigma, \Gamma)$. To prove this, assume that $\Delta \subseteq \Gamma$. Then, for some $\Delta_1 \subseteq \Gamma_1$ it holds either $\Delta = \Delta_1$ or $\Delta = \Delta_1 \cup \{\alpha_2\}$ or $\Delta = \{\alpha_2\}$. Since $AU(A, \sigma_1, \Gamma_1)$ we know that $A \stackrel{\sigma_1\gamma_1}{\Longrightarrow}$ for $\gamma_1 = \bigcup_{\delta_1 \in \Delta_1} \delta_1$, and with $B \stackrel{\sigma_2\alpha_2}{\Longrightarrow}$ this implies $A|B \stackrel{\sigma\gamma}{\Longrightarrow}$ for $\gamma = \bigcup_{\delta \in \Delta} \delta_1$ as required.

(c) $|\alpha_1| \leq 2, |\alpha_2| \geq 3$. Symmetric with (b) above.

(d) $|\alpha_1| \geq 3, |\alpha_2| \geq 3$. Since A and B are partition closed, there exist a partition Γ_1 of α_1 and a partition Γ_2 of α_2, satisfying $AU(A, \sigma_1, \Gamma_1)$ and $AU(B, \sigma_2, \Gamma_2)$ respectively. Then the partition

$$\Gamma = \Gamma_1 \cup \Gamma_2$$

of α satisfies $AU(A|B, \sigma, \Gamma)$. A proof of this is similar to case (b) above.

3. Let A be partition closed, $a, b \in L(A)$, $A\langle a^\frown b\rangle \stackrel{\sigma\alpha}{\Longrightarrow}$ and $|\alpha| \geq 3$. Then $a, b \notin \alpha$. So for some $\sigma' \in \sigma||\{a, b\}$ it holds either $A \stackrel{\sigma'\alpha}{\Longrightarrow}$ or $\mathrm{A} \stackrel{\sigma'(\alpha\cup\{a,b\})}{\Longrightarrow}$. We consider these cases in turn.

(a) $A \stackrel{\sigma'\alpha}{\Longrightarrow}$. Since A is partition closed, there is a partition Γ of α satisfying $AU(A, \sigma', \Gamma)$. We prove that also $AU(A\langle a^\frown b\rangle, \sigma, \Gamma)$. Let γ be an arbitrary union of elements in Γ; we then know that $A \stackrel{\sigma'\gamma}{\Longrightarrow}$. Since neither a nor b is in any element in Γ it follows that $a, b \notin \gamma$. From the construction of σ' we then infer $A\langle a^\frown b\rangle \stackrel{\sigma\gamma}{\Longrightarrow}$.

(b) $A \stackrel{\sigma'(\alpha\cup\{a,b\})}{\Longrightarrow}$. Since A is partition closed there is a partition $\Gamma' = \{\alpha_1, \ldots, \alpha_n\}$ of $\alpha \cup \{a, b\}$ satisfying $AU(A, \sigma', \Gamma')$. Thus, for some $i, j \leq n$ it must hold that $a \in \alpha_i$ and $b \in \alpha_j$. There are two subcases:

i. $i = j$. Then since $|\alpha_i| \leq 2$ it must hold that $\alpha_i (= \alpha_j) = \{a, b\}$. Let

$$\Gamma = \Gamma' - \{\alpha_i\}$$

Thus Γ is a partition of α. We prove $AU(A\langle a^\frown b\rangle, \sigma, \Gamma)$. Let γ be an arbitrary union of elements of Γ. Then γ is also a union of elements of Γ', whence $A \stackrel{\sigma'\gamma}{\Longrightarrow}$. Since $a, b \notin \gamma$ this implies $A\langle a^\frown b\rangle \stackrel{\sigma\gamma}{\Longrightarrow}$.

ii. $i \neq j$. Let

$$\beta = (\alpha_i \cup \alpha_j) - \{a, b\}$$

Note that $|\beta| \leq 2$. Let

$$\Gamma = \Gamma' - \{\alpha_i, \alpha_j\} \cup \{\beta\}$$

Thus Γ is a partition of α. We prove $AU(A\langle a^\frown b\rangle, \sigma, \Gamma)$. Let γ be an arbitrary union of elements of Γ. There are two subsubcases:

A. $\beta \not\subseteq \gamma$. Then γ is also a union of elements of Γ', whence $A \stackrel{\sigma'\gamma}{\Longrightarrow}$. Since $a, b \notin \gamma$ this implies $A\langle a^\frown b\rangle \stackrel{\sigma\gamma}{\Longrightarrow}$.

B. $\beta \subseteq \gamma$. Let

$\gamma' = (\gamma - \beta) \cup \alpha_i \cup \alpha_j$

Now γ' is a union of elements of Γ'. Thus $A \stackrel{\sigma'\gamma'}{\Longrightarrow}$ whence, since $a, b \in \gamma'$, it holds that $A\langle a \frown b \rangle \stackrel{\sigma(\gamma' - \{a,b\})}{\Longrightarrow}$. But

$$\begin{aligned} \gamma' - \{a, b\} &= ((\gamma - \beta) \cup \alpha_i \cup \alpha_j) - \{a, b\} \\ &= \gamma \end{aligned}$$

whence it follows $A\langle a \frown b \rangle \stackrel{\sigma\gamma}{\Longrightarrow}$.

By 1-4 above, if a term is equivalent with a term in the closure of $\{\text{AR}, \text{AL}\}$, then it is partition closed. But by 5, $\text{SY}(a, b, c)$ is not partition closed. This concludes the proof of Lemma 12. □

The proof of Theorem 2 now is immediate from Lemmas 9–12.

Proof of Theorem 3

Let f be a function from port names to port names, and let f also extend pointwise to actions by $f(\alpha) = \{f(a) : a \in \alpha\}$. Consider the port name relabelling operator $[f]$ defined by the rules

$$\frac{A \xrightarrow{\alpha} A'}{A[f] \xrightarrow{f(\alpha)} A'[f]}$$

Lemma 13 Let L be a sort and let $[f]$ be a port name relabelling. Then there exists a unary context $\mathcal{S}_{L,[f]}$ over $\{\text{SY}, \text{AR}, \text{AL}\}$ such that

$$\mathcal{S}_{L,[f]}(A) \approx A[f]$$

holds for all A of sort L. □

Proof: For each a in L, let a_1, a_2, and a_3 be new port names. Let $C_{L,[f]}$ and $C'_{L,[f]}$ be terms with exactly the following transitions

$$\begin{array}{ll} C_{L,[f]} \xrightarrow{\alpha} C_{L,[f]} & \text{for all } \alpha \text{ such that } a_1 \in \alpha \text{ iff } a_2 \in \alpha \text{ (for all } a \in L) \\ C'_{L,[f]} \xrightarrow{\beta} C'_{L,[f]} & \text{for all } \beta \text{ such that } a_3 \in \beta \text{ iff } f(a) \in \beta \text{ (for all } a \in L) \end{array}$$

From Theorem 2 we know that such terms (or at least observationally equivalent terms) exist in $\mathcal{T}_{\{\text{SY},\text{AR},\text{AL}\}}$. Let $\langle \mathit{LINKS} \rangle$ be the sequence of linking operators $\langle a \frown a_1 \rangle$ for all $a \in L$, and let $\langle \mathit{LINKS}' \rangle$ be the sequence $\langle a_2 \frown a_3 \rangle$ for all $a \in L$. Define $\mathcal{S}_{L,[f]}$ by

$$\mathcal{S}_{L,[f]}(A) \;=\; ((A | C_{L,[f]})\langle \mathit{LINKS} \rangle | C'_{L,[f]})\langle \mathit{LINKS}' \rangle$$

We can now prove that $\mathcal{S}_{L,[f]}(A) \approx A[f]$ by proving that

$$\{(A[f], \mathcal{S}_{L,[f]}(A)) : A \text{ has sort } L\}$$

is a bisimulation; this is straightforward since

$$A[f] \xrightarrow{f(\alpha)} A'[f]$$
iff
$$A \xrightarrow{\alpha} A'$$
iff
$$((A|C_{L,[f]})\langle LINKS\rangle|C'_{L,[f]})\langle LINKS'\rangle \xrightarrow{f(\alpha)} ((A'|C_{L,[f]})\langle LINKS\rangle|C'_{L,[f]})\langle LINKS'\rangle$$

$\square$

Proof of Theorem 3: Let **op** be an asynchronous set of operators of sort $L_1, \ldots L_n \to L$. Assume fresh and distinct port names a_k and a_k' for each $k \leq n$ and $a \in L_k$. Let $\mathbf{M} = \{M_{op} : op \in \mathbf{op}\}$ where each M_{op} has an instance C_{op} with precisely the following behaviour: for each rule

$$\frac{A_1 \xrightarrow{\mu_1} A_1' \quad \cdots \quad A_n \xrightarrow{\mu_n} A_n'}{op(\widetilde{A}) \xrightarrow{\alpha} op'(\widetilde{A'})}$$

in the semantics of op, there is a transition

$$C_{op} \xrightarrow{\beta} C_{op'}$$

where

$$\beta = \alpha \cup \bigcup_{1 \leq k \leq n} \bigcup_{a \in \mu_k} \{a_k\}$$

Clearly this can be achieved by choosing $\mathbf{M}$ in an appropriate way. Note that $\mathbf{M}$ is finite, and hence all modules in $\mathbf{M}$ are finite-state, if **op** is finite.

For each $k \leq n$, let $[f]_k$ be a port name relabelling which relabels a to a_k', for all $a \in L_k$. Let $\langle LINKS\rangle$ be a sequence of linking operators containing for all $k \leq n$ and $a \in L_k$ the link $\langle a_k' \frown a_k\rangle$. The defining context for op can now be written:

$$\mathcal{C}_{op}(\widetilde{A}) \;=\; (C_{op}|\mathcal{S}_{L_1,[f]_1}(A_1)|\cdots|\mathcal{S}_{L_n,[f]_n}(A_n))\langle LINKS\rangle$$

where $\mathcal{S}_{L_k,[f]_k}$ are defined as in Lemma 13. Intuitively, C_{op} encodes the rules of op: it controls the transitions of the operands and generates the appropriate external actions. The important point is that if $op(\widetilde{A})$ can do an action α on condition that all A_k can do μ_k, then C_{op} can do an action containing α provided it does it in synchrony with an action representing the ports in all μ_k.

We will now prove that this indeed is a defining context. Let

$$\mathcal{C}'_{op}(\widetilde{A}) \;=\; (C_{op}|A_1[f]_1|\cdots|A_n[f]_n)\langle LINKS\rangle$$

We will prove that

$$\mathcal{R} \;=\; (\mathcal{C}'_{op}(\widetilde{A}),\ op(\widetilde{A})) \;:\; op \in \mathbf{op},\ \widetilde{A} \text{ has sort } \widetilde{L}\}$$

is a bisimulation; the result then follows from Lemma 13 and the fact that $\simeq$ is a congruence.

We first prove that $\mathcal{R}$ is a simulation. So assume $\mathcal{C}'_{op}(\widetilde{A}) \xrightarrow{\alpha} B$. We have to prove that for some $\widetilde{A'}$ it holds $B = \mathcal{C}'_{op'}(\widetilde{A'})$ and $op(\widetilde{A}) \overset{\alpha}{\Longrightarrow} op'(\widetilde{A'})$. There are two cases:

1. The transition $\mathcal{C}'_{op}(\widetilde{A}) \xrightarrow{\alpha} B$ does not involve any transition in C_{op}. By definition of $\mathcal{C}'_{op}(\widetilde{A})$ and the rules for parallel and linking, this implies that $\alpha = \emptyset$ and that no synchronisation occurs along any of the links connecting any of $\widetilde{A}$ with any other subterm; in other words, $A_k \xrightarrow{\emptyset} A'_k$ or $A_k = A'_k$ holds for all k and $B = (C'_{op}|A'_1[f]_1|\cdots|A'_n[f]_n)\langle LINKS\rangle = \mathcal{C}_{op}(\widetilde{A'})$. Since op is asynchronous, it then holds that $op(\widetilde{A}) \xrightarrow{\emptyset} op(\widetilde{A'})$ or $op(\widetilde{A}) = op(\widetilde{A'})$; this implies as required that $op(\widetilde{A}) \overset{\emptyset}{\Longrightarrow} op(\widetilde{A'})$.

2. The transition $\mathcal{C}'_{op}(\widetilde{A}) \xrightarrow{\alpha} B$ involves a transition $C_{op} \xrightarrow{\beta} C'$. By definition of $\mathcal{C}'_{op}(\widetilde{A})$ and the rules for parallel and linking, we get that there are $\widetilde{A'}, \mu_1, \ldots, \mu_n$ such that

$$
\begin{array}{l}
A_1 \xrightarrow{\mu_1} A'_1 \text{ or } \mu_1 = *, A_1 = A'_1 \\
\quad\vdots \\
A_n \xrightarrow{\mu_n} A'_n \text{ or } \mu_n = *, A_n = A'_n \\
B = (C'|A'_1[f]_1|\cdots|A'_n[f]_n)\langle LINKS\rangle \\
a \in \mu_k \text{ iff } a_k \in \beta
\end{array}
$$

From the transitions of C_{op} we get that $C' = C_{op'}$ for some op', thus $B = \mathcal{C}'_{op'}(\widetilde{A'})$, and that the transition $C_{op} \xrightarrow{\beta} C_{op'}$ corresponds to a rule in the definition of op. There may be several such rules; if so the rules differ only in that $*$ replaces $\emptyset$ or vice versa. Since op is asynchronous one such rule must be $\langle op, \mu_1, \ldots, \mu_n, \alpha, op'\rangle$. Since the premises of that rule are fulfilled, we can conclude $op(\widetilde{A}) \xrightarrow{\alpha} op'(\widetilde{A'})$.

We next prove that $\mathcal{R}^{-1}$ is a simulation. The proof is essentially the proof above run backwards. So assume that $op(\widetilde{A}) \xrightarrow{\alpha} B$. We must prove that $B = op'(\widetilde{A'})$ and $\mathcal{C}'_{op}(\widetilde{A}) \overset{\alpha}{\Longrightarrow} \mathcal{C}'_{op'}(\widetilde{A'})$. The transition $op(\widetilde{A}) \xrightarrow{\alpha} B$ must be inferred from some rule $\langle op, \mu_1, \ldots, \mu_n, \alpha, op'\rangle$. Thus $B = op'(\widetilde{A'})$; moreover the premises of this rule must hold, so $A_k \xrightarrow{\mu_k} A'_k$ (or $\mu_k = *$ and $A_k = A'_k$) for all $k \leq n$. From the existence of this rule we know that $C_{op} \xrightarrow{\beta} C_{op'}$ where $a \in \mu_k$ iff $[a]_k \in \beta$. The rules for parallel and linking then give that $\mathcal{C}'_{op}(\widetilde{A}) \xrightarrow{\alpha} \mathcal{C}'_{op'}(\widetilde{A'})$. $\Box$

Proof of Theorem 6

We prove by structural induction on contexts that for each context $\mathcal{C}$ there is an asynchronous operator $op^{\mathcal{C}}$ such that $\mathcal{C}$ is a $\approx$-defining context for $op^{\mathcal{C}}$. If the modules in $\mathcal{C}$ are finite state then $op^{\mathcal{C}}$ will also be finite state. The idea of the induction is that the identity context and instances of modules can be given equivalent asynchronous operators, and parallel composition and linking preserve this property. There are four cases to consider:

1. $\mathcal{C} = \bullet$, i.e. $\mathcal{C}(A) = A$. The operator id defined by the rules

$$\frac{A \xrightarrow{\alpha} A'}{id(A) \xrightarrow{\alpha} id(A')}$$

(one rule for each $\alpha \subseteq L(A)$) is asynchronous and finite-state; moreover $id(A) \approx A$. Choose $op^{\mathcal{C}} = id$.

2. $\mathcal{C} = M(a_1, \ldots, a_n)$. But $M(a_1, \ldots, a_n)$ can be viewed as a 0-ary operator, and as such it is trivially asynchronous and finite state if M is finite state. Choose $op^{\mathcal{C}} = M(a_1, \ldots, a_n)$.

3. $\mathcal{C} = \mathcal{C}_1 | \mathcal{C}_2$. We then know by induction that there are asynchronous operators $op^{\mathcal{C}_1}$ and and $op^{\mathcal{C}_2}$ with $\approx$-defining contexts $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. Call their corresponding asynchronous sets of rules R_1 and R_2. Introduce new operators named "(op_1, op_2)" for all op_i that occur in R_i for $i = 1, 2$. Construct a set of rules R for these operators as follows. For each rule $\langle op_1, \mu_{1,1}, \ldots, \mu_{1,n_1}, \alpha_1, op_1' \rangle$ in R_1 and each rule $\langle op_2, \mu_{2,1}, \ldots, \mu_{2,n_2}, \alpha_2, op_2' \rangle$ in R_2 let there be the following three rules in R:

   (a) $\langle (op_1, op_2), \mu_{1,1}, \ldots, \mu_{1,n_1}, *, \ldots, *, \alpha_1, (op_1', op_2) \rangle$
   (b) $\langle (op_1, op_2), *, \ldots, *, \mu_{2,1}, \ldots, \mu_{2,n_2}, \alpha_2, (op_1, op_2') \rangle$
   (c) $\langle (op_1, op_2), \mu_{1,1}, \ldots, \mu_{1,n_1}, \mu_{2,1}, \ldots, \mu_{2,n_2}, \alpha_1 \cup \alpha_2, (op_1', op_2') \rangle$

   If R_1 and R_2 are asynchronous then so is R, and if they are finite then so is R. In particular $(op^{\mathcal{C}_1}, op^{\mathcal{C}_2})$ is an asynchronous operator, and finite-state if $op^{\mathcal{C}_1}$ and $op^{\mathcal{C}_2}$ are finite-state. It is now straightforward to see that

   $$\{(op_1(\widetilde{A_1}) | op_2(\widetilde{A_2}), (op_1, op_2)(\widetilde{A_1}\widetilde{A_2})) : op_i \text{ is used in } R_i \text{ and } \widetilde{A_1}\widetilde{A_2} \text{ has correct sort}\}$$

   is a bisimulation. So for all $\widetilde{A_1}\widetilde{A_2}$ of correct sort it holds that

   $$(op^{\mathcal{C}_1}, op^{\mathcal{C}_2})(\widetilde{A_1}\widetilde{A_2}) \approx op^{\mathcal{C}_1}(\widetilde{A_1}) | op^{\mathcal{C}_2}(\widetilde{A_2}) \approx \mathcal{C}_1(\widetilde{A_1}) | \mathcal{C}_2(\widetilde{A_2}) = \mathcal{C}(\widetilde{A_1}\widetilde{A_2})$$

   This proves that $\mathcal{C}$ is a $\approx$-defining context for $(op^{\mathcal{C}_1}, op^{\mathcal{C}_2})$; choose $op^{\mathcal{C}} = (op^{\mathcal{C}_1}, op^{\mathcal{C}_2})$.

4. $\mathcal{C} = \mathcal{C}_1 \langle a \frown b \rangle$. We then know by induction that there is an asynchronous operator $op^{\mathcal{C}_1}$ with $\approx$-defining context $\mathcal{C}_1$. Call the corresponding asynchronous set of rules R_1. Introduce new operators named "$op_1 \langle a \frown b \rangle$" for all op_1 that occur in R_1. Construct a set of rules R for these operators as follows. For each rule $\langle op_1, \mu_{1,1}, \ldots, \mu_{1,n_1}, \alpha_1, op_1' \rangle$ in R_1 such that $a, b \notin \alpha_1$ let there be a rule $\langle op_1 \langle a \frown b \rangle, \mu_{1,1}, \ldots, \mu_{1,n_1}, \alpha_1, op_1' \langle a \frown b \rangle \rangle$ in R. Also, for each rule $\langle op_1, \mu_{1,1}, \ldots, \mu_{1,n_1}, \alpha_1, op_1' \rangle$ such that $a, b \in \alpha_1$ let there be a rule $\langle op_1 \langle a \frown b \rangle, \mu_{1,1}, \ldots, \mu_{1,n_1}, \alpha_1 - \{a, b\}, op_1' \langle a \frown b \rangle \rangle$ in R. If R_1 is asynchronous then so is R, and if it is finite then so is R. In particular $op^{\mathcal{C}_1} \langle a \frown b \rangle$ is an asynchronous operator, and finite-state if $op^{\mathcal{C}_1}$ is finite-state. It is now straightforward to see that

   $$\{(op_1(\widetilde{A}) \langle a \frown b \rangle, op_1 \langle a \frown b \rangle (\widetilde{A})) : op_1 \text{ is used in } R_1 \text{ and } \widetilde{A} \text{ has correct sort}\}$$

   is a bisimulation. So for all $\widetilde{A}$ of correct sort it holds that

   $$op^{\mathcal{C}_1} \langle a \frown b \rangle (\widetilde{A}) \approx op^{\mathcal{C}_1}(\widetilde{A}) \langle a \frown b \rangle \approx \mathcal{C}_1(\widetilde{A}) \langle a \frown b \rangle = \mathcal{C}(\widetilde{A})$$

   This proves that $\mathcal{C}$ is a $\approx$-defining context for $op^{\mathcal{C}_1} \langle a \frown b \rangle$; choose $op^{\mathcal{C}} = op^{\mathcal{C}_1} \langle a \frown b \rangle$. $\square$

Proof of the Results in Table 3

We have already proved that the Relabelling operator $[f]$ is definable (Lemma 13). We begin by formally defining the other operators in Table 3. To avoid confusion we write the CSP hiding operator as "$\backslash\backslash a$", and CCS parallel composition as $|^c$.

Strictly speaking, these are all families of operators (in the same way as $|$ is a family of operators); there is one operator for each (combination of) sort(s) of the operand(s). We say that such a family is definable if all members of it are definable. For convenience we will continue to refer to such families as "operators". The definitions of resulting sorts are:

- For binary operators *op*: $L(op(A, B)) = L(A) \cup L(B)$
- $L(a.A) = L(a * A) = L(A) \cup \{a\}$
- $L(A\backslash a) = L(A\backslash\backslash a) = L(A) - \{a\}$

For the purpose of CCS parallel, assume a bijection $^-$ on Λ which is its own inverse. For the purpose of CSP sequential composition, assume a distinguished port name $\surd$ in Λ. For the purpose of SCCS synchronous parallel, assume a function s which when given two actions α and β returns an action representing the simultaneous occurrence of α and β. We disqualify trivial such functions s by requiring that for some α it holds that $s(\emptyset, \alpha) \neq \emptyset$.

Rules defining the operational semantics of the operators are presented in Table 4. Rules marked † have a symmetric form. Note that we have modified some of the usual rules slightly in order to make them fit in our framework (where actions are sets of port names). For example, for most operators we only consider the cases where an action contains at most one port name. The reason is that in the original algebras actions contain at most one port name. But other formulations of the rules are also possible; these variations do not significantly affect the results.

In what follows we assume that at least all finite state terms are definable. We say that an operator is definable (without qualifying equivalence) to mean that it is $\approx$-definable.

Lemma 14 The operators $\backslash a$, $\backslash\backslash a$, $|^c$, and $||$ are definable. $\square$

Proof: The proof goes by giving asynchronous finite-state definitions to equivalent operators. The situations for all these operators are similar; we present only the proof for $|^c$ here. Define a new operator $|^{c\prime}$ which inherits the rules for $|^c$ and has one additional rule

$$[\,|^{c\prime}]: \quad \frac{A \xrightarrow{\alpha} A' \quad B \xrightarrow{\emptyset} B'}{A \,|^{c\prime}\, B \xrightarrow{\alpha} A' \,|^{c\prime}\, B'}\,^{\dagger}$$

Define $L(A \,|^{c\prime}\, B) = L(A) \cup L(B)$. Clearly, $\{\,|^{c\prime}\,\}$ is asynchronous and finite-state. Note that $L(A \,|^{c}\, B) = L(A \,|^{c\prime}\, B)$. It is straightforward to prove that

$$\mathcal{R} = \{\langle A \,|^{c}\, B, A \,|^{c\prime}\, B\rangle : A, B \text{ terms}\}$$

$$[\,|^c_1]:\ \frac{A \xrightarrow{\alpha} A'}{A \,|^c\, B \xrightarrow{\alpha} A' \,|^c\, B}\ \dagger \qquad [\,|^c_2]:\ \frac{A \xrightarrow{\{a\}} A' \quad B \xrightarrow{\{\bar{a}\}} B'}{A \,|^c\, B \xrightarrow{\emptyset} A' \,|^c\, B'} \qquad [+]:\ \frac{A \xrightarrow{\alpha} A'}{A+B \xrightarrow{\alpha} A'}\ \dagger$$

$$[.]:\ \frac{-}{a.A \xrightarrow{\{a\}} A} \qquad [\backslash]:\ \frac{A \xrightarrow{\alpha} A' \quad b \notin \alpha}{A \backslash b \xrightarrow{\alpha} A' \backslash b} \qquad [\,||_1]:\ \frac{A \xrightarrow{\alpha} A' \quad \alpha \cap L(B) = \emptyset}{A \,||\, B \xrightarrow{\alpha} A' \,||\, B}\ \dagger$$

$$[\,||_2]:\ \frac{A \xrightarrow{\{a\}} A' \quad B \xrightarrow{\{a\}} B'}{A \,||\, B \xrightarrow{\{a\}} A' \,||\, B'} \qquad [\backslash\backslash_1]:\ \frac{A \xrightarrow{\alpha} A' \quad b \notin \alpha}{A \backslash\backslash b \xrightarrow{\alpha} A' \backslash\backslash b} \qquad [\backslash\backslash_2]:\ \frac{A \xrightarrow{\{a\}} A'}{A \backslash\backslash a \xrightarrow{\emptyset} A' \backslash\backslash a}$$

$$[\Box_1]:\ \frac{A \xrightarrow{\emptyset} A'}{A \Box B \xrightarrow{\emptyset} A' \Box B}\ \dagger \qquad [\Box_2]:\ \frac{A \xrightarrow{\{a\}} A'}{A \Box B \xrightarrow{\{a\}} A'}\ \dagger \qquad [\sqcap]:\ \frac{-}{A \sqcap B \xrightarrow{\emptyset} A}\ \dagger$$

$$[;_1]:\ \frac{A \xrightarrow{\alpha} A' \quad \surd \notin \alpha}{A;B \xrightarrow{\alpha} A';B} \qquad [;_2]:\ \frac{A \xrightarrow{\{\surd\}} A'}{A;B \xrightarrow{\emptyset} B} \qquad [\triangleright_1]:\ \frac{A \xrightarrow{\alpha} A'}{A \triangleright B \xrightarrow{\alpha} A' \triangleright B}$$

$$[\triangleright_2]:\ \frac{B \xrightarrow{\alpha} B'}{A \triangleright B \xrightarrow{\alpha} B'} \qquad [*]:\ \frac{A \xrightarrow{\alpha} A'}{a * A \xrightarrow{\alpha \cup \{a\}} a * A'} \qquad [\times]:\ \frac{A \xrightarrow{\alpha} A' \quad B \xrightarrow{\beta} B'}{A \times B \xrightarrow{s(\alpha,\beta)} A' \times B'}$$

Table 4: Rules for operators from other process algebras. Rules marked † have a symmetric form.

is a bisimulation; the only nontrivial case is $A \,|^{c'}\, B \xrightarrow{\alpha} A' \,|^{c'}\, B'$ because of the new rule $[\,|^{c'}]$; this transition is simulated by $A \,|^{c}\, B \xrightarrow{\alpha} A' \,|^{c}\, B \xrightarrow{\emptyset} A' \,|^{c}\, B'$. This proves as required that $A \,|^{c}\, B \approx A \,|^{c'}\, B$ for all A, B. $\square$

Lemma 15 The operator $\square$ is definable. $\square$

Proof: Again the proof goes by defining an equivalent asynchronous operator, but this operator is slightly more involved than in the previous lemma. First define two auxiliary operators Π_1 and Π_2 with the rules

$$\frac{A \xrightarrow{\alpha} A' \quad B \xrightarrow{\emptyset} B'}{\Pi_1(A,B) \xrightarrow{\alpha} \Pi_1(A',B')} \qquad \frac{A \xrightarrow{\alpha} A'}{\Pi_1(A,B) \xrightarrow{\alpha} \Pi_1(A',B)} \qquad \frac{B \xrightarrow{\emptyset} B'}{\Pi_1(A,B) \xrightarrow{\emptyset} \Pi_1(A,B')}$$

$$\frac{A \xrightarrow{\emptyset} A' \quad B \xrightarrow{\alpha} B'}{\Pi_2(A,B) \xrightarrow{\alpha} \Pi_2(A',B')} \qquad \frac{B \xrightarrow{\alpha} B'}{\Pi_2(A,B) \xrightarrow{\alpha} \Pi_2(A,B')} \qquad \frac{A \xrightarrow{\emptyset} A'}{\Pi_2(A,B) \xrightarrow{\emptyset} \Pi_2(A',B)}$$

Let $L(\Pi_1(A,B)) = L(A)$ and $L(\Pi_2(A,B)) = L(B)$. Next define the operator $\square'$ with seven rules where the three first rules are

$$[\square'_1]\ \frac{A \xrightarrow{\emptyset} A' \quad B \xrightarrow{\emptyset} B'}{A \square' B \xrightarrow{\emptyset} A' \square' B'} \qquad [\square'_2]\ \frac{A \xrightarrow{\{a\}} A' \quad B \xrightarrow{\emptyset} B'}{A \square' B \xrightarrow{\{a\}} \Pi_1(A',B')} \qquad [\square'_3]\ \frac{A \xrightarrow{\emptyset} A' \quad B \xrightarrow{\{a\}} B'}{A \square' B \xrightarrow{\{a\}} \Pi_2(A',B')}$$

and the remaining four rules are obtained from these three by replacing $\xrightarrow{\emptyset}$ with $=$ in a premise: we get $[\square'_4]$ and $[\square'_5]$ from $[\square'_1]$ by doing this in the lefthand premise and righthand premise respectively, $[\square'_6]$ from $[\square'_2]$, and $[\square'_7]$ from $[\square'_3]$. Define $L(A \square' B) = L(A) \cup L(B)$. Clearly, $\{\Pi_1, \Pi_2, \square'\}$ is an asynchronous (and finite) set. It is straightforward to show that $\Pi_1(A,B) \approx A$ and $\Pi_2(A,B) \approx B$. We will show that $A \square B \approx A \square' B$. Clearly, $L(A \square B) = L(A \square' B)$. Define the relation $\mathcal{R}$ by

$$\mathcal{R} = \{\langle A \square B, A \square' B \rangle : A, B \text{ terms}\} \cup \approx$$

We first show that $\mathcal{R}$ is a simulation. So assume $C \mathcal{R} D$ and $C \xrightarrow{\alpha} C'$; we must show $D \stackrel{\alpha}{\Longrightarrow} D'$ and $C' \mathcal{R} D'$. If $C \approx D$ then this is trivial. So assume $C = A \square B$ and $D = A \square' B$. There are four cases for $C \xrightarrow{\alpha} C'$:

1. $A \xrightarrow{\emptyset} A'$, $\alpha = \emptyset$, $C' = A' \square B$. Then by $[\square'_4]$ we get that $A \square' B \xrightarrow{\emptyset} A' \square' B$.

2. $B \xrightarrow{\emptyset} B'$ — this case is symmetric with 1.

3. $A \xrightarrow{\{a\}} A'$, $\alpha = \{a\}$, $C' = A'$. Then by $[\square'_6]$ we get that $A \square' B \xrightarrow{\{a\}} \Pi_1(A', B)$. But $\Pi_1(A', B) \approx A'$, hence $\Pi_1(A', B) \mathcal{R} A'$.

4. $B \xrightarrow{\{a\}} B'$ — this case is symmetric with 3.

We next show that $\mathcal{R}^{-1}$ is a simulation. So assume $C \mathcal{R} D$ and $D \xrightarrow{\alpha} D'$; we must show $C \stackrel{\alpha}{\Longrightarrow} C'$ and $C' \mathcal{R} D'$. If $C \approx D$ then this is trivial. So assume $C = A \square B$ and $D = A \square' B$. There are seven cases for $D \xrightarrow{\alpha} D'$ and we only show the nontrivial cases here.

1. $A\xrightarrow{\emptyset}A'$, $B\xrightarrow{\emptyset}B'$, $\alpha = \emptyset$, $D' = A'\Box' B'$. Then by two applications of $[\Box_1]$ we get that $A\Box B\xrightarrow{\emptyset}A'\Box B\xrightarrow{\emptyset}A'\Box B'$, i.e. $A\Box B\overset{\emptyset}{\Longrightarrow}A'\Box B'$.

2. $A\xrightarrow{\{a\}}A'$, $B\xrightarrow{\emptyset}B'$, $\alpha = \{a\}$, $D' = \Pi_1(A', B')$. Then by $[\Box_2]$ we infer that $A\Box B\xrightarrow{\{a\}}A'$. But $\Pi_1(A', B') \approx A'$, hence $\Pi_1(A', B')\mathcal{R}A'$.

3. $B\xrightarrow{\{a\}}B'$ — this case is symmetric with 2.

This proves that $A\Box B \approx A\Box' B$. By Theorem 4, $\Box'$ and hence $\Box$ is definable. $\Box$

Lemma 16 All operators in Table 3 except $a*$ and $\times$ are $=_T$-definable. $\Box$

Proof: Let op be any operator in Table 3 except $a*$ or $\times$. Define a new operator op' by modifying the rules for op as follows:

1. For each rule where some operand A of the operator is not mentioned in the premise, add a new rule (unless it is already present) which only differs in that "$A\xrightarrow{\emptyset}A'$" is added to the premise, and in that A' is used for A in the conclusion.

2. Conversely, for each rule containing a premise $A\xrightarrow{\emptyset}A'$, add a new rule (unless it is already present) which only differs in that this premise is removed, and in that A is used for A' in the conclusion.

3. For each rule where the conclusion is of type $op(\widetilde{A})\xrightarrow{\alpha}A_k$, replace the conclusion with $op(\widetilde{A})\xrightarrow{\alpha}\Pi_k(\widetilde{A})$ (Π_k is as defined in Lemma 15).

4. Add an idling rule

$$\frac{A_1\xrightarrow{\emptyset}A_1' \cdots A_n\xrightarrow{\emptyset}A_n'}{op(\widetilde{A})\xrightarrow{\emptyset}op(\widetilde{A'})}$$

   if this is not subsumed by the other rules.

For example, the prefixing operator $a.$ will with this scheme produce an operator $a.'$ with the following three rules:

$$\frac{-}{a.'A\xrightarrow{\{a\}}\Pi_1(A)} \qquad \frac{A\xrightarrow{\emptyset}A'}{a.'A\xrightarrow{\{a\}}\Pi_1(A')} \qquad \frac{A\xrightarrow{\emptyset}A'}{a.'A\xrightarrow{\emptyset}a.'A'}$$

Clearly, $\{op', \Pi_1, \Pi_2\}$ is asynchronous and finite-state, and hence definable. It can be proven that for all op it is the case that $op =_T op'$. The proof is straightforward in all cases. For example, for $+$ the proof amounts to showing that

$$A + B\overset{\sigma}{\Longrightarrow} \quad \text{iff} \quad A\overset{\sigma}{\Longrightarrow} \text{ or } B\overset{\sigma}{\Longrightarrow} \quad \text{iff} \quad A +' B\overset{\sigma}{\Longrightarrow}$$

The full proof is omitted. $\Box$

For the following lemmas let $\mathbf{M}$ be a basis for finite-state terms.

Lemma 17 The operator $+$ is not definable. $\square$

Proof: Let op be an asynchronous operator and assume $op(A, B) \approx A + B$ for all A, B. We will derive a contradiction. Let $\mathbf{0}$ stand for a term which has no transitions, and consider the terms A, B, and C whose only transitions are:

$$A \xrightarrow{\emptyset} C \qquad A \xrightarrow{\{a\}} \mathbf{0} \qquad C \xrightarrow{\{c\}} \mathbf{0} \qquad B \xrightarrow{\{b\}} \mathbf{0}$$

These terms are finite-state and hence definable, i.e. there are equivalent terms A', B' and C' in $\mathcal{T}_{\mathbf{M}}$. For these terms it must then hold that

$$\begin{array}{lll} A' \stackrel{\emptyset}{\Longrightarrow} A'' & \text{such that} & A'' \approx C \\ A' \stackrel{\emptyset}{\Longrightarrow} A'' & \text{implies} & A'' \approx A \text{ or } A'' \approx C \\ B' \stackrel{\emptyset}{\Longrightarrow} B'' & \text{implies} & B'' \approx B \end{array}$$

Now op is asynchronous, so it must hold that $op(A', B') \stackrel{\emptyset}{\Longrightarrow} op(A'', B')$ for some $A'' \approx C'$. Since $op(A', B') \approx A' + B'$, there exists a D' such that $A' + B' \stackrel{\emptyset}{\Longrightarrow} D'$ and $D' \approx op(A'', B') \approx C' + B'$. The following possibilities exist for $A' + B' \stackrel{\emptyset}{\Longrightarrow} D'$:

1. $D' = A' + B'$. But $A' + B' \not\approx C' + B'$, since $A' + B' \stackrel{\{a\}}{\Longrightarrow}$.
2. $D' \approx A'$ But $A' \not\approx C' + B'$, since $C' + B' \stackrel{\{b\}}{\Longrightarrow}$.
3. $D' \approx C'$. But $C' \not\approx C' + B'$, since $C' + B' \stackrel{\{b\}}{\Longrightarrow}$.
4. $D' \approx B'$. But $B' \not\approx C' + B'$, since $C' + B \stackrel{\{c\}}{\Longrightarrow}$.

Hence, no such D' exists. $\square$

Remark: A simpler proof of this lemma would use the fact that $+$ does not preserve $\approx$, i.e. that there are terms A, B, and C such that $A \approx B$ and $A + C \not\approx B + C$. It then immediately follows that there is no defining context for $+$ since all contexts must preserve $\approx$. However, such a proof would rely on the existence of terms A, B, and C with the indicated property. Such terms do indeed exist in e.g. $\mathcal{T}_{\{\mathbf{SY},\mathbf{AR},\mathbf{AL}\}}$. But the existence of such terms does *not* follow from the assumption that all finite-state terms are definable w.r.t $\approx$; hence our proof of the lemma is more general.

Lemma 18 The operator $\sqcap$ is not definable. $\square$

Proof: The proof is similar to the proof of Lemma 17. Assume the terms A', B', and C' of that lemma, and assume that an asynchronous operator op is equivalent with $\sqcap$. Then $op(A', B') \stackrel{\emptyset}{\Longrightarrow} op(A'', B') \approx C' \sqcap B'$, so for some D' it must hold that $A' \sqcap B' \stackrel{\emptyset}{\Longrightarrow} D' \approx C' \sqcap B'$. The possibilities for D' turn out to be precisely as presented in the proof of Lemma 17, so no such D' can exist. $\square$

Lemma 19 The operator $\rhd$ is not definable. $\square$

Proof: The proof is similar to the proof of Lemma 17. Let *op* be an asynchronous operator and assume $op(A, B) \approx A \rhd B$ for all A, B. We will derive a contradiction. Consider the terms A, B, C, and $\mathbf{0}$ from the proof of Lemma 17, and the corresponding equivalent terms A', B' and C'. Because *op* is asynchronous it holds that $op(B', A') \overset{\emptyset}{\Longrightarrow} op(B', A'')$ for some $A'' \approx C$. Thus $B' \rhd A' \overset{\emptyset}{\Longrightarrow} D'$ for some $D' \approx B' \rhd C'$. The cases for $B' \rhd A' \overset{\emptyset}{\Longrightarrow} D'$ are:

1. $D' = B' \rhd A'$. But $B' \rhd A' \not\approx B' \rhd C'$, since $B' \rhd A' \overset{\{a\}}{\Longrightarrow}$.
2. $D' \approx A'$. But $A' \not\approx B' \rhd C'$, since $A' \overset{\{a\}}{\Longrightarrow}$.
3. $D' \approx C'$. But $C' \not\approx B' \rhd C'$, since $B' \rhd C' \overset{\{b\}}{\Longrightarrow}$.

Hence, no such D' exists. $\square$

Lemma 20 The operator ; is not definable. $\square$

Proof: The proof is similar to the proof of Lemma 17. Let *op* be an asynchronous operator and assume $op(A, B) \approx A; B$ for all A, B. We will derive a contradiction. Let $\mathbf{0}$ stand for a term which has no transitions, and consider the terms A, B, and C whose only transitions are:

$$A \xrightarrow{\emptyset} \mathbf{0} \qquad A \xrightarrow{\{a\}} \mathbf{0} \qquad B \xrightarrow{\{b\}} C \qquad C \xrightarrow{\{\surd\}} \mathbf{0}$$

These terms are finite-state and hence definable, i.e. there are equivalent terms A', B' and C' in $\mathcal{T}_{\mathbf{M}}$. In particular for B' it must hold that $B' \overset{\emptyset}{\Longrightarrow} B''$ implies $B'' \approx B'$, and for A' it must hold that $A' \overset{\emptyset}{\Longrightarrow} A''$ for some $A'' \approx \mathbf{0}$. Now *op* is asynchronous, so it must hold that $op(B', A') \overset{\emptyset}{\Longrightarrow} op(B', A'')$ for some $A'' \approx \mathbf{0}$. Since $op(B', A') \approx B'; A'$, there exists a D' such that $B'; A' \overset{\emptyset}{\Longrightarrow} D'$ and $D' \approx op(B', A'') \approx B'; \mathbf{0}$. The only possibility for $B'; A' \overset{\emptyset}{\Longrightarrow} D'$ is that $D' \approx B'; A'$. But $B'; A' \not\approx B'; \mathbf{0}$ since $B'; A' \overset{\{b\}\{a\}}{\Longrightarrow}$. $\square$

Lemma 21 The operator $a.$ is not definable. $\square$

The proof is given in the text of Section 7.

For the next two lemmas we assume the existence of a term $\mathbf{0}$ which has no transitions. For example, $\mathbf{0}$ could be $\mathrm{AR}(a, b, c)\langle a^\frown b\rangle$. Notice that this is the first use we make of any of the modules (apart from the property that all finite-state terms are definable). For the following lemmas it is important that such a term $\mathbf{0}$ (and not merely an equivalent term) exists in $\mathcal{T}_{\mathbf{M}}$.

Lemma 22 The operator $a*$ is not $=_T$-definable. $\square$

Proof: The proof idea is to demonstrate that $a*$ does not respect $=_T$. We will prove this using only the term $\mathbf{0}$ and the fact that all finite-state terms are definable w.r.t $=_T$. Consider the term A which has the only transition $A \xrightarrow{\{a,b\}} \mathbf{0}$. By assumption there is an equivalent ($=_T$) term A' in $\mathcal{T}_{\mathbf{M}}$; hence $A' \overset{\{a,b\}}{\Longrightarrow}$ is the only nonempty trace from this term. It then holds that

$$\mathbf{0} \quad =_T \quad A'\langle a^\frown b\rangle$$

since the only trace possible from either of the terms is the empty trace. However,

$$c * \mathbf{0} \;\; \neq_T \;\; c * (A'\langle a^\frown b\rangle)$$

The reason is that $A'\langle a^\frown b\rangle \xrightarrow{\emptyset}$ holds, so $c * (A'\langle a^\frown b\rangle) \xRightarrow{\{c\}}$ holds. However, $\mathbf{0}$ has no transitions at all, so $c * \mathbf{0}$ has also no transitions at all and consequently only the empty trace.

By Theorem 1, all contexts must respect $=_T$, so no context can be equivalent with $a*$. $\square$

Lemma 23 the operator $\times$ is not $=_T$-definable. $\square$

Proof: The proof is similar to the proof of Lemma 22. Let α be such that $s(\emptyset, \alpha) \neq \emptyset$. Let A be a term with the only transition $A \xrightarrow{\{a,b\}} A$. There then exists an equivalent term A' in $\mathcal{T}_{\mathbf{M}}$. Clearly,

$$\mathbf{0} \quad =_T \quad A'\langle a^\frown b\rangle$$

since the only trace from any of these terms is the empty trace. Let B be a term with the only transition $B \xrightarrow{\alpha} \mathbf{0}$. Then there exists an equivalent term B' in $\mathcal{T}_{\mathbf{M}}$. We now have that for all n,

$$A' \underbrace{\xRightarrow{\{a,b\}} \cdots \xRightarrow{\{a,b\}}}_{n}$$

Hence for all n,

$$A'\langle a^\frown b\rangle \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{n}$$

We also have, since $B' =_T B$, that for some m,

$$B' \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{m} \xrightarrow{\alpha}$$

So we then infer that for some m,

$$A'\langle a^\frown b\rangle \times B' \underbrace{\xrightarrow{s(\emptyset,\emptyset)} \cdots \xrightarrow{s(\emptyset,\emptyset)}}_{m} \xrightarrow{s(\emptyset,\alpha)}$$

Since $s(\emptyset, \alpha) \neq \emptyset$ this implies that $A'\langle a^\frown b\rangle \times B' \;\; \neq_T \;\; \mathbf{0} \times B'$, since $\mathbf{0} \times B'$ has only the empty trace. It follows that $\times$ does not respect $=_T$. By Theorem 1, all contexts must respect $=_T$, so no context can be equivalent with $\times$. $\square$

This concludes the results summarised in Table 3.