LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Synchronisation Flow Algebra

by

Joachim Parrow

ECS-LFCS-87-35
(also published as CSR-243-87

LFCS Report Series

August 1987

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Copyright © 1987, LFCS

Synchronisation Flow Algebra

Joachim Parrow

Abstract

A *synchronisation flow network* is a network where independent processing units engage in synchronisations with each other and with the environment of the network. We explore an algebra for description of such networks. This algebra has two operators: *disjoint parallelism*, where the arguments are composed in parallel without any interconnections, and *linking*, whereby an interconnection is formed between two ports. The intention is that these operators correspond to the primitive steps when constructing interconnection structures. The results on this algebra are: (1) Expressibility of terms: with only three simple processing units, every finite state behaviour can be constructed. (2) Expressibility of operators: we characterise the operators which are expressible within the algebra. (3) Axiomatisation: We present an axiomatisation for valid equations which is sound and complete with respect to a large class of behaviour equivalences.

1 Introduction

In this paper we will combine two well established ideas. The first idea is that of *data flow networks*: a number of simple functional units execute in parallel and transmit data to each other. The second idea is that of *synchronisation* and *reactive behaviour*: the behaviour of a process is described operationally in terms of the synchronisations it engages in. Both these ideas have been extensively treated in the literature; the original contribution here is the way in which they are combined.

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. The interconnection structure formed by the links can be viewed as a graph, the *flowgraph*, where the units are nodes and the links are edges. Typically, a data flow network realises some complex function, using only a set of standard (predefined) functional units.

Good expositions of ideas and concepts related to data flow can be found in e.g. [Com82]. From a theoretical point of view data flow has been the subject of many papers, e.g. [Kah74,Par83,KP85,SN85,Jon85]; we will have reason to mention some of them again below.

Aspects related to synchronisation and reactive behaviour have been treated in process algebras ([Mil80,Mil83,AB84,BT85,BHR84,Win80,Mil85]). We will in the rest of this paper assume that the reader is at least vaguely familiar with some process algebra. In particular, we will use operational semantics defined by state transition systems, where transitions involve synchronisation events. Excellent motivations for this type of semantics can be found in the references mentioned above. We will also use some existing behaviour equivalences (notably bisimulation equivalence and trace equivalence); they will be formally defined but not intuitively motivated.

By a *synchronisation flow network* we will mean a network consisting of units interconnected by links. The difference between a data flow network and a synchronisation flow network is that in the former, the units perform computations and communicate asynchronously, while in the latter, the units engage in synchronisations with each other and with the environment of the network. One potential application for synchronisation flow networks is in the design of complicated hardware synchronisation structures, e.g. bus management protocols. Another potential application is in the design of software structures where the set of available processes is predetermined and the designer can only decide which to employ and how to interconnect them. As for data flow networks, we assume that the links remain unchanged during execution, and we use the term "flowgraph" for the graphical description of a network.

Our aim is to develop and explore an algebraic description language for synchronisation flow networks. In such an algebra there are elements which correspond to the predefined units, and operators which correspond to ways in which larger networks can be built from smaller networks by combining or modifying the corresponding flowgraphs. In principle, any existing process algebra could serve as such a description language. However, the operators in existing process algebras are chosen either to mimic established programming language constructs, or to achieve some expressive completeness result. Thus, there is no reason to expect that all operators of an existing process algebra correspond to ways of combining networks viewed as flowgraphs. Indeed, in section 7 below we will determine that some of these operators are not expressible as network combinators. Another difficulty is that existing process algebras (and also the work on flowgraphs in [Mil79]) use port names to implicitly determine interprocess links. This means that the whole algebra is parameterised on a linking structure. We have found it more natural to follow recent work on semantics of data flow net-

works ([KP85,SN85]) and introduce links as operators, i.e. as part of network constructions.

In our synchronisation flow algebra there are only two operators. The intention is that these operators correspond to primitive steps when constructing networks, and that they are simple enough to be available in any practical design situation. One operator 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 operator corresponds to "disjoint union" of [SN85] and "aggregation" of [KP85]. The other operator is *linking*: two ports of a network are 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. This linking operator corresponds to "linking" of [SN85] and "loop" of [KP85]. We will largely develop our results independently of any particular set of predetermined units. An exception is in section 6 below where we will give an expressiveness result for a set of three simple units.

In summary this paper differs from existing treatments of process algebras by using only very simple operators, one of which is an explicit linking operator, and in the assumption that the links are binary and may not change during execution. It differs from treatments of data flow by considering reactive behaviours rather than functions on data values. Finally it differs from the treatment of flowgraphs in [Mil79] in that an operational semantics is used.

In sections 2 and 3 below we present the syntax and semantics respectively of the synchronisation flow algebra. Section 4 is devoted to some examples of networks and behaviours. In section 5 we define some notions of behaviour equivalence. Section 6 contains an expressiveness result: with only three simple behaviours, any finite state behaviour can be constructed up to bisimulation equivalence. In section 7 we present a class of operators which are definable within the algebra, and discuss to what extent some operators from other algebras are definable. In section 8 we consider valid equations, i.e. equations which are true no matter what operational semantics is given to the predefined units. We provide a sound and complete axiomatisation for valid equations. An interesting result is that this axiomatisation is sound and complete with respect to many different behaviour equivalences. Finally, in section 9 we recapitulate some interesting open problems and discuss alternative formulations of the algebra. All proof details are relegated to an appendix.

2 Syntax

By a *network* we will mean either a single predetermined unit, or a network composed of such units. The purpose of this section is to define an algebra where the terms correspond to networks.

A network has a finite set of *ports* through which its behaviour is manifested. Ports can be thought of as possible endpoints for links or, borrowing concepts from hardware construction, as wires dangling from the network waiting to be connected somewhere. Each predetermined unit has a set of ports, and the ports of a network are the non-linked ports of its components.

We will assume that each port in a network has a unique name. The port names serve to identify ports when defining interconnection structures and behaviours of networks. The (potentially infinite) set of *port names* is denoted Σ. In the following, $a, b, \ldots$ etc. will range over Σ.

The set of non-linked port names in a particular network is called the *sort* of the network. Thus, a sort is a finite subset of Σ. We use $L, L_1, \ldots$ to range over sorts.

We adopt the viewpoint that the predetermined units are parameterised on the names of their ports. Thus, if $M(a, b)$ is a unit with two ports named a and b, then $M(c, d)$ is another instance of the same unit with ports named c and d.[1] The object M in such constructs can be thought of as a "pre-unit" or "schematic unit" — we will simply call it a *module*. A module is a predetermined unit where the ports have not yet been named. We require that different instances of the same module have the same number of ports, and that the port names which parameterise a module are distinct.

Formally, let $\mathbf{M}$ be a set of *modules*, typically ranged over by $M, M_1, \ldots$. Each module M has an *arity* written $\#M$, a nonnegative integer representing the number of ports of the module. If M is a module with arity n and $a_1, \ldots, a_n$ are n distinct port names, then $M(a_1, \ldots, a_n)$ is a *unit* with sort $\{a_1, \ldots, a_n\}$. In the following, $C_1, C_2, \ldots$ will range over units.

The algebra has one binary infix operator $|$, and a set of postfix operators $\langle a \frown b \rangle$, one for each $a, b \in \Sigma, a \neq b$. We call $|$ the *parallel* operator and $\langle a \frown b \rangle$ the *linking* operators. The operators are partial: they are only defined for certain sorts of their arguments. A definition of the domain of the operators together with a definition of the function L which assigns sorts to terms is given in table 1. In this table as in the rest of this paper $A, B, \ldots$ range over terms of the algebra.

[1] Another possibility here would be to assign each unit an arbitrary sort and then use renaming operators in the algebra to obtain different instances of the unit. Our approach with explicit parameterisation on port names is technically simpler: port renaming is syntactic substitution. See section 9 for an alternative formulation of the algebra with renaming operators.

A		$L(A)$
$M(a_1,\ldots,a_n)$	$\#M = n,\ \ i \neq j \Rightarrow a_i \neq a_j$	$\{a_1,\ldots,a_n\}$
$B\|B'$	$L(B) \cap L(B') = \emptyset$	$L(B) \cup L(B')$
$B\langle a^\frown b\rangle$	$a \neq b,\ \ a,b \in L(B)$	$L(B) - \{a,b\}$

Table 1: Syntax of the synchronisation flow algebra. For each term A, the corresponding sort $L(A)$ is defined.

The parallel operator is only defined for arguments with disjoint sorts, hence the name *disjoint parallel* can be used to distinguish it from other parallel combinators in the literature. This sort constraint ensures that distinct ports will have distinct names. In practice the constraint will not be severe: by first applying a suitable substitution of port names, any two terms may be composed in parallel.

The linking operators are also partial: it is required that both linked ports are present in the sort of the term. Intuitively, it is meaningless to link ports which do not exist in the network. The resulting sort is the original sort with the linked ports removed; this means that once a port has been linked, it can not be affected by another linking operator.

Essentially, what we have is a *sorted algebra* (see e.g. [GTWW77]). We could make a more formal presentation of the algebra and talk about "variables indexed by sequences of port names" instead of units. We will not here indulge in the fine points of sorted algebras; simply note that each term has a well defined sort.

A term has a natural interpretation as a *flowgraph*. A flowgraph consists of a set of nodes where each node has a sequence of ports, and a set of edges where each edge interconnects exactly two ports. The ports not connected by edges have labels from Σ. Thus a unit $M(\tilde{a})$ is interpreted as a node M with ports labelled $\tilde{a}$, a term $B|B'$ is interpreted as the union of the graphs of B and B', and the interpretation of $B\langle a^\frown b\rangle$ is obtained from the graph of B by removing port labels a and b and adding an edge between these ports. It is clear that any flowgraph can be described by a term in this way. Examples of flowgraphs and corresponding terms are presented in section 4

3 Semantics

The interpretation of terms into flowgraphs is interesting in itself, and it would be possible to explore the algebra with only this interpretation in mind. A similar algebra with an interpretation into flowgraphs is explored in [Mil79]. We want

to go one step further and give an even more specific interpretation with an operational semantics. This is done in a way that has become standard for process algebras: as a family of labelled transition relations on terms. The label of a transition is called the *event* of the transition. An event is a set of port names:[2] the intended meaning of $A \xrightarrow{\alpha} B$ is that the network corresponding to A can evolve to the network corresponding to B by participating in synchronisations on all ports in α. There may be several different transitions from a term with the same event; this amounts to nondeterminism in the term. When loosely referring to the *behaviour* of a term we mean its operational interpretation.

We will focus this paper on results which are independent of any particular set of modules or particular behaviours of units. Therefore we let the behaviours of units be defined by an *interpretation*. An interpretation assigns to each unit C a set of transitions of type $C \xrightarrow{\alpha} C'$, where α is a set of port names and C' is a unit. Statements of type $A \xrightarrow{\alpha} B$ are always taken with respect to some (sometimes implicit) interpretation.

We require an interpretation to satisfy the following properties (here and in the rest of the paper θ ranges over bijective substitutions of port names):

1. If $M(\tilde{a}) \xrightarrow{\alpha} M'(\tilde{b})$ then $\tilde{a} = \tilde{b}$.
2. If $M(a_1, \ldots, a_n) \xrightarrow{\alpha} M'(a_1, \ldots, a_n)$ then $\alpha \subseteq \{a_1, \ldots, a_n\}$.
3. If $M(\tilde{a}) \xrightarrow{\alpha} M'(\tilde{a})$ then $M(\tilde{a}\theta) \xrightarrow{\alpha\theta} M'(\tilde{a}\theta)$.
4. Each unit C has an *inaction* transition $C \xrightarrow{\emptyset} C$.

The requirements deserve some comments. Requirement 1 says that the ports in a unit may not change names as the result of a transition. As mentioned in section 1, we are only interested in fixed interconnection structures which do not change as networks execute. Since links (the linking operators) refer to ports through their names, this implies that port names may not change during execution. Requirement 2 simply states that events in transitions from a unit are subsets of the port names of that unit. Requirement 3 is a uniformity requirement which relates the behaviours of different instances of the same module: the particular choice of port names should not affect the behaviour. For example, if $M(a, b, c) \xrightarrow{\{a,c\}} M'(a, b, c)$ then it must also be the case that $M(d, e, a) \xrightarrow{\{d,a\}} M'(d, e, a)$. Note that we only need consider bijective substitutions since the port names in a unit must be distinct.

[2] Events as sets of port names are also used in CIRCAL ([Mil85]); other process algebras tend to use either single port names or multisets.

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

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

Table 2: The operational semantics of operators

Requirement 4 says that each unit has an inaction transition. Intuitively, if $C \xrightarrow{\emptyset} C'$, then either this is because of some internal state change within the unit not involving synchronisations on any ports (cf. the τ action in CCS), or the transition corresponds to complete inaction: $C' = C$ and the unit has done nothing at all. Thus, in our algebra we identify internal action with inaction. The reason for this is purely technical: the definitions and proofs become more concise. At the expense of two more rules for the parallel operator (see below) this requirement could be omitted.[3]

The semantics of arbitrary terms is defined by extending the transition relations $\xrightarrow{\alpha}$ to terms through structural induction as defined in table 2. The rule for parallel says that if both A and B can do something, then $A|B$ can do the union of the events. Note that α and β must be disjoint because of the sort constraint on $|$. Also, note that either (or both) of A and B may "inact" (any term possesses a transition derived from inaction transitions only); this makes specific rules for the case that only one of A and B executes unnecessary. 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 event neither involving a nor b, then $A\langle a^\frown b\rangle$ can do the same event (the link has no effect at all). If A can do an event involving both a and b, then $A\langle a^\frown b\rangle$ can do the same event, now with a and b removed (intuitively this event involves a synchronisation over the link). As a consequence, events of A involving exactly one of a and b are disallowed in $A\langle a^\frown b\rangle$ (such events 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.

The following properties of terms are trivial to verify from the requirements

[3]For a further discussion of this, and a formulation of the algebra where internal action is not the same as inaction, see section 9.

on interpretation and by induction on the rules:

1. If $A \xrightarrow{\alpha} A'$ then $L(A) = L(A')$.
2. If $A \xrightarrow{\alpha} A'$ then $\alpha \subseteq L(A)$.
3. If $A \xrightarrow{\alpha} A'$ and θ is a bijective substitution of port names then $A\theta \xrightarrow{\alpha\theta} A'\theta$.
4. $A \xrightarrow{\emptyset} A$ for all terms A.

A remark is in order here on the use of sets of port names in events. The intuitive meaning is that if a term A has a transition with event α, then the elements in α may potentially be synchronised. Thus, co-occurrence in an event is not directly related to "simultaneity" or "true concurrency" (two usual interpretations in similar formalisms), but rather to "potentiality for synchronisation". It would be possible to use multisets (instead of sets) of port names. While this would be in accordance with algebras as SCCS and MEIJE, the corresponding synchronisation flow algebra would be more complicated and we leave this as a topic for further investigations.

4 Examples

In this section we present some examples of synchronisation flow networks and corresponding reactive behaviours. To begin, consider the three simple units in figure 1. For each of the units we give the intended behaviour in an interpretation as a transition system (top), a symbol to be used in flowgraphs (middle), and the formal symbol to be used in the terms of the algebra (bottom). When the transition diagrams have more than one state, the initial state is marked by an unlabelled arrow. Inaction transitions (labelled $\emptyset$ and leading back to the same state) are tacitly present from all states.

The first unit SY(a,b,c) is a three way synchroniser. It has three ports, and require synchronisations on all three. While the linking operators in the algebra are binary, i.e. link exactly two ports, a three way synchroniser can be used as an intermediate unit to link three ports. In the same way, it is possible to define four way synchronisers, or n way synchronisers for arbitrary n. However, n way synchronisers are definable from three way synchronisers in the following sense: for each n way synchroniser there exists an equivalent term, containing only three way synchronisers. For example, the term

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

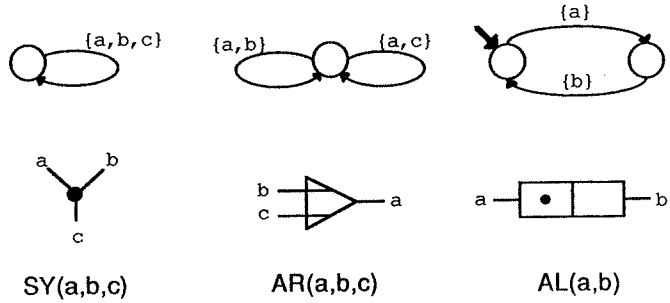


Figure 1: Three units: A three way synchroniser, an arbiter, and an alternator.

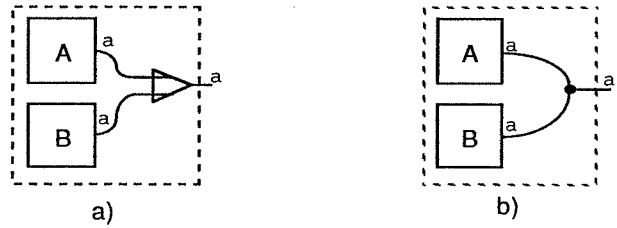


Figure 2: a) CCS parallel composition: the implicit use of an arbiter. b) CSP parallel composition: the implicit use of a three way synchroniser.

is equivalent with a four way synchroniser: it requires synchronisations on all four ports `a,b,c,d`. A precise definition of what "equivalence" means in general will be given in the next section.

The second unit `AR(a,b,c)` is an arbiter. It requires synchronisations on its righthand port and exactly one of its lefthand ports. This represents another way to interconnect three ports: imagine the lefthand ports to be competing for the privilege to synchronise with the righthand port. It is possible to define n way arbiters with n lefthand ports as a combination of arbiters. For example, a three way arbiter would be

$$(\mathtt{AR(a,b,f)} \mid \mathtt{AR(g,c,d)})\langle \mathtt{f \frown g} \rangle$$

It is interesting here to briefly consider the parallel operators in CCS and CSP (figure 2). In CCS, two agents which share a port name a will compete for the use of a in synchronisations. 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

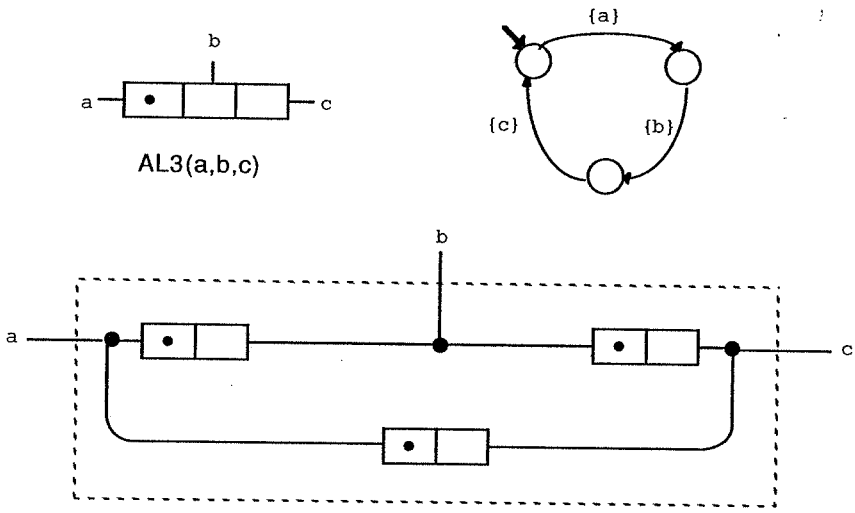


Figure 3: A three way alternator and its implementation using ordinary alternators and synchronisers.

the use of shared port names. On the other hand, in CSP two 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. It might be that our framework, where the primitive links are binary and other synchronisation mechanisms must be defined through intermediate units, can serve as a common basis or at least as a vehicle for understanding of higher level description formalisms.

The third unit `AL(a,b)` in figure 1 is an alternator: it alternatingly synchronises on its two ports. In the flowgraph symbol we use a small filled circle to indicate the state of the alternator: when this circle is to the left, the next synchronisation will be on the lefthand port. Formally, we should use two different module symbols in the algebra to distinguish between alternators in different states. For example, if `AL(a,b)` denotes an alternator in its lefthand state, then `AL'(a,b)` could be used to denote an alternator in the righthand state. Since this `AL'(a,b)` is equivalent with `AL(b,a)` we will only use the symbol `AL`.

Since synchronisers and arbiters can be combined to n way synchronisers and arbiters, it might be expected that n way alternators can be defined from ordinary alternators. This is not quite the case — the reader is encouraged to try to deduce the behaviour of the term

$$(\mathtt{AL(a,f)} \mid \mathtt{AL(g,b)})\langle \mathtt{f}^\frown \mathtt{g} \rangle$$

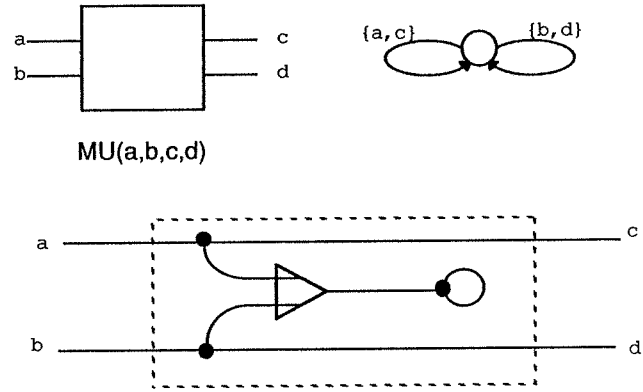


Figure 4: A multiplexer and its implementation using three way synchronisers and an arbiter.

A three way alternator AL3(a,b,c), i.e. a unit which repeatedly synchronises on a, b, and c in sequence, is depicted in figure 3. In fact, the three way alternator is definable by using three ordinary alternators and three synchronisers:

$$(\mathrm{SY}(a,f,g) \mid \mathrm{AL}(h,i) \mid \mathrm{SY}(j,b,k) \mid \mathrm{AL}(l,m) \mid \mathrm{SY}(n,c,o) \mid \mathrm{AL}(q,p))$$
$$\langle f^\frown h\rangle\langle i^\frown j\rangle\langle k^\frown l\rangle\langle m^\frown n\rangle\langle o^\frown p\rangle\langle q^\frown g\rangle$$

Even simple networks such as this give rise to syntactically long terms. The purpose of the algebra is not primarily to serve as a description language, but rather as a mathematical tool for the study of flowgraphs and behaviours. The flowgraph corresponding to this long term is also presented in figure 3.

Another example of a simple construction is the multiplexer MU(a,b,c,d) in figure 4. The behaviour is that either an event involves both a and c, or it involves both b and d. However, all four ports may not synchronise in the same event. The multiplexer can be regarded as a unit with the ability to propagate signals between a and c and between b and d, but not simultaneously. This unit is definable in terms of three three way synchronisers and one arbiter:

$$(\mathrm{SY}(a,c,f) \mid \mathrm{AR}(g,h,i) \mid \mathrm{SY}(b,j,d) \mid \mathrm{SY}(k,l,m))\langle f^\frown h\rangle\langle i^\frown j\rangle\langle g^\frown k\rangle\langle l^\frown m\rangle$$

The flowgraph corresponding to this term is also depicted in figure 4. Note the rightmost three way synchroniser working as a "source": it is always ready to supply the arbiter with synchronisations on the righthand port.

As a final and slightly more complicated example consider figure 5. The flip flop FF(r,s,q,q') has the following behaviour: following a signal on s it accepts signals on q' and following a signal on r it accepts signals on q. The use of a flip flop is that it "remembers" which of s and r was last performed, and this "memory" can be recalled through q and q'. As with the alternator, a small filled

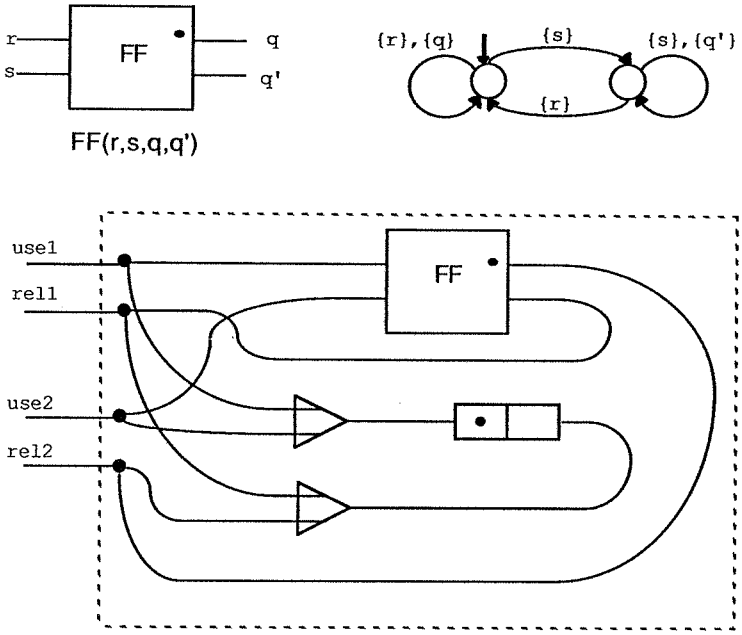


Figure 5: A flip flop: flowgraph symbol and behaviour. The flip flop is used in a flowgraph representing a controller for a critical resource.

circle is used to indicate which state the flip flop is in. With one such flip flop, one alternator, two arbiters, and four three way synchronisers, a critical resource controller can be built. This controller has four ports: `use1`, `rel1`, `use2`, and `rel2`. The intention is that `use1` and `rel1` are connected to one process, and `use2` and `rel2` to another process. A process signals the use of the critical resource with a signal on its `use` port and the release of the resource with a signal 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. The controller employs a flip flop to remember which process last acquired the resource, and an alternator to remember whether the resource is available.

Before continuing with the next section, an interested reader might enjoy trying to construct a flip flop from the other units presented here.

5 Equivalences

Intuitively, in a given interpretation two terms should be equivalent iff their respective behaviours are "sufficiently similar". We will mainly use an adaptation of *observation equivalence*, $\approx$, based on bisimulation. Since bisimulations have been used extensively in similar contexts (e.g. [Mil83]) we will simply state the definition (1-4 below). Note that we require equivalent terms to have the same sort. Without this requirement, the congruence result below would not hold.

In section 7 we will also use a much weaker equivalence, *trace equivalence*, $=_T$. The corresponding definitions are 5–7 below. Finally, the notion of deadlock (8–9 below) will figure in section 8. We use σ to range over sequences of events.

1.

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

2. $\mathcal{R}$ is a *simulation* iff $A \mathcal{R} B$ 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* iff $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations.

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

5. $A \stackrel{\sigma}{\Longrightarrow} A'$ $(\sigma = \alpha_1 \ldots \alpha_n)$ iff $A \stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow} A'$.

6. $A \stackrel{\sigma}{\Longrightarrow}$ iff for some A', $A \stackrel{\sigma}{\Longrightarrow} A'$.

7. $A =_T B$ iff ($A \stackrel{\sigma}{\Longrightarrow}$ iff $B \stackrel{\sigma}{\Longrightarrow}$) and $L(A) = L(B)$.

8. A is *deadlocked* iff not $A \stackrel{\alpha}{\Longrightarrow}$ for any $\alpha \neq \emptyset$.

9. $A \stackrel{\sigma}{\Longrightarrow} \bullet$ iff $A \stackrel{\sigma}{\Longrightarrow} A'$ for some deadlocked A'.

All definitions in this section must be taken with respect to some particular interpretation I assigning behaviours to units. When stressing the dependency of $\approx$ on I, we will write $I \models A \approx B$ to mean that A and B are observation equivalent in I.

We say that an equivalence $\simeq$ is a *congruence* if it is preserved by bijective substitutions of port names and 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 $A\langle a^\frown b\rangle \simeq A'\langle a^\frown b\rangle$.

3. If θ is a bijective substitution of port names, then $A\theta \simeq A'\theta$.

Our first theorem is that observation equivalence is a congruence:

Theorem 1 *$\approx$ is a congruence.*

Proof idea: Establish that $\{\langle A|B, A'|B\rangle : A \approx A',\ L(A) \cap L(B) = \emptyset\}$ and $\{\langle A\langle a^\frown b\rangle, A'\langle a^\frown b\rangle\rangle : A \approx A',\ a, b \in L(A)\}$ are bisimulations. □

6 A Basis for Finite State Terms

In the following, assume some fixed interpretation for defining the behaviours of units, containing at least the units described in the preceding section, and assigning arbitrary behaviours to other units. In a practical design situation, it will be useful to know what modules are necessary to build a certain set of behaviours (up to equivalence). Conversely, it will be useful to know what behaviours can be constructed given a certain set of modules. This prompts the following definitions. With equivalence, we will in this section always mean observation equivalence.

For $\mathbf{M}' \subset \mathbf{M}$, the *closure* of $\mathbf{M}'$ is the set of terms containing modules only from $\mathbf{M}'$. We say that $\mathbf{M}'$ is a *basis* for a set of terms $\mathbf{T}$, if each term in $\mathbf{T}$ is equivalent with a term in the closure of $\mathbf{M}'$. Also, $\mathbf{M}'$ is a *proper basis* if it is a basis and each term in the closure of $\mathbf{M}'$ is equivalent with a term in $\mathbf{T}$. A set of modules $\mathbf{M}'$ is *independent* iff for all modules $M \in \mathbf{M}'$, any unit $M(\tilde{a})$ is not in the closure of $\mathbf{M}' - \{M\}$, i.e. no unit with a module in $\mathbf{M}'$ is equivalent with a term in the closure of the rest of $\mathbf{M}'$.

Intuitively, "$\mathbf{M}'$ is a basis for $\mathbf{T}$" means that the set of behaviours represented by $\mathbf{T}$ can be constructed (up to equivalence) with modules in $\mathbf{M}'$. If in addition $\mathbf{M}'$ is proper, then $\mathbf{T}$ 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 $\mathbf{T}$, what is an independent proper basis? Conversely, given an interesting set of modules $\mathbf{M}'$, is it independent, and for which set of terms is it a proper basis? We will answer only one such question here, and leave the rest as topics of further research.

Say that a term A is *finite state* iff $\{B : \text{ for some } \sigma : A \stackrel{\sigma}{\Longrightarrow} B\}$ is finite, i.e. the state space of the corresponding behaviour is finite. Recall the interpretation of the three modules in figure 1: the three way synchroniser (SY), arbiter (AR), and alternator (AL). The main theorem of this section is:

Theorem 2 *Assume an interpretation which assigns* SY, AR *and* AL *the behaviours as indicated in figure 1. Then* $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$ *is an independent proper basis for the set of finite state terms.*

Proof idea: To show that this set of modules is a basis, we must show that for every finite state term, there is an equivalent term containing only these modules. The proof is by direct construction of this equivalent term. Essentially, the operational semantics of a finite state term can be captured by a finite transition diagram. This diagram can be transformed to a flowgraph, where the nodes of the transition diagram correspond to alternators, and transitions correspond to links, possibly involving three way synchronisers (to generate external synchronisations) and arbiters (to make choices in case there are several outgoing transitions from a node). The term corresponding to this flowgraph is the required equivalent term. To prove that this basis is proper, it suffices to observe that SY, AR and AL are all finite state, and that the operators preserve the property of being finite state. Finally, to prove it independent, we establish for each of the modules a property, possessed by only the two other modules, and preserved by the operators and equivalence. For example, for AL, the property is "the term has exactly one state" (formally, "$A \stackrel{\alpha}{\Longrightarrow} A'$ implies $A \approx A'$"). □

A perhaps intuitively more appealing way to formulate theorem 2 is to say that a behaviour (regarded as a transition system) is *expressible* in an interpretation iff there is a term with that behaviour. The theorem then implies that any finite state behaviour is expressible in an interpretation containing SY, AR and AL, i.e. that any finite state behaviour can be built from three way synchronisers, arbiters, and alternators.

One other result is that $\{\mathrm{SY}, \mathrm{AR}\}$ 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 (cf. [Mil80]) are not closed under the linking operator, and can hence not have a proper basis. Of course, there might be derived operators which preserve confluence or determinacy, and these behaviours might have an interesting basis with respect to those operators.

In summary there are many interesting problems related to expressibility of terms; we have only solved one. Since such problems depend critically on particular sets of modules and their interpretations, it might be that experience with practical design work based on our ideas is necessary to determine which problems are important.

7 Expressibility of Operators

The linking operators in the algebra are binary, i.e. they connect exactly two ports. Assume that three ports d, e, f should be linked in a module A. Although there is no primitive three way linking operator, the effect of three way linking can be achieved with a three way synchroniser (cf. figure 1): in the term

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

an event will involve either none of d, e, f or all three of them. Similar constructions can be given for n way linking for arbitrary n by using $n-2$ three way synchronisers in parallel. Thus we can say that if $\mathrm{SY}(a,b,c)$ or a term with equivalent behaviour is expressible, then multi way linking is definable, or expressible, in the algebra and a designer can confidently use multi way links when constructing networks. In this section we will investigate which operators are expressible in this sense.

As might be expected, whether an operator is expressible depends critically on which equivalence is used. It turns out that strictly more operators are expressible up to trace equivalence than to observation equivalence. We will give a general expressibility result, and discuss some examples of operators from other process algebras. The only similar result known to us is [dS85], where expressibility in the synchronous process algebras MEIJE and SCCS is investigated.

In the following, let $\tilde{A}$ be a sequence of terms $A_1, \ldots, A_n$, and let $\tilde{L}$ be a sequence of sorts $L_1, \ldots, L_n$. We say that $\tilde{A}$ has sort $\tilde{L}$ iff $L(A_k) = L_k$ for all k $(1 \leq k \leq n)$. By an *n-ary context* C we mean a term with n holes in it[4], and we write $C(\tilde{A})$ for the term obtained by inserting $A_1, \ldots, A_n$ in C.

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

Let op be an n-ary operator symbol. We are only interested in operators which conform to the framework of sorted algebra, thus we assume that for each sequence of sorts $\tilde{L}$ one of the following holds: either (1) $op(\tilde{A})$ is undefined for all $\tilde{A}$ of sort $\tilde{L}$ — in this case we call *op undefined for* $\tilde{L}$, or (2) $op(\tilde{A})$ is defined and has the same sort L for all $\tilde{A}$ of sort $\tilde{L}$ — in this case we call *op defined for* $\tilde{L}$, and say that it returns sort L given sort $\tilde{L}$.

An n-ary operator op is *expressible w.r.t. the equivalence* $\simeq$ (or simply *expressible* if the equivalence is implicit) iff for each sequence of sorts $\tilde{L}$ for which it is defined, there exists a context C and bijective port name substitutions $\theta_1, \ldots, \theta_n$ with the property that for all $\tilde{A}$ of sort $\tilde{L}$,

$$C(\widetilde{A\theta}) \simeq op(\tilde{A})$$

Here, $\widetilde{A\theta} = A_1\theta_1, \ldots, A_n\theta_n$. We call this C an *expressing context* for op. Note that an expressible operator may have different expressing contexts for different sequences of sorts of its argument. Intuitively, an operator is expressible if a designer can construct a corresponding network whenever he knows the sort(s) of its argument(s).

An n-ary operator op is *contextual*[5] iff for each sequence of sorts $\tilde{L}$ for which it is defined, the semantics is defined as follows:

1. There are (possibly infinitely many) n-ary operator symbols $op_1, op_2, \ldots$; each of these return the same sort L given sort $\tilde{L}$, and $op = op_1$.

2. There is a set of inference rules determining the semantics of each op_i. Each rule takes the following form, where i and j are natural numbers, $\alpha \subseteq L$, and $\alpha_k \subseteq L_k$ for all $1 \leq k \leq n$:

$$\frac{A_1 \xrightarrow{\alpha_1} A_1' \quad \cdots \quad A_n \xrightarrow{\alpha_n} A_n'}{op_i(\tilde{A}) \xrightarrow{\alpha} op_j(\widetilde{A'})}$$

   Here, $\widetilde{A'} = A_1', \ldots, A_n'$. As usual in such rules the terms $\tilde{A}, \widetilde{A'}$ are implicitly universally quantified, i.e. they should be read "for all $\tilde{A}$ and $\widetilde{A'}$ of the correct sorts satisfying the premise, the conclusion holds".

3. Each op_i has at least one rule, the *idling rule*:

$$\frac{A_1 \xrightarrow{\emptyset} A_1' \quad \cdots \quad A_n \xrightarrow{\emptyset} A_n'}{op_i(\tilde{A}) \xrightarrow{\emptyset} op_i(\widetilde{A'})}$$

[5] I am grateful to Kim Larsen for suggesting this definition. A similar idea of a "general static process construction" is presented in [LT87], which attributes the idea to Aczel ([Acz84]).

Note that each contextual operator may have many different sets of rules: one set for each sequence of sorts of its arguments for which the operator is defined. Intuitively, an operator is contextual if the derivative of a transition contains the derivatives of the arguments contributing to the transition. The idling rule ensures that if all arguments idle (do nothing externally), then the whole term has the possibility to idle.

All arguments of the operator must be present in the premises of the rules. This is sufficiently general since any argument may "inact" (i.e. perform the event $\emptyset$ and remain in the same state). This is one point where the assumption of inaction transitions result in a more concise formulation than would otherwise be possible.

For particular operators it may be possible to present the rules in a more attractive way than by giving one rule for each $i, j, \alpha, \alpha_1, \ldots, \alpha_k$. For example, the parallel operator in our calculus is contextual, as are the linking operators.

A contextual operator is *finite state* iff for each sequence of sorts for which the operator is defined, only finitely many operator symbols $op_1, \ldots, op_N$ are needed in the definition of the semantics. We can now formulate a theorem for expressibility w.r.t. $\approx$ of operators:

Theorem 3 (a) *Assume a module set and an interpretation such that any behaviour is expressible. Then all contextual operators are expressible.*

(b) *Assume a module set and an interpretation such that any finite state behaviour is expressible (by theorem 2, the modules* SY, AR, AL *are enough). Then all finite state contextual operators are expressible.*

Proof idea: Let *op* be a contextual operator, and consider a sequence of sorts $\tilde{L}$ for which *op* is defined. It is possible to directly establish the expressing context from the set of rules in the definition of *op*. This context is of the form

$$C(\widetilde{A\theta}) = (OP|A_1\theta_1|\cdots|A_n\theta_n)\langle LINKS\rangle$$

where OP is a term encoding the rules and functioning as a "controller": it controls the transitions of the arguments of *op* according to the rules, and generates the appropriate external events. If *op* is finite state, then OP is finite state. □

A converse result is not difficult: all expressible operators can be defined as contextual operators. The proof is that units, parallel, and linking can be given contextual definitions (units are 0-ary operators), and that compositions of contextual operators yield contextual operators. Similarly, if all units are finite state, then all expressible operators have equivalent contextual finite state definitions. Thus we get the following corollary:

Operator	Usual notation	$\approx$	$=_T$
CCS parallel	$A\|B$	Yes	Yes
CCS nondeterministic choice	$A+B$	No	Yes
CCS prefixing	$a.A$	No	Yes
CCS restriction	$A\backslash a$	Yes	Yes
SCCS synchronous parallel	$A \times B$	No	No
CSP parallel	$A \parallel B$	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: Expressibility of some operators.

Corollary 4 *Assume that all behaviours (exactly the finite state behaviours) are expressible, and let $\simeq$ be any equivalence weaker than or equal to $\approx$. Then an operator op is expressible w.r.t. $\simeq$ iff $op \simeq op'$ for some (finite state) contextual op'.*

Here, by $op \simeq op'$ we mean that for all $\tilde{A}$, $op(\tilde{A}) \simeq op(\tilde{A})$. Unfortunately, it is not in general easy to determine whether a particular operator can be given a contextual definition.

It is interesting to consider some popular operators from other process algebras here. Table 3 summarises the expressibility of operators w.r.t. $\approx$ and $=_T$, assuming that all finite state behaviours are expressible. We use the standard operational definitions of these operators as elaborated in the appendix.

As can be seen, the operators fall into three groups. The first group consists of operators which are expressible w.r.t. $\approx$, and hence also w.r.t. the strictly weaker $=_T$. The proof that these operators are expressible follows from theorem 3 by providing finite state contextual definitions.

The second group consists of operators expressible w.r.t. $=_T$ but not w.r.t. $\approx$. Again, the proof of expressibility is by giving equivalent (w.r.t. $=_T$) contextual definitions. The inexpressibility proofs are perhaps more interesting — some of the results may seem surprising. Consider as a typical example the CCS prefixing operator. This operator has the only rule

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

The proof that prefixing is not expressible w.r.t. $\approx$ goes as follows. Assume that there is a contextual operator op such that $op(A) \approx a.A$ for all A. Choose A such

that $A \xrightarrow{\emptyset} A'$ with $a.A \not\approx a.A'$. By the idling rule, we get that $op(A) \xrightarrow{\emptyset} op(A')$. By definition, $op(A) \approx a.A$ requires a term B such that $a.A \stackrel{\emptyset}{\Longrightarrow} B$ and $op(A') \approx B$. We deduce from $a.A \stackrel{\emptyset}{\Longrightarrow} B$ that $B = a.A$. 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 expressible w.r.t. $=_T$, and hence not w.r.t. $\approx$. The inexpressibility proofs for these cases are simple: it can be shown that these operators do not preserve $=_T$. This means that they can not be expressed by our disjoint parallel and linking operators since these preserve $=_T$.

It would be interesting to study expressibility with respect to some other equivalence, e.g. failure equivalence or testing equivalence. We leave this as a topic for further study.

8 Axiomatisation

It is clear that there are some equalities which hold no matter what the interpretation of the behaviours of units are. For example, the equation

$$C_1|C_2 \approx C_2|C_1$$

is true regardless of how C_1 and C_2 are interpreted. The purpose of this section is to formally characterise the equations which are valid in this sense.

Let $\simeq$ be any equivalence, and let I be an interpretation. Recall that $I \models A \simeq A'$ means that $A \simeq A'$ holds in I. We say that $A \simeq A'$ is *valid*, written $\models A \simeq A'$, iff $I \models A \simeq A'$ for all interpretations I.

Consider the axiom system in table 4. We say that A is *provably equivalent* with A', written $\vdash A = A'$, iff $A = A'$ can be derived from these axioms and equational reasoning (i.e. that $=$ is an equivalence and is preserved by the operators). The axioms are impure since they use side conditions. This is of no major concern: the side conditions always relate to the syntactic structure of terms.

The axioms (C1) and (C2) state that $|$ is commutative and associative. The axioms (L1) and (L2) state that the order of a and b is unimportant in $\langle a \frown b \rangle$, and that linking is commutative. (D) is a distributive law: linking distributes over parallel. (S) says that the names of linked ports can be arbitrarily chosen as long as distinct ports get distinct names. Here we require that c does not occur in A, i.e. that it is a fresh port name. Alternatively we could require that $[c/a]$ is a bijective substitution which does not affect any name in $L(A)$ except a. Finally, $(\emptyset)$ is a law for removal of "useless" terms: any term B with the empty sort can be removed from a composition $A|B$. Note that from (C1) and $(\emptyset)$ any two terms with the empty sort are provably equivalent.

(C1)	$A\|B$	$=$	$B\|A$	
(C2)	$(A\|B)\|C$	$=$	$A\|(B\|C)$	
(L1)	$A\langle a^\frown b\rangle$	$=$	$A\langle b^\frown a\rangle$	
(L2)	$A\langle a^\frown b\rangle\langle c^\frown d\rangle$	$=$	$A\langle c^\frown d\rangle\langle a^\frown b\rangle$	
(D)	$(A\|B)\langle a^\frown b\rangle$	$=$	$(A\langle a^\frown b\rangle)\|B$	$(a, b \notin L(B))$
(S)	$A\langle a^\frown b\rangle$	$=$	$A[c/a]\langle c^\frown b\rangle$	(c does not occur in A)
($\emptyset$)	$A\|B$	$=$	A	$(L(B) = \emptyset)$

Table 4: Axioms for parallel and linking. Here $[c/a]$ represents syntactic substitution of c for a.

Theorem 5 (Soundness) *If* $\vdash A = A'$, *then* $\models A \approx A'$.

Proof idea: Establish bisimulations corresponding to the axioms. For example, for (C1) the corresponding bisimulation is $\{\langle A|B, B|A\rangle : L(A) \cap L(B) = \emptyset\}$. It is easy to show that $\approx$ is an equivalence, and by theorem 1 it is a congruence. □

Recall the definition of deadlock and $A \stackrel{\sigma}{\Longrightarrow} \bullet$ from section 5. An equivalence $\simeq$ is a *behaviour equivalence* iff it is a congruence, weaker than or equal to $\approx$, and

$$A \simeq A' \text{ and } A \stackrel{\sigma}{\Longrightarrow} \bullet \text{ implies that } A' \stackrel{\sigma}{\Longrightarrow} \bullet$$

Note that if $\simeq$ is a congruence, then it must be sensitive to sorts, i.e. $A \simeq A'$ implies $L(A) = L(A')$.[6]

The perhaps most interesting, surprising, and difficult result presented in this paper is:

Theorem 6 (Completeness) *Let* $\simeq$ *be any behaviour equivalence. Then*

$$\models A \simeq A' \text{ implies } \vdash A = A'$$

Proof idea: As in other related completeness proofs, the idea is to show that any term is reducible to a normal form, and that equivalent normal forms are provably equivalent. However, the details in this proof are substantially different from related proofs. For example, in [HM85] it is possible to reduce terms to a normal form which is very close to the intended semantics. We can not use that idea since we lack the operators (prefixing and nondeterministic choice) for

[6] Also, note that trace equivalence ($=_T$) is not a behaviour equivalence with this definition. Results in [Mai86] and [Hen87] seem to indicate that equivalences known as *failure equivalence* and *testing equivalence* coincide as the largest behaviour equivalence.

such a normal form. In [Mil79] another normal form, similar to our normal form, is used. However, there the semantics and the notion of validity is related to flowgraphs rather than behaviours, and hence the completeness proof in that paper is comparatively straightforward.

The normal form used in this proof is

$$(C_1|\cdots|C_n)\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$$

where $C_1, \ldots, C_n$ are units. Say that such a normal form has an *isolated part* $\tilde{C} \subseteq \{C_1, \ldots, C_n\}$ if all the ports in $\tilde{C}$ are linked to each other by $\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$, i.e. no ports are external or linked to any unit outside $\tilde{C}$. In a flowgraph, an isolated part is a closed subgraph without any external port. Using (D) and $(\emptyset)$, any normal form with nonempty sort is provably equivalent with a normal form without any isolated parts.

The difficult part of the proof is to establish a set of properties of terms such that for any normal forms A, A' without isolated parts:

1. If A but not A' has one of the properties, then there exist an interpretation I and a sequence of events σ, such that in I it holds that $A \stackrel{\sigma}{\Longrightarrow} \bullet$ and not $A' \stackrel{\sigma}{\Longrightarrow} \bullet$.

2. If A and A' have exactly the same properties, then they are isomorphic.

Here, two normal forms are isomorphic if they differ only in the order of components and names of linked ports. It is not difficult to show that two isomorphic normal forms are provably equivalent.

The major difficulty of the proof is to establish the "distinguishing" interpretation I under 1 above. One source of trouble is that several different units containing the same module might occur in A and A'; in that case the interpretation of these units must be uniform. Another problem is that the linking structure is static. It might be that a property possessed by A but not by A' relates to some part of the terms, and that the only external ports are in another part of the terms. The task of I is to manifest this property with a transition sequence involving the external ports, and to do so using only the given links. □

Corollary 7 *Validities for all behaviour equivalences coincide.*

Proof: Let $\simeq$ and $\simeq'$ be two behaviour equivalences. By completeness, soundness, and the fact that $\simeq'$ is weaker than $\approx$, we get

$$\models A \simeq A' \text{ implies } \vdash A = A' \text{ implies } \models A \approx A' \text{ implies } \models A \simeq' A' \quad \square$$

It is interesting to consider the axioms in relation to the "structural" interpretation of terms into flowgraphs. This type of interpretation is used in [Mil79],

and a complete axiomatisation for isomorphism of flowgraphs is presented. This axiomatisation is similar to ours in spirit — the main difference is the axiom $(\emptyset)$, which of course would not be sound with respect to a structural interpretation. Our axiomatisation can be said to capture "isomorphism of flowgraphs modulo isolated parts", i.e. two flowgraphs are equated iff there is an isomorphism relating their non isolated parts. The import of the completeness result is that two flowgraphs are so equated iff their behaviours are equivalent in all operational interpretations. In other words, the only way for flowgraphs to be different while being operationally equivalent in all interpretations is to differ in their isolated parts.

9 Conclusion

We have explored an algebraic language for description of synchronisation flow networks. There are many interesting questions left unanswered:

- For which sets of terms are $\{\mathrm{SY}, \mathrm{AL}\}$ and $\{\mathrm{AR}, \mathrm{AL}\}$ proper bases? 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?

- Is it possible to characterise the operators which are *not* expressible w.r.t. $\approx$, or w.r.t. $=_T$? Our present result (corollary 4) says that these are exactly the operators without equivalent contextual operators, but a more constructive characterisation would be interesting.

- What is a complete axiomatisation for observation equivalence in some particular interpretation, e.g. with the modules $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$?

- Is our axiomatisation complete with respect to trace equivalence? We believe it is but have not found a proof.

Finally, three notes about alternative formulations of the algebra. The first note concerns substitution. We have not introduced port name substitution (or "renaming") as an explicit operator in the algebra. The reason for this is partly aesthetic: we want to have as few operators as possible, and want all operators to correspond to construction steps. Renaming operators could be added to the algebra as follows: for each bijective substitution of port names θ, add the postfix operator θ to the algebra. Define $L(A\theta) = (L(A))\theta$, and define the semantics by

$$\frac{A \xrightarrow{\alpha} A'}{A\theta \xrightarrow{\alpha\theta} A'\theta}$$

(R1)	$(A\|B)\theta$	$=$	$A\theta\|B\theta$	
(R2)	$A\langle a^\frown b\rangle\theta$	$=$	$A\theta\langle\theta(a)^\frown\theta(b)\rangle$	
(R3)	$A\theta\theta'$	$=$	$A(\theta'\circ\theta)$	
(R4)	$A\theta$	$=$	A	$(\theta(a)=a$ for all $a\in L(A)$

Table 5: Axioms for explicit renaming operators.

Now, all results in this paper carry over to the extended algebra, and there is no longer any need for substitution of port names at the meta level. The completeness result still holds if axiom (S) is replaced by axioms (R1-R4) in table 5. The appropriate normal form is

$$(C_1\theta_1|\cdots|C_n\theta_n)\langle a_1{}^\frown b_1\rangle\cdots\langle a_k{}^\frown b_k\rangle$$

and the rest of the completeness proof is as before.

One possible advantage with the renaming operators is that there is no longer any need for different instances of modules. Each module can be given some arbitrary sort, and different instances can be expressed with renaming operators.

The second note concerns inaction. In our algebra, the event $\emptyset$ corresponds to either some internal action or complete inaction (the inaction transition in the units). The reason for equating inaction with internal action is again aesthetic: it leads to simpler formulations of the results and of the proofs. This is in contrast with e.g. CCS, where the τ action corresponds to internal action but not complete inaction. In CCS it is important to maintain a distinction: through the + operator, internal actions may preempt other actions while inactions may not. In our calculus, events $\emptyset$ may also preempt other actions, but there are no contexts where such preemptions can be introduced. However, it can be argued that equating internal action with inaction precludes the study of aspects related to divergence and finer equivalences such as strong congruence. We therefore briefly indicate how the distinction can be maintained within our algebra. Omit the requirement of the inaction transition of units, and add two rules to the parallel operator (otherwise, the parallel operator would be synchronous rather than asynchronous):

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

All the results in this paper carry over to this variant. Care has to be taken in the definition of contextual operators: although inaction and internal action are now distinct, no contextual operator may exploit this distinction. One way to

formulate this is to define $\xrightarrow{\epsilon}$ to be the union of $\xrightarrow{\emptyset}$ and the identity relation on terms, and use $\xrightarrow{\epsilon}$ instead of $\xrightarrow{\emptyset}$ in the premises of the semantic rules for contextual operators.

The third and final note concerns formal variables. It is clear that in section 8, a unit $M(\tilde{a})$ is treated approximately as a variable which is parameterised on a sequence of port names. Nevertheless we have avoided the word "variable" in this paper. The reason is that usually, two open terms (i.e. terms with variables) are considered equivalent if all substitutions of closed terms (terms without variables) for variables make them equivalent. Now, if units were variables, then our algebra would not contain any closed terms in this sense. It would be possible to remedy this by introducing a set of "constants", and formulate the completeness result with respect to truth in all interpretations of the constants. While this would possibly conform better with standard treatments of algebras, it would in our particular case be technically less elegant and seems not to yield any substantial insights.

The standard way of introducing variables in algebras is as placeholders for arbitrary terms (or in sorted algebras for arbitrary terms of correct sort). This can easily be done in our algebra: assume for each sort L a set of *variables* $\mathbf{X}_L$. The main difference between variables and units will then be that only the latter are explicitly parameterised on the port names. Since one of our axioms, namely (S), uses a form of syntactic substitution that does not affect variables, our axiomatisation is no longer sound if taken as it stands. One reasonable restriction on (S) is to apply it only to terms without variables; this would make the axiomatisation sound but incomplete. More specifically, there seems to be no good candidate for a normal form. As an example of two terms which are equivalent in all interpretations but not provably equivalent, let X be a variable with sort $\{a\}$ and consider

$$A = \Big(((X|M(b,c,d))\langle a^\frown b\rangle)|X\Big)\langle a^\frown c\rangle$$

and

$$A' = \Big(((X|M(b,c,d))\langle a^\frown c\rangle)|X\Big)\langle a^\frown b\rangle$$

Now the flowgraphs corresponding to A and A' are isomorphic; hence they must be behaviour equivalent in all interpretations. But A and A' are not provably equivalent, since there is no way to apply a "syntactic" substitution to X. A complete axiomatisation for the algebra with variables can be obtained by adding the explicit renaming operators — for example, both A and A' would then be provably equivalent with

$$(X[e/a]|M(b,c,d)|X)\langle e^\frown b\rangle\langle a^\frown c\rangle$$

Acknowledgments

I am deeply grateful to Robin Milner and David Walker for innumerable 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 also been received from Kim Larsen.

This work was carried out under a grant from the British Science and Engineering Research Council.

References

[AB84] D Austry and G Boudol. Algèbre de processus et synchronisation. *Theoretical Computer Science*, 30(1):91–131, 1984.

[Acz84] Peter Aczel. A simple version of SCCS and its semantics. 1984. Unpublished Notes, University of Edinburgh.

[BHR84] S. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[BT85] J. Bergstra and J. Tucker. Top-down design and the algebra of communicating processes. *Science of Computer Programming*, 5(2):171–199, 1985.

[Com82] *Computer*, 15(2). February 1982. Special issue on data flow machines.

[dS85] R de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37(3):245–267, 1985.

[GTWW77] J. A. Gougen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977.

[Hen87] Matthew Hennessy. Why testing equivalence is natural. 1987. Handwritten Notes.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[Jon85] Bengt 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.

[Kah74] G Kahn. The semantics of a simple language for parallel programming. In *IFIP 74*, pages 471–475, 1974.

[KP85] R M 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.

[LT87] Kim Larsen and Bent Thomsen. *Compositional Proofs by Partial Specification of Processes.* Technical Report R 87-20, Institut for Elektroniske Systemer, Aalborg University, 1987.

[Mai86] Michael Main. Demons, catastrophes and communicating processes. 1986. Department of Computer Science, University of Colorado.

[Mil79] Robin Milner. Flowgraphs and flow algebras. *J. ACM*, 26(4):794–818, 1979.

[Mil80] Robin Milner. *A Calculus of Communicating Systems.* Volume 92 of *Lecture Notes of Computer Science*, Springer Verlag, 1980.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Mil85] George Milne. CIRCAL and the representation of communication, concurrency and time. *ACM Transactions on Programming Languages and Systems*, 7(2):270–298, 1985.

[Par83] David 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.

[SN85] J Staples and V L Nguyen. A fixpoint semantics for nondeterministic data flow. *J. ACM*, 32(2):411–444, 1985.

[Win80] G Winskel. *Events in Computation.* PhD thesis, Dep. of Computer Science, University of Edinburgh, 1980.

Appendix: Proof Details

Proof of theorem 1

The proof of theorem 1 follows directly from lemmas 8 – 11 below.

Lemma 8 *If $A \approx B$ and $L(A) \cap L(C) = \emptyset$ then $A|C \approx B|C$.*

Proof: Clearly $L(A|C) = L(B|C) = L(A) \cup L(C)$. Let $\mathcal{R}$ be defined by

$$\mathcal{R} = \{\langle A|C, B|C\rangle : A \approx B \text{ and } L(A) \cap L(C) = \emptyset\}$$

We prove that $\mathcal{R}$ is a bisimulation. Since $\mathcal{R}$ is symmetric it suffices to prove $\mathcal{R}$ a simulation. Assume $A|C\,\mathcal{R}\,B|C$ and $A|C \xrightarrow{\alpha} D$. Then, by the semantic rule for $|$, it must hold that $D = A'|C'$ and $A \xrightarrow{\alpha_1} A'$ and $C \xrightarrow{\alpha_2} C'$ and $\alpha = \alpha_1 \cup \alpha_2$. Since $A \approx B$, we get $B \stackrel{\alpha_1}{\Longrightarrow} B'$ with $A' \approx B'$. Also, $L(A) = L(A')$ and $L(C) = L(C')$, thus $L(A') \cap L(C') = \emptyset$. There are two cases:

1. $\alpha_1 \neq \emptyset$. Thus, for some $n, m \geq 0$:

$$B \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{n \text{ times}} \xrightarrow{\alpha_1} \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{m \text{ times}} B'$$

   By n inaction transitions of C and m inaction transitions of C':

$$C \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{n \text{ times}} \xrightarrow{\alpha_2} \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{m \text{ times}} C'$$

   whence by $n + m + 1$ applications of the rule for $|$ gives

$$B|C \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{n \text{ times}} \xrightarrow{\alpha_1 \cup \alpha_2} \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{m_1 \text{ times}} B'|C'$$

   That is, $B|C \stackrel{\alpha}{\Longrightarrow} B'|C'$. From $A' \approx B'$ it follows $A'|C'\,\mathcal{R}\,B'|C'$ as required.

2. $\alpha_1 = \emptyset$, i.e. $\alpha = \alpha_2$. Then for some $n \geq 0$:

$$B \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{n \text{ times}} B'$$

   By possibly adding an inaction transition, this holds for some $n \neq 0$. By $n - 1$ inaction transitions of C and n applications of the rule for $|$ we get

$$B|C \underbrace{\xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset}}_{n-1 \text{ times}} \xrightarrow{\alpha} B'|C'$$

   That is, $B|C \stackrel{\alpha}{\Longrightarrow} B'|C'$. From $A' \approx B'$ it follows $A'|C'\,\mathcal{R}\,B'|C'$ as required. □

Lemma 9 *If $A \approx B$ and $L(A) \cap L(C) = \emptyset$ then $C|A \approx C|B$*

Proof: Symmetric with the proof of lemma 8. □

Lemma 10 *If $A \approx B$ and $a, b \in L(A)$, $a \neq b$ then $A\langle a^\frown b\rangle \approx B\langle a^\frown b\rangle$.*

Proof: Clearly $L(A\langle a^\frown b\rangle) = L(B\langle a^\frown b\rangle) = L(A) - \{a, b\}$. Define $\mathcal{R}$ by

$$\mathcal{R} = \{\langle A\langle a^\frown b\rangle, B\langle a^\frown b\rangle\rangle : A \approx B \text{ and } a, b \in L(A)\}$$

We prove that $\mathcal{R}$ is a bisimulation. Since $\mathcal{R}$ is symmetric it suffices to prove $\mathcal{R}$ a simulation. Assume $A\langle a^\frown b\rangle \mathcal{R} B\langle a^\frown b\rangle$ and $A\langle a^\frown b\rangle \xrightarrow{\alpha} C$. Note that this implies $a, b \notin \alpha$. By the rules for linking there are two cases:

1. $A \xrightarrow{\alpha\cup\{a,b\}} A'$ and $C = A'\langle a^\frown b\rangle$. From $A \approx B$ follows $B \overset{\alpha\cup\{a,b\}}{\Longrightarrow} B'$ with $A' \approx B'$. This means that
$$B \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} \xrightarrow{\alpha\cup\{a,b\}} \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} B'$$
By repeated applications of the rules for linking we get
$$B\langle a^\frown b\rangle \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} \xrightarrow{\alpha} \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} B'\langle a^\frown b\rangle$$
That is, $B\langle a^\frown b\rangle \overset{\alpha}{\Longrightarrow} B'\langle a^\frown b\rangle$. From $A' \approx B'$ and $L(A) = L(A')$ it follows that $A'\langle a^\frown b\rangle \mathcal{R} B'\langle a^\frown b\rangle$ as required.

2. $A \xrightarrow{\alpha} A'$ and $C = A'\langle a^\frown b\rangle$. From $A \approx B$ follows $B \overset{\alpha}{\Longrightarrow} B'$ with $A' \approx B'$. This means that
$$B \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} \xrightarrow{\alpha} \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} B'$$
By repeated applications of the rules for linking (remember $a, b \notin \alpha$) we get
$$B\langle a^\frown b\rangle \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} \xrightarrow{\alpha} \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} B'\langle a^\frown b\rangle$$
That is, $B\langle a^\frown b\rangle \overset{\alpha}{\Longrightarrow} B'\langle a^\frown b\rangle$. From $A' \approx B'$ and $L(A) = L(A')$ it follows that $A'\langle a^\frown b\rangle \mathcal{R} B'\langle a^\frown b\rangle$ as required. □

Lemma 11 *Substitution of port names preserves $\approx$.*

Proof: Let θ be a bijective substitution of port names. Define $\mathcal{R}$ by

$$\mathcal{R} = \{\langle A\theta, B\theta\rangle : A \approx B\}$$

We prove $\mathcal{R}$ a bisimulation. Since $\mathcal{R}$ is symmetric it suffices to prove that $\mathcal{R}$ a simulation. So assume $A\theta \mathcal{R} B\theta$ and $A\theta \xrightarrow{\alpha} C$. Then $A \xrightarrow{\beta} A'$ where $\beta\theta = \alpha$ and $C = A'\theta$. From $A \approx B$ follows $B \overset{\beta}{\Longrightarrow} B'$ with $A' \approx B'$. This means $B\theta \overset{\beta\theta}{\Longrightarrow} B'\theta$, i.e. $B\theta \overset{\alpha}{\Longrightarrow} B'\theta$. From $A' \approx B'$ follows $A'\theta \mathcal{R} B'\theta$ as required. □

Proof of theorem 2

The proof of theorem 2 stretches over several lemmas. In the following we will use flowgraphs as graphical representations of terms; in particular we will use the flowgraph symbols for the modules in figure 1. We assume an interpretation which assigns behaviours according to that figure, and assigns arbitrary behaviours to other modules.

Let A be a term. The *state space* of A is defined to be the set

$$\{A' : \exists \sigma : A \stackrel{\sigma}{\Longrightarrow} A'\}$$

The *transition diagram corresponding to* A, written $T(A)$, is the transition diagram where the set of nodes is the state space of A, the initial node is A, and $B \stackrel{\alpha}{\longrightarrow} B'$ is an edge in $T(A)$ iff B has a transition $B \stackrel{\alpha}{\longrightarrow} B'$.

With the *sort* $L(T)$ of a transition diagram T we will mean the set of port names occurring in events of transitions in T. Thus, $a \in L(T)$ iff $\exists A, A', \alpha$ such that $a \in \alpha$ and $A \stackrel{\alpha}{\longrightarrow} A'$ 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 event of any transition in $T(A)$. However, it is always the case that $L(T(A)) \subseteq L(A)$.

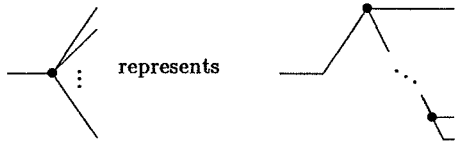
In the following we extend the notion of bisimulation to transition diagrams in the obvious way. If G and G' are transition diagrams or terms, and there exists a bisimulation between G and G', we write $G \approx_b G'$ and say that G and G' are *bisimilar*. Thus, if A and B are terms, then $A \approx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

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

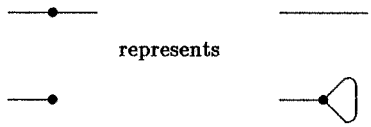
Lemma 12 *Let T be a transition diagram. Then there exists a bisimilar self-loop free transition diagram T'.*

Proof: Build T' from T as follows: for every node A in T, add a new node A' to T'. For every transition $A \stackrel{\alpha}{\longrightarrow} B$ in T, add the transition $A' \stackrel{\alpha}{\longrightarrow} B$ in T'. Finally, replace every transition $A \stackrel{\alpha}{\longrightarrow} A$ with the transition $A \stackrel{\alpha}{\longrightarrow} A'$. It is straightforward to demonstrate a bisimulation between T and T': a node A in T bisimulates the nodes A and A' in T'. □

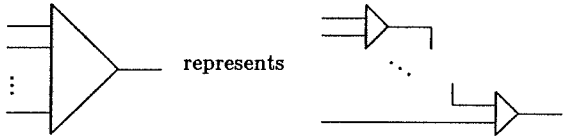
We next introduce flowgraph symbols for multi way synchronisers (constructed from three way synchronisers, cf. figure 1) and multi way arbiters (constructed from ordinary arbiters). A *multi way synchroniser* is depicted as a filled circle with arbitrarily many external ports, and represents a combination of three way synchronisers:



Thus, a multi way synchroniser requires that all ports are involved in the event of 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 event involving its only port. Graphically:



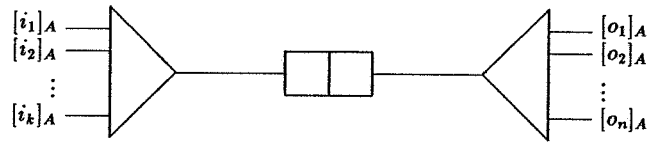
A *multi way arbiter*, depicted as a big triangle, represents a combination of arbiters:



Thus, the multi way arbiter requires that the righthand port and exactly one of the lefthand 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 event involving the righthand port. Graphically:

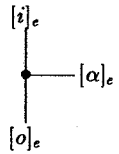
Let T be a finite self-loop free transition diagram. A *canonic term* corresponding to T, written $CT(T)$, is any term corresponding to the flowgraph constructed as follows:

1. For each node A in T there is a flowgraph 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. The flowgraph N_A has ports[7] $[i_1]_A, \ldots, [i_k]_A$ 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 diagram, then the alternator is originally in its righthand state. This means that N_A may do a transition involving exactly one of its righthand ports $[o_1]_A, \ldots, [o_n]_A$; after this transition the alternator will be in the lefthand state. If A is not the initial node, then the alternator is originally in its lefthand state. This means that N_A may do a transition involving exactly one of its lefthand ports $[i_1]_A, \ldots, [i_k]_A$; after this transition the alternator will be in the righthand state.

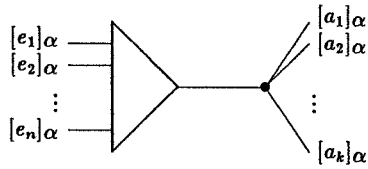
2. For each edge $e : A \xrightarrow{\alpha} A'$ in T there is a flowgraph N_e with three ports $[i]_e$, $[o]_e$, and $[\alpha]_e$ defined with a three way synchroniser:



Thus, N_e requires an event involving all three ports.

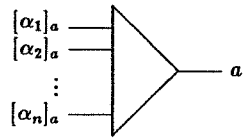
3. For each $\alpha \subseteq L(T)$ there is a flowgraph N_α as follows. Let $e_1, \ldots, e_n$ be the edges with event α in T. Let $\alpha = \{a_1, \ldots, a_k\}$. Then N_α has ports $[e_1]\alpha, \ldots, [e_n]\alpha$ and $[a_1]\alpha, \ldots, [a_k]\alpha$ and is defined as:

[7] We use port names enclosed in square brackets to represent fresh and unique port names.



Thus, if $n, k \geq 1$ then N_α consists of $n-1$ arbiters and $k-1$ three way synchronisers. An event in N_α will involve exactly one of the lefthand ports and all of the righthand ports.

4. For each $a \in L(T)$ there is a flowgraph N_a as follows. Let $\alpha_1, \ldots, \alpha_n$ be the subsets of $L(T)$ containing a. The flowgraph N_a has one port a and n ports $[\alpha_1]_a, \ldots, [\alpha_n]_a$ and is defined as:



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

5. The flowgraph corresponding to the canonic term $CT(T)$ is obtained by composing in parallel all N_A, N_e, N_α and N_a from steps 1-4 above, and linking the ports as follows:

   (a) For each edge $e = A \xrightarrow{\alpha} A'$ in T, link
      i. $[i]_e$ to $[e]_A$.
      ii. $[o]_e$ to $[e]_{A'}$.
      iii. $[\alpha]_e$ to $[e]_\alpha$.

   (b) For each $\alpha \subseteq L(T)$ and $a \in \alpha$, link port $[\alpha]_a$ to $[a]_\alpha$.

   Thus, the only non linked ports of $CT(T)$ are the ports in $L(T)$ in step 4 above.

This completes the definition of canonic term. Actually,we have described how to build a "canonic flowgraph" rather than term, but the translation from this flowgraph to a term is trivial. The only modules in $CT(T)$ are three way synchronisers, arbiters, and alternators.

Lemma 13 *Let T be a finite self-loop free transition diagram. Then*

$$T \approx_b CT(T)$$

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 AL_B, in $CT(T)$. By the definition of canonic term, it is easily seen that in $CT(T_B)$ all alternators except AL_B are in their lefthand states, and AL_B is in its righthand state (here, "right" and "left" refer to the orientation in the definition of canonic term, step 1). Thus, $CT(T_B)$ and $CT(T_{B'})$ differ only in that AL_B and $\mathrm{AL}_{B'}$ 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 $CT(T_B)$, where we use the flowgraph notation from the definition of canonic term:

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

It is clear from the definition of canonic term, step 5, that all ports in these transitions are linked, with the exception $a_1, \ldots, a_n$. Also, in $CT(T_B)$ all networks 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 event α. As a result of this transition, the alternators AL_B and $\mathrm{AL}_{B'}$ 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 $T_B \stackrel{\alpha}{\Longrightarrow} T_{B'}$. There are two cases:

1. All subterms corresponding to N_e (for edges e in T) do inaction transitions. It is evident from the linking structure of $CT(T)$ that in this case, all subterms must do inaction transitions. Hence $\alpha = \emptyset$ and $C = CT(T_B)$. But $T_B \stackrel{\emptyset}{\Longrightarrow} T_B$ is trivially true.

2. At least one subterm corresponding to N_e does a transition with event $\{[i]_e, [o]_e, [\beta]_e\}$. Then $[i]_e$ must be linked with a righthand port of N_B and $[o]_e$ with a lefthand port of $N_{B'}$ for some $B' \neq B$. From the definition of N_B, at most one such subterm N_e can be involved in this transition. The effect of the transition is that AL_B and AL'_B change states, i.e. the resulting term C is $CT(T_{B'})$. Furthermore, by steps 3 and 4 in the definition of canonic term, the event of the transition is β. Thus, $\beta = \alpha$. From steps 2 and 5 in the same definition, the reason for the existence of N_e and its links with N_B and $N_{B'}$ must be that T has an edge $B \stackrel{\alpha}{\longrightarrow} B'$. This, of course, implies $T_B \stackrel{\alpha}{\Longrightarrow} T_{B'}$.

This concludes the proof that $\mathcal{R}$ is a bisimulation, and hence the proof of lemma 13. □

The proof that $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$ is independent follows from the lemmas below.

Lemma 14 *No term in the closure of* $\{\mathrm{SY}, \mathrm{AR}\}$ *is equivalent with* $\mathrm{AL}(a, b)$.

Proof: Say that a term A is *single state* iff

$$A \stackrel{\alpha}{\Longrightarrow} A' \text{ implies } A \approx A'$$

The lemma follows from the following observations:

1. The units $\mathrm{SY}(a, b, c)$ and $\mathrm{AR}(a, b, c)$ 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 $A \approx B$ then B is single state.

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

The proofs of these observations are trivial, and we prove only 2 and 4 here. For 2, $A|B \stackrel{\alpha}{\Longrightarrow} C$ implies $C = A'|B'$ with $A \stackrel{\alpha_1}{\Longrightarrow} A'$ and $B \stackrel{\alpha_2}{\Longrightarrow} B'$, which since $A \approx A'$ and $B \approx B'$ implies $A'|B' \approx A|B$. For 4, $B \stackrel{\alpha}{\Longrightarrow} B'$ implies (since $A \approx B$) that $A \stackrel{\alpha}{\Longrightarrow} A'$ with $A' \approx B'$; thus $B \approx A \approx A' \approx B'$.

By 1-4 above, if a term is equivalent with a term in the closure of $\{\mathrm{SY}, \mathrm{AR}\}$, then it is single state. But by 5, $\mathrm{AL}(a, b)$ is not single state. □

Lemma 15 *No term in the closure of* $\{\mathrm{SY}, \mathrm{AL}\}$ *is equivalent with* $\mathrm{AR}(a,b,c)$.

Proof: Say that a term A is *intersection closed* iff for all $\alpha, \beta \subseteq L(A)$ it holds that

$$A \stackrel{\alpha}{\Longrightarrow} \text{ and } A \stackrel{\beta}{\Longrightarrow} \text{ implies } A \stackrel{\alpha\cap\beta}{\Longrightarrow}$$

The lemma follows from the following observations:

1. The units $\mathrm{SY}(a,b,c)$ and $\mathrm{AL}(a,b)$ 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 $A \approx B$ then B is intersection closed.
5. The unit $\mathrm{AR}(a,b,c)$ is not intersection closed.

The proofs of the nontrivial observations are:

2. Assume $A|B \stackrel{\alpha}{\Longrightarrow}$ and $A|B \stackrel{\beta}{\Longrightarrow}$. By the rule for the $|$ operator, there exist $\alpha_1, \alpha_2, \beta_1, \beta_2$ such that

$$\begin{aligned} \alpha &= \alpha_1 \cup \alpha_2 \text{ and } A \stackrel{\alpha_1}{\Longrightarrow} \text{ and } B \stackrel{\alpha_2}{\Longrightarrow} \\ \beta &= \beta_1 \cup \beta_2 \text{ and } A \stackrel{\beta_1}{\Longrightarrow} \text{ and } B \stackrel{\beta_2}{\Longrightarrow} \end{aligned}$$

Since A is intersection closed, $A \stackrel{\alpha_1\cap\beta_1}{\Longrightarrow}$. Also, since B is intersection closed, $B \stackrel{\alpha_2\cap\beta_2}{\Longrightarrow}$. Thus, by possibly adding inaction transitions and applying the rule for $|$, we get

$$A|B \stackrel{(\alpha_1\cap\beta_1)\cup(\alpha_2\cap\beta_2)}{\Longrightarrow}$$

Since $L(A) \cap L(B) = \emptyset$ and $\alpha_1, \beta_1 \subseteq L(A)$ and $\alpha_2, \beta_2 \subseteq L(B)$ we get

$$\alpha_1 \cap \beta_2 = \alpha_2 \cap \beta_1 = \emptyset$$

Calculating:

$$\begin{aligned} (\alpha_1 \cap \beta_1) \cup (\alpha_2 \cap \beta_2) &= (\alpha_1 \cap \beta_1) \cup (\alpha_2 \cap \beta_2) \cup (\alpha_1 \cap \beta_2) \cup (\alpha_2 \cap \beta_1) \\ &= (\alpha_1 \cap (\beta_1 \cup \beta_2)) \cup (\alpha_2 \cap (\beta_1 \cup \beta_2)) \\ &= (\alpha_1 \cup \alpha_2) \cap (\beta_1 \cup \beta_2) \\ &= \alpha \cap \beta \end{aligned}$$

This proves $A|B \stackrel{\alpha\cap\beta}{\Longrightarrow}$ as required.

3. Assume $A\langle a^\frown b\rangle \stackrel{\alpha}{\Longrightarrow}$ and $A\langle a^\frown b\rangle \stackrel{\beta}{\Longrightarrow}$. Thus, either $A \stackrel{\alpha}{\Longrightarrow}$ or $A \stackrel{\alpha\cup\{a,b\}}{\Longrightarrow}$, and also either $A \stackrel{\beta}{\Longrightarrow}$ or $A \stackrel{\beta\cup\{a,b\}}{\Longrightarrow}$. Since A is intersection closed one of the following four cases must hold:

   (a) $A \stackrel{\alpha\cap\beta}{\Longrightarrow}$

   (b) $A \stackrel{(\alpha\cup\{a,b\})\cap\beta}{\Longrightarrow}$

   (c) $A \stackrel{\alpha\cap(\beta\cup\{a,b\})}{\Longrightarrow}$

   (d) $A \stackrel{(\alpha\cup\{a,b\})\cap(\beta\cup\{a,b\})}{\Longrightarrow}$

   Since $a, b \notin \alpha, \beta$, this means that either $A \stackrel{\alpha\cap\beta}{\Longrightarrow}$ or $A \stackrel{(\alpha\cap\beta)\cup\{a,b\}}{\Longrightarrow}$. In either case, $A\langle a^\frown b\rangle \stackrel{\alpha\cap\beta}{\Longrightarrow}$ as required.

By 1-4 above, if a term is equivalent with a term in the closure of $\{\mathrm{SY}, \mathrm{AL}\}$, then it is intersection closed. But by 5, $\mathrm{AR}(a,b,c)$ is not intersection closed. This concludes the proof of lemma 15. □

Lemma 16 *No term in the closure of* $\{\mathrm{AR}, \mathrm{AL}\}$ *is equivalent with* $\mathrm{SY}(a,b,c)$.

Proof: Let A be a term and $\alpha \subseteq L(A)$. Let Γ be a partition of α into disjoint sets. Say that (A,Γ) has the ***arbitrary union*** property, written $AU(A,\Gamma)$, iff

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

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

The lemma follows from the following observations:

1. The units $\mathrm{AR}(a,b,c)$ and $\mathrm{AL}(a,b)$ 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 $A \approx B$ then B is partition closed.

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

The proofs of the nontrivial observations are:

2. Assume $A|B\stackrel{\alpha}{\Longrightarrow}$ and $|\alpha|\geq 3$. This means that there are α_1 and α_2 such that $\alpha=\alpha_1\cup\alpha_2$ and $A\stackrel{\alpha_1}{\Longrightarrow}$ and $B\stackrel{\alpha_2}{\Longrightarrow}$. From $L(A)\cap L(B)=\emptyset$ it follows that α_1 and α_2 are disjoint. There are four cases:

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

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

   of α satisfies $AU(A|B,\Gamma)$.

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

   $$\Gamma'=\Gamma\cup\{\alpha_2\}$$

   of α satisfies $AU(A|B,\Gamma')$.

   (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 exists a partition Γ of α_1 and a partition Γ' of α_2, satisfying $AU(A,\Gamma)$ and $AU(B,\Gamma')$ respectively. Then the partition

   $$\Gamma''=\Gamma\cup\Gamma'$$

   of α satisfies $AU(A|B,\Gamma'')$.

3. Assume $A\langle a^\frown b\rangle\stackrel{\alpha}{\Longrightarrow}$ and $|\alpha|\geq 3$. Then $a,b\notin\alpha$. There are two cases:

   (a) $A\stackrel{\alpha}{\Longrightarrow}$. Since A is partition closed, there is a partition Γ of α satisfying $AU(A,\Gamma)$. Since neither a nor b is in any element in Γ, and $a,b\notin\gamma$ and $A\stackrel{\gamma}{\Longrightarrow}$ implies $A\langle a^\frown b\rangle\stackrel{\gamma}{\Longrightarrow}$, it follows that Γ also satisfies $AU(A\langle a^\frown b\rangle,\Gamma)$.

   (b) $A\stackrel{\alpha\cup\{a,b\}}{\Longrightarrow}$. Since A is partition closed there is a partition $\Gamma=\{\alpha_1,\ldots,\alpha_n\}$ of $\alpha\cup\{a,b\}$ satisfying $AU(A,\Gamma)$. 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 $AU(A\langle a^\frown b\rangle,\Gamma')$. Let γ be an arbitrary union of elements of Γ'. Then γ is also a union of elements of Γ, whence $A\stackrel{\gamma}{\Longrightarrow}$. Since $a,b\notin\gamma$ this implies $A\langle a^\frown b\rangle\stackrel{\gamma}{\Longrightarrow}$.

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 $AU(A\langle a^\frown b\rangle, \Gamma')$. 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 $A \stackrel{\gamma}{\Longrightarrow}$. Since $a, b \notin \gamma$ this implies $A\langle a^\frown b\rangle \stackrel{\gamma}{\Longrightarrow}$.

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

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

Now δ is a union of elements of Γ. Thus $A \stackrel{\delta}{\Longrightarrow}$ whence, since $a, b \in \delta$, it holds that $A\langle a^\frown b\rangle \stackrel{\delta - \{a,b\}}{\Longrightarrow}$. But

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

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

By 1-4 above, if a term is equivalent with a term in the closure of $\{\mathrm{AR}, \mathrm{AL}\}$, then it is partition closed. But by 5, $\mathrm{SY}(a, b, c)$ is not partition closed. This concludes the proof of lemma 16. □

Proof of theorem 2: To prove $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$ a basis for the finite state terms, we construct for any given finite state term A an equivalent term in the closure of $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$ as follows. Let T be the transition diagram corresponding to A. Clearly, $A \approx_b T$. By lemma 12 there exists a self-loop free transition diagram 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 $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$. By lemma 13, $T' \approx_b CT$, 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 flowgraph N_a consisting of one arbiter:

Let CT^+ be CT in parallel with all terms corresponding to N_a for $a \in L^+$. Then $CT^+ \approx_b CT$, since all such terms corresponding to N_a can only do inaction transitions. Hence $CT^+ \approx_b A$. Also,

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

This proves $CT^+ \approx A$.

To prove $\{\mathrm{SY}, \mathrm{AR}, \mathrm{AL}\}$ a proper basis, it suffices to observe that $\mathrm{SY}(a,b,c)$, $\mathrm{AR}(a,b,c)$, and $\mathrm{AL}(a,b)$ are all finite state for any a, b, c and that the operators preserve the property of being finite state.

Finally, lemmas 14 - 16 prove that the basis is independent. □

Proof of theorem 3

In this proof we write *expressible* to mean expressible w.r.t. $\approx$. Let op be an n-ary contextual operator. Let $\tilde{L}$ be a sequence of sorts such that op is defined for $\tilde{L}$ and returns sort L given sort $\tilde{L}$. Let $op_1, op_2, \ldots$ be the operator symbols used in the definition of the semantics of op (thus $op_1 = op$). A rule in the semantics for op must be of type

$$\frac{A_1 \xrightarrow{\alpha_1} A_1' \quad \cdots \quad A_n \xrightarrow{\alpha_n} A_n'}{op_i(\tilde{A}) \xrightarrow{\alpha} op_j(\tilde{A'})}$$

where i and j are natural numbers, $\alpha \subseteq L$, and $\alpha_k \subseteq L_k$ for $1 \leq k \leq n$. Note that the rule is uniquely identified by the tuple

$$\langle i, j, \alpha, \alpha_1, \ldots, \alpha_n \rangle$$

In the following we will use such tuples to represent rules.

Assume fresh and distinct port names $[a]_k$ and $[a]_k'$ for each $k \leq n$ and $a \in L_k$. Write $[L_k]$ for $\{[a]_k : a \in L_k\}$ and $[L_k]'$ for $\{[a]_k' : a \in L_k\}$. Assume further that there are terms $OP_1, OP_2, \ldots$ each with sort

$$L \cup \bigcup_{1 \leq k \leq n} [L_k]$$

with precisely the following transitions: for each rule $\langle i, j, \alpha, \alpha_1, \ldots, \alpha_n \rangle$ in the definition of op, there is a transition

$$OP_i \xrightarrow{\beta} OP_j$$

where

$$\beta = \alpha \cup \bigcup_{1 \leq k \leq n} \bigcup_{a \in \alpha_k} \{[a]_k\}$$

Note that OP_i must have an inaction transition which corresponds to the idling rule for op_i. The term OP_i can be regarded as "enforcing" the rules of op_i: if $op_i(\tilde{A})$ can do an event α on condition that all A_k can do α_k, then OP_i can do an event containing α provided it does it in synchrony with an event representing the ports in all α_k.

For each $k \leq n$, let θ_k be a port name substitution which substitutes $[a]_k'$ for a. The proof of theorem 3 follows directly from the following lemma:

Lemma 17 *Let $\langle LINKS\rangle$ be a sequence of linking operators containing for all $k \leq n$ and $a \in L_k$ the link $\langle [a]'_k \frown [a]_k\rangle$. Then for all $\tilde{A}$ of sort $\tilde{L}$,*

$$(OP_1|A_1\theta_1|\cdots|A_n\theta_n)\langle LINKS\rangle \;\approx\; op(\tilde{A})$$

Proof: Write $C_i(\widetilde{A\theta})$ for $(OP_i|A_1\theta_1|\cdots|A_n\theta_n)\langle LINKS\rangle$. We prove $C_1(\widetilde{A\theta}) \approx op(\tilde{A})$. It is clear that $L(C_1(\widetilde{A\theta})) = L(op(\tilde{A})) = L$. Define the relation $\mathcal{R}$ by

$$\mathcal{R} = \{\langle C_i(\widetilde{A\theta}), op_i(\tilde{A})\rangle : i \text{ is a natural number and } \tilde{A} \text{ has sort } \tilde{L}\}$$

We first prove that $\mathcal{R}$ is a simulation. So assume $C_i(\widetilde{A\theta}) \xrightarrow{\alpha} B$. We have to prove that for some j and $\widetilde{A'}$ it holds $B = C_j(\widetilde{A'\theta})$ and $op_i(\tilde{A}) \stackrel{\alpha}{\Longrightarrow} op_j(\widetilde{A'})$. By definition of $C_i(\widetilde{A\theta})$ and the rules for parallel and linking, we get that there are $C, \widetilde{A'}, \beta, \alpha_1, \ldots, \alpha_n$ such that

$$\begin{array}{l} OP_i \xrightarrow{\beta} C \\ A_1 \xrightarrow{\alpha_1} A'_1 \\ \quad\vdots \\ A_n \xrightarrow{\alpha_n} A'_n \\ B = (C|A'_1\theta_1|\cdots|A'_n\theta_n)\langle LINKS\rangle \\ a \in \alpha_k \text{ iff } [a]_k \in \beta \end{array}$$

From the properties of OP_i we get that $C = OP_j$ for some j, thus $B = C_j(\widetilde{A'\theta})$, and that the transition $OP_i \xrightarrow{\beta} OP_j$ corresponds to a rule in the definition of op. Since α must be the non linked ports in β, and since $a \in \alpha_k$ iff $[a]_k \in \beta$, the rule must be precisely as represented by $\langle i, j, \alpha, \alpha_1, \ldots, \alpha_n\rangle$. But since $A_k \xrightarrow{\alpha_k} A'_k$, the premises of that rule are fulfilled, and we can conclude $op_i(\tilde{A}) \xrightarrow{\alpha} op_j(\widetilde{A'})$.

We next prove that $\mathcal{R}^{-1}$ is a simulation. The proof is essentially the proof above run backwards. So assume that $op_i(\tilde{A}) \xrightarrow{\alpha} B$. We must prove that for some j and $\widetilde{A'}$, it holds that $B = op_j(\widetilde{A'})$ and $C_i(\widetilde{A\theta}) \stackrel{\alpha}{\Longrightarrow} C_j(\widetilde{A'\theta})$. Since the transition $op_i(\tilde{A}) \xrightarrow{\alpha} op_j(\widetilde{A'})$ must be inferred from some rule for op_i, we get that there is a rule $\langle i, j, \alpha, \alpha_1, \ldots, \alpha_n\rangle$, and that $A_k \xrightarrow{\alpha_k} A'_k$ for all $k \leq n$, and that $B = op_j(\widetilde{A'})$. From the existence of this rule we know that $OP_i \xrightarrow{\beta} OP_j$ for β as defined above. The rules for parallel and linking finally give that $C_i(\widetilde{A\theta}) \xrightarrow{\alpha} C_j(\widetilde{A'\theta})$. □

Proof of theorem 3: Let op be an arbitrary contextual operator, and $\tilde{L}$ a sequence of sorts for which it is defined. For (a), since any behaviour can be expressed (up to $\approx$), then in particular there is a term equivalent with OP_1, and C in lemma 17 provides the expressing context for op. For (b), if op is finite state, i.e. only makes use of symbols $op_1, \ldots, op_N$, then the corresponding term OP_1 has a finite state space $\{OP_1, \ldots, OP_N\}$, whence OP_1 is expressible, and C in lemma 17 provides the expressing context for op. □

Proof of the results in table 3

All operators in table 3 except "$a*$" (MEIJE pilot) and "$\times$" (SCCS synchronous parallel) come from algebras where at most one external port can be involved in a transition. Thus, in the rest of this section except when dealing with "$a*$" or "$\times$" we will use events of cardinality at most one. To avoid confusion we write the CSP hiding operator as "$\backslash\backslash a$". All operators are total, i.e. there are no sort constraints on any arguments. 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 Σ which is its own inverse. For the purpose of CSP sequential composition, assume a distinguished port name $\surd$ in Σ.

The operators $\times$ and $a*$ (SCCS synchronous parallel and MEIJE pilot) come from synchronous process algebras. In order to make a reasonably faithful representation of these operators in our framework, we assume that the inaction transitions of our units are in some way distinguished, and that the premises of the inference rules for these operators require transitions which are not derived only from inactions.[8] For the purpose of SCCS synchronous parallel, assume a function s which when given two events α and β returns an event representing the simultaneous occurrence of α and β.

Rules defining the operational semantics of the operators are presented in table 6. Rules marked † have a symmetric form.

In what follows we assume that at least all finite state behaviours are expressible. We say that an operator is expressible (without qualifying equivalence) to mean that it is expressible w.r.t. $\approx$.

Lemma 18 *The operators $\backslash a$ and $\backslash\backslash a$ are expressible.*

Proof: Immediate from theorem 3 since these operators are contextual. □

Lemma 19 *The operators $|$ and $||$ are expressible.*

Proof: The proof goes by giving contextual definitions to equivalent operators. The situations for $|$ and $||$ are similar; we present only the proof for $|$ here.

[8] It should be noted that in the alternative formulation of our algebra without inaction transitions (cf. section 9) the representations of synchronous operators pose no problems.

$[|_1]:\ \dfrac{A\xrightarrow{\alpha}A'}{A\,|\,B\xrightarrow{\alpha}A'\,|\,B}\ \dagger$ $\qquad$ $[|_2]:\ \dfrac{A\xrightarrow{\{a\}}A'\quad B\xrightarrow{\{\bar{a}\}}B'}{A\,|\,B\xrightarrow{\emptyset}A'\,|\,B'}$ $\qquad$ $[+]:\ \dfrac{A\xrightarrow{\alpha}A'}{A+B\xrightarrow{\alpha}A'}\ \dagger$

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

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

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

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

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

Table 6: Rules for operators from other process algebras. Rules marked † have a symmetric form. There is an extra condition on the rules for "*a" and "×": the premises may not be derived only from inaction transitions.

Define a new operator $|'$ by

$$[\,|'_1]:\ \frac{A\xrightarrow{\alpha}A'\ \ B\xrightarrow{\emptyset}B'}{A\,|'\,B\xrightarrow{\alpha}A'\,|'\,B'}{}^{\dagger}\qquad [\,|'_2]:\ \frac{A\xrightarrow{\{a\}}A'\ \ B\xrightarrow{\{\bar{a}\}}B'}{A\,|'\,B\xrightarrow{\emptyset}A'\,|'\,B'}$$

and define $L(A\,|'\,B)=L(A)\cup L(B)$. Clearly, $|'$ is contextual (the idling rule is subsumed by $[\,|'_1]$), and we will prove that $A\,|\,B\approx A\,|'\,B$ for all A,B. It is clear that $L(A\,|\,B)=L(A\,|'\,B)$. Define

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

We first prove that $\mathcal{R}$ is a simulation. So assume $A\,|\,B\xrightarrow{\alpha}C$. There are two cases:

1. $A\xrightarrow{\alpha}A'$ and $C=A'\,|\,B$. In this case we infer from the inaction transition of B and $[\,|'_1]$ that $A\,|'\,B\xrightarrow{\alpha}A'\,|'\,B$.

2. $\alpha=\emptyset$, $A\xrightarrow{\{a\}}A'$, $B\xrightarrow{\{\bar{a}\}}B'$, and $C=A'\,|\,B'$. In this case we infer from $[\,|'_2]$ that $A\,|'\,B\xrightarrow{\emptyset}A'\,|'\,B'$.

We next prove that $\mathcal{R}^{-1}$ is a simulation. So assume $A\,|'\,B\xrightarrow{\alpha}C$. Again there are two cases:

1. $A\xrightarrow{\alpha}A'$ and $B\xrightarrow{\emptyset}B'$ and $C=A'\,|'\,B'$. Then, by two applications of $[\,|_1]$ we infer $A\,|\,B\xrightarrow{\alpha}A'\,|\,B\xrightarrow{\emptyset}A'\,|\,B'$, i.e. $A\,|\,B\xRightarrow{\alpha}A'\,|\,B'$.

2. $\alpha=\emptyset$, $A\xrightarrow{\{a\}}A'$, $B\xrightarrow{\{\bar{a}\}}B'$, and $C=A'\,|'\,B'$. In this case we infer from $[\,|_2]$ that $A\,|\,B\xrightarrow{\emptyset}A'\,|\,B'$.

This proves that $A\,|\,B\approx A\,|'\,B$. By theorem 3, $|'$ and hence $|$ is expressible. □

Lemma 20 *The operator* $\Box$ *is expressible.*

Proof: The proof goes by defining an equivalent contextual operator. Define two auxiliary operators Π_1 and Π_2 with the rules

$$\frac{A\xrightarrow{\alpha}A'\ \ B\xrightarrow{\emptyset}B'}{\Pi_1(A,B)\xrightarrow{\alpha}\Pi_1(A',B')}\qquad \frac{A\xrightarrow{\emptyset}A'\ \ B\xrightarrow{\alpha}B'}{\Pi_2(A,B)\xrightarrow{\alpha}\Pi_2(A',B')}$$

and define $L(\Pi_1(A,B))=L(A)$ and $L(\Pi_2(A,B))=L(B)$. Next define the operator $\Box'$ by

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

and define $L(A\Box'B) = L(A) \cup L(B)$. Clearly, $\Box'$ is contextual. It is straightforward to show that $\Pi_1(A,B) \approx A$ and $\Pi_2(A,B) \approx B$. We will show that $A\Box B \approx A\Box'B$. Clearly, $L(A\Box B) = L(A\Box'B)$. Define the relation $\mathcal{R}$ by

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

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 $C = A\Box B$ and $D = A\Box'B$. There are four cases for $C \xrightarrow{\alpha} C'$:

1. $A \xrightarrow{\emptyset} A'$, $\alpha = \emptyset$, $C' = A'\Box B$. Then, by inaction of B and $[\Box_1']$ we get that $A\Box'B \xrightarrow{\emptyset} A'\Box'B$.

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

3. $A \xrightarrow{\{a\}} A'$, $\alpha = \{a\}$, $C' = A'$. Then by $[\Box_2']$ we get that $A\Box'B \xrightarrow{\{a\}} \Pi_1(A', B')$. 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 $C = A\Box B$ and $D = A\Box'B$. There are three cases for $D \xrightarrow{\alpha} D'$:

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. $A\Box B \stackrel{\emptyset}{\Longrightarrow} A'\Box B'$

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 3, $\Box'$ and hence $\Box$ is expressible. $\Box$

Lemma 21 *All operators in table 3 except $a*$ and $\times$ are expressible w.r.t. $=_T$.*

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 argument of the operator is not mentioned in the premise, add "$A \xrightarrow{\emptyset} A'$" to the premise for each argument A not otherwise mentioned (in the premise), and use A' for A in the conclusion.

2. For each rule where the conclusion is of type $op(\tilde{A}) \xrightarrow{\alpha} A$, replace the conclusion with $op(\tilde{A}) \xrightarrow{\alpha} \Pi_k(\tilde{A})$ for the suitable k (Π_k is as defined in lemma 20).

3. Add the idling rule, if this is not subsumed by the other rules.

Clearly, op' is contextual and hence expressible. 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 $+$:

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

The full proof is omitted. □

Lemma 22 *The operator $+$ is not expressible.*

Proof: Let *op* be a contextual operator and assume $op(A, B) \approx A + B$ for all A, B. We will derive a contradiction. Consider terms A, A', B, C with the following transitions (in addition to inaction transitions):

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

By the idling rule for *op*, we infer that $op(A, B) \xrightarrow{\emptyset} op(A', B)$. Since $op(A, B) \approx A + B$, there exists a D such that $A + B \stackrel{\emptyset}{\Longrightarrow} D$ and $D \approx op(A', B) \approx A' + B$. The following possibilities exist for $A + B \stackrel{\emptyset}{\Longrightarrow} D$:

1. $D = A + B$. But $A + B \not\approx A' + B$, since $A + B \stackrel{\{b\}}{\Longrightarrow}$.

2. $D = A$ (by inaction of A). But $A \not\approx A' + B$, since $A \stackrel{\{b\}}{\Longrightarrow}$.

3. $D = A'$. But $A' \not\approx A' + B$, since $A' + B \stackrel{\{c\}}{\Longrightarrow}$.

4. $D = B$ (by inaction of B). But $B \not\approx A' + B$, since $A' + B \stackrel{\{a\}}{\Longrightarrow}$.

Hence, no such D exists. □

Remark: The operator $+$ actually preserves observation equivalence in our algebra. This might appear to contradict the well established fact in CCS that $+$ does not in general preserve equivalence. However, in our algebra we require of all units, and hence of all terms, to have inaction transitions. This means that the examples which demonstrate that $+$ does not preserve equivalence are not expressible. Were it not for this fact, there would be a much simpler proof that $+$ is not expressible: an operator which does not preserve equivalence can not be expressed in terms of operators which preserve equivalence.

Lemma 23 *The operator a. is not expressible.*

Proof: Let *op* be a contextual operator and assume $op(A) \approx a.A$ for all A. We will derive a contradiction. Consider terms A, C with the following transitions (in addition to inaction transitions):

$$A \xrightarrow{\emptyset} C \qquad A \xrightarrow{\{b\}} C$$

By the idling rule for *op*, we infer that $op(A) \xrightarrow{\emptyset} op(C)$. Since $op(A) \approx a.A$, there exists a D such that $a.A \stackrel{\emptyset}{\Longrightarrow} D$ and $D \approx op(A') \approx a.A'$. But $a.A \stackrel{\emptyset}{\Longrightarrow} D$ implies $D = a.A$, and $a.A \not\approx a.A'$ since $a.A \stackrel{\{a\}}{\Longrightarrow} \stackrel{\{b\}}{\Longrightarrow}$. □

Lemma 24 *The operator* $\sqcap$ *is not expressible.*

Proof: We will prove that $A \sqcap B \approx A + B$, hence the lemma follows from lemma 22. Let **Id** be the identity relation on terms, and define the relation $\mathcal{R}$ by

$$\mathcal{R} = \{\langle A + B, A \sqcap B\rangle : A, B \text{ terms}\} \cup \mathbf{Id}$$

We first prove that $\mathcal{R}$ is a simulation. Assume $A + B \xrightarrow{\alpha} C$. Then either $A \xrightarrow{\alpha} C$ or $B \xrightarrow{\alpha} C$. In either case, $A \sqcap B \xrightarrow{\emptyset} \xrightarrow{\alpha} C$, i.e. $A \sqcap B \stackrel{\alpha}{\Longrightarrow} C$.

We next prove that $\mathcal{R}^{-1}$ is a simulation. Assume $A \sqcap B \xrightarrow{\alpha} C$. Then $\alpha = \emptyset$ and $C = A$ or $C = B$. By inaction transitions, $A \xrightarrow{\emptyset} A$ and $B \xrightarrow{\emptyset} B$ always hold, hence $A + B \xrightarrow{\emptyset} C$.

Thus $\mathcal{R}$ is a bisimulation whence the result follows. □

Remark: For $A + B \approx A \sqcap B$ to hold, it is essential that all terms can do inaction transitions. In CCS or CSP, this would not be true.

Lemma 25 *The operator* ; *is not expressible.*

Proof: Let *op* be a contextual operator and assume $op(A, B) \approx A; B$ for all A, B. We will derive a contradiction. Consider terms A, A', B, B', C with the following transitions (in addition to inaction transitions):

$$A \xrightarrow{\{a\}} A' \qquad A' \xrightarrow{\{\surd\}} C \qquad B \xrightarrow{\{b\}} C \qquad B \xrightarrow{\emptyset} B' \qquad B' \xrightarrow{\{c\}} C$$

By the idling rule for *op*, we infer that $op(A, B) \xrightarrow{\emptyset} op(A, B')$. Since $op(A, B) \approx A; B$, there exists a D such that $A; B \stackrel{\emptyset}{\Longrightarrow} D$ and $D \approx op(A, B') \approx A; B'$. The only possibility for $A; B \stackrel{\emptyset}{\Longrightarrow} D$ is $D = A; B$. Thus $A; B \approx A; B'$ must hold. This is contradicted by the fact that $A; B \stackrel{\{a\}}{\Longrightarrow} \stackrel{\{b\}}{\Longrightarrow}$ and not $A; B' \stackrel{\{a\}}{\Longrightarrow} \stackrel{\{b\}}{\Longrightarrow}$. □

Lemma 26 *The operator $\triangleright$ is not expressible.*

Proof: Let *op* be a contextual operator and assume $op(A,B) \approx A \triangleright B$ for all A, B. We will derive a contradiction. Consider terms A, B, B', C with the following transitions (in addition to inaction transitions):

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

By the idling rule for *op*, we infer that $op(A,B) \xrightarrow{\emptyset} op(A,B')$. Since $op(A,B) \approx A \triangleright B$, there exists a D such that $A \triangleright B \stackrel{\emptyset}{\Longrightarrow} D$ and $D \approx op(A,B') \approx A \triangleright B'$. There are three possibilities for $A \triangleright B \stackrel{\emptyset}{\Longrightarrow} D$:

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

Hence, no such D exists. □

Lemma 27 *The operator $a*$ is not expressible w.r.t. $=_T$.*

Proof: Consider terms A, B, B', C with the same sort and the following transitions (in addition to inaction transitions):

$$A \xrightarrow{\{b\}} C \qquad B \xrightarrow{\emptyset} B' \qquad B \xrightarrow{\{b\}} C$$

Clearly, $A =_T B$. But it is not the case that $a*A =_T a*B$, since $a*A \stackrel{\{a,b\}}{\Longrightarrow}$ and not $a*B \stackrel{\{a,b\}}{\Longrightarrow}$. Thus, $a*$ does not preserve $=_T$. But parallel and linking preserve $=_T$ (the proof of this fact is easy and is omitted). It follows that $a*$ is not expressible w.r.t. $=_T$. □

Lemma 28 *The operator $\times$ is not expressible w.r.t. $=_T$.*

Proof: Consider terms A, B with the following transitions (in addition to inaction transitions), and let C be a term with no transitions except inaction transitions:

$$A \xrightarrow{\{a\}} C \qquad B \xrightarrow{\emptyset} C$$

Clearly, $B =_T C$. But it is not the case that $A \times B =_T A \times C$, since $A \times B \stackrel{s(\{a\},\emptyset)}{\Longrightarrow}$ and not $A \times C \stackrel{s(\{a\},\emptyset)}{\Longrightarrow}$. Thus, $\times$ does not preserve $=_T$, which means that it is not expressible. □

This concludes the results summarised in table 3.

Proof of theorem 6

In the course of this proof we will need to explicitly represent particular interpretations. Since modules represent schematic units where the ports have not yet been named, it is natural to consider the semantics of modules as schematic transition diagrams which, when instantiated with a sequence of port names, will yield labelled transition diagrams. We call such a schematic transition diagram a *template*, and represent it as a transition diagram where the labels of the transitions are sets of integers. Formally, an n-ary template T is a tuple (here $\mathcal{P}$ denotes powerset, i.e. $\mathcal{P}K$ is the set of subsets of K):

$$\langle D, E, d \rangle$$

where

D is a nonempty set, the set of *states*.
E is a subset of $D \times \mathcal{P}\{1, \ldots n\} \times D$, the set of *transitions*.
$d \in D$ is the *initial* state.
$\langle d, \emptyset, d \rangle \in E$, the *inaction* transition.

If T is an n-ary template as above and $\tilde{a} = a_1, \ldots, a_n$ is a sequence of port names, then $T(\tilde{a})$ is the transition diagram

$$\langle D, E(\tilde{a}), d \rangle$$

where

$$\langle d, \{a_k : k \in K\}, d' \rangle \in E(\tilde{a}) \text{ iff } \langle d, K, d' \rangle \in E$$

An *interpretation* I is a mapping from modules to templates: each n-ary module is assigned an n-ary template. The operational semantics of a unit $M(\tilde{a})$ is defined to be $I(M)(\tilde{a})$. This means that

$$M(\tilde{a}) \xrightarrow{\alpha} M'(\tilde{a})$$

iff

$$\begin{aligned} I(M)(\tilde{a}) &= \langle D, E, d \rangle \\ I(M')(\tilde{a}) &= \langle D, E, d' \rangle \\ \langle d, \alpha, d' \rangle &\in E \end{aligned}$$

It is clear that in this way an interpretation conforms to the requirements in section 3.

We now proceed with the proof, first by establishing a normal form for terms. The reader is advised to keep the proof outline as presented in section 8 in mind. A term is on *normal form* iff it is on the form

$$(C_1 | \cdots | C_n) \langle a_1 \frown b_1 \rangle \cdots \langle a_k \frown b_k \rangle$$

for some $n \geq 1$ and $k \geq 0$, where $C_1, \ldots, C_n$ are units.

Lemma 29 *Every term is provably equivalent with a term on normal form.*

Proof: Let A be a term. We prove that there is a provably equivalent normal form by induction on the structure of A.

1. $A = M(\tilde{a}), M \in \mathbf{M}$. Then A is on normal form.

2. $A = A'|A''$. By induction, A' and A'' have provably equivalent normal forms:

$$\begin{aligned} \vdash A' &= (C'_1|\cdots|C'_{n'})\langle a'_1{}^\frown b'_1\rangle\cdots\langle a'_{k'}{}^\frown b'_{k'}\rangle \\ \vdash A'' &= (C''_1|\cdots|C''_{n''})\langle a''_1{}^\frown b''_1\rangle\cdots\langle a''_{k''}{}^\frown b''_{k''}\rangle \end{aligned}$$

   Now, by (S) and (L1) all a'_i, b'_i can be substituted by port names which are distinct from any port name in A''. Similarly, all a''_i, b''_i can be substituted by fresh port names which are distinct from any port name in A'. Thus we get that $\vdash A' = B'$ and $\vdash A'' = B''$ where

$$\begin{aligned} B' &= (D'_1|\cdots|D'_{n'})\langle c'_1{}^\frown d'_1\rangle\cdots\langle c'_{k'}{}^\frown d'_{k'}\rangle \\ B'' &= (D''_1|\cdots|D''_{n''})\langle c''_1{}^\frown d''_1\rangle\cdots\langle c''_{k''}{}^\frown d''_{k''}\rangle \end{aligned}$$

   where c'_i, d'_i do not occur in B'' and c''_i, d''_i do not occur in B'. By k' applications of (D) we get

$$\vdash B'|B'' = ((D'_1|\cdots|D'_{n'})|B'')\langle c'_1{}^\frown d'_1\rangle\cdots\langle c'_{k'}{}^\frown d'_{k'}\rangle$$

   By (P1) and k'' applications of (D) we get

$$\vdash B'|B'' = ((D''_1|\cdots|D''_{n''})|(D'_1|\cdots|D'_{n'}))\langle c''_1{}^\frown d''_1\rangle\cdots\langle c''_{k''}{}^\frown d''_{k''}\rangle\langle c'_1{}^\frown d'_1\rangle\cdots\langle c'_{k'}{}^\frown d'_{k'}\rangle$$

   By (P2), the operands of $|$ in this term can be organised to yield a normal form.

3. $A = A'\langle a^\frown b\rangle$. By induction A' has a provably equivalent normal form B'. Then $\vdash A = B'\langle a^\frown b\rangle$, and $B'\langle a^\frown b\rangle$ is on normal form. □

Let A be a normal form $(C_1|\cdots|C_n)\langle a_1{}^\frown b_1\rangle\cdots\langle a_k{}^\frown b_k\rangle$. Let $\tilde{C} \subseteq \{C_1, \ldots, C_n\}$ and write $L(\tilde{C})$ for $\bigcup_{C\in\tilde{C}} L(C)$. Say that $\tilde{C}$ is an *isolated part* iff $\tilde{C}$ is nonempty and

$$L(\tilde{C}) \subseteq \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$$

and

$$\forall i(1 \leq i \leq k): \quad a_i \in L(\tilde{C}) \text{ iff } b_i \in L(\tilde{C})$$

Thus, in an isolated part all ports are linked to ports within the part: there are no external (non linked) ports, and no ports linked to ports outside the part.

Lemma 30 *Let A be a normal form such that $L(A) \neq \emptyset$. Then A is provably equivalent with a normal form without isolated parts.*

Proof: Let A be a normal form with nonempty sort, and with an isolated part. We prove that there exists a provably equivalent and strictly smaller normal form. By repeated application of this result, either there is an infinite chain of normal forms strictly decreasing in size — this is clearly impossible — or A has an equivalent normal form without isolated parts.

So, let

$$A = (C_1 | \cdots | C_n)\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$$

and let $\widetilde{C} \subseteq \{C_1, \ldots, C_n\}$ be an isolated part. Write $\overline{C}$ for the parallel composition of the elements in $\widetilde{C}$, and $\overline{D}$ for the parallel composition of the C_i which are not in $\widetilde{C}$. By (P1) and (P2) we get

$$\vdash A = (\overline{C}|\overline{D})\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$$

By repeated application of (L2) the linking operators can be permuted to get

$$\vdash A = (\overline{C}|\overline{D})\langle a'_1 \frown b'_1\rangle \cdots \langle a'_k \frown b'_k\rangle$$

where for some m $(0 \leq m \leq k)$ it holds that

$$a'_1, \ldots, a'_m \in L(\overline{C}) \text{ and } a'_{m+1}, \ldots, a'_k \in L(\overline{D})$$

Since $\widetilde{C}$ is isolated, this means that $b'_1, \ldots, b'_m \in L(\overline{C})$ and $b'_{m+1}, \ldots, b'_k \in L(\overline{D})$. Also, we get that $L(\overline{C}) = \{a'_1, \ldots, a'_m, b'_1, \ldots, b'_m\}$. By m applications of (D) we get

$$\vdash A = \left(\overline{C}\langle a'_1 \frown b'_1\rangle \cdots \langle a'_m \frown b'_m\rangle | \overline{D}\right) \langle a'_{m+1} \frown b'_{m+1}\rangle \cdots \langle a'_k \frown b'_k\rangle$$

Note that $L(\overline{C}\langle a'_1 \frown b'_1\rangle \cdots \langle a'_m \frown b'_m\rangle) = \emptyset$. If $L(\overline{D}) = \emptyset$ then $L(A) = \emptyset$, contradicting the premise. Hence $\overline{D}$ must consist of at least one unit. Applying $(\emptyset)$ yields

$$\vdash \overline{C}\langle a'_1 \frown b'_1\rangle \cdots \langle a'_m \frown b'_m\rangle | \overline{D} = \overline{D}$$

whence

$$\vdash A = \overline{D}\langle a'_{m+1} \frown b'_{m+1}\rangle \cdots \langle a'_k \frown b'_k\rangle$$

Since $\widetilde{C} \neq \emptyset$, this equation demonstrates that A is provably equivalent with a strictly smaller normal form. □

Lemma 32 below is the heart of the completeness proof. We will need some auxiliary definitions. Say that a port a is *linked* to a port b in A iff either $\langle a \frown b\rangle$ or $\langle b \frown a\rangle$ is a link in A. A *chain* is a sequence

$$\overset{\varepsilon \frown p_1}{\longleftrightarrow} M_1 \overset{q_1 \frown p_2}{\longleftrightarrow} M_2 \overset{q_2 \frown p_3}{\longleftrightarrow} \ldots \overset{q_{m-1} \frown p_m}{\longleftrightarrow} M_m$$

for $m \geq 1$ (if $m = 1$ then the chain is simply $\overset{\varepsilon \frown p_1}{\longleftrightarrow} M_1$) where

1. $M_1, \ldots, M_m$ are modules; the arity of M_i is written m_i.
2. $e \in \Sigma$.
3. $1 \leq p_i \leq m_i$ for all i $(1 \leq i \leq m)$.
4. $1 \leq q_i \leq m_i$ for all i $(1 \leq i \leq m-1)$.

A normal form

$$A \;=\; (C_1|\cdots|C_n)\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$$

possesses the chain

$$CH \;=\; \overset{e \frown p_1}{\longleftrightarrow} M_1 \overset{q_1 \frown p_2}{\longleftrightarrow} M_2 \overset{q_2 \frown p_3}{\longleftrightarrow} \cdots \overset{q_{m-1} \frown p_m}{\longleftrightarrow} M_m$$

iff there exists a function $\ell : \{1, \ldots, m\} \to \{1, \ldots, n\}$ satisfying:

1. For all i $(1 \leq i \leq m)$ there is a sequence of port names $\widetilde{a^{(i)}} = a_1^{(i)}, \ldots, a_{m_i}^{(i)}$ such that
$$M_i(\widetilde{a^{(i)}}) = C_{\ell(i)}$$
i.e. M_i is the module name in $C_{\ell(i)}$.
2. $e = a_{p_1}^{(1)}$, and e is not linked to any port in A, i.e. $e \neq a_i, b_i$.
3. For all i $(1 \leq i \leq m-1)$: $a_{q_i}^{(i)}$ is linked to $a_{p_{i+1}}^{(i+1)}$ in A
4. One of the following holds:
   (a) For all $i \neq j$, $1 \leq i, j \leq m$, it is the case that $\ell(i) \neq \ell(j)$. If so, the chain is called *linear*.
   (b) For all $i \neq j$, $1 \leq i, j \leq m-1$, it is the case that $\ell(i) \neq \ell(j)$, and $\ell(m-1) = \ell(m)$. In this case, the chain is called *loop ended*.

We will sometimes say that the chain CH is in A, or that A has CH, to mean that A possesses CH. The function ℓ is called the *evidence* of CH in A.

Intuitively, a chain is a sequence of modules and links. Each module M_i except the *bottom* M_m has port number q_i going "downwards", i.e. it is connected with a port of M_{i+1}. Conversely, each module M_i has port number p_i going "upwards", i.e. it is connected with a port of M_{i-1} or, if $i = 1$, it corresponds to the external port e in A. A chain is linear if all units involved in the chain are distinct (note that this does not imply that the modules are distinct), and it is loop ended if all units except "bottom" and "next to bottom" are distinct.

Lemma 31 *Let A be a normal form and let*

$$CH \;=\; \overset{e\frown p_1}{\longleftrightarrow} M_1 \overset{q_1\frown p_2}{\longleftrightarrow} M_2 \overset{q_2\frown p_3}{\longleftrightarrow} \cdots \overset{q_{m-1}\frown p_m}{\longleftrightarrow} M_m$$

be a chain in A with evidence ℓ. Let CH' with evidence ℓ' be the prefix of length m' $(1 \leq m' \leq m)$ of CH, i.e.

$$CH' \;=\; \overset{e\frown p_1}{\longleftrightarrow} M_1 \overset{q_1\frown p_2}{\longleftrightarrow} M_2 \overset{q_2\frown p_3}{\longleftrightarrow} \cdots \overset{q_{m'-1}\frown p_{m'}}{\longleftrightarrow} M_{m'}$$

Then for all i $(1 \leq i \leq m')$

$$\ell(i) = \ell'(i)$$

Proof: By induction on i. If $i = 1$ the result is immediate since A can have at most one unit with an external port e not covered by a link. For the inductive step assume $\ell(i-1) = \ell'(i-1)$. Let $C_{\ell(i-1)} = C_{\ell'(i-1)} = M_{i-1}(\tilde{a})$. Let $a = a_{q_{i-1}}$. By definition of chain, either $\langle a\frown b\rangle$ or $\langle b\frown a\rangle$ is a link in A for some b. Let C_j have this b in its sort. Then $\ell(i) = \ell'(i) = j$. □

By lemma 31 the evidence of any chain is unique in a given normal form. We will sometimes say that the unit C_j is *used* in CH iff $j = \ell(i)$ for some i where ℓ is the evidence of CH. Similarly, we will say that a link $\langle a\frown b\rangle$ is used in a chain if this link is one of the links under 3 in the definition of chain.

Lemma 32 *Let A and A' be two normal forms such that*

$$\models A \simeq A'$$

for some behaviour equivalence $\simeq$. Then each linear chain in A is also in A'.

Proof: Let

$$A = (C_1|\cdots|C_n)\langle a_1\frown b_1\rangle\cdots\langle a_k\frown b_k\rangle$$

be a normal form and

$$CH \;=\; \overset{e\frown p_1}{\longleftrightarrow} M_1 \overset{q_1\frown p_2}{\longleftrightarrow} M_2 \overset{q_2\frown p_3}{\longleftrightarrow} \cdots \overset{q_{m-1}\frown p_m}{\longleftrightarrow} M_m$$

be a linear chain with evidence ℓ in A. We will establish an interpretation I in which

1. $A \overset{\{e\}}{\Longrightarrow} \bullet$.
2. $A' \overset{\{e\}}{\Longrightarrow} \bullet$ implies that A' has the chain CH.

From $\models A \simeq A'$, and in particular from $I \models A \simeq A'$, it then follows $A' \stackrel{\{c\}}{\Longrightarrow} \bullet$, whence A' must also have the chain CH.

The main part of the proof is devoted to constructing the interpretation I, and proving that it has the property mentioned above. The construction is complicated by the following fact: even though for distinct i and j the units $C_{\ell(i)}$ and $C_{\ell(j)}$ are distinct, it might well be the case that M_i and M_j are identical. As an example, let M have arity two and consider the chain

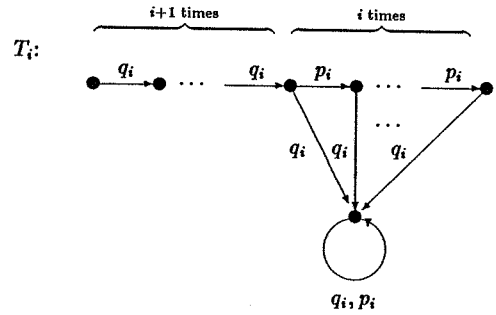
$$\stackrel{a_1 \frown 1}{\longleftrightarrow} M \stackrel{2 \frown 1}{\longleftrightarrow} M$$

An example of a normal form with that chain is

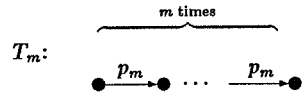
$$(M(a_1, b_1) | M(a_2, b_2)) \langle b_1 \frown a_2 \rangle$$

An ambitious reader might want to pause here and try to construct an interpretation I such that $A' \stackrel{\{a_1\}}{\Longrightarrow} \bullet$ implies that A' has that chain.

In the rest of this proof we will omit set brackets in events and in templates where this can not cause confusion. When describing the interpretation I we will graphically describe the templates by drawing transition diagrams, and we will not distinguish between a template and its transition diagram. Also, we will not distinguish between isomorphic templates. For clarity we will not show the inaction transitions; these are tacitly present in all states in all transition diagrams. Consider the following transition diagrams T_i for $1 \leq i \leq m-1$:

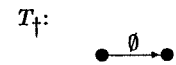


In addition, the transition diagram T_m is:



All labels of transitions are singleton sets; we omit the set brackets. The bottom state of T_i $(1 \leq i \leq m-1)$ has two transitions leading back to the same state: one is labelled q_i and one is labelled p_i. The initial states are the leftmost states. For $1 \leq i \leq m-1$ the behaviour of T_i is as follows: T_i begins by doing $i+1$ q_i events, after which it can do i p_i events. If it does an additional q_i (after the $i+1$ first), then it ends up looping in the bottom state where arbitrarily many q_i and p_i are possible. T_m is simpler: it can do the event p_m not more than m times, after which it deadlocks.

Finally, the transition diagram $T_\dagger$ is



The diagram $T_\dagger$ has one empty transition, leading from the initial (leftmost) state to a state with no further transitions.

Formally, we will use several different versions of these transition diagrams with different arities; we will let the contexts decide what is the proper arity in each occurrence below.

Let $T = \langle D, E, d \rangle$ and $T' = \langle D', E', d' \rangle$ be two templates where D and D' are disjoint. The *union* of T and T', written $T \cup T'$, is defined as follows:

$$T \cup T' \;=\; \langle D \cup D' - \{d'\},\ (E \cup E')[d/d'],\ d \rangle$$

Thus, $T \cup T'$ is the template with the union of the state sets and edges in T and T', and where the initial states of T and T' are identified and becomes the initial state of $T \cup T'$. If C is a unit whose module is mapped to such a union, then any transition sequence of C will correspond to either a transition sequence of T or a transition sequence of T'. Evidently, the operation of taking unions of templates is commutative and associative (w.r.t. isomorphism of templates).

Define the templates U_i for $1 \leq i \leq m$ as follows:

$$U_i = T_\dagger \cup \bigcup_{j:M_i=M_j} T_j$$

Thus each U_i will have several transitions from its initial state: one empty transition leading to a state with no further transitions (corresponding to $T_\dagger$), and for each $j : M_j = M_i$ a transition corresponding to the first transition of T_j. It is clear that if $M_i = M_j$, then $U_i = U_j$.

Now define the interpretation I as follows:

1. For each i $(1 \leq i \leq m)$, the module M_i is mapped to U_i.
2. For each $U_i = \langle D, E, d \rangle$ and each $d' \in D$ there is a module M, not occurring in A or A', such that M is mapped to $\langle D, E, d' \rangle$.
3. All other modules are mapped to $T_\dagger$.

Write $M_i^{(r)}$ for a module which is mapped to a template with initial state derived from the rightmost state of T_i (for $1 \leq i \leq m$ or $i = \dagger$). Formally, there might be for any given i several modules $M_i^{(r)}$, but they are all equivalent and we will not bother to distinguish between them. The only transition leading from the initial state of $M_i^{(r)}$ is q_i for $1 \leq i \leq m-1$.

We will now proceed to prove that $A \stackrel{e}{\Longrightarrow} \bullet$ in I. Each $C_{\ell(i)} = M_i(\widetilde{a^{(i)}})$ has transitions corresponding to transitions in U_i. For $1 \leq i \leq m$, write f_i for $a_{p_i}^{(i)}$. For $1 \leq i \leq m-1$, write g_i for $a_{q_i}^{(i)}$. Thus, for $1 \leq i \leq m-1$, $C_{\ell(i)}$ has port g_i and $C_{\ell(i+1)}$ has port f_{i+1}, and g_i is linked to f_{i+1} Furthermore, $f_1 = e$. Write $C_{\ell(i)}^{(r)}$ for $M_i^{(r)}(\widetilde{a^{(i)}})$ (where $1 \leq i \leq m$). Consider the sequence of transitions which takes $C_{\ell(i)}$ to $C_{\ell(i)}^{(r)}$:

$$C_{\ell(i)} \overbrace{\xrightarrow{g_i} \cdots \xrightarrow{g_i}}^{i+1 \text{ times}} \overbrace{\xrightarrow{f_i} \cdots \xrightarrow{f_i}}^{i \text{ times}} C_{\ell(i)}^{(r)} \quad \text{if } i \neq m$$

$$C_{\ell(i)} \overbrace{\xrightarrow{f_i} \cdots \xrightarrow{f_i}}^{i \text{ times}} C_{\ell(i)}^{(r)} \quad \text{if } i = m$$

Now consider the following sequence of transitions, obtained from the above sequences, inaction transitions, and the rule for parallel:

$$(C_{\ell(1)} | \cdots | C_{\ell(m)}) \overbrace{\xrightarrow{\{f_m, g_{m-1}\}} \cdots \xrightarrow{\{f_m, g_{m-1}\}}}^{m \text{ times}} \overbrace{\xrightarrow{\{f_{m-1}, g_{m-2}\}} \cdots \xrightarrow{\{f_{m-1}, g_{m-2}\}}}^{m-1 \text{ times}}$$

$$\overbrace{\cdots \xrightarrow{\{f_i, g_{i-1}\}} \cdots \xrightarrow{\{f_i, g_{i-1}\}}}^{i \text{ times}} \cdots \xrightarrow{f_1} (C_{\ell(1)}^{(r)} | \cdots | C_{\ell(m)}^{(r)})$$

Recall that the ports g_i and f_{i+1} are linked in A, and furthermore $f_1 = e$. So A has a sequence of transitions

$$A \xrightarrow{\emptyset} \cdots \xrightarrow{\emptyset} \xrightarrow{e} B$$

where B is A with all $C_{\ell(i)}$ replaced by $C^{(r)}_{\ell(i)}$. Finally, for all units C_j in A with a module M_i for some $1 \leq i \leq m$ but not used by the chain (i.e. they are not $C_{\ell(i)}$ — there may or may not be such units) consider the transition $C_j \xrightarrow{\emptyset} C'_j$ derived from the empty transitions (corresponding to the transition in $T_\dagger$) of all such units. We thus get that

$$A \stackrel{e}{\Longrightarrow} C$$

where C is obtained from A by replacing all $C_{\ell(i)}$ by $C^{(r)}_{\ell(i)}$, and replacing all other modules M_i by modules mapped to the rightmost state of a template $T_\dagger$. We call this transition sequence a *full execution* of CH in I. It is now the case that C has no transitions except those derived from inaction transitions of all units — all $C^{(r)}_{\ell(i)}$ are blocked since they can only do g_i events and no unit can do any f_i event. This proves

$$A \stackrel{e}{\Longrightarrow} \bullet$$

The final and most difficult part of the proof of lemma 32 is the proof that if $A' \stackrel{e}{\Longrightarrow} \bullet$, then A' has the chain CH. The proof idea is to establish that the only possible reason for $A' \stackrel{e}{\Longrightarrow} \bullet$ is a full execution of CH. We first prove an auxiliary lemma:

Lemma 33 Let C be any unit in A', and assume $C \stackrel{\sigma}{\Longrightarrow} D$ where σ is a sequence with exactly i $(1 \leq i \leq m)$ occurrences of a for some port name a, and that not $D \stackrel{a}{\Longrightarrow}$. Then

1. D must be $M_i(c_1, \ldots, c_{m_i})$ for some $c_1, \ldots, c_{m_i}$.
2. a must be c_{p_i}.
3. If $i \neq m$ then
   (a) σ must be $(c_{q_i})^{i+1} a^i$.
   (b) For some E it is the case that $D \xrightarrow{c_{q_i}} E$ and $E \xrightarrow{a}$.

Proof: Let $C = M(c_1, \ldots, c_{m'})$ for some $c_1, \ldots, c_{m'}$. Thus, $a = c_p$ for some p $(1 \leq p \leq m')$, and in the template to which M is mapped there must be a path containing exactly i p-events, leading to a state from which no p-event is possible. By definition of I, this path must correspond to a path in some transition diagram T_k. Among these transition diagrams there is only one with the property that for some p, there is a path containing exactly i p-events leading to a state where no p-event is possible:[9] this diagram is T_i, and $p = p_i$. The only module mapped to a template containing T_i is M_i. Hence M must be M_i. This proves 1 and 2 of the lemma.

[9]Our particular choice of the transition diagrams T_i is motivated by the need for exactly one T_i to satisfy this property.

To prove 3, assume $i \neq m$. The path through T_i must terminate in the rightmost state of T_i, hence the accumulated labels of the path must be $(q_i)^{i+1}(p_i)^i$. This proves (a). From this rightmost state there is a transition labelled q_i to the bottom state, from which there is a transition labelled p_i. This proves (b). □

We now conclude the proof of lemma 32 by proving that if $A' \stackrel{e}{\Longrightarrow} \bullet$, then A' has the chain CH. Let A' be the normal form

$$A' = (C'_1| \cdots |C'_{n'})\langle a'_1 \frown b'_1\rangle \cdots \langle a'_{k'} \frown b'_{k'}\rangle$$

Assume that $A' \stackrel{e}{\Longrightarrow} \bullet$. We will describe a procedure which establishes a function ℓ' as evidence of the chain CH in A'.

For some B', $A' \stackrel{e}{\Longrightarrow} B'$ and not $B' \stackrel{e}{\Longrightarrow}$. This must be because for each i $(1 \leq i \leq n')$, $C'_i \stackrel{\sigma_i}{\Longrightarrow} D'_i$ where all σ_i contain one e altogether, and B' is A' with D'_i for C'_i. We will examine the possibilities for the C'_i and σ_i.

Clearly, for some unit, say C'_j, in A' we have $C'_j \stackrel{\sigma_j}{\Longrightarrow} D'_j$ where σ_j contains exactly one e, and not $D'_1 \stackrel{e}{\Longrightarrow}$. According to lemma 33, $C'_j = M_1(c_1^{(1)}, \ldots, c_{m_1}^{(1)})$ and $c_{p_1}^{(1)} = e$. Define $\ell'(1) = j$. If $m = 1$ we are done — we have just established the (trivial) chain

$$\stackrel{e \frown p_1}{\longleftrightarrow} M_1$$

in A'.

So assume $m > 1$. Then, again by lemma 33:3(a) we have $\sigma_j = (c_{q_1}^{(1)})^2 e$. Write c for $c_{q_1}^{(1)}$. First assume that c does not occur in a link in A', i.e. that c is different from all a'_i, b'_i. In this case $C'_j \stackrel{\sigma_j}{\Longrightarrow}$ can not possibly be used to infer $A' \stackrel{e}{\Longrightarrow}$, since c precedes the e in σ_j. So assume that there is a port d and a link $\langle c \frown d\rangle$ or $\langle d \frown c\rangle$ in A'. Then there is a unit C'_{j_2} in A' and $C'_{j_2} \stackrel{\sigma_{j_2}}{\Longrightarrow} D'_{j_2}$ where

1. σ_{j_2} contains exactly two d events.

2. Not $D'_{j_2} \stackrel{d}{\Longrightarrow}$ — otherwise $D'_{\ell'(1)}$ and D'_{j_2} can interact in B', leading to (by lemma 33:3(b)) a term which can do another event e, i.e. $B' \stackrel{e}{\Longrightarrow}$.

So, by lemma 33 we infer that $C'_{j_2} = M_2(c_1^{(2)}, \ldots, c_{m_2}^{(2)})$ and $c_{p_2}^{(2)} = d$. Define $\ell'(2) = j_2$.

If $m = 2$ we are done — we have just established the chain

$$\stackrel{e \frown p_1}{\longleftrightarrow} M_1 \stackrel{q_1 \frown p_2}{\longleftrightarrow} M_2$$

in A', as evidenced by the function ℓ'.

So assume $m > 2$. In steps analogous to the preceding two paragraphs, the chain is established. At step i, the fact that $\sigma_{\ell'(i-1)}$ begins with $(c_{q_{i-1}}^{(i-1)})^i$ is inferred from lemma 33:3(a), and used to establish that there is a port d linked to $c_{q_{i-1}}^{(i-1)}$,

which means that for some σ_{j_i} containing exactly i events d it is the case that $C'_{j_i} \stackrel{\sigma_{j_i}}{\Longrightarrow} D'_{j_i}$ and not $D'_{j_i} \stackrel{d}{\Longrightarrow}$. By lemma 33, the module in such an C'_{j_i} is must be M_i. This establishes the i:th module in the chain. These steps continue until $i = m$, when the whole chain has been established.

Finally, we must prove that in the chain so constructed, $i \neq j$ implies that $\ell'(i) \neq \ell'(j)$. This follows from considering the different $\sigma_{\ell'(i)}$ in the proof above: by lemma 33, $i \neq j$ implies $\sigma_{\ell'(i)} \neq \sigma_{\ell'(j)}$

This concludes the proof that A' has the chain CH, and hence the proof of lemma 32. □.

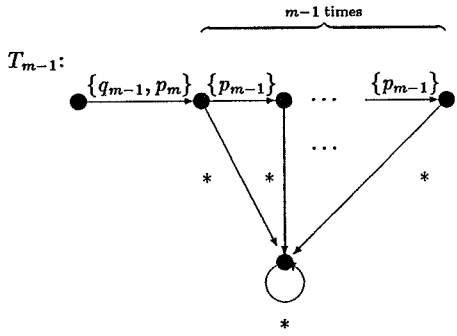
Lemma 34 *Let A and A' be two normal forms such that*

$$\models A \simeq A'$$

for some behaviour equivalence $\simeq$. Then each loop ended chain in A is also in A'.

Proof: The proof is similar to the previous lemma; we here indicate the differences. Let A and CH be as in lemma 32, but let CH be loop ended. Thus, $m \geq 2$. Note that since $\ell(m-1) = \ell(m)$ it follows that $C_{\ell(m-1)} = C_{\ell(m)}$ which implies $M_{m-1} = M_m$.

Define the templates T_i for $1 \leq i \leq m-2$ as in lemma 32, and define T_{m-1} by



The transitions labelled "*" actually correspond to two transitions: one labelled $\{q_{m-1}\}$ and one labelled $\{p_m\}$. Thus, T_{m-1} represents a behaviour that begins by one $\{q_{m-1}, p_m\}$ transition — for the first time we make use of the fact

that events may be labelled with sets of cardinality greater than one — and continues with $m-1$ transitions labelled $\{p_{m-1}\}$. Following the first transition it can always do $\{q_{m-1}\}$ or $\{p_m\}$ transitions, thereby ending up in the bottom state where $\{q_{m-1}\}$ and $\{p_m\}$ transitions are possible.

The construction of the templates U_i and the interpretation I is as in lemma 32, only that the new T_{m-1} is used. The proof that $A \stackrel{\{e\}}{\Longrightarrow} \bullet$ is similar and we omit it here. Lemma 33 is still true in a modified form: point 3 says "If $i \leq m-2 \ldots$" and there is an additional point 4: If $i = m-1$ then $\sigma = (\{c_{q_{m-1}}, c_{p_m}\})a^{m-1}$.

Finally, the proof that $A' \stackrel{\{e\}}{\Longrightarrow} \bullet$ implies that A' has CH is modified as follows. The first $m-1$ steps are similar, constructing the function ℓ' as evidence of the linear chain

$$\stackrel{e\frown p_1}{\longleftrightarrow} M_1 \stackrel{q_1\frown p_2}{\longleftrightarrow} M_2 \stackrel{q_2\frown p_3}{\longleftrightarrow} \ldots \stackrel{q_{m-2}\frown p_{m-1}}{\longleftrightarrow} M_{m-1}$$

which is the prefix of CH. According to the modified lemma 33:4, $C'_{\ell'(m-1)} \stackrel{\sigma_{\ell'(m-1)}}{\Longrightarrow} D'_{\ell'(m-1)}$ where $\sigma_{\ell'(m-1)} = (\{c_{q_{m-1}}, c_{p_m}\}) \ldots$ (for brevity we here write c for $c^{(m-1)}$). This means that both $c_{q_{m-1}}$ and c_{p_m} must be linked in A', or one of them would precede the e in $A' \stackrel{\{e\}}{\Longrightarrow}$. There are three possibilities:

1. Either $c_{q_{m-1}}$ or c_{p_m} are linked to a port *not* in $C'_{\ell'(m-1)}$. Then this can be used to establish a linear chain in A' which is not in A. This contradicts lemma 32.

2. $c_{q_{m-1}}$ and c_{p_m} are linked to ports, call them d and d', in $C'_{\ell'(m-1)}$, but they are not linked to each other. But then $\sigma_{\ell'(m-1)}$ can not begin with $\{c_{q_{m-1}}, c_{p_m}\}$: since $c_{q_{m-1}}$ and d are linked in A', either none or both of them occur in any event used in a derivation of a transition of A'.

3. $c_{q_{m-1}}$ is linked to c_{p_m}. This establishes the full chain CH in A'. □

The proof of lemma 32 can be understood as follows: given any chain CH, we construct an interpretation I such that $A \stackrel{e}{\Longrightarrow} \bullet$ iff A has that chain. The structure of this interpretation is that the top of the chain, $C_{\ell(1)}$, awaits exactly two synchronisations from below, and then emits the e. If it experiences more than two synchronisations from below, then it runs berserk and produces an infinite number of e:s without ever being able to deadlock. The second unit in the chain, $C_{\ell(2)}$, requires three synchronisations from below before engaging $C_{\ell(1)}$ with two synchronisations. In general, the ith unit in the chain $C_{\ell(i)}$ requires exactly $i+1$ synchronisations from below in order to engage in i synchronisations upwards. If it may take part in more than $i+1$ synchronisations from below, it runs berserk and may take part in more than i synchronisations upwards. Thus, if any unit in the chain has the ability to run berserk, it will enable all units above it in the chain to run berserk, whence A will not deadlock after having done its e event.

In a full execution of the chain CH of length m, the bottom element engages in m synchronisations with the next to bottom element, which engages in $m-1$ synchronisations with the next to next to bottom element etc. It is clear from the structure of the proof that the sequence of numbers of synchronisations between units $(m, m-1, \ldots, 2$ followed by 1 external event $e)$ is a property of I which can be arbitrarily chosen. It is possible to define I with *any* sequence $n_m, n_{m-1}, \ldots, n_1$ of distinct numbers as follows: redefine T_i in lemma 32 (for $1 \leq i \leq m-1$) to do n_{i+1} events q_i followed by n_i events p_i, and redefine T_m to do n_m events p_m. As a result, a full execution will now result in n_m synchronisations between bottom and next to bottom etc., and finally n_1 events e. If the numbers n_i are distinct, a lemma corresponding to lemma 33 holds, and $A \stackrel{(e)^{n_1}}{\Longrightarrow} \bullet$ iff A has the chain CH. The sequence $n_m, \ldots, n_1$ is called the *characteristic sequence* of I for CH.

The fact that the characteristic sequence can be almost arbitrarily chosen is critical for the final difficult lemma below. Let A be a normal form, let CH be any chain with evidence ℓ in A, and a a port name in a link $\langle a^\frown b\rangle$ or $\langle b^\frown a\rangle$ in A. Say that a is a *downwards port* in CH iff $a = a_q$, $b = b_p$, $M_i \stackrel{q^\frown p}{\longleftrightarrow} M_{i+1}$ is in CH, $C_{\ell(i)} = M_i(\tilde{a})$, and $C_{\ell(i+1)} = M_{i+1}(\tilde{b})$. Thus, a is a downwards port if it is used in CH to connect a module to a module below it in the chain. Conversely, b is an *upwards port* iff it is used to connect a module to a module above it. Say that two chains are in *conflict* over a iff a is an upwards port in one chain and a downwards port in the other. Two chains are *conflict free* in A iff they are not in conflict over any port in A.

Lemma 35 *Let A and A' be two normal forms such that $\models A \simeq A'$ for some behaviour equivalence $\simeq$. Let CH_1 and CH_2 be two conflict free linear chains of lengths m_1 and m_2 respectively in A. By lemma 32 these chains are also in A'. Let the evidences for these chains in A be ℓ_1 and ℓ_2 respectively, and let the evidences in A' be ℓ'_1 and ℓ'_2 respectively. Then*

1. *CH_1 and CH_2 are conflict free in A'.*
2. *$\ell_1(m_1) = \ell_2(m_2)$ iff $\ell'_1(m_1) = \ell'_2(m_2)$.*

Proof: The lemma says that if there are two conflict free chains in A, then these chains are also conflict free in B, and that if the bottom elements are identified in A then they are also identified in B. Let

$$CH_1 = \stackrel{e_1^\frown p_{11}}{\longleftrightarrow} M_{11} \stackrel{q_{11}^\frown p_{12}}{\longleftrightarrow} \cdots \stackrel{q_{1(m_1-1)}^\frown p_{1m_1}}{\longleftrightarrow} M_{1m_1}$$

$$CH_2 = \stackrel{e_2^\frown p_{21}}{\longleftrightarrow} M_{21} \stackrel{q_{21}^\frown p_{22}}{\longleftrightarrow} \cdots \stackrel{q_{2(m_2-1)}^\frown p_{2m_2}}{\longleftrightarrow} M_{2m_2}$$

Let $T_{11}, \ldots, T_{1m_1}$ be the transition diagrams obtained from lemma 32 in proving that A' has CH_1. Let $U_{11}, \ldots, U_{1m_1}$ be the templates as defined in that lemma by

$$U_{1i} = T_\dagger \cup \bigcup_{j:M_{1i}=M_{1j}} T_{1j}$$

Let I_1 be the interpretation which maps each module M_{1i} to U_{1i} and all other modules to $T_\dagger$. Thus, I_1 is the interpretation in which $A' \stackrel{e}{\Longrightarrow} \bullet$ iff A' has CH_1. The characteristic sequence of I_1 is

$$m_1,\ m_1 - 1,\ \ldots, 1$$

Correspondingly, let $T_{21}, \ldots, T_{2m_2}$ be transition diagrams obtained from lemma 32 in proving that A' has CH_2, and let $U_{21}, \ldots, U_{2m_2}$ be defined by

$$U_{2i} = T_\dagger \cup \bigcup_{j:M_{2i}=M_{2j}} T_{2j}$$

and let I_2 be the interpretation which maps each module M_{2i} to U_{2i} and all other modules to $T_\dagger$. However, let the characteristic sequence of I_2 for CH_2 be

$$m_2(m_1+1),\ (m_2-1)(m_1+1),\ \ldots, m_1+1$$

Thus all elements in this sequence are distinct from each other and from $m_1, \ldots, 1$. The important property is that any number between 0 and $(m_2+1)(m_1+1)-1$ inclusive has a *unique* decomposition as a sum of two numbers, one number being either 0 or in the characteristic sequence of I_1, and the other number being either 0 or in the characteristic sequence of I_2. It is clear that in I_2, both $A \stackrel{(e_2)^{m_1+1}}{\Longrightarrow} \bullet$ and $A' \stackrel{(e_2)^{m_1+1}}{\Longrightarrow} \bullet$.

Define the *merge product* $\times$ of templates as follows:

$$\langle D_1, E_1, d_{(1)} \rangle \times \langle D_2, E_2, d_{(2)} \rangle = \langle D, E, (d_{(1)}, d_{(2)}) \rangle$$

where

$$\begin{aligned} D &= \{(d_1, d_2) : d_1 \in D_1, d_2 \in D_2\} \\ ((d_1, d_2), K, (d_1', d_2')) &\in E' \text{ iff } (d_1, K, d_1') \in D_1 \text{ or } (d_2, K, d_2) \in D_2 \end{aligned}$$

Thus, $T_1 \times T_2$ is a template which behaves like T_1 and T_2 in parallel under interleaving. If C is a unit whose module is mapped to such a product template, then any transition sequence of C will correspond to an interleaving of a transition sequence in T_1 and a sequence in T_2.

Now define the interpretation $I_1 \times I_2$ by mapping M to $I_1(M) \times I_2(M)$ for all M.

Consider the behaviour of A in $I_1 \times I_2$. Specifically, consider the sequence of transitions corresponding to first a full execution of CH_1, and then a full execution of CH_2. This way we get

$$A \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} B$$

for some B, where all units in B involved in either CH_1 or CH_2 can only do synchronisations on downwards ports. Since downwards ports are linked to upwards ports, and since the chains are conflict free, i.e. no port can be upwards in one chain and downwards in another chain, B is deadlocked. This proves $A \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} \bullet$. Note that this holds even if the chains overlap, and even if $e_1 = e_2$.

Since $\models A' \simeq A$, and in particular $I_1 \times I_2 \models A' \simeq A$, we must have $A' \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} \bullet$, i.e. $A' \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} B'$ for some deadlocked B'. With an argument similar to that in lemma 32, such a transition sequence must correspond to a full execution of CH_1 and a full execution of CH_2. These executions may to some extent be interleaved, but since they have completely different characteristic sequences they can not interfere in any significant way. This holds even if the chains overlap, and even if $e_1 = e_2$. Here, the unique decomposition property mentioned above is critical.

It follows that in B', all units can do synchronisations only on downwards ports. Now assume that there is a port a, linked to b in A', such that both a and b are downwards ports. Then a synchronisation over this link would be possible. The result of this synchronisation is to take the units with ports a and b to their "berserk" states, where arbitrarily many synchronisations on an upwards port is possible. As remarked above, if one unit reaches its berserk state, then all units above it in the chain may go berserk. This contradicts the fact that B' is deadlocked. Hence no such ports a and b can exist; whence it follows that CH_1 and CH_2 are conflict free in A'. This proves 1 of the lemma.

To prove 2, assume that $\ell_1(m_1) = \ell_2(m_2)$. We will prove $\ell'_1(m'_1) = \ell'_2(m'_2)$ (the converse is symmetric). To begin, observe that $C_{\ell_1(m_1)} = C_{\ell_2(m_2)}$, and as a consequence $M_{1m_1} = M_{2m_2}$. Write M for $M_{1m_1} (= M_{2m_2})$ in the rest of this proof. Through the product interpretation $I_1 \times I_2$ the module M is mapped to $U_{1m_1} \times U_{2m_2}$. Recall that each of the factors is formed as a union:

$$U_{1m_1} = T_\dagger \cup \bigcup_{j:M_{1j}=M_{1m_1}} T_{1j}$$

$$U_{2m_2} = T_\dagger \cup \bigcup_{j:M_{2j}=M_{2m_2}} T_{2j}$$

Now consider the interpretation J which is the same as $I_1 \times I_2$ on all modules except M, where it is

$$J(M) = \left(\left(T_\dagger \cup \bigcup_{j \neq m_1 : M_{1j}=M_{1m_1}} T_{1j} \right) \times T_{2m_2} \right)$$

$$\bigcup\left(\left(T_\dagger \cup \bigcup_{j\neq m_2 : M_{2j}=M_{2m_2}} T_{2j}\right) \times T_{1m_1}\right)$$
$$\bigcup\left(\left(T_\dagger \cup \bigcup_{j\neq m_1 : M_{1j}=M_{1m_1}} T_{1j}\right) \times \left(T_\dagger \cup \bigcup_{j\neq m_2 : M_{2j}=M_{2m_2}} T_{2j}\right)\right)$$

The important thing to note about $J(M)$ is that it can *not* act as T_{1m_1} and T_{2m_2} in parallel. Thus, if C is a unit with module M, then in any particular execution, C may behave as either T_{1m_1} or T_{2m_2} but not as an interleaving of both.

Recalling that $\ell_1(m_1) = \ell_2(m_2)$, we can now prove that $A \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} \bullet$ is impossible in J. Such a transition sequence would correspond to full executions of both CH_1 and CH_2. This means that $C_{\ell_1(m_1)}$ must contribute a transition sequence according to T_{1m_1}, and $C_{\ell_2(m_2)}$ must contribute a transition sequence according to T_{2m_2}. But these units are actually identical, and according to J, there is no unit which can behave as both T_{1m_1} and T_{2m_2} in the same execution.

Assume that $\ell'_1(m_1) \neq \ell'_2(m_2)$. Then A' can perform first the full execution corresponding to CH_1, and then the full execution corresponding to CH_2. The reason is that $C'_{\ell'_1(m_1)}$ and $C'_{\ell'_2(m2)}$ are distinct; thus one of them may contribute a transition sequence corresponding to T_{1m_1} and the other a sequence corresponding to T_{2m_2}. This implies $A' \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} \bullet$.

But from $J \models A \simeq A'$ it is impossible that $A' \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} \bullet$ and not $A \stackrel{e_1(e_2)^{m_1+1}}{\Longrightarrow} \bullet$.

Thus, we must have $\ell'_1(m_1) = \ell'_2(m_2)$. This concludes the proof of lemma 35. □

Lemma 36 *Let A and A' be two normal forms such that $\models A \simeq A'$ for some behaviour equivalence $\simeq$. Let CH_1 and CH_2 be two conflict free linear chains of lengths m_1 and m_2 respectively in A. By lemma 32 these chains are also in A'. Let the evidence for these chains in A be ℓ_1 and ℓ_2 respectively, and let the evidence in A' be ℓ'_1 and ℓ'_2 respectively. Then for all i, j ($1 \leq i \leq m_1$ and $1 \leq j \leq m_2$):*

$$\ell_1(i) = \ell_2(j) \text{ iff } \ell'_1(i) = \ell'_2(j)$$

Proof: Let CH'_1 and CH'_2 be the prefixes of CH_1 and CH_2 of length i and j respectively. Assume $\ell_1(i) = \ell_2(j)$. Then, by lemma 31, the evidences of CH'_1 and CH'_2 in A agree on the bottom elements. Thus, by lemma 35, their evidences in A' also agree on the bottom elements. Again by lemma 31, $\ell'_1(i) = \ell'_2(j)$. □

Let A be a normal form

$$A = (C_1| \cdots |C_n)\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$$

and S a set of chains in A. The set S is said to *span* A iff each unit and link in A is used in at least one chain in CH. For the purpose of spanning external ports

are counted as trivial links, hence a spanning set S must also contain chains with all external ports.

Lemma 37 *Let A be a normal form without isolated parts. Then there is a set S of chains which spans A such that any two chains in S are conflict free.*

Proof: By direct construction of S. Let the *rank* of a unit C in A be 1 iff C has an external port, and i ($i \geq 2$) iff the rank of C is not $i-1$, and there is a link in A connecting C with a unit of rank $i-1$. In this way since A is finite and without isolated parts, each unit in C is assigned a finite rank.

Now build S as follows. Form S_1 as the set of chains of length one. Form S_i from S_{i-1} in the following way: for each chain CH with evidence ℓ in S_{i-1} add CH and, if CH is linear and of length $i-1$, also add the following extensions of CH to chains of length i to S_i:

1. All possible extensions such that the rank of $C_{\ell(i-1)}$ is strictly greater than the rank of $C_{\ell(i)}$.
2. All possible extensions such that $C_{\ell(i-1)}$ and $C_{\ell(i)}$ have the same rank and $\ell(i) > \ell(i-1)$.
3. All possible extensions $CH \overset{q\frown p}{\longleftrightarrow} M$ such that $\ell(i) = \ell(i-1)$ and $q < p$ (note that the result is a loop ended chain).

The sequence $S_1, S_2, \ldots$ converges to a set S. It is easily established that S has the desired properties: S_i spans the units of A with rank less than or equal to i, and all links connected with a unit of rank less than i. For each link $\langle a \frown b \rangle$ in A, it is uniquely determined which of a and b can be an upwards port, hence any two chains in S_i are conflict free. □

Lemma 38 *Let A and A' be two normal forms without isolated parts such that $\models A \simeq A'$ for some behaviour equivalence $\simeq$. Let S be a set of chains which spans A. Then S spans A'.*

Proof: By lemmas 32 and 34 all chains in S must be in A'. Let x be an arbitrary element (unit or link) in A'. Since A' has no isolated parts, x must be used by a chain CH in A'. Then, by the same lemmas, CH is in A. By the definition of span, in A it holds that any element used by CH is used by a chain in S. Then by lemma 36 the same holds in A', whence x is used by a chain in S. □

Let two normal forms A and A' be

$$A = \left(M_1(\widetilde{a^{(1)}})|\cdots|M_n(\widetilde{a^{(n)}})\right)\langle a_1 \frown b_1\rangle \cdots \langle a_k \frown b_k\rangle$$
$$A' = \left(M'_1(\widetilde{a'^{(1)}})|\cdots|M'_{n'}(\widetilde{a'^{(n)}})\right)\langle a'_1 \frown b'_1\rangle \cdots \langle a'_{k'} \frown b'_{k'}\rangle$$

Say that a function $h : \{1,\ldots,n\} \rightarrow \{1,\ldots n'\}$ is a *homomorphism* from A to A' iff it satisfies

1. $M_i = M'_{h(i)}$.
2. If $a_p^{(i)}$ is in $L(A)$ then $a_p^{(i)} = a_p^{\prime(h(i))}$ and it is in $L(A')$.
3. If $a_p^{(i)}$ and $a_q^{(j)}$ are linked in A then $a_p^{\prime(h(i))}$ and $a_q^{\prime(h(j))}$ are linked in A'. .

The function h is an *isomorphism* from A to A' iff h is a homomorphism from A to A' and h^{-1} is a homomorphism from A' to A. The normal forms A and A' are *isomorphic* iff there exists an isomorphism from A to A'. In essence, two isomorphic normal form may differ only in the order of units and links and the choice of name of the linked ports — the linking structure and the external ports must be identical. As expected, two normal forms are isomorphic iff their flowgraphs are isomorphic (in the standard sense of graph isomorphism).

Lemma 39 *Let A and A' be two normal forms without isolated parts such that $\models A \simeq A'$ for some behaviour equivalence $\simeq$. Then A and A' are isomorphic.*

Proof: By lemma 37 there is a set S of chains which spans A and where any two chains are conflict free. By lemma 38 this set also spans A', and by lemma 35 any two chains in S are conflict free in A'. Define the function h by

$$h(\ell(i)) = \ell'(i)$$

iff ℓ and ℓ' are the evidences in A and A' respectively for a chain CH of length greater than or equal to i in S. By lemma 36 and the definition of spanning set, this uniquely defines h, and h is a bijection from the indices of the units in A to the indices of the units in A'. The required properties of h and h^{-1} follow directly from the definition of chain. □

Lemma 40 *Let A and A' be two isomorphic normal forms. Then $\vdash A = A'$.*

Proof: Let h be an isomorphism between A and A'. With (P1) and (P2) it is possible to reorder the units of A' to conform with the order in A so that $C'_{h(i)}$ becomes the ith unit in A'. By further applying (S) it is possible to substitute all linked port names in A' to conform with A, so that the ith unit in A' is identical with the ith unit in A. By the properties of h, the link $\langle a^\frown b\rangle$ is in A iff either $\langle a^\frown b\rangle$ or $\langle b^\frown a\rangle$ is a link in the so transformed A'. Hence A' can be made syntactically identical with A through application of (P1), (P2), (S), (L1) and (L2). □

Proof of theorem 6: Let A and A' be two terms such that $\models A \simeq A'$ for some behaviour equivalence $\simeq$. By definition of behaviour equivalence we know that $L(A) = L(A')$. If $L(A) = L(A') = \emptyset$, then by $(\emptyset)$ and (P1) we get

$$\vdash A = A|A' = A'|A = A'$$

So assume that $L(A) = L(A') \neq \emptyset$. By lemma 29 there are terms B, B' on normal form such that $\vdash A = B$ and $\vdash A' = B'$. By lemma 30 there are normal forms C and C' such that $\vdash B = C$ and $\vdash B' = C'$ and neither C nor C' have any isolated parts. By lemma 39 C and C' are isomorphic, whence by lemma 40 we conclude $\vdash C = C'$. □

Copyright © 1987, Laboratory for Foundations of Computer Science,
University of Edinburgh. All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.