# Submodule Construction as Equation Solving in CCS *

Joachim Parrow
Swedish Institute of Computer Science
Box 1263, S-16428 Kista, Sweden

**Abstract**

In top-down design methodologies the following problem arises: given specifications of a system and of some of its submodules, derive a specification for the remaining submodules. We formulate this problem in CCS as an equation $(A|X)\backslash L \approx B$, where $X$ is unknown, $B$ represents the whole system, $A$ the known submodules, and $L$ the channels over which the submodules interact. We present a procedure for solving such equations by successive transformation of equations into simpler equations in parallel with generation of a solution. The procedure has been implemented as a semiautomatic program, where the user may interact in order to guide the transformations towards particular solutions. As an example we demonstrate the automatic generation of receivers of two versions of the Alternating Bit protocol.

## 1 Introduction

One of the most important and difficult fields in computer science is to develop methods for construction of complex systems. Most design methodologies rely on *modularization*: systems are partitioned into submodules, and each submodule is given a specification. These specifications contain sufficient information for proving correctness of the combined system.

In this paper, we will consider a particular problem in the top-down design of systems containing several nondeterministic modules executing in parallel. A specification of such a system typically defines the behaviour of the system when it interacts with its environment. In a top-down design methodology, the designer will begin with a specification of the system to be constructed, and proceed to construct the submodules one by one. As an example, consider a communication protocol, where the submodules are a sender, a receiver, and a medium. In the beginning, the designer knows only the specification of the behaviour of the entire protocol, i.e. the protocol service specification. He partitions the protocol into its submodules and proceeds by giving specifications for each of them. The main idea in this paper is that when

*This work was carried out when the author was visiting the Laboratory for Foundations of Computer Science, University of Edinburgh, supported by a grant from the British Science and Engineering Research Council.

two submodules, say the sender and the medium, have been given specifications, it may be possible to infer a specification for the receiver.

This idea has been presented and elaborated by Merlin and Bochmann in [10]. Their specifications are given as transition systems, and the submodules are considered correct if the combined system has the same set of possible execution sequences as the original specification. One limitation is that this notion of correctness does not capture some aspects, e.g. potential deadlocks, of the behaviour of the system. Thus, a receiver satisfying the automatically generated specification may cause deadlocks.

We will here use a more refined notion of correctness, namely *observation equivalence* as developed by Milner in e.g. [11]. Essentially, two systems are observation equivalent if they cannot be distinguished by an external observer. This equivalence is more discriminating than comparing execution sequences; in particular it is sensitive to potential deadlocks. Following [10] our specifications are transition systems with finite state spaces. We have found it convenient to use the syntax of CCS (Calculus of Communicating Systems; see [11]) rather than representing specifications directly as transition graphs. Our results are in principle equally applicable to any other notation.

In CCS specifications are called *agents*. Observation equivalence between agents is written $\approx$. There is a formal syntax for combining agents: a system composed of agents $A_1, A_2, \ldots, A_n$ executing in parallel and communicating over channels $L$ is written

$$(A_1|A_2|\cdots|A_n)\backslash L$$

In a top-down design methodology, the designer starts with an agent, call it $B$, representing the behaviour of the whole system to be constructed. He divides the system into $n$ modules, communicating over channels $L$, and proceeds to construct for each module $i$ an agent $A_i$. The criterion for a correct construction is that

$$(A_1|\cdots|A_n)\backslash L \approx B$$

Now, assume that the designer has actually constructed all agents but one, say $A_n$. The missing agent can then be obtained as a solution for $X$ of the CCS equation (for clarity, put $A = A_1|\cdots|A_{n-1}$):

$$(A|X)\backslash L \approx B \tag{†}$$

The theory of such equations has to some extent been studied by Shields ([15]); we defer a discussion on this and other related work to section 6.

Equations of type (†) may in general have several solutions, some of which are unsuitable for implementation. For example, some solutions may be unnecessarily complex. Consider a retransmission protocol where the sender always retransmits each message at least a million times, even if the message gets through on the first attempt. This sender might be formally correct, but would be highly inefficient.

In this paper we present a method for solving equations of type (†) where $A$ and $B$ are finite state agents, and $B$ is deterministic. The method has been implemented in a program which will attempt to automatically find a solution. Having tested the program on nontrivial examples, we conclude that it may produce solutions which are not suitable for implementation. We have not formulated general criteria for suitable solutions. Instead our program can be run in a semiautomatic mode: a designer may interact with it in order to guide it towards suitable solutions. The program helps the designer to find agents which are guaranteed to be (formally) correct, or convinces him that no such agents exist.

The procedure is based on stepwise transformation of equations into simpler equations. As an example (we here assume the reader to be familiar with the fundamental concepts of CCS), assume that we want to solve

$$a.NIL \mid X \;\approx\; a.b.NIL + b.a.NIL$$

The first step of the procedure is to guess the initial actions of $X$. Consider the initial actions of the two sides of the equation: the right hand side can do actions $a$ and $b$, thus the left hand side should be able to do the same actions. Since $a.NIL$ can only do $a$, the unknown $X$ must supply the $b$ action. Guessing that $X$ has $b$ as the only initial action we may write $X = b.Y$ where $Y$ is a new unknown. Substituting $b.Y$ for $X$ in the original equation we get

$$a.NIL \mid b.Y \;\approx\; a.b.NIL + b.a.NIL$$

and by applying the expansion theorem on the left hand side we get

$$a.(NIL \mid b.Y) + b.(a.NIL \mid Y) \;\approx\; a.b.NIL + b.a.NIL$$

but by congruence properties, this is implied by

$$\begin{cases} NIL \mid b.Y & \approx \quad b.NIL \\ a.NIL \mid Y & \approx \quad a.NIL \end{cases}$$

We have now transformed the original equation into two simpler equations, and by similar reasoning we will find that $Y = NIL$ solves them both. Hence, $X = b.NIL$ is a solution.

The rest of this paper is structured as follows. In section 2 we define the syntax and semantics of the part of CCS which is related to our work. In section 3 we present the tableau method: section 3.1 contains the general definitions of tableaus and tableau transformations, and sections 3.2 and 3.3 particular transformations for solving equations of type (†). The transformations are proven sound and complete. In section 4 we explore ways to automate the transformations, and describe an implementation in the form of a semiautomatic program. This program is applied in section 5 to nontrivial examples: generating the receiver of two versions of the Alternating Bit protocol. Section 6 contains ideas for extending this work, and comparisons with similar efforts.

# 2 Preliminaries

In this section, we establish the notation for the rest of the paper. Although all concepts will be formally defined, a reader unfamiliar with CCS is advised to consult some introduction to CCS such as [11] for more extensive explanations.

Assume a set **Act** of *actions* where the *inverse* of the action $a$ is the action $\overline{a}$. The unobservable action $\tau$ has no inverse. Let **Id** be a set of *identifiers*. The set **Ag** of *agents* is the smallest set containing **Id** and closed under the following operators:

$a.$ (called prefixing) for $a \in \mathbf{Act}$; prefix operator.

$+$ (called nondeterministic choice); binary infix operator.

$|$ (called parallel composition); binary infix operator.

$\backslash L$ (called restriction) for $L \subseteq \mathbf{Act} - \{\tau\}$, $L$ finite; postfix operator.

$NIL$ agent constant.

As a shorthand for $(A|B)\backslash L$ we will write $A \parallel_L B$, or even $A \parallel B$ when $L$ is unimportant or understood from the context. As a shorthand for $A_1 + A_2 + \cdots + A_n$ we write

$$\sum_{i=1}^{n} A_i$$

If $n = 1$, the above sum is just $A_1$, and if $n = 0$ it is $NIL$.

An *environment* $\mathcal{E}$ is a partial function from **Id** to **Ag**. An identifier is said to be *bound* by $\mathcal{E}$ if it is in the domain of $\mathcal{E}$, otherwise it is *free* in $\mathcal{E}$. The behaviour of an agent (i.e. the transitions that the agent can perform) is always determined with respect to an environment. Thus, we have for each $a \in \mathbf{Act}$ and environment $\mathcal{E}$ the binary transition relation $\xrightarrow{a}_{\mathcal{E}}$ on agents. These relations are defined to be the smallest relations satisfying the following clauses, which can be regarded as the operational semantics of the operators:

$$\frac{}{a.A \xrightarrow{a}_{\mathcal{E}} A} \qquad \frac{A \xrightarrow{a}_{\mathcal{E}} A'}{B + A \xrightarrow{a}_{\mathcal{E}} A'} \qquad \frac{A \xrightarrow{a}_{\mathcal{E}} A'}{A + B \xrightarrow{a}_{\mathcal{E}} A'}$$

$$\frac{A \xrightarrow{a}_{\mathcal{E}} A'}{A|B \xrightarrow{a}_{\mathcal{E}} A'|B} \qquad \frac{A \xrightarrow{a}_{\mathcal{E}} A'}{B|A \xrightarrow{a}_{\mathcal{E}} B|A'} \qquad \frac{A \xrightarrow{a}_{\mathcal{E}} A',\ \ B \xrightarrow{\bar{a}}_{\mathcal{E}} B'}{A|B \xrightarrow{\tau}_{\mathcal{E}} A'|B'}$$

$$\frac{A \xrightarrow{a}_{\mathcal{E}} A',\ \ a \notin L,\ \bar{a} \notin L}{A\backslash L \xrightarrow{a}_{\mathcal{E}} A'\backslash L} \qquad \frac{\mathcal{E}(X) \xrightarrow{a}_{\mathcal{E}} A}{X \xrightarrow{a}_{\mathcal{E}} A}$$

An agent $B$ is a *derivative* of $A$ (w.r.t. an environment $\mathcal{E}$) if for some $n \geq 0$ there are actions $a_1, \ldots, a_n$ such that

$$A \xrightarrow{a_1}_{\mathcal{E}} \cdots \xrightarrow{a_n}_{\mathcal{E}} B$$

If $n = 0$, this formula is interpreted as $A = B$, thus $A$ is a derivative of itself.

We define the *experiment* relation $\overset{a}{\Longrightarrow}_{\mathcal{E}}$, $a \in \mathbf{Act}$ on agents in the following way:

$$A \stackrel{a}{\Longrightarrow}_{\mathcal{E}} B \text{ if } \begin{cases} \text{for some } n \geq 0\text{: } A \underbrace{\stackrel{\tau}{\rightarrow}_{\mathcal{E}} \cdots \stackrel{\tau}{\rightarrow}_{\mathcal{E}}}_{n \text{ times}} B & \text{if } a = \tau \\ \text{for some } n, m \geq 0\text{: } A \underbrace{\stackrel{\tau}{\rightarrow}_{\mathcal{E}} \cdots \stackrel{\tau}{\rightarrow}_{\mathcal{E}}}_{n \text{ times}} \stackrel{a}{\rightarrow}_{\mathcal{E}} \underbrace{\stackrel{\tau}{\rightarrow}_{\mathcal{E}} \cdots \stackrel{\tau}{\rightarrow}_{\mathcal{E}}}_{m \text{ times}} B & \text{if } a \neq \tau \end{cases}$$

When the environment is understood from the context, or unimportant, we will drop the index $\mathcal{E}$ in $\rightarrow$ and $\Longrightarrow$. Note that $A \stackrel{\tau}{\Longrightarrow} A$ always holds.

An agent $A$ is *deterministic* in the environment $\mathcal{E}$ if all its derivatives $B$ satisfy the following: for no $C$ does it hold that $B \stackrel{\tau}{\rightarrow}_{\mathcal{E}} C$, and if $B \stackrel{a}{\rightarrow}_{\mathcal{E}} C_1$ and $B \stackrel{a}{\rightarrow}_{\mathcal{E}} C_2$ then $C_1 = C_2$. Essentially, this means that the agent can never do any $\tau$-actions (in the terminology of [11] it is *rigid*), and that for each action $a$ there is at most one transition labelled $a$ from any given derivative.

Let $\sqsubseteq$ be the usual partial order on partial functions, i.e. $f \sqsubseteq g$ if for all $x$ such that $f(x)$ is defined, $g(x)$ is defined and $g(x) = f(x)$. Applied on environments, we say that $\mathcal{F}$ *extends* $\mathcal{E}$ if $\mathcal{E} \sqsubseteq \mathcal{F}$.

Let $\mathcal{E}$ be an environment. Define the relation $\triangleright_{\mathcal{E}}$ on $\mathbf{Ag} \times \mathbf{Id}$ to be the least relation satisfying

(1) $X \triangleright_{\mathcal{E}} X$ for all free identifiers $X$
(2) $A \triangleright_{\mathcal{E}} X$ implies $A + B \triangleright_{\mathcal{E}} X$, $B + A \triangleright_{\mathcal{E}} X$, $A|B \triangleright_{\mathcal{E}} X$, $B|A \triangleright_{\mathcal{E}} X$, $A \backslash L \triangleright_{\mathcal{E}} X$
(3) $A \triangleright_{\mathcal{E}} X$ and $\mathcal{E}(Y) = A$ implies $Y \triangleright_{\mathcal{E}} X$

Say that $A$ has an *unguarded* $X$ in $\mathcal{E}$ if $A \triangleright_{\mathcal{E}} X$. An agent is *well guarded* in $\mathcal{E}$ if it has no unguarded identifiers in $\mathcal{E}$. This means that the initial transitions from the agent are not affected by extending $\mathcal{E}$. An agent is *closed* if all identifiers which occur as subexpressions in it (here $\mathcal{E}(X)$ counts as a subexpression of $X$) are bound. This means that neither transitions from the agent, nor transitions from any derivative, are affected by extending $\mathcal{E}$.

With respect to a particular environment, a binary relation $R$ on closed agents is a *simulation* if $ARB$ implies:

for all $a$ and $A'$ such that $A \stackrel{a}{\Longrightarrow} A'$
there exists a $B'$ such that $B \stackrel{a}{\Longrightarrow} B'$ and $A'RB'$

If both $R$ and $R^{-1}$ are simulations, $R$ is said to be a *bisimulation*. Two closed agents $A$ and $B$ are *observation equivalent*, written $A \approx B$, if there exists a bisimulation $R$ such that $ARB$. Observation equivalence is extended to arbitrary agents by $A \approx B$ if in all extensions where $A$ and $B$ are closed, $A \approx B$.

We will in the following use $a$, $b$, ... to denote actions, $A$, $B$, ... to denote agents, $X$, $Y$, ... to denote identifiers, and $\mathcal{E}$, $\mathcal{F}$, ... to denote environments.

# 3 The Tableau Method

## 3.1 Tableaux

As mentioned in the introduction, our aim is to present a procedure for solving equations of type $A \parallel X \approx B$ by successive transformations of equations. We formalise this reasoning by using tableaux. A *tableau* consists of two parts: a *goal* $\Gamma$ and an environment $\mathcal{E}$. The intuition behind a tableau is that it represents an intermediate stage in producing a solution: the goal says what remains to be done, and the environment records the solution produced so far. A goal is a unary predicate over environments. For example, the goal "$X \approx Y$" is true of the environments assigning observationally equivalent agents to $X$ and $Y$. As another example, the goal "$X + X \approx X$" is true of all environments.

In order to solve an equation of type $A \parallel X \approx B$ we will start with an initial tableau with goal "$A \parallel X \approx B$" and an environment where $A$ and $B$ are closed and $X$ is free, meaning "it remains to find an extension satisfying $A \parallel X \approx B$." The procedure then works by successively simplifying the equations and extending the environment until a tableau with goal *true* is reached. The environment of that tableau will contain the desired solution.

In the rest of this section we will make these ideas formally rigorous. We use $\Gamma$, $\Gamma'$ etc. to range over goals; the syntax for goals is defined by

$$\Gamma \quad ::= \quad \mathit{true} \quad | \quad A \approx B \quad | \quad \Gamma \wedge \Gamma' \quad | \quad \Gamma \supset \Gamma'$$

The *satisfaction relation* $\models$ between environments and goals is defined inductively as the least relation satisfying:

$$\begin{array}{ll}
\mathcal{E} \models \mathit{true} & \text{always} \\
\mathcal{E} \models A \approx B & \text{if } A \approx B \text{ holds in } \mathcal{E} \\
\mathcal{E} \models \Gamma \wedge \Gamma' & \text{if } \mathcal{E} \models \Gamma \text{ and } \mathcal{E} \models \Gamma' \\
\mathcal{E} \models \Gamma \supset \Gamma' & \text{if for all } \mathcal{F} \sqsupseteq \mathcal{E},\ \mathcal{F} \models \Gamma \text{ implies } \mathcal{F} \models \Gamma'
\end{array}$$

Note the definition of implication: the idea is that "$\Gamma$ implies $\Gamma'$" is true when any evidence of $\Gamma$, i.e. extension of the environment where $\Gamma$ holds, is also evidence of $\Gamma'$. Here, this can be thought of as implicit universal quantification over free identifiers. Actually, $\supset$ will never occur in any of our tableaux, but this notion of logical implication is convenient when formulating results about tableau transformations. In particular, we will use it when formulating the soundness result.

An important property of the satisfaction relation is the *preservation property*: if an environment satisfies a goal, then all extensions of the environment also satisfy the goal.

**Proposition 1 (preservation property)** *If $\mathcal{E} \models \Gamma$ and $\mathcal{E} \sqsubseteq \mathcal{F}$, then $\mathcal{F} \models \Gamma$*

*Proof:* By induction on $\Gamma$. It is important that the atomic goals (of type $A \approx B$) possess the preservation property; our definition of observation equivalence ensures this. □

We will write $\langle \Gamma, \mathcal{E} \rangle$ for a tableau with goal $\Gamma$ and environment $\mathcal{E}$. A tableau $\langle \Gamma, \mathcal{E} \rangle$ is *satisfiable* if there exists an extension $\mathcal{F}$ of $\mathcal{E}$ that satisfies $\Gamma$.

A *transformation* is a binary relation on tableaux. We say that a transformation $\longrightarrow$ is *safe* if whenever $\langle \Gamma, \mathcal{E} \rangle \longrightarrow \langle \Gamma', \mathcal{E}' \rangle$, then $\mathcal{E} \sqsubseteq \mathcal{E}'$ and $\mathcal{E}' \models \Gamma' \supset \Gamma$. This means that safe transformations can only add to the environment, and only strengthen the goal. Thus, if the resulting tableau is satisfiable by a particular environment, then the original tableau is also satisfiable by the same environment. Note, however, that a safe transformation might transform a satisfiable tableau into an unsatisfiable one. An example of a safe transformation, by the expansion theorem in CCS:

$$\langle (a.A \parallel_{\{a\}} X) \approx B,\ \ \emptyset \rangle \longrightarrow \langle \tau.(A \parallel_{\{a\}} Y) \approx B,\ \ \langle X \mapsto \bar{a}.Y \rangle \rangle$$

We now prove that safe tableau transformations are indeed sound: if we start with a tableau and successively transform it until we arrive at the goal *true*, then we have derived an extension of the environment which satisfies the original goal.

**Proposition 2 (Soundness)** *Let $\longrightarrow^*$ be the reflexive transitive closure of the transformation $\longrightarrow$. If $\longrightarrow$ is safe, then for all tableaux $\langle \Gamma, \mathcal{E} \rangle$:*

$$\text{If } \langle \Gamma, \mathcal{E} \rangle \longrightarrow^* \langle \mathit{true}, \mathcal{F} \rangle \ \text{ then } \mathcal{E} \sqsubseteq \mathcal{F} \text{ and } \mathcal{F} \models \Gamma$$

*Proof:* If $\longrightarrow$ is safe, then $\longrightarrow^*$ is safe. This is proven by induction on the length of sequences of transformations; here the preservation property is crucial. Hence, by the definition of safe transformations we get that

$$\langle \Gamma, \mathcal{E} \rangle \longrightarrow^* \langle \mathit{true}, \mathcal{F} \rangle \ \text{ implies } \ \mathcal{E} \sqsubseteq \mathcal{F} \text{ and } \mathcal{F} \models \ \mathit{true} \supset \Gamma$$

But $\mathcal{F} \models \ \mathit{true} \supset \Gamma$ is easily seen to be true iff $\mathcal{F} \models \Gamma$ is true. □

The tableau framework is not limited to solving equations of type $A \parallel X \approx B$. Indeed, it can be applied to any equation in CCS, or even to any predicates which have the preservation property.

## 3.2 Finite Agents

We will present a tableau transformation for solving $A \parallel X \approx B$ where $A$ and $B$ are *finite state*, i.e. having only finitely many syntactically different derivatives. The initial tableau is $\langle A \parallel X \approx B, \quad \mathcal{E} \rangle$, where $\mathcal{E}$ only contains definitions of identifiers appearing in $A$ and $B$. We will pay special attention to the case where $B$ is deterministic, as it turns out that the transformation can then be formulated in a computationally simple way. To present our tableau transformation $\longrightarrow$ in a readable way, we first consider the case where $A$ and $B$ are *finite*, i.e. do not contain any recursively defined identifiers.

In general, a goal will be a conjunction of equations of type $E \approx C$, where $C$ is closed in the environment and $E$ is an agent containing free variables. There are two types of transformation rules: *instantiations* extend the environment by guessing the initial transitions of an unknown, and *reductions* strengthen the goal.

- *Instantiation*: If $\langle \Gamma, \mathcal{E} \rangle$ is a tableau and $X$ is free in $\mathcal{E}$, then

  $$X \mapsto \sum_{i=1}^{n} a_i.X_i$$

  can be added to $\mathcal{E}$. Here, $a_1, \ldots, a_n$ are actions and $X_1, \ldots, X_n$ are fresh (free in $\mathcal{E}$) and distinct identifiers.

- *Reduction*: $\langle \Gamma, \mathcal{E} \rangle \longrightarrow \langle \Gamma', \mathcal{E} \rangle$ if $\Gamma'$ is obtained from $\Gamma$ in one of the following ways:

- *Equivalence*: An equation $E \approx B$, where $E$ and $B$ are observation equivalent (w.r.t. $\mathcal{E}$) can be removed. Here, if $E \approx B$ is the only equation in the goal, then the goal becomes *true*.
- *Splitting*: If $\Gamma$ contains an equation $E \approx B$ where $E$ is well guarded, and the transitions from $E$ are $E \xrightarrow{e_j}_{\mathcal{E}} E_j$ for $j \in [1, \ldots, n]$, and there are agents $B_1, \ldots, B_n$ such that:

  $$\text{for all } j\text{: } B \overset{e_j}{\Longrightarrow}_{\mathcal{E}} B_j$$

  and also

  $$\mathcal{E} \quad \models \quad \left(\sum_{j=1}^{n} e_j.B_j\right) \approx B$$

  then the equation $E \approx B$ can be replaced by the equations

  $$\bigwedge_{j=1}^{n} (E_j \approx B_j)$$

  We here also allow the case $n = 0$ (i.e. $E$ has no transitions); then the only requirement is that $\mathcal{E} \models NIL \approx B$ and the resulting empty conjunction is simply *true*.

The rules deserve some comments. The instantiation transformation amounts to guessing the initial actions $a_1, \ldots, a_n$ of $X$. If applied carelessly an instantiation may result in an unsatisfiable tableau. Therefore, in section 4.2 we will provide heuristics for instantiations. The equivalence transformation will be applied sparingly, since it is computationally expensive to check observation equivalence.

The purpose of the splitting rule is to split an equation $E \approx B$ into a set of equations $E_j \approx B_j$, where all $E_j$ are derivatives of $E$ and all $B_j$ are derivatives of $B$. In essence this is the same as finding an "embryo" of a bisimulation between $E$ and $B$, by demonstrating how some derivatives of $E$ and $B$ should be related. As observed in [13], building a bisimulation in this way is easy if one of the agents $E$ and $B$ is deterministic. Indeed, when $B$ is deterministic, the requirements on $B_j$ can be simplified as demonstrated by the following proposition:

**Proposition 3** *Assume an environment $\mathcal{E}$ in which the agent $E$ is well guarded with initial transitions $E \xrightarrow{e_j} E_j$ for $j \in [1, \ldots, n]$, and in which $B$ is deterministic and closed with initial actions $\tilde{b} = \{b_1, \ldots, b_m\}$. Put $\tilde{e} = \{e_1, \ldots, e_n\}$. Then, the premises of the splitting rule*

$$\text{for all } j \in [1, \ldots, n]\text{: } B \overset{e_j}{\Longrightarrow}_{\mathcal{E}} B_j \tag{A1}$$

$$\mathcal{E} \quad \models \quad \left(\sum_{j=1}^{n} e_j.B_j\right) \approx B \tag{A2}$$

*are equivalent with*

$$\text{for all } j \in [1, \ldots, n]\text{: } \begin{cases} B = B_j & \text{if } e_j = \tau \\ B \xrightarrow{e_j}_{\mathcal{E}} B_j & \text{if } e_j \neq \tau \end{cases} \tag{B1}$$

$$\tilde{e} - \{\tau\} \subseteq \tilde{b}, \text{ and if } \tau \notin \tilde{e} \text{ then } \tilde{e} = \tilde{b} \tag{B2}$$

*Proof:*

(A1 $\Rightarrow$ B1) By A1, for each $j$ there are two cases:

1. $e_j = \tau$. Since $B \stackrel{\tau}{\Longrightarrow} B$ is always true for any $B$, and $B$ is deterministic, it follows $B = B_j$.
2. $e_j \neq \tau$. Then, by A1, $B \stackrel{\tau}{\rightarrow} \cdots \stackrel{\tau}{\rightarrow} B' \stackrel{e_j}{\rightarrow} B'' \stackrel{\tau}{\rightarrow} \cdots \stackrel{\tau}{\rightarrow} B_j$. Since $B$, and hence $B''$, are deterministic, it follows $B = B'$ and $B'' = B_j$, i.e. $B \stackrel{e_j}{\rightarrow} B_j$.

(A2 $\Rightarrow$ B2) Since $B$ is deterministic, it follows that $\tilde{b}$ is the set of observable experiments possible from $B$, i.e. $\tilde{b} = \{b \neq \tau : \text{for some } B'\text{: } B \stackrel{b}{\Longrightarrow} B'\}$. By A2, this implies $\tilde{e} - \{\tau\} \subseteq \tilde{b}$. Also, if $e_j \neq \tau$ for all $j$ then $\tilde{e}$ is the set of observable experiments possible from $\sum_j e_j.B_j$, whence by A2, $\tilde{e} = \tilde{b}$.

(B1 $\Rightarrow$ A1) Immediate.

(B1 and B2 $\Rightarrow$ A2) Let $I$ be the identity relation on agents which are closed in $\mathcal{E}$. It is straightforward to verify that under conditions B1 and B2, the relation

$$I \quad \cup \quad \left\{ \left( \sum_{j=1}^{n} e_j.B_j \ , \ B \right) \right\}$$

is a bisimulation, whence A2 follows. $\Box$

The important consequence of proposition 3 is the following:

**Proposition 4** *Assume an environment $\mathcal{E}$ and agents $E$ and $B$ as in proposition 3. Then there is at most one way to split the equation $E \approx B$.*

*Proof*: The result of splitting $E \approx B$ is uniquely determined by the agents $B_j$ in the splitting rule, and these are uniquely determined by condition B1 in proposition 3. $\Box$

Thus, to perform a splitting of $E \approx B$ when $B$ is deterministic, first compute (by the operational semantics in section 2) the transitions from $E$ and $B$. Then, if condition B2 holds, condition B1 uniquely gives the agents $B_j$. If B2 does not hold, no splitting transformation is applicable.

A simple example might be illuminating at this point: assume that we want to solve

$$(a.b.NIL|X)\backslash\{b\} \approx a.c.NIL$$

In the following, we write tableaux as boxes with goals to the left and environments to the right. For this particular example, we write $\|$ for $\|_{\{b\}}$. Hence, the original tableau is:

| | |
|---|---|
| $a.b.NIL \parallel X \approx a.c.NIL$ | $\emptyset$ |

Only an instantiation transformation is applicable here. By the heuristics (to be described in section 4.2), $X$ should have an initial $\bar{b}$ action to match the potential $b$ transition in $a.b.NIL$. Instantiating $X$ to $\bar{b}.Y$ gives

$$\begin{array}{|c|c|}\hline a.b.NIL \parallel X \approx a.c.NIL & X \mapsto \bar{b}.Y \\ \hline\end{array}$$

The left hand side of the equation is now well guarded, and the right hand side is deterministic. The only transition from the left hand side is

$$a.b.NIL \parallel X \xrightarrow{a} b.NIL \parallel X$$

We apply the splitting rule with $n = 1$, $e_1 = a$, and $E_1 = b.NIL \parallel X$. The only transition from the right hand side is

$$a.c.NIL \xrightarrow{a} c.NIL$$

Thus condition (B2) of proposition 3 is fulfilled, and condition (B1) gives $B_1 = c.NIL$. As a result of the splitting we get

$$\begin{array}{|c|c|}\hline b.NIL \parallel X \approx c.NIL & X \mapsto \bar{b}.Y \\ \hline\end{array}$$

The left hand side is still well guarded, and has the only transition

$$b.NIL \parallel X \xrightarrow{\tau} NIL \parallel Y$$

By another splitting:

$$\begin{array}{|c|c|}\hline NIL \parallel Y \approx c.NIL & X \mapsto \bar{b}.Y \\ \hline\end{array}$$

Of course, it is now easy to see that $Y = c.NIL$ will solve this goal. Following our method strictly, it is time to supply the initial actions of $Y$. By the heuristics, Y should have an initial $c$ to conform with the initial $c$ transition in the right hand side:

$$\begin{array}{|c|c|}\hline NIL \parallel Y \approx c.NIL & \begin{array}{c} X \mapsto \bar{b}.Y \\ Y \mapsto c.Z \end{array} \\ \hline\end{array}$$

By a splitting we get

$$\begin{array}{|c|c|}\hline NIL \parallel Z \approx NIL & \begin{array}{c} X \mapsto \bar{b}.Y \\ Y \mapsto c.Z \end{array} \\ \hline\end{array}$$

After instantiating $Z$ to $NIL$, and applying a final splitting transformation (with $n = 0$) we get

$$\begin{array}{|c|c|}\hline true & \begin{array}{c} X \mapsto \bar{b}.Y \\ Y \mapsto c.Z \\ Z \mapsto NIL \end{array} \\ \hline\end{array}$$

We now have the goal *true* and the environment contains the desired solution, which can be written $\bar{b}.c.NIL$.

There seems to be a fair amount of tedium in applying the tableau method even to simple problems. Our point is that this tedium can be automated. Indeed, the program described in section 4.3 will do the above steps automatically.

To demonstrate the soundness of this tableau method (i.e. that when the goal *true* is reached, the environment will satisfy the original goal), we prove that $\longrightarrow$ is safe, and appeal to proposition 2.

**Proposition 5** $\longrightarrow$ *is safe.*

*Proof:* For the instantiation and equivalence transformations this is obvious. For the splitting transformation, if $E \xrightarrow{e_j} E_j$ are the transitions from $E$ and $E$ is well guarded, then $E \approx \sum_j e_j.E_j$. Also, by congruence properties of $\approx$,

$$\bigwedge_{j=1}^{n} (E_j \approx B_j) \quad \text{implies} \quad \left(\sum_{j=1}^{n} e_j.E_j\right) \approx \left(\sum_{j=1}^{n} e_j.B_j\right)$$

Thus, from $\bigwedge_j E_j \approx B_j$ and the premises of the splitting rule we infer $E \approx B$, i.e. the goal is made stronger by replacing $E \approx B$ with $\bigwedge_j E_j \approx B_j$. $\square$

For finite agents, $\longrightarrow$ is complete in the following sense: starting with any satisfiable equation $A \parallel X \approx B$ where $A$ and $B$ are finite, the goal *true* can eventually be reached.

**Proposition 6 (Completeness)** *Let $\mathcal{E}$ be an environment where $X$ is free and $A$ and $B$ are finite agents. Assume that the goal $\Gamma = A \parallel X \approx B$ is satisfiable in $\mathcal{E}$. Then there exists an extension $\mathcal{F}$ of $\mathcal{E}$ such that*

$$\langle \Gamma, \mathcal{E} \rangle \longrightarrow^* \langle true, \mathcal{F} \rangle$$

*Proof sketch:* Since $A$ and $B$ are finite and $A \parallel X \approx B$ is satisfiable, it is satisfiable by an environment which maps $X$ to a finite agent. This environment can be obtained by a finite sequence of instantiations, since any finite agent can be expressed (up to equivalence) using only finitely many guarded summations. This sequence of instantiations followed by the removal of $A \parallel X \approx B$ according the equivalence rule results in the goal *true*. $\square$

The completeness result ensures us that the transformation presented in this section is in principle sufficient for solving any equation. However, the above proof is of no computational value, since knowledge of the solution is required for choosing the correct sequence of instantiations.

## 3.3 Finite State Agents

Obviously, with the transformations presented so far it is impossible to generate environments with recursively defined identifiers. The following extension of the instantiation transformation will amend this situation:

- *Instantiation by identification*: If $\langle \Gamma, \mathcal{E} \rangle$ is a tableau and $X$ is free in $\mathcal{E}$, and $Y$ is an identifier bound by $\mathcal{E}$, then

  $$\langle \Gamma, \mathcal{E} \rangle \longrightarrow \langle \Gamma[Y/X], \quad \mathcal{E}[Y/X] \rangle$$

  I.e., $X$ may be uniformly replaced by $Y$ in the tableau.

Note that replacing $X$ with $Y$ has the same effect as adding $X \mapsto Y$ to the environment. Thus, the tableau transformation allowing identifications is still safe, and hence sound. In fact, any transformation of the form

$$\langle \Gamma, \mathcal{E} \rangle \longrightarrow \langle \Gamma[E/X], \;\; \mathcal{E}[E/X] \rangle$$

(where $E$ is an arbitrary agent) is sound by the same argument, but we will only need the simpler form.

A simple example will illustrate how the identification transformation is used. Let $A \mapsto a.A$ and $B \mapsto a.B + b.B$ be an environment and assume we want to find an $X$ satisfying $A|X \approx B$. The initial tableau is (for convenience, we do not show $A \mapsto a.A$ and $B \mapsto a.B + b.B$; these are tacitly present in the environment):

| | |
|---|---|
| $A\|X \approx B$ | $\emptyset$ |

According to the heuristics, $X$ should have an initial $b$ action to conform with the initial $b$ in $B$. By instantiating $X$ to $b.Y$ we get

| | |
|---|---|
| $A\|X \approx B$ | $X \mapsto b.Y$ |

Now the left hand side is well guarded. Both sides of the equation can do $a$ and $b$ transitions; after a splitting we get

| | |
|---|---|
| $A\|X \approx B$<br>$A\|Y \approx B$ | $X \mapsto b.Y$ |

The first equation in the goal is identical with the original equation. In the second equation $Y$ is free, and should be instantiated. However, the equations containing $X$ and $Y$ are similar: a solution for $X$ will always also be a solution for $Y$. This suggests that $Y$ could successfully be identified with $X$. Identifying $Y$ with $X$ yields:

| | |
|---|---|
| $A\|X \approx B$<br>$A\|X \approx B$ | $X \mapsto b.X$ |

There are now no free identifiers in the tableau. The goal contains two (identical) equations, these are true in the environment and can be removed. Hence, "$X \mapsto b.X$" is the desired solution.

The transformation allowing identifications can be proven complete in the same way as proposition 6: the key fact is that guarded summation and recursion is sufficient to express any finite state agent up to equivalence (as proven in [12]). Again, this proof is of no computational value, and we will not go into it.

# 4 Implementing the Tableau Method

In this section we will describe a program which implements the tableau transformations in a semiautomatic way. The main obstacle to such an implementation is that there are in general several applicable transformations from a given tableau: any equation with a well guarded left hand side may be split, and any free identifier may be instantiated. We will therefore first investigate principles for choosing appropriate transformations.

We restrict attention to equations $A \parallel X \approx B$ where the right hand side $B$ is deterministic; this limits the scope of the method but facilitates automation. In particular, we will prove that under this assumption all splitting transformations are harmless in the sense that they preserve satisfiability, and that there is an essentially unique way to repeatedly perform splitting transformations. We will also prove that if an equation has a solution, then it has a deterministic solution. This facilitates the instantiations: when instantiating $X$ with $\sum a_i.X_i$, we may assume that all $a_i$ are distinct and that none of them is $\tau$. We will provide some heuristics for the instantiations, and conclude this section with a description of the program.

For convenience, we will assume that $\mathcal{E}(X)$ is always an agent of type $\sum_j a_j.X_j$, i.e. a sum of prefixed identifiers. Since we only consider finite state agents this implies no loss of generality. We can then prove the following:

**Proposition 7** *Assume* $\langle A \parallel X \approx B, \quad \mathcal{E}\rangle \longrightarrow^* \langle \Gamma, \mathcal{F}\rangle$ *where* $X$ *is free in* $\mathcal{E}$*. Then* $\Gamma$ *is a conjunction of equations* $A_1 \parallel X_1 \approx B_1, \ldots, A_n \parallel X_n \approx B_n$*, where each* $A_j$ *is a derivative of* $A$*, each* $B_j$ *is a derivative of* $B$*, and each* $X_j$ *is a derivative (w.r.t.* $\mathcal{F}$*) of* $X$*.*

*Proof*: Through induction over the transformation rules. $\square$

## 4.1 Splitting Transformations

As has been noted in section 3.2, instantiations are dangerous since they might result in unsatisfiable tableaux. We will prove that splittings are harmless in this sense. We first establish that an equation which may be split can always be treated by a splitting transformation which preserves satisfiability:

**Proposition 8** *Let* $\langle \Gamma, \mathcal{E}\rangle$ *be satisfiable by* $\mathcal{F}$*, i.e.* $\mathcal{F}$ *is an extension of* $\mathcal{E}$ *such that* $\mathcal{F} \models \Gamma$*. Let* $E \approx B$ *be an equation in* $\Gamma$*, where* $E$ *is well guarded and* $B$ *is closed in* $\mathcal{E}$*, and the initial transitions from* $E$ *are* $E \xrightarrow{e_j} E_j$ *for* $j \in [1, \ldots, n]$*. Then there exist agents* $B_1, \ldots, B_n$ *which satisfy the premises of the splitting rule, i.e:*

$$\text{for all } j\text{: } B \stackrel{e_j}{\Longrightarrow}_{\mathcal{E}} B_j$$

*and*

$$\mathcal{E} \quad \models \quad \left(\sum_{j=1}^{n} e_j.B_j\right) \approx B$$

*and such that the splitting rule preserves satisfiability by* $\mathcal{F}$*, i.e:*

$$\mathcal{F} \models \bigwedge_{j=1}^{n} (E_j \approx B_j)$$

*Proof:* In all extensions of $\mathcal{E}$ we have that $E \stackrel{e_j}{\Longrightarrow} E_j$; in particular this must hold in $\mathcal{F}$. By the premises, $E \approx B$ is true in $\mathcal{F}$. Thus, for all $j = 1, \ldots, n$ the following diagram (where we write $\Updownarrow$ for $\approx$ in environment $\mathcal{F}$)

$$\begin{array}{lcl} E & \stackrel{e_j}{\Longrightarrow}_{\mathcal{F}} & E_j \\ \Updownarrow & & \\ B & & \end{array}$$

can be completed with agents $B_j$ to a commuting diagram

$$\begin{array}{lcl} E & \stackrel{e_j}{\Longrightarrow}_{\mathcal{F}} & E_j \\ \Updownarrow & & \Updownarrow \\ B & \stackrel{e_j}{\Longrightarrow}_{\mathcal{F}} & B_j \end{array}$$

This implies that

$$\mathcal{F} \models \bigwedge_{j=1}^{n} (E_j \approx B_j)$$

Since $B$ is closed in $\mathcal{E}$, $B \stackrel{e_j}{\Longrightarrow}_{\mathcal{F}} B_j$ implies $B \stackrel{e_j}{\Longrightarrow}_{\mathcal{E}} B_j$. By congruence properties of $\approx$ with respect to guarded sum, we get (w.r.t. $\mathcal{F}$):

$$\sum_{j=1}^{n} e_j.B_j \approx \sum_{j=1}^{n} e_j.E_j \approx E \approx B$$

However, since $B$ and all $B_j$ are closed under $\mathcal{E}$, this implies that w.r.t. $\mathcal{E}$:

$$\sum_{j=1}^{n} e_j.B_j \approx B$$

□

An interesting consequence of proposition 8 is that if an equation is in a form suitable for splitting (i.e. the left hand side is well guarded) but no splitting transformation applies to it (condition B2 in proposition 3 does not hold), then the equation is not satisfiable. This is a useful way to detect unsatisfiable goals. Another immediate consequence is:

**Proposition 9** *All splitting transformations preserve satisfiability.*

*Proof*: Directly from propositions 4 and 8. □

We will next prove that there is an essentially unique way to apply splitting transformations until no more new equations are generated. Let $\longrightarrow_s$ be the subset of $\longrightarrow$ corresponding to splitting transformations on tableaux. We will consider finite transformation sequences $\sigma$ consisting only of splittings:

$$\sigma = \langle \Gamma_1, \mathcal{E}_1 \rangle \longrightarrow_s \cdots \longrightarrow_s \langle \Gamma_i, \mathcal{E}_i \rangle$$

Define $Eq(\sigma)$ to be the set of equations in the goals in $\sigma$. Say that $\sigma$ is *splitting complete* if all equations with a well guarded left hand side in $Eq(\sigma)$ have been treated by a splitting transformation in $\sigma$. This implies that no new equations can be obtained by a further splitting from the final tableau, and that an instantiation is necessary in order to make progress.

**Proposition 10** *Any transformation sequence can be extended to a splitting complete sequence.*

*Proof*: Extend $\sigma$ by repeatedly applying splitting transformations on all equations which have not previously been treated by splittings. Continue this until no new equations are generated; this must eventually happen since all equations must be formed from the derivatives of the agents in the original equation (proposition 7), and there are only finitely many such derivatives. □

We additionally prove that two splitting complete extensions of a given transformation sequence cannot differ in any significant way. We already know (from proposition 8) that splittings do not affect satisfiability; in fact the final tableaux of two splitting complete sequences are satisfiable by exactly the same environments. We will demonstrate in section 4.2 that the equations with free identifiers on the left hand side are the only equations which affect the heuristics for instantiations. Thus, we say that two transformation sequences are *instantiation equivalent* if they have the same set of equations with free identifiers on the left hand side in the last tableau. The following proposition means that all splitting complete extensions of a given sequence are equivalent in this sense; hence the extension to a splitting complete sequence requires no particular insights.

**Proposition 11** *Let $\sigma'$ and $\sigma''$ be splitting complete extensions of a given transformation sequence $\sigma$. Then $\sigma'$ and $\sigma''$ are instantiation equivalent.*

*Proof*: We first prove that the premises of the proposition imply $Eq(\sigma') = Eq(\sigma'')$. Define the equivalence relation $\simeq$ on transformation sequences by $\sigma \simeq \sigma'$ if $Eq(\sigma) = Eq(\sigma')$ and the last goal in $\sigma$ is also the last goal in $\sigma'$. Define $\prec$ on transformation sequences by $\sigma \prec \sigma'$ if $\sigma'$ is obtained by extending $\sigma$ with exactly one splitting transformation. Since the splitting of any particular equation is unique and only affects one equation in the goal it is not difficult to show that $\prec$ is confluent in the following sense: if $\sigma_1 \simeq \sigma_2$ and $\sigma_1 \prec \sigma_1'$ and $\sigma_2 \prec \sigma_2'$, then there exist $\sigma_1^* \simeq \sigma_2^*$ such that $\sigma_1' \prec \sigma_1^*$ and $\sigma_2' \prec \sigma_2^*$. It is then standard to establish that the reflexive transitive closure of $\prec$ is confluent in the same sense. Thus, if $\sigma'$ and $\sigma''$ both extend $\sigma$, then we can find $\sigma^*$ extending $\sigma'$ and $\sigma^{**}$ extending $\sigma''$ such that $\sigma^* \simeq \sigma^{**}$. If $\sigma'$ and $\sigma''$ are splitting complete it follows $Eq(\sigma') = Eq(\sigma^*)$ and $Eq(\sigma'') = Eq(\sigma^{**})$, which implies $Eq(\sigma') = Eq(\sigma'')$.

The proposition now follows from the observation that an equation with a free identifier on the left hand side may not be split (its left hand side is not well guarded), and hence any such equation in $Eq(\sigma)$ must remain in the last goal of $\sigma$. □

## 4.2 Instantiations

We will proceed with some results concerning instantiation transformations. The first result says that if an equation has a solution then it has a deterministic solution. This means that when instantiating $X$ with $\sum a_i.X_i$, we may assume that all $a_i$ are distinct and that none of them is $\tau$.

**Proposition 12** *If $A \parallel X \approx B$ has a solution and $B$ is deterministic, then it has a deterministic solution.*

*Proof*: See the appendix. □

In the following we will assume that deterministic solutions are always preferable to non-deterministic ones, and restrict attention to generation of deterministic solutions. Thus, when instantiating an identifier $X$ to $\sum_{i=1}^{n} a_i.X_i$ it suffices to know the set $\{a_1, \ldots, a_n\}$ of initial actions of $X$. Unfortunately there are in general several different such sets corresponding to different solutions for $X$. We will therefore be satisfied with some heuristics for choosing this set.

One such heuristic is to avoid the actions which could not possibly be initial actions of $X$. Formally, we say that an action $a$ is *inadmissible* for $X$ in the equation $A \parallel X \approx B$ if the following holds: if $X$ were instantiated with an initial action $a$, then the goal would become unsatisfiable. Inadmissibility is in general hard to check, but it can be approximated by $Tr_k$-inadmissibility as follows: for an agent $A$, let $Tr_k(A)$ be the set of traces (transition sequences where $\tau$ transitions have been deleted) of length $\leq k$. An action $a$ is $Tr_k$-inadmissible for $X$ if $Tr_k(A \parallel a.NIL) \not\subseteq Tr_k(B)$. This means that if $X$ were instantiated with an initial action $a$, then there would be traces (of length $\leq k$) of $A \parallel X$ which are not traces of $B$. For every $k$, $Tr_k$-inadmissibility implies inadmissibility. As $k$ increases, $Tr_k$-inadmissibility becomes a more accurate approximation of inadmissibility (and of course more expensive to compute).

Another heuristic is to avoid actions which will never be used in any solution for $X$. Formally, an action $a$ is *useful* for $X$ in the equation $A \parallel_L X \approx B$, if there is a transition of a derivative of $A \parallel_L X$ which depends on the fact that $X$ can do an initial $a$ transition. The useful actions can be computed as follows: say that $a$ is *covered* by $L$ if $a \in L$ or $\bar{a} \in L$. Then, all actions not covered by the restriction $L$ are useful for $X$. Furthermore, an action $a$ is useful for $X$ if $A$ can perform a transition sequence, containing $\bar{a}$ but no other action covered by $L$. In this case an initial $a$ in $X$ can result in a synchronisation with this $\bar{a}$ in $A$.

As an example of these concepts, consider the equation (from section 3.2):

$$a.b.NIL \parallel_{\{b\}} X \approx a.c.NIL$$

Here, the action $\bar{b}$ is useful for $X$ (it can result in a synchronisation with $b$). Also, $\bar{b}$ is admissible. The action $c$ is useful, but not admissible — in fact, it is even $Tr_1$-inadmissible.

In general, a goal may contain several equations with the same free identifier $X$. When instantiating this $X$ each initial action should be admissible for all equations and useful for at least one equation. Thus, each equation where $X$ occurs contributes information for the instantiation of $X$.

Finally, we need a heuristic for instantiation by identification. Say that a bound identifier $Y$ is *adequate* for a free identifier $X$ if the equations containing $X$ constitute a subset of the equations containing $Y$, and the admissible and useful actions of $X$ agree with the initial actions of $Y$. The intuition is that this is a strong indication that $X$ could successfully be identified with $Y$: a solution for $Y$ will always also be a solution for $X$.

### 4.3 An Implementation

Our program for solving equations with the tableau method works in the following way: first, the user enters the equation $A \parallel_L X \approx B$ that he wants to solve. $A$ and $B$ must be finite state, and $B$ must be deterministic. Also, the expected sort of the solution must be given (alternatively, the program will guess an expected sort). Thereafter, a semiautomatic procedure will start.

The procedure works as follows: a free identifier (in the beginning there is only one, namely $X$) is chosen for instantiation, and the program computes its admissible and useful actions and adequate identifiers, taking all equations in all goals into account. This information is presented to the user, who decides on the proper instantiation. Then the program proceeds by automatically performing a splitting complete transformation sequence. As demonstrated in propositions 10 and 11, this is always possible, and the resulting set of equations is independent of the particular sequence. Following a splitting complete sequence an instantiation is unavoidable in order to make progress, and the user is consulted again in the same way. The procedure repeats until no more free identifiers remain in the goal. Then the goal contains only closed equations, and these are checked automatically by the program. If each equation is true then a solution has been generated.

During this procedure the program might discover that the goal is unsatisfiable e.g. by finding it impossible to perform a splitting transformation (proposition 8), or by finding a closed equation which is false. The program then backtracks automatically to the previous instantiation. Also, the user may backtrack at will in order to explore different possibilities. Alternatively, he can run the program in an automatic mode, where all instantiations are made according to the *maximal* strategy:

1. If there is at least one adequate bound identifier, then identify with one of them.

2. If there are no adequate bound identifiers, then instantiate with the set of *all* admissible and useful actions.

The strategy is called "maximal" because the solutions will in general be agents that have maximal freedom: if more transitions are added, then they would either cause inadmissible behaviour, or would never be exercised. Maximality might, or might not, be a sensible criterion for good solutions. For most small examples, such as those presented in this paper (excepting section 5), the strategy produces the expected solutions automatically. It should be noted that the maximal strategy is not complete for satisfiable goals: it sometimes results in a diverging sequence of choices.

A simple example will illustrate the program. Assume that we seek an agent $X$, which in parallel with a buffer of capacity one yields a buffer of capacity two. A buffer of capacity one on channels $a$ and $b$ is defined by

$$A \mapsto a.b.A$$

and a buffer of capacity two on channels $a$ and $c$ is

$$\begin{cases} B \mapsto a.B' \\ B' \mapsto c.B + a.c.B' \end{cases}$$

The user also has to supply the restriction $L$ for solving $A \parallel_L X \approx B$, in this case the restriction is $\{b\}$. The program infers a sort for the solution, in this case it is $\{\bar{b}, c\}$, and the user acknowledges this.

Now the tableau method begins. We will here display the tableaux where the user needs to interact. To the left are the equations which contain the free identifier under consideration (in general the goal also contains other equations that are not immediately relevant for this identifier; these equations are not shown), in the middle is the solution generated so far, and to the right are the heuristics computed by the program. The first tableau is the initial one:

| | | |
|---|---|---|
| $A \parallel X \approx B$ | | No adequate bound identifiers<br>Admissible and useful actions: $\{\bar{b}\}$ |

The user is presented with a menu of various alternatives; in this case he chooses to instantiate $X$ according to the heuristics. The next interaction takes place at the following tableau:

| | | |
|---|---|---|
| $A \parallel X_1 \approx B'$ | $X \mapsto \bar{b}.X_1$ | No adequate bound identifiers<br>Admissible and useful actions: $\{c\}$ |

Again, following the heuristics leads to the following situation:

| | | |
|---|---|---|
| $A \parallel X_2 \approx B$<br>$b.A \parallel X_2 \approx B'$ | $X \mapsto \bar{b}.X_1$<br>$X_1 \mapsto c.X_2$ | Adequate bound identifiers: $X$<br>Admissible and useful actions: $\{\bar{b}\}$ |

Now the user decides to identify $X_2$ with $X$. The program then discovers that there are no more free identifiers, and proceeds to check the environment against the goal[1]. In this case, the environment satisfies the goal, and the program reports the solution to the user:

$$\begin{cases} X \mapsto \bar{b}.X_1 \\ X_1 \mapsto c.X \end{cases}$$

The solution can be written $X \mapsto \bar{b}.c.X$, i.e. it defines as expected a buffer of capacity one.

# 5 The Alternating Bit Protocol

In this section we study the effects of applying the the maximal strategy of the program to a nontrivial example: the Alternating Bit protocol.

The purpose of the Alternating Bit protocol (originally presented in [2]) is to provide reliable data transmission over an imperfect medium. Figure 1 shows the general structure of the protocol. It consists of three modules: a sender, a medium and a receiver. There are several versions of this protocol; we will begin by studying the protocol as presented in [10]. There, the medium can corrupt but not lose messages. A message is delivered to the sender through the primitive *put*, and accepted from the receiver through the primitive *use*. The service of the protocol is that of a perfect one place buffer, i.e. *put* and *use* alternate:

$$Service \mapsto put.use.Service$$

Figure 2 depicts state transition diagrams for the modules in the protocol. We will in this section consistently use such diagrams to represent agents; the transformation between diagrams and a system of recursive agent identifier definitions is trivial.

The protocol works as follows. The sender adds a one bit sequence number to an incoming message (starting with 0 for the first message) and transmits it to the medium. We will not explicitly represent message contents, but the sequence numbers are important for the synchronisation properties of the protocol. Thus, we use $\underline{d_0}$ to represent transmission of messages with

[1] Actually, it checks against the original goal rather than the current goal; this has turned out to be more efficient in practice.

Notation:

| | |
|---|---|
| *put* / *use* | submitting/receiving a message to/from protocol |
| $\underline{d_i}$ / $d_i$ | transmitting/receiving a message to/from medium |
| $\underline{a_i}$ / $a_i$ | transmitting/receiving an acknowledgment to/from medium |
| $d_e$ / $a_e$ | receiving corrupt message/acknowledgment from medium |

Figure 1: The structure of the modules in the Alternating Bit protocol.

Figure 2: The modules in the Alternating Bit protocol. $(A)$ The Sender. $(B)$ The Receiver. $(C)$ The Medium.

sequence number 0, and $\underline{d_1}$ for messages with sequence number 1. Following a transmission, the sender awaits an acknowledgment (actions $a_0$ and $a_1$) with the same sequence number. After reception of the correct acknowledgment, the procedure is repeated: a new message can be accepted for transmission. This time the sequence number is inverted. If the sender receives an acknowledgment with the wrong sequence number, or a corrupt acknowledgment (action $a_e$), then it retransmits the last message.

The receiver acknowledges all messages $(d_0, d_1)$ by transmitting an acknowledgment with the same sequence number as the message $(\underline{a_0}, \underline{a_1})$. If the sequence number differs from the preceding one, then the message is not a retransmission, and is delivered to the user through the primitive *use*. If a corrupt message arrives $(d_e)$, then the last acknowledgment is retransmitted.

The medium can contain at most one message or acknowledgment at a time, i.e. it is half duplex. Thus, following an action $\overline{\underline{d_i}}$ (the inverse of $\underline{d_i}$), it either delivers the message through $\overline{d_i}$ or delivers a corrupt message through $\overline{d_e}$. Similarly, acknowledgments may be corrupted.

Our model of the protocol differs from that in [10] in one respect: in the states where acknowledgments are not expected, the sender may accept and discard spurious acknowledgments. A similar modification of the receiver could also be made without affecting the service of the protocol.

The program for equation solving can be used to generated any one unknown module of the protocol. As an example, we have generated the receiver by solving the equation

$$(Sender|Medium) \parallel X \approx Service$$

Here, $\parallel$ means parallel composition and restriction over the internal actions ($\underline{d_i}$ and $\underline{a_i}$ for $i = 0$ and 1; $d_i$ and $a_i$ for $i = 0$, 1, and $e$). When applying the maximal strategy to solve the equation, the result is the rather surprising receiver in figure 3.

This receiver is a most general receiver in the sense that from any state, additional actions will either never be exercised or will lead to inadmissible behaviour of the protocol. It is clear that it is much more general than the expected solution in figure 2. For example, in the initial state, the receiver may begin by transmitting any sequence of acknowledgments. Of course, in a real implementation this would be ridiculous. Nevertheless, the receiver satisfies the formal problem. Indeed, with the medium being half duplex and of capacity one element, these extra acknowledgments are harmless: when the receiver has not accepted any message and it transmits an acknowledgment, then no message has been sent, and hence the sender is in a state where it discards the incoming acknowledgments. There are other similar paradoxical aspects of the behaviour of this receiver. Note, however, that the expected solution is contained as a subgraph (highlighted transitions). We take this example as a good illustration of our point: a completely automatic procedure for generating submodule behaviours is not always desirable.

A variation on the Alternating Bit protocol is to use a full-duplex medium, with capacity one element in each direction, and ability to lose messages. For simplicity, we assume that messages are either lost or delivered intact (this is a realistic assumption; there could be an error detection mechanism that discards all corrupt messages). The medium is modelled as the parallel composition of two independent simplex media, where $\tau$ actions correspond to message loss. The sender and receiver modules are modified by deleting all transitions dealing with corrupt messages ($d_e$ and $a_e$), and by adding timeout transitions to the sender ($\tau$ transitions

Figure 3: A most general receiver.

leading from states where the sender waits for acknowledgments to states where it can do retransmissions). These modules are shown in figure 4.

Again, using the maximal strategy to solve the equation where the receiver is unknown, yields a most general solution as shown in figure 5. This solution does not depart very much from the expected solution. The initial state is unreachable from the other states (in the initial state there is no useful $d_1$ action), and in all states it is harmless to retransmit the last acknowledgment or accept a duplicate of the last accepted message.

In a similar way, and with similar results, the sender of the protocol can be generated when the receiver is known. It is even possible to generate a medium if both sender and receiver are known. Naturally, it is unlikely that the medium is unknown in a real protocol design project. Instead, this result indicates the worst possible conditions under which the protocol will work. In the Alternating Bit protocol it turns out that the medium may not only lose messages, but also generate spurious messages in certain situations, without harming the protocol.

# 6 Conclusions and Comparisons with Related Work

We have in this paper indicated one way to give meaning to CCS equations of type $(A|X)\backslash L \approx B$, and presented a method for solving such equations. The method is based on a general tableau framework where a solution is derived via a sequence of transformations. These transformations form a basis for an implementation, by which we have generated the receiver of different versions of the Alternating Bit protocol.

Figure 4: Second version of the modules in the Alternating Bit protocol. ($A$) The Sender. ($B$) The Receiver. ($C, D$) Simplex media. The Medium in the protocol consists of these two media in parallel.

Figure 5: A most general receiver of the second version of the protocol.

Typically, an equation $(A|X)\backslash L \approx B$ has many solutions. We do not know if there always is exactly one "most general" solution (i.e. a solution which simulates every other solution). Our implementation attempts to find such a solution, but even if it exists it is not certain that it is suitable for implementation. It might be expected that "least general" solutions, i.e. solutions which do not contain unnecessary nondeterministic choices, are preferable. Unfortunately there are often many such least general solutions, and it is not clear how a "good" solution could be generated in a general way.

Our conclusion is that a completely automatic procedure for solving equations is not always desirable. When generating a solution, some criteria for what constitutes a good solution must be used. Obviously, such criteria are dependent on the particular equation to be solved. With our method, the program performs some of the transformations automatically, but a user can effect critical transformations in order to guide the program towards a suitable solution.

One interesting way to extend this work is to consider a larger class of equations. Already, the method is sufficiently powerful to treat several equations simultaneously. Hence, it can be used to solve problems as "find an $X$ such that $A_1 \parallel X \approx B_1$ and also $A_2 \parallel X \approx B_2$". Similarly, the method can in principle handle equations of the more general form $\mathcal{C}(\widetilde{X}) \approx B$, where $\mathcal{C}$ is any CCS context. However, our program and heuristics for determining admissible and useful actions only apply for the class of equations considered in this paper.

Another way to extend the scope of this method is to consider other operators. For example, in TCSP ([4]) there are other types of parallel operators and other types of nondeterministic choices. We conjecture that the tableau method would work well also in these systems. A congruence property of guarded sum would be sufficient for a sound splitting transformation.

It would be exciting to extend our method to include communication with value passing. The tableau transformation is easily extended by including events with value parameters and parametrised identifiers in the instantiations. The difficulty would be to provide good heuristics for choosing the value expressions in the output events.

Since there is a vast literature on generating modules of complex systems, we will here only comment on some approaches related to our method. To our knowledge, the only work on solving CCS equations is [15] and [14]. There, equations of type $(A|X)\backslash L \approx B$ are called "interface equations". For the case where $B$ is deterministic, and under some requirements (not very restrictive) on the sorts of $A$ and $B$, necessary and sufficient conditions for the existence of solutions of such equations are given. In the case where there exist solutions, an explicit construction of a solution is presented. This construction, and also the requirements for existence of a solution, are formulated in terms of the state spaces of $A$ and $B$. There is, however, no indication that this method can be used interactively and guided towards solutions which are suitable for implementation.

We have already mentioned the work in [10]. There, a similar problem is considered with finite automata instead of agents, and with trace equivalence instead of observation equivalence. Another difference is that according to the definition of parallel composition the simultaneous execution of two actions does not always result in an unobservable action. Within this formalism, the authors derive a solution in terms of the "complement" operation on automata (the complement of an automaton $A$ accepts the complement of the language accepted by $A$). They apply this method to generate the receiver of the Alternating Bit protocol, and remark that the most general solution is not always the best one. Their suggested remedy is to start by generating a most general solution, and proceed by deleting states and transitions which are unnecessary (i.e. can be deleted while preserving trace equivalence of the system). Also, they remark that trace equivalence is not sufficient to demonstrate properties like deadlock freedom.

The recent [3] goes one step further. There, the authors present a method to automatically partition an overall system behaviour $B$ into submodules $A_1, \ldots, A_n$. These submodules, when composed in parallel, yield a behaviour which is trace equivalent with $B$. The idea is to partition the set of actions in $B$ into different locations, and generate one module $A_i$ for each location. The method assumes that the modules communicate over unbounded perfect channels.

In [16], a semiautomatic procedure is given on how to complete partly specified modules into a system which will be guaranteed to be free of certain unwanted properties such as deadlocks. In [6], an algorithm is presented for generating one module of a protocol when a second module is given. However, in neither of these is there any formal specification of the expected service of the combined system. Algorithms for synthesis of concurrent programs from service specifications in temporal logic are presented in [5] and [9]. A new direction is taken in [1]. There, specifications are formulated in knowledge logic (where assertions can be of type "module $A$ knows the contents of message $m$").

Our method is based on transformation of tableaux. The main inspiration for this is [8], where (sequential) functional programs are generated in a similar way by transforming predicate logic formulas. Later, this idea was extended to synthesis of asynchronously communicating networks ([7]). The approach is to first generate one single module, defined as a functional program, and subsequently transform this module into several modules working in parallel. This transformation is specifically aimed at generating dataflow networks.

# Acknowledgments

I am grateful to Bengt Jonsson, Robin Milner, Colin Stirling, and David Walker for constructive discussions and for a critical reading of the manuscript. I also wish to thank an anonymus referee for many helpful comments.

# References

[1] Foto Afrati, Christos Papadimitriou, and George Papageorgiou. The synthesis of communication protocols. In *Proceedings of the fifth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 263–271, 1986.

[2] K Bartlett, R Scantlebury, and P Wilkinson. A note on reliable full-duplex transmissions over half duplex lines. *Communications of the ACM*, 2(5):260–261, 1969.

[3] Gregor von Bochmann and Reinhard Gotzhein. Deriving protocol specifications from service specifications. In *Proceedings of the ACM SIGCOM Symposium*, pages 148–156, 1986.

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

[5] E. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[6] Mohamed Gouda and Yao-Tin Yu. Synthesis of communicating finite-state machines with guaranteed progress. *IEEE Transactions on Communications*, COM-32(7):779–788, 1984.

[7] Bengt Jonsson, Zohar Manna, and Richard Waldinger. Towards deductive synthesis of dataflow networks. In *Proceedings of Symposium on Logic in Computer Science*, pages 26–37, 1986.

[8] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1), 1980.

[9] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.

[10] Philip Merlin and Gregor von Bochmann. On the construction of submodule specifications and communication protocols. *ACM Transactions on Programming Languages and Systems*, 5(1):1–25, 1983.

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

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

[13] Michael Sanderson. *Proof Techniques for CCS*. PhD thesis, University of Edinburgh, 1982. CST-19-82.

[14] M W Shields. *Extending the Interface Equation*. Technical Report SE/079/3, Electronic Engineering Laboratories, University of Kent at Canterbury, August 1986.

[15] M W Shields. *Solving the Interface Equation*. Technical Report SE/079/2, Electronic Engineering Laboratories, University of Kent at Canterbury, July 1986.

[16] P Zafiropulo, C H West, H Rudin, D D Cowan, and D Brand. Towards analyzing and synthesizing protocols. *IEEE Transactions on Communications*, COM-28(4):651–661, 1980.

# Appendix

In this appendix we prove proposition 12: every satisfiable equation has a deterministic solution. The proof idea is to start with any solution and remove each instance of nondeterminism. We first prove a lemma saying that the solutions of an equation are closed under summation:

**Lemma 13** *If $A \parallel X \approx B$ and $A \parallel Y \approx B$ and $B$ is deterministic then $A \parallel (X+Y) \approx B$.*

*Proof*: Define the relation $R$ by $CRD$ if $D$ is deterministic and for some $A$ it holds $C = A \parallel (X+Y)$ and $A \parallel X \approx D$ and $A \parallel Y \approx D$. We will prove that $R\cup \approx$ is a bisimulation.

We first prove that $R\cup \approx$ is a simulation: assume $C(R\cup \approx)D$ and $C \stackrel{a}{\Longrightarrow} C'$. The case $C \approx D$ is trivial, so assume $CRD$. Then for some $A$ we must have $C = A \parallel (X+Y)$ and $A \parallel X \approx D$ and $A \parallel Y \approx D$. There are three cases:

1. The inference of $C \stackrel{a}{\Longrightarrow} C'$ uses a transition from $X$. With essentially the same inference we then get $A \parallel X \stackrel{a}{\Longrightarrow} C'$, whence by $A \parallel X \approx D$ it follows $D \stackrel{a}{\Longrightarrow} D'$ with $C' \approx D'$ as required.

2. The inference of $C \stackrel{a}{\Longrightarrow} C'$ uses a transition from $Y$. This case is symmetric with case 1 above.

3. $C' = A' \parallel (X+Y)$ and $A \stackrel{a}{\Longrightarrow} A'$, and both $A \parallel X \stackrel{a}{\Longrightarrow} A' \parallel X$ and $A \parallel Y \stackrel{a}{\Longrightarrow} A' \parallel Y$. Then from $A \parallel X \approx D$ and $A \parallel Y \approx D$ we may infer $D \stackrel{a}{\Longrightarrow} D_1$ and $D \stackrel{a}{\Longrightarrow} D_2$ with $A' \parallel X \approx D_1$ and $A' \parallel Y \approx D_2$. But since $D$ is deterministic $D_1 = D_2$, which implies $A' \parallel X \approx A' \parallel Y \approx D_1$. This proves $C'RD_1$ as required.

We next prove that $(R\cup \approx)^{-1} = R^{-1}\cup \approx$ is a simulation: assume $CR^{-1}D$ and $D \stackrel{a}{\Longrightarrow} D'$. As above for some $A$ we must have $C = A \parallel (X+Y)$ and $A \parallel X \approx D$ and $A \parallel Y \approx D$. From $A \parallel X \approx D$ we get $A \parallel X \stackrel{a}{\Longrightarrow} C' \approx D'$. There are two cases:

1. The inference of $A \parallel X \stackrel{a}{\Longrightarrow} C'$ uses a transition from $X$. With a similar inference we then get $C \stackrel{a}{\Longrightarrow} C'$ as required.

2. $C' = A' \parallel X$ and $A \stackrel{a}{\Longrightarrow} A'$, and thus also $A \parallel Y \stackrel{a}{\Longrightarrow} A' \parallel Y$ and $C \stackrel{a}{\Longrightarrow} A' \parallel (X+Y)$. With a reasoning analogous with case 3 above we infer that $A' \parallel X \approx A' \parallel Y$, which implies $A' \parallel (X+Y)RD'$ as required. $\square$

We continue with two lemmas with which any nondeterminism in a solution can be removed. In these lemmas we assume that $\mathcal{E}$ is an environment in which $A \parallel X \approx B$ holds and $B$ is deterministic.

Say that $Y$ *occurs* in $Z$ if $Y$ is a subexpression of $Z$ (recall from section 2 that $\mathcal{E}(Z)$ counts as a subexpression of $Z$). Without loss of generality we assume that no identifier occurring in $X$ also occurs in $A$ or $B$.

**Lemma 14** *Assume that for an identifier $Y$ occurring in $X$ it holds*

$$\mathcal{E}(Y) = \sum_{i=1}^{n} a_i.X_i$$

*where $a_j = a_k$ for some $j \neq k$. Let $Z$ be an identifier which is free in $\mathcal{E}$ and define the environment $\mathcal{F}$ by*

$$\mathcal{F}(V) = \begin{cases} a_j.Z + \sum_{i\neq j, i\neq k} a_i.X_i & \text{if } V = Y \\ \mathcal{E}(X_j) + \mathcal{E}(X_k) & \text{if } V = Z \\ \mathcal{E}(V) & \text{otherwise} \end{cases}$$

*Then $A \parallel X \approx B$ holds in $\mathcal{F}$.*

*Proof*: In the following $C$ will range over derivatives of $A \parallel X$. Thus $C$ is always $A' \parallel V$ where $V$ is an identifier and $A'$ is a derivative of $A$. We define $C[X_j + X_k/Z]$ as follows: if $C = A' \parallel Z$ then $C[X_j + X_k/Z] = A' \parallel (X_j + X_k)$, otherwise $C[X_j + X_k/Z] = C$. The following facts are easily verified:

1. If $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel Z$ then $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_j$ and $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_k$.

2. For $V \neq Z$: if $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel V$ then $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel V$.

3. If $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_j$ then either $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel X_j$ or $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel Z$.

4. If $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_k$ then either $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel X_k$ or $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel Z$.

5. For $V \neq X_j, X_k$: if $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel V$ then $V \neq Z$ and $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel V$.

Define the relation $R$ by $CRD$ if $C$ and $D$ are derivatives (w.r.t $\mathcal{F}$) of $A \parallel X$ and $B$ respectively, and $C[X_j + X_k/Z] \approx D$ (here as in the rest of this proof $\approx$ means with respect to $\mathcal{E}$ unless stated otherwise). Evidently $(A \parallel X)RB$, and we will prove that $R$ is a bisimulation with respect to $\mathcal{F}$. We first prove that $R$ is a simulation. So assume $CRD$ and $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} C'$. There are two cases:

1. $C'$ is $A' \parallel Z$. Then by fact 1, $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_j$ and $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_k$. Since $C[X_j + X_k/Z] \approx D$ there must exist $D_1$ and $D_2$ such that $D \stackrel{a}{\Longrightarrow}_{\mathcal{E}} D_1$ and $D \stackrel{a}{\Longrightarrow}_{\mathcal{E}} D_2$ and $D_1 \approx A' \parallel X_j$ and $D_2 \approx A' \parallel X_k$. But $D$ is deterministic and hence $D_1 = D_2$. We thus infer $A' \parallel X_j \approx A' \parallel X_k \approx D_1$, whence by lemma 13 $C'[X_j + X_k/Z] = A' \parallel (X_k + X_j) \approx D_1$; this proves $C'RD_1$. Finally, since $X$ does not occur in $D$ we infer $D \stackrel{a}{\Longrightarrow}_{\mathcal{F}} D_1$.

2. $C'$ is $A' \parallel V$ for some $V \neq Z$. Then by fact 2, $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} C'$, and since $C[X_j + X_k/Z] \approx D$ we infer $D \stackrel{a}{\Longrightarrow}_{\mathcal{E}} D'$ with $C' \approx D'$. Since $Z$ cannot occur in $A'$ and since $X$ does not occur in $D$ this implies that $D \stackrel{a}{\Longrightarrow}_{\mathcal{F}} D'$ with $C'[X_j + X_k/Z] \approx D'$ as required.

We next prove that $R^{-1}$ is a simulation. So assume $CR^{-1}D$ and $D \stackrel{a}{\Longrightarrow}_{\mathcal{F}} D'$. First, since $X$ does not occur in $D$ we get $D \stackrel{a}{\Longrightarrow}_{\mathcal{E}} D'$, and by definition of $R$ we get $C[X_j + X_k/Z] \approx D$. This means that $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} C'$ with $C' \approx D'$. Again there are two cases:

1. $C'$ is either $A' \parallel X_j$ or $A' \parallel X_k$; these cases are symmetric and we consider only $C' = A' \parallel X_j$. Then by fact 3 we infer that either $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} C'$, in which case we are done since $C'[X_j + X_k/Z] = C' \approx D'$, or $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel Z$. So assume $C \stackrel{a}{\Longrightarrow}_{\mathcal{F}} A' \parallel Z$. By fact 1 we get $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{E}} A' \parallel X_k$. Since $D$ is deterministic and $C[X_j + X_k/Z] \approx D$ we get that $A' \parallel X_j \approx A' \parallel X_k \approx D'$. By lemma 13 this implies $(A' \parallel Z)[X_j + X_k/Z] = A' \parallel (X_j + X_k) \approx D'$ as required.

2. $C'$ is neither $A' \parallel X_j$ nor $A' \parallel X_k$. In this case fact 5 applies and we immediately get $C[X_j + X_k/Z] \stackrel{a}{\Longrightarrow}_{\mathcal{F}} C'$, and conclude $C'[X_j + X_k/Z] = C' \approx D'$ as required.

This concludes the proof of lemma 14. $\square$

**Lemma 15** *Assume that for an identifier $Y$ occurring in $X$ it holds*

$$\mathcal{E}(Y) = \sum_{i=1}^{n} a_i.X_i$$

*where for some $j$ we have $a_j = \tau$. Define the environment $\mathcal{F}$ by*

$$\mathcal{F}(V) = \begin{cases} \sum_{i \neq j} a_i.X_i & \text{if } V = Y = X_j \\ \mathcal{E}(X_j) + \sum_{i \neq j} a_i.X_i & \text{if } V = Y \neq X_j \\ \mathcal{E}(V) & \text{otherwise} \end{cases}$$

*Then $A \parallel X \approx B$ holds in $\mathcal{F}$.*

*Proof*: Define the relation $R$ by $CRD$ if $C$ and $D$ are derivatives (w.r.t $\mathcal{F}$) of $A \parallel X$ and $B$ respectively, and $C \approx D$ w.r.t $\mathcal{E}$. We can prove that $R$ is a bisimulation w.r.t. $\mathcal{F}$. The proof is similar to (but simpler than) the proof of lemma 14; we omit the details. □

We now conclude the proof of proposition 12. Assume that $A \parallel X \approx B$ is true in $\mathcal{E}$. Apply lemma 15 repeatedly to remove all $\tau$ actions from $X$ and its derivatives. Then apply lemma 14 repeatedly until $X$ becomes deterministic. We additionally prove that if $X$ is finite state in $\mathcal{E}$ it is sufficient to apply these lemmas a finite number of times: for lemma 15 this is immediate since each application of the lemma removes one $\tau$ action from $\mathcal{E}$. For lemma 14 a more elaborate proof is needed. For a derivative $V$ of $X$, define

$$\begin{aligned} indeg_{\mathcal{E}}(V) &= \text{the number of occurrences of a term } a.V \text{ in } \mathcal{E} \\ outdeg_{\mathcal{E}}(V) &= n \text{ where } \mathcal{E}(V) = \textstyle\sum_{i=1}^{n} a_i.V_i \end{aligned}$$

and define a measure over environments by

$$|\mathcal{E}| = \sum_{V} (indeg_{\mathcal{E}}(V) + 1) \times outdeg_{\mathcal{E}}(V)$$

where the summation is over all derivatives of $X$. It is easy to verify that each application of lemma 14 decreases this measure; hence the lemma can only be applied a finite number of times if $X$ has a finite number of derivatives in the original environment . □