Representing Closed CCS Systems by Petri Nets

Jacek Olszewski[1]
Microsoft Institute of Advanced Software Technology
65 Epping Rd, North Ryde, 2113 Australia
E-mail: jacek@cse.unsw.edu.au

UNSW-CSE-TR-9412 — 31 OCTOBER 1994

[1]on leave from the School of Computer Science and Engineering, University of New South Wales, Kensington, Australia.

Abstract

This paper describes and proves a simple transformation of CCS compositions into Petri nets. Under certain conditions, additional to the CCS syntax rules, the resulting Petri nets are finite, and firing of their transitions corresponds to handshakes in CCS compositions. Such correspondence also holds between simultaneous firing of several transitions and multiple handshakes. The transformation has proved useful in a fast deadlock detection tool developed for CCS specifications.

1 Introduction

Pure Petri nets [10], i.e. without coloured tokens, labelled edges, inhibitor edges, etc., have been proven inadequate as representations of CCS programs (cf. for example [8, 12]). Semantics of such Petri nets is not as powerful as that of CCS. However, they may adequately serve certain purposes whose scope is less than modeling full semantics of CCS. For instance, they may be used to investigate possible sequences of handshakes in certain CCS compositions, or even sequences of sets of handshakes that can be performed together. In particular, they may help in analysis of closed systems for presence or absence of deadlocks. Closed means that the specification includes both the composition of concurrent processes and their environment. The environment may also consist of concurrent processes. In such a system, all interactions between processes are modeled as handshakes. If the system as a whole becomes ready for an action and waits for it, it is a deadlock. On the other hand, if handshakes are always possible, the system is deadlock free.

Petri nets are not the only modeling tool for CCS compositions. Another such tool is automata that have been used for a wider range of analysis than just deadlock detection (see for instance the Concurrency Workbench [1]). However, automata seem inferior to Petri nets with respect to one important aspect. Namely, they cannot model possible simultaneity of actions within the system. Because of their single locum of control, they necessarily represent so-called interleaving model of concurrency. Petri nets may be used to model concurrent actions within the system by simultaneous firing of more than one transition at the same time (see for instance [13, 2]). Hence, they may represent a non-interleaving model of concurrency. If a set of actions that may happen together replaces all of their possible interleavings, the system state space may be reduced considerably, thus leading to more efficient analysis tools. For instance, replacement of 2 possible interleavings of two actions by one double action reduces the number of possible states by half.

The rest of the paper is organized as follows. The next section describes syntax and transitional semantics of CCS. It also presents conditions upon CCS specifications, under which they can be modeled by pure Petri nets. Section 3 shows how a Petri net is constructed for a given CCS composition, and how handshakes between processes are modeled by firing transitions of the net. Section 4 presents a deadlock detection tool for CCS compositions, based on the reachability analysis of their Petri net models. Section 5 concludes the paper.

2 CCS

The notation for CCS specifications and its transitional semantics is taken directly from [6]. Let A be a set of actions, and $\overline{A}$ a set of co-actions. Also, let $Act = A \cup \overline{A} \cup \{\tau\}$, where τ is a so-called silent action (handshake). Further, let $\{E_i \ : \ i \in I\}$ be a family of expressions indexed by I. Such expressions can be combined into specifications of concurrent systems by means of the following

operators:

1. $a.E$, a Prefix ($a \in Act$)
2. $\sum\{a.E_i : i \in I\}$, a Summation ($a \in A$)
3. $\prod\{E_i : i \in I\}$, a Composition ($E_{i_1}|E_{i_2}|\ldots$)
4. $E\backslash L$, a Restriction ($L \subseteq Act$)
5. $E[f]$, a Relabelling (f a relabelling function)

Generally, actions represent input to be performed by processes, while co-actions – output. To denote a process incapable of any actions or co-actions, a special identifier **0** is used.

Milner [6] explains why (2) has to be the sum of all expressions E_i. Here, for reasons explained further, (2) allows only choices guarded by input actions (c.f. Assumption 1 below).

The transition rules are as follows:

Act $\dfrac{}{a.E \xrightarrow{a} E}$

Sum$_j$ $\dfrac{E_j \xrightarrow{a} E'_j}{\sum\{E_i : i \in I\} \xrightarrow{a} E'_j}$ $\quad (j \in I)$

Com1 $\dfrac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F}$

Com2 $\dfrac{F \xrightarrow{a} F'}{E|F \xrightarrow{a} E|F'}$

Com3 $\dfrac{E \xrightarrow{l} E', F \xrightarrow{\overline{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$

Com4 $\dfrac{E \xrightarrow{\tau} E', F \xrightarrow{\tau} F'}{E|F \xrightarrow{\tau} E'|F'}$

Res $\dfrac{E \xrightarrow{a} E'}{E\backslash L \xrightarrow{a} E'\backslash L}$ $\quad (a, \overline{a} \notin L)$

Rel $\dfrac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$

Com4 states that if there are more than one pair of components capable of the silent action τ, all of such pairs can engage in τ, and for the whole composition it is still the same silent action.

What is actually represented by τ may be denoted by an index attached to τ. For instance, let τ_a denote a handshake between $a.E_i$ and $\overline{a}.E_j$, and $\tau_{a,b}$ – handshakes between $a.E_i$ and $\overline{a}.E_j$, and $b.E_k$ and $\overline{b}.E_l$.

As an example, let us consider

$$D = (E_1|E_2|E_3|E_4)\backslash L$$

where

$$\begin{aligned} E_1 &= a.b.\overline{a}.E_1 \\ E_2 &= \overline{a}.\overline{a}.a.E_2 \\ E_3 &= b.a.b.E_3 \\ E_4 &= \overline{b}.E_4 \\ L &= \{a, b\} \end{aligned}$$

Behaviour of D may depend on the required degree of parallelism. In the case of its maximum, it is the following cycle:

$$\begin{aligned} D &\xrightarrow{\tau_{a,b}} (b.\overline{a}.E_1|\overline{a}.a.E_2|a.b.E_3|E_4)\backslash L \\ &\xrightarrow{\tau_{b,a}} (\overline{a}.E_1|a.E_2|b.E_3|E_4)\backslash L \\ &\xrightarrow{\tau_{a,b}} D \end{aligned}$$

However, with less than the maximum degree of parallelism, the following sequence is also possible:

$$\begin{aligned} D &\xrightarrow{\tau_b} (E_1|E_2|a.b.E_3|E_4)\backslash L \\ &\xrightarrow{\tau_a} (E_1|\overline{a}.a.E_2|b.E_3|E_4)\backslash L \\ &\xrightarrow{\tau_b} (E_1|\overline{a}.a.E_2|E_3|E_4)\backslash L \\ &\xrightarrow{\tau_b} (E_1|\overline{a}.a.E_2|a.b.E_3|E_4)\backslash L \\ &\xrightarrow{\tau_a} (E_1|a.E_2|b.E_3|E_4)\backslash L \\ &\xrightarrow{\tau_b} (E_1|a.E_2|E_3|E_4)\backslash L \\ &\xrightarrow{\tau_b} (E_1|a.E_2|a.b.E_3|E_4)\backslash L = \mathbf{0} \quad (deadlock) \end{aligned}$$

To formulate conditions imposed upon CCS specifications and their transformation into Petri nets, the following notation is used:

$\ldots P$ – a process whose first and further actions or co-actions are irrelevant at the moment,

$a \ldots P$ – a process whose first action is a, and whose second and further actions are irrelevant at the moment,

τ_a – a single handshake over a,

$\tau_{a_1,a_2,\ldots,a_n}$ – a multiple handshake over $a_1, a_2, \ldots, a_n$.

The conditions can be formulated as follows:

Assumption 1 *Choices are guarded by input actions*[1].

In other words, only choices of the form $a.E_i + b.E_j$ are allowed. Choices of the forms $a.E_i + \overline{b}.E_j$, $\overline{a}.E_i + b.E_j$, $\overline{a}.E_i + \overline{b}.E_j$ are considered syntactically incorrect.

Assumption 2 *Each pair of action – co-action names occurs only in two processes.*

In other words, in a composition $a.E_i|\overline{a}.E_j|R$, the names a and $\overline{a}$ can occur only in E_i and E_j, not in R. This assumption effectively rules out compositions of processes that are defined recursively as compositions themselves. A process defined as follows:

$$P = a \ldots P|b \ldots Q$$

violates the assumption because each substitution of its definition for its name in the composition creates new components in which the same action – co-action names occur.

Notice that the constraints exclude specifications like the example D above.

It is easy to see that Assumptions 1 and 2 constrain any CCS specification to a composition whose components are single processes or action guarded choices of processes; they are not compositions themselves:

$$E = E_1|E_2|\ldots|E_n$$

where each of $E_i, i = 1, 2, \ldots, n$, has the syntax P (here $\mid$ is a meta-symbol that denotes a choice of syntax terms):

$$P = \overline{a}.P \mid \sum a.P \mid P[f] \mid \mathbf{0}$$

As we will see in the next section, compositions that satisfy Assumptions 1 and 2 can easily be represented by finite Petri nets where firing of transitions corresponds to handshakes, and where multiple transition firing corresponds to a multiple handshake. However, to make use of the latter in a deadlock detection tool, we have to constrain compositions even further. They are assumed to satisfy also the following:

Assumption 3 *Each component of the composition is a cyclic process.*

It effectively excludes components defined as, eg. $P = a.Q$, where $Q = \ldots Q$.

It has been proved in [9] that for compositions satisfying all 3 assumptions, it is possible to prove the following theorems:

[1] A similar constraint can be found in Ada or Occam where choices are allowed between input actions only (cf. [11, 4])

- A multiple handshake $\tau_{a_1,a_2,\ldots,a_n}$ is equivalent to single handshakes $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_n}$ performed in any order.
- If there is a sequence of single handshakes that leads to deadlock, there is also a sequence of multiple handshakes at the maximum degree of parallelism, that leads to deadlock as well.

Without Assumption 3, it is necessary to consider all possible sequences of single handshakes, and consequently, of single transition firings.

3 Petri nets

Petri net is a quadruple:

$$N = (Pl, Tr, Ar, M_0)$$

where Pl is a set of places, Tr – of transitions, Ar – of arcs between places and transitions, and between transitions and places, $M_0 \subseteq Pl$ – initial marking. Marking M is a multiset (bag) of places that hold tokens. Each place can hold a number tokens (0 or more), and it is included in M that number of times. Ar is denoted by a set of subsets of Pl, defined for elements of Tr. For each $t \in Tr$, there are two subsets of Pl:

$\cdot t$ – input places for transition t,
$t\cdot$ – output places for transition t.

Actions that can be represented by such a net take the form of changes of its marking M. An elementary change is called firing of a transition which means that tokens are taken from the transition input places, and put into its output places. Firing of $t \in Tr$ is denoted by $M_i \stackrel{t}{\rightarrow} M_j$ where M_i and M_j are markings before and after firing of t correspondingly. Such firing is possible if the input places hold tokens, i.e. $\cdot t \subseteq M_i$. The effect of firing of $t \in Tr$ can be presented as

$$M_j = (M_i - \cdot t) \sqcup t\cdot$$

where $-$ and $\sqcup$ denote difference and sum of multisets respectively.

Transformation of a given CCS composition E into a Petri net can be presented as the following function;

$$Pn(E) = (Pl_E, Tr_E, Ar_E, \{E\})$$

where $E \in Pl$ is a starting place of the net, i.e. $M_0 = \{E\}$, and where Pl_E, Tr_E, Ar_E are defined by another function Nt. The identifier E plays a double role; it denotes the CCS composition and it is also a name of a place in the Petri net. Generally, certain places of the net are named after corresponding parts of the CCS specification.

The definition of Nt is given case by case as follows:

Composition
For a composition $E = E_1|E_2|\ldots|E_n$,

$$\begin{aligned} Nt(E) \;=\; & (\{E_i | i = 1,2,\ldots,n\} \cup \bigcup_{i=1,2,\ldots,n} Pl_i, \\ & \{t\} \cup \bigcup_{i=1,2,\ldots,n} Tr_i, \\ & \{\cdot t, t\cdot\} \cup \bigcup_{i=1,2,\ldots,n} Ar_i) \end{aligned}$$

where

$$\begin{aligned} \cdot t &= \{E\} \\ t\cdot &= \{E_1, E_2, \ldots, E_n\} \\ (Pl_i, Tr_i, Ar_i) &= Nt(E_i),\ i = 1,2,\ldots,n \end{aligned}$$

In other words, $n+1$ places and 1 transition t are created, firing of which means that the places $E_1, E_2, \ldots, E_n$ become marked. Fig. 1 illustrates the corresponding fragment of the net, where places are drawn as circles, and transitions as bars.

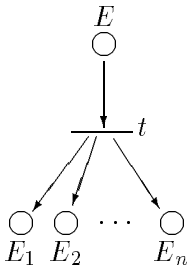

Fig. 1 Composition

Process definition
Suppose that for some index i, $E_i = P$ where P is defined separately as $P = Def$. In that case, the corresponding place in the net has the name P, and

$$Nt(P) = (\{Def\} \cup Pl_{Def},\ \ \{t\} \cup Tr_{Def},\ \ \{\cdot t, t\cdot\} \cup Ar_{Def})$$

where

$$\begin{aligned} \cdot t &= \{P\} \\ t\cdot &= \{Def\} \\ (Pl_{Def}, Tr_{Def}, Ar_{Def}) &= Nt(Def) \end{aligned}$$

The process P may be used (called) by more than one component of the composition. However, the corresponding fragment of the net is created once – Nt is applied to P once – and the place named after P is the same in all such cases.

Action
For a process with an input prefix, $E = a.F$,

$$Nt(E) = (\{F, p_a, q_a\} \cup Pl_F, \quad \{t_a\} \cup Tr_F, \quad \{\cdot t_a, t_a \cdot\} \cup Ar_F)$$

where

$$\begin{aligned} \cdot t_a &= \{p_a, E\} \\ t_a \cdot &= \{q_a, F\} \\ (Pl_F, Tr_F, Ar_F) &= Nt(F) \end{aligned}$$

Places p_a and q_a do not belong to any process thread. They are related to input/output actions performed by processes. Further explanations follow the next case of Nt.

Co-action
For a process with an output prefix, $E = \overline{a}.F$,

$$Nt(E) = (\{F, p_a, q_a, \underline{a}.F\} \cup Pl_F, \quad \{u_a, w_a\} \cup Tr_F, \quad \{\cdot u_a, u_a \cdot, \cdot w_a, w_a \cdot\} \cup Ar_F)$$

where

$$\begin{aligned} \cdot u_a &= \{E\} \\ u_a \cdot &= \{p_a, \underline{a}.F\} \\ \cdot w_a &= \{q_a, \underline{a}.F\} \\ w_a \cdot &= \{F\} \\ (Pl_F, Tr_F, Ar_F) &= Nt(F) \end{aligned}$$

Places p_a and q_a are the same as for the process that has an input prefix a. Since the set union $\cup$ is used in the formula for Nt, there are only as many pairs of places p_{a_i}, q_{a_i} in the net, as there are different actions/co-actions a_i in the composition. Other places, $\underline{a}.F, F$, and transitions, u_a, w_a, are unique, i.e. they are not repeated in transformation of any other CCS processes. The symbol $\underline{a}.F$ does not come from CCS. It used here just to name the place between the transitions u_a and w_a.

Communication represented by fragments of a Petri net created by Nt is illustrated in Fig 2. There, the composition $a.D|\overline{a}.F$ is transformed by Nt into 2 threads of the net, and the places p_a, q_a between the threads. The handshake τ_a is interpreted as firing a sequence of 3 transitions: t – beginning of output, w – input, and u – end of output.

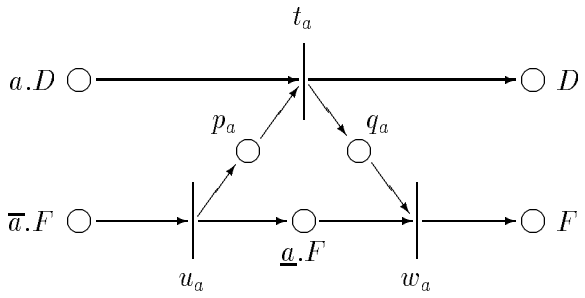

Fig. 2 Communication

Choice

For a choice of processes, $E = a_1.E_1 + a_2.E_2 + \ldots + a_n.E_n$,

$$\begin{aligned} Nt(E) \quad = \quad & (\{p_{a_i}, q_{a_i}, E_i | i = 1, 2, \ldots, n\} \cup \bigcup_{i=1,2,\ldots,n} Pl_i, \\ & \{t_{a_i} | i = 1, 2, \ldots, n\} \cup \bigcup_{i=1,2,\ldots,n} Tr_i, \\ & \{\cdot t_{a_i}, t_{a_i} \cdot | i = 1, 2, \ldots, n\} \cup \bigcup_{i=1,2,\ldots,n} Ar_i) \end{aligned}$$

where

$$\begin{aligned} \cdot t_{a_i} &= \{E, p_{a_i}\}, \quad i = 1, 2, \ldots, n \\ t_{a_i} \cdot &= \{q_{a_i}, E_i\}, \quad i = 1, 2, \ldots, n \\ (Pl_i, Tr_i, Ar_i) &= Nt(E_i), \quad i = 1, 2, \ldots, n \end{aligned}$$

Fig. 3 illustrates transformation of a choice into the corresponding fragment of a Petri net. It shows how the actual choice depends on communication. It is non-deterministic, if more than one of the places p_{a_i} is marked (there is choice of handshakes with such a process).

Relabeling

For a process defined as a relabeling of another process, $F = E[f]$,

$$Nt(F) = Nt(E')$$

where E' stands for the definition of E to which the relabeling function f has been applied, i.e. corresponding actions and/or co-actions have been relabeled, and all occurrences of E have been replaced by E'.

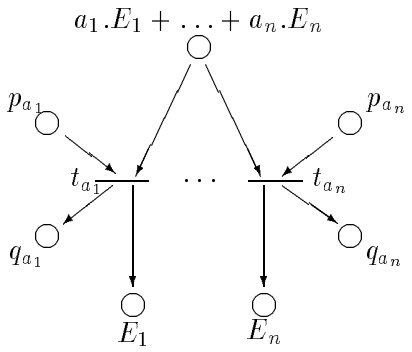

Fig 3 Choice

As mentioned in the introduction, CCS compositions that are considered here are closed, i.e. restricted by their entire alphabets. If such a composition becomes ready for an action, and waits for it, it is a deadlock. If, however, handshakes can be performed ad infinitum, the composition is deadlock free. Restriction of every composition by its entire alphabet makes it unnecessary to define Nt for the restriction case – it will not be used.

For the proposed transformation Tn, the following lemma is not difficult to prove.

Lemma 1 *If a handshake τ_a is possible for a given composition E, so is the firing sequence u_a, t_a, w_a for $Tn(E)$.*

Proof
The proof examines markings of $Tn(E)$ and shows that the transition sequence can be fired.

Assume τ_a to be the first handshake of E. Its possibility means that either

$$E = \overline{a}.D|a.F|\ldots$$

or

$$E = \overline{a}.D|(a.F + b.G + \ldots)|\ldots$$

Consider the latter alternative as more general. From the definition of Tn we have:

$$Tn(E) = (Pl_E, Tr_E, Ar_E, \{E\})$$

where

$$\begin{aligned} Pl_E &= \{E, p_a, q_a, \overline{a}.D, \underline{a}.D, D, a.F + b.G + \ldots, F, G, \ldots\} \\ Tr_E &= \{t, u_a, w_a, t_a, \ldots\} \\ \cdot t &= \{E\} \\ t\cdot &= \{\overline{a}.D, a.F + b.G + \ldots, \ldots\} \end{aligned}$$

$$
\begin{aligned}
\cdot u_a &= \{\overline{a}.D\} \\
u_a\cdot &= \{p_a, \underline{a}.D\} \\
\cdot w_a &= \{q_a, \underline{a}.D\} \\
w_a\cdot &= \{D\} \\
\cdot t_a &= \{a.F + b.G + \ldots, p_a\} \\
t_a\cdot &= \{q_a, F\}
\end{aligned}
$$

Since $M_k \xrightarrow{t_k} M_{k+1}$ where $M_{k+1} = (M_k - \cdot t_k) \sqcup t_k\cdot$, it is easy to see that the sequence of firings

$$
\begin{aligned}
\{E\} &\xrightarrow{t} \{\overline{a}.D, a.F + b.G + \ldots, \ldots\} \\
&\xrightarrow{u_a} \{\underline{a}.D, p_a, a.F + b.G + \ldots, \ldots\} \\
&\xrightarrow{t_a} \{\underline{a}.D, q_a, F, \ldots\} \\
&\xrightarrow{w_a} \{D, F, \ldots\}
\end{aligned}
$$

can be performed.

If the given handshake is not the first, but occurs after a sequence of handshakes $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_k}$, the proof is carried out as above for τ_{a_1}. Then, it remains to prove that the firing sequence

$$u_{a_2}, t_{a_2}, w_{a_2}, \ldots, u_{a_k}, t_{a_k}, w_{a_k}, u_a, t_a, w_a$$

is possible for $Tn(D|F|\ldots)$. Since τ_{a_2} is the next handshake performed by the composition, we have:

$$D|F|\ldots = \overline{a_2}.G|a_2.H|\ldots$$

or

$$D|F|\ldots = \overline{a_2}.G|(a_2.H + c.I + \ldots)|\ldots$$

Notice that proving $u_{a_2}, t_{a_2}, w_{a_2}$ for $\overline{a_2}.G|(a_2.H + c.I + \ldots)|\ldots$ is analogous to proving u_a, t_a, w_a for $\overline{a}.D|(a.F + b.G + \ldots)|\ldots$ above. The same can be shown for the rest of the sequence, i.e. for $\tau_{a_3}, \tau_{a_4}, \ldots, \tau_{a_k}$, and for τ_a.
□

Lemma 1 states only that for any handshake sequence of the composition E there is a corresponding firing sequence of the Petri net $Tn(E)$. To use $Tn(E)$ as an analysis tool for E, we need a reverse – a theorem stating that every possible firing sequence of $Tn(E)$ has a defined interpretation in terms of handshake sequences of E. Before such a theorem is formulated and proven, we shall try to reduce the number of possible firings that have to be taken into the consideration. Without any reduction, firing of the transitions u_a, t_a, w_a in that order may be understood as representing a single handshake τ_a. Moreover, each of the following firing sequences:

$$
\begin{aligned}
&\langle u_a, t_a, w_a, u_b, t_b, w_b\rangle, \langle u_a, t_a, u_b, w_a, t_b, w_b\rangle, \langle u_a, u_b, t_a, w_a, t_b, w_b\rangle, \\
&\langle u_a, u_b, t_a, t_b, w_a, w_b\rangle, \langle u_a, u_b, t_b, t_a, w_a, w_b\rangle, \langle u_b, u_a, t_b, t_a, w_a, w_b\rangle, \\
&\langle u_b, t_b, u_a, t_a, w_a, w_b\rangle
\end{aligned}
$$

may be understood as representing the same handshake sequence τ_a, τ_b. Reduction of the number of firings can be achieved by firing certain transitions invisibly (automatically), i.e. as soon as they become enabled, thus leaving them out of the consideration.

A transition can be left out of the consideration if it is not involved in any choices, i.e. its firing does not exclude firing of any other transitions. If two or more such transitions are enabled, the order in which they are fired is irrelevant to the subsequent firing possibilities of the net. Furthermore, they may be fired simultaneously. The following lemma serves as a formal statement of the observation just made.

Lemma 2 *If 2 transitions of a Petri net can be fired in any order, both firing orders give the same marking.*

Proof
Let x, y be the two enabled transitions. The premise can formalized as follows:

1. $\cdot x, \cdot y \subseteq M$, i.e. both are enabled,
2. $\cdot x \cap \cdot y = \emptyset$, i.e. they do not exclude one another.

Firing x, then y gives a new marking:

$$M' = ((M - \cdot x) \sqcup x\cdot - \cdot y) \sqcup y\cdot$$

Since $\cdot x$ and $\cdot y$ are both disjoint subsets of M (points 1 and 2 above), we can perform subtractions first, and then additions:

$$M' = (M - \cdot x - \cdot y) \sqcup x\cdot \sqcup y\cdot$$

We can also change the order of subtractions, and the order of additions:

$$M' = (M - \cdot y - \cdot x) \sqcup y\cdot \sqcup x\cdot$$

M' does not change if we first add $y\cdot$, and then subtract $\cdot x$:

$$M' = ((M - \cdot y) \sqcup y\cdot - \cdot x) \sqcup x\cdot$$

The last formula is that of firing y, then x, thus proving the lemma.
□

For any CCS composition E, transitions of $Tn(E)$ can be classified as follows:

1. single input transitions (composition, beginning of co-action),
2. double input-output transitions (action),

3. single output transitions (call of a process, end of co-action).

From the definition of Tn above, it follows that input sets of every transition of class 1 are disjoint with input sets of all other transitions of the net. Therefore, transitions of that class can be reduced by invisible firing (as soon as they are enabled). Transitions of class 2, obviously, cannot be reduced because they may be involved in choices, i.e. their input sets may not be disjoint (cf. the choice case of Nt and Fig. 3 above). For transitions of class 3 we have to prove or disprove their involvement in choices. Those that represent calls of processes are involved in no choices because of Assumption 1 above (choices in CCS specifications are guarded by input actions). The question whether transitions representing end of output, w_a, can be reduced is a little more complicated. A choice between such transitions, say w_a^i and w_a^j, can occur if both are enabled, i.e. $\cdot w_a^i, \cdot w_a^j \subseteq M$, $\cdot w_a^i \cap \cdot w_a^j = \{q_a\}$. For both to be enabled, the transitions u_a^i, u_a^j, t_a^k must have been fired. Assumption 2 above (each pair of action – co-action names occurs only in two processes) excludes the possibility of the 3 fired transitions belonging to 3 different threads of the net. Hence, we may assume, for instance, $k = j$. Therefore, the order of the transition firing must have been u_a^i, t_a^j, u_a^j because:

1. $u_a^i \cdot \cap \cdot t_a^j = \{p_a\}$, i.e. t_a^j is preceded by u_a^i,
2. $u_a^i \cdot \cap \cdot u_a^j = u_a^j \cdot \cap \cdot u_a^i = \emptyset$, i.e. neither can u_a^i be followed immediately by u_a^j, nor u_a^j by u_a^i; t_a^j must be fired between u_a^i and u_a^j.

The firing order u_a^i, t_a^j, u_a^j for $Tn(E)$ means that E includes components of the following form:

- $\overline{a}.E_i$, transformed by Tn into thread i,
- $a.\overline{a}.E_j$, transformed by Tn into thread j,

A corresponding fragment of $Tn(\overline{a}.E_i | a.\overline{a}.E_j)$ may be shown as:

$$
\begin{aligned}
Pl_E &= \{\overline{a}.E_i, \underline{a}.E_i, E_i, a.\overline{a}.E_j, \overline{a}.E_j, \underline{a}.E_j, E_j, p_a, q_a, \ldots\} \\
Tr_E &= \{u_a^i, w_a^i, t_a^j, u_a^j, w_a^j, \ldots\} \\
\cdot u_a^i &= \{\overline{a}.E_i\} \\
u_a^i \cdot &= \{\underline{a}.E_i, p_a\} \\
\cdot w_a^i &= \{\underline{a}.E_i, q_a\} \\
w_a^i \cdot &= \{E_i\} \\
\cdot t_a^j &= \{a.\overline{a}.E_j, p_a\} \\
t_a^j \cdot &= \{\overline{a}.E_j, q_a\} \\
\cdot u_a^j &= \{\overline{a}.E_j\} \\
u_a^j \cdot &= \{\underline{a}.E_j, p_a\} \\
\cdot w_a^j &= \{\underline{a}.E_j, q_a\} \\
w_a^j \cdot &= \{E_j\}
\end{aligned}
$$

Now we can see that the choice between w_a^i and w_a^j is "false". It must not be considered a choice at all. w_a^i should be fired as soon as it is enabled, thus finishing the first handshake τ_a between the two components. The second handshake τ_a, already begun as firing of u_a^j, may be finished by firing of w_a^j only after t_a^i (representing input by E_i, not shown in the formulae above) is fired.

Another, simpler solution to the problem would be to require that communication between components of E be one way only, i.e. the use of both a and $\overline{a}$ in the definition of any single component would be regarded as a syntax error[2]. Then, the situation presented above could not occur. However, a slight complication of proofs and implementation seems to be a small price for less restrictive notation.

Having solved the "false" choice problem, we can apply invisible firing to the net, thus leaving transitions of class 1 and 3 out of the consideration. Notice that such transitions may enable one another. For instance, firing of a transition representing a composition may enable a number of those representing the beginning of output. If more than one are enabled, they may be fired in an arbitrary order – cf. Lemma 2 above. They should be fired repeatedly until the only enabled transitions in the net are those of class 2. "False" choices can be eliminated practically as follows: after firing of t^i (class 2), do not fire w^i (class 3) despite the fact that it may be enabled. The superscript i indicates the same thread of the net. Some other transition of class 3 of a different thread, w^j, will get fired invisibly instead.

Rules of invisible firing for a composition E of m components and n actions/co-actions can be formalized as follows (arrows are intentionally shown with no labels):

$$\{E\} \rightarrow \{E_i | i = 1, 2, ..., m\} \quad (1)$$

$$\{\overline{a_j}.E_i, \ldots\} \rightarrow \{\underline{a_j}.E_i, p_{a_j}, \ldots\}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n \quad (2)$$

$$\{\underline{a_j}.E_i, q_{a_j}, \ldots\} \rightarrow \{E_i, \ldots\}, \quad i = 1, 2, \ldots, k-1, k+1, \ldots, m, \quad j = 1, 2, \ldots, n \quad (3)$$

$$\{P, \ldots\} \rightarrow \{Def, \ldots\} \quad (4)$$

where k in (3) indicates the visible transition

$$\{a_j.E_k, p_{a_j}, \ldots\} \xrightarrow{t_{a_j}} \{E_k, q_{a_j}, \ldots\}$$

fired prior to (3). To avoid a "false" choice, for $i = k$, (3) is not fired.

With invisible firing, the only transitions to be considered are those of class 2. Their firings may well represent corresponding handshakes, as the following theorems state below. In the proof of Theorem 1, we shall use a notion of reverse firing, denoted by $\leftarrow$:

$$M_k \xleftarrow{t} M_{k-1} \quad where \quad M_{k-1} = (M_k - t\cdot) \sqcup \cdot t$$

[2]Such a restriction can be found in the original version of CSP [3].

For all 4 theorems, the rules of invisible firing are assumed to be in force, applied to the Petri net $Tn(E)$.

Theorem 1 *If firing of t_a is possible for $Tn(E)$, so is the handshake τ_a for E.*

Proof
Let M_k be the marking of $Tn(E)$ after t_a is fired.

1. $t_a \in Tr_E \Rightarrow \{a.E_i, E_i, p_a, q_a\} \subseteq Pl_E \wedge (E = a.E_i | \ldots \vee E = a.E_i + b.F + \ldots)$, from the input or choice case of the definition of Nt,
2. $M_k = \{E_i, q_a, \ldots\} \stackrel{t_a}{\longleftarrow} \{a.E_i, p_a, \ldots\} = M_{k-1}$, reverse firing of t_a,
3. $p_a \in M_{k-1} \Rightarrow u_a \in Tr_E$, a token may be placed in p_a only by firing of u_a,
4. $u_a \in Tr_E \Rightarrow \{\overline{a}.\{\underline{a}.E_j, \{\underline{a}.E_j, E_j, p_a, q_a\} \subseteq Pl_E \wedge (E = a.E_i | \overline{a}.E_j | \ldots \vee E = a.E_i + b.F + \ldots | \overline{a}.E_j | \ldots)$, from the output case of the definition of Nt, and from 1,
5. for either of $a.E_i | \overline{a}.E_j | \ldots$ and $a.E_i + b.F + \ldots | \overline{a}.E_j | \ldots$, τ_a is possible.

□

Theorem 2 *If no visible transition of $Tn(E)$ is enabled, no handshake is possible for E either.*

Proof
Assume the contrary, eg. τ_a is possible for E. Hence, from Lemma 1 we have that the firing sequence u_a, t_a, w_a is possible for $Tn(E)$. u_a is an invisible transition, fired as soon as enabled. So, we have to show that u_a is never enabled. Lemma 1 and the rules of invisible firing preclude the case of τ_a to be the first handshake of E, because u_a is enabled either immediately, or after t is fired, where

$$\begin{aligned} {\cdot t} &= \{E\} \\ t{\cdot} &= \{a.E_i + b.F + \ldots, \overline{a}.E_j, \ldots\} \\ M &= \{E\} \end{aligned}$$

and where t is also invisible. Therefore, there has to be some sequence of handshakes, $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_k}$ performed by E prior to τ_a. From the rules of invisible firing and Theorem 1 we know that such a sequence corresponds to the visible firing sequence $t_{a_1}, t_{a_2}, \ldots, t_{a_k}$, accompanied by invisible firings of all transitions $u_{a_i}, w_{a_i}, i = 1, 2, \ldots, k$. According to Lemma 1, after all transitions $u_{a_i}, t_{a_i}, w_{a_i}, i = 1, 2, \ldots, k$ are fired, the next firing sequence is u_a, t_a, w_a. Therefore, u_a is enabled.
□

Theorem 3 *If a handshake $\tau_{a,b}$ is possible for E, so is firing of t_a and t_b simultaneously in $Tn(E)$.*

Proof
$\tau_{a,b}$ means that τ_a and τ_b are performed together. Therefore, Lemma 1 can be applied to both of them, i.e. the firing sequences u_a, t_a, u_a and u_b, t_b, u_b are possible together. In other words, the transitions u_a and u_b are both enabled, and they do not exclude one another. Since they are invisible, they are fired, thus enabling t_a and t_b.
□

Theorem 4 *If two visible transitions of $Tn(E)$, t_a and t_b, can be fired together, then the handshake $\tau_{a,b}$ is possible for E.*

Proof
Applying Theorem 1 to both transitions t_a and t_b, we conclude that the handshakes τ_a and τ_b are possible. From Lemma 1 we know that they correspond to the sequences u_a, t_a, w_a and u_b, t_b, w_b. According to the rules of invisible firing, the transitions u_a and u_b can be fired simultaneously. So can w_a and w_b. Since t_a and t_b can also be fired simultaneously, $\tau_{a,b}$ is possible.
□

Theorems 3 and 4 can easily be extended to cover possibilities of more than 2 simultaneous firings and, respectively, multiple handshakes of more than 2 single handshakes. To consider, for instance, 3 simultaneous firings t_a, t_b, t_c, we can take them as two pairs, t_a, t_b and t_b, t_c. Correspondingly, a handshake $\tau_{a,b,c}$ can be taken as $\tau_{a,b}$ and $\tau_{b,c}$.

Summary of this section
Every closed (restricted by its entire alphabet) CCS composition satisfying Assumptions 1 and 2 can be transformed into a Petri net $Tn(E)$ whose visible transitions model handshakes of E. Firing of t_a corresponds to the handshake τ_a. Firing $t_{a_1}, t_{a_2}, \ldots, t_{a_k}$ simultaneously corresponds to $\tau_{a_1,a_2,\ldots,a_k}$.

4 Deadlock detection

Equivalence between handshakes of a closed CCS system E and firing of visible transitions of its transformation into a Petri net $Tn(E)$ makes it possible to use established Petri nets techniques to analyse the system. One such technique is generation of so-called reachability tree of the net (cf. [10]). The tree gives information on possible firing sequences, including their loops and dead ends. In cases of so-called safe nets (the number of tokens in any place never exceeds 1), the tree represents the net language, i.e. all possible firing sequences.

Construction of the reachability tree may be described as follows. Markings of the net are placed in the tree nodes, its initial marking – in the root. Children of the root are generated by firing transitions that are enabled under the initial marking. Every child is labelled by the corresponding transition name, and

contains the new marking. Then, it becomes a parent of other nodes the same way. However, if the new marking is the same as some other marking already present in the tree, the new node is marked as a repetition of the other node. Instead of its children, a pointer to the other node is attached to it. If marking in some node does not allow any firing, the node becomes a leaf of the tree, representing deadlock. If markings along some path from the root "grow", i.e. token numbers in certain places are larger, and in other places not fewer than in previous markings along the path, the larger numbers are replaced by the symbol ω (may be understood as infinity). This causes some loss of information about possible firing sequences, but allows the tree to be kept finite.

For our purposes, the tree generation algorithm summarized above has been modified in two ways. First – only visible transitions are taken into the consideration. Practically, it means that at the very beginning, and after firing of a visible transition, all invisible transitions are repeatedly fired for as long as possible (cf. the rules of invisible firing of the previous section). Only then, the new marking is recorded in the new node. Second – not single, but multiple firings are considered, i.e. for a given marking, instead of single transitions, sets of simultaneously fireable transitions are fired, and become labels of the corresponding nodes. An order in which transitions of such a set are fired is arbitrary (cf. Lemma 2 of the previous section).

The second modification is not new (eg. [7, 13]). In [7], bags of transitions are fired. They are bags rather than sets because of so-called autoconcurrency, i.e. assumption that one transition may be fired as many times in one multiple firing as marking of its input places allows. In our case, however, bags of net-transitions would make rather little sense. Assuming that one communication action between two processes takes a finite time, and one process can engage in one such action at a time, we have to exclude any possibility of one process engaging in more than one communication action at the same time. Thus, sets of transitions, instead of bags, are objects of our analysis.

Upon completion, the reachability tree may be traversed from its root along various paths. A sequence of label sets along each path is interpreted as a sequence of multiple handshakes possible for the CCS composition.

The transformation Tn and the reachability tree construction have been implemented as Gofer [5] programs that are included in a package ccsdd.tar.gz available by anonymous ftp from ftp.cse.unsw.edu.au:pub/users/jacek. The package facilitates:

- syntax checks and conversion of CCS specifications into Petri nets,
- conformance checks of CCS specifications with Assumptions 1, 2, and 3,
- construction of reachability trees with multiple firings, for CCS specifications satisfying all 3 assumptions,
- construction of reachability trees with single firings, for CCS specifications satisfying Assumptions 1 and 2 only.

Construction of reachability trees finishes as soon as deadlock is detected. In cases of no deadlock, complete trees are produced.

The package also includes four examples – CCS specifications and log files of the program runs.

Example 1 specifies the problem of five philosophers (eg. [3]) as a composition of 10 processes – 5 philosophers each behaving the same way, and 5 forks. Its analysis is completed very quickly. The program finishes after the first step of the algorithm, when a set of five transitions is fired. The set corresponds to a multiple handshake between philosophers and forks – each philosopher picks up the left fork. Then, they wait for the other fork forever.

Example 2 is a modification of the same problem, in which one of the philosophers behaves in a different way to the others. He picks up his right fork first. Such a composition is always in progress (no deadlock). The reachability tree constructed for this example has 55 nodes, and represents all possible sequences of multiple handshakes under the maximum degree of parallelism (at every step, as many single handshakes together as possible). Such a degree of parallelism may be understood as the most crowded situations at the philosophers table, i.e. when they get into the way of one another in the most obstructive manner.

Examples 3 and 4 are two different versions of a scheduler (cf. [6] p. 113, Specification of a Simple Scheduler). The purpose of the scheduler is to allow a number of tasks to run in parallel, but to ensure that they are started in a certain order. They may finish in any order, and then, restarted, but again in the same order as before. The scheduler has also been subject to analysis with the use of Concurrency Workbench [1]. The maximum number of tasks with which Concurrency Workbench could cope was 7. Our Example 3 is such a scheduler for 4 tasks. Its specification satisfies Assumptions 1 and 2 only, thus precluding the use of multiple firings in the reachability tree construction. The tree has 239 nodes, i.e. the scheduler has 239 possible states.

Example 4 is a modification of Example 3. The modified scheduler satisfies all 3 Assumptions, and consequently, allows for multiple transition firing in the reachability tree construction. It also allows 9 tasks to be run in parallel (beyond possibilities offered by Concurrency Workbench). The tree has only 47 nodes, far from the number of possible states of the scheduler. Nevertheless, it proves that the scheduler is deadlock free.

5 Conclusion

The scheduler example of the previous section illustrates well how crucial, in verification of CCS compositions for deadlocks, the question of single versus multiple handshakes may be. If we can use multiple handshakes, in other words, if a given CCS composition E satisfies Assumptions 1, 2 and 3 of section 2, then:

- the composition E can be transformed into a finite pure Petri net $Tn(E)$ where transition firing models a handshake, and where multiple transition

firing models a multiple handshake,

- deadlocks can be detected by an attempt to generate a reachability tree of $Tn(E)$ with multiple transition firings (the attempt can be abandoned as soon as deadlock is detected),
- deadlocks can be disproved by construction of a complete reachability tree of $Tn(E)$ with multiple transition firings.

If the composition E satisfies Assumptions 1 and 2, but not 3, it can still be transformed into a Petri net $Tn(E)$, and analysed by reachability tree construction for $Tn(E)$. However, no multiple transition firing can be used for that purpose. That nullifies the advantage of Petri nets over other tools of analysis, eg. automata, of being able to model possible simultaneity of handshakes in E.

Acknowledgement

The author thanks John Potter for his valuable suggestions on how to improve the manuscript.

References

[1] R. Cleveland, J. Parrow, B. Steffen: The Concurrency Workbench: A Semantics-Based Tool for Verification of Concurrent Systems, ACM TOPLAS 15, No. 1, 36-72, 1993.

[2] Droste M.: Concurrent Automata and Domains, Intern. J. of Foundations of Comp. Sc. 3, No. 4, pp. 389-417, 1992.

[3] C.A.R. Hoare C.A.R.: Communicating Sequential Processes, Prentice-Hall, 1985

[4] Inmos Limited: Occam 2 reference manual, New York: Prentice-Hall 1988

[5] M. Jones: Gofer, Functional Programming Environment, v. 2.30a, available from nebula.cs.yale.edu:/pub/haskell/gofer, 1994.

[6] R. Milner: Communication and Concurrency. Prentice-Hall, 1985.

[7] M. Mukund: Petri Nets and Step Transition Systems, Intern. Journal of Foundations of Computer Science 3, No. 4, pp. 443-478, 1992.

[8] E.-R. Olderog: Nets, Terms and Formulas, CUP 1991.

[9] J. Olszewski: Non-Interleaving Semantics for CCS and Fast Deadlock Detection, SCS&E Report 9317, Univ. of New South Wales, Australia, Dec 1993.

[10] J.L. Peterson: Petri Net Theory and the Modeling of Systems. Prentice-Hall 1981.

[11] I.C. Pyle: The Ada Programming Language, Prentice Hall, 1981.

[12] D. Taubner: Representing CCS Programs by Finite Predicate/Transition Nets, Acta Informatica 27, 533-565, 1990.

[13] A. Valmari: Stubborn Sets for Reduced State Space Generation, in (Rozenberg G., ed.) Advances in Petri Nets 1990, LNCS 483, Springer 1991.