SCS&E Report 9317
December, 1993

Non-Interleaving Semantics for CCS and Fast Deadlock Detection

Jacek Olszewski

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
THE UNIVERSITY OF NEW SOUTH WALES

SCIENTIA

MANU ET MENTE

Abstract

This paper proposes a non-interleaving semantics for CCS, and defines a class of CCS compositions for which interleaving and non-interleaving semantics are equivalent. It also presents a fast software tool for analysis of CCS compositions under non-interleaving semantics. The tool employs Petri net techniques with multiple transition firings.

Contact: jacek@cse.unsw.oz.au

1 Introduction

The first step of analysis of parallel system specifications is usually an attempt to detect or to disprove deadlocks (cf. for instance examples of specifications in [1]). It is also the most time consuming function of any available analysis tool set. Known deadlock detection techniques suffer from so-called state space explosion problem. The problem lies in a very large number of states in which the system under analysis can be found. Any reduction in the number of states may significantly enlarge the class of systems that can be analysed automatically. So far, only moderately sized systems have been analysed successfully with the use of existing tools.

Recently, new analysis techniques have been proposed, that take advantage of concurrency with which state transitions can take place. For instance, Valmari [15] proposes such a technique for analysis of Petri nets. It requires determination of which transitions of a given net can be fired simultaneously, without any change to the terminal states of the net and to the existence of non-termination. The possibility of firing a transition simultaneously with others reduces the number of states by half. In another example of such an approach, Droste [2] also proposes models of parallel systems where interactions between components take place simultaneously.

For CCS, the same idea can be expressed as a possibility of simultaneous actions performed by different components of a given composition (cf. eg. [4, 5]). However, the main aim of the cited works is to extend CCS with information concerning required timing of process interactions, so as to allow specification and modelling of real time systems. The aim of this work is to provide a tool for fast analysis of untimed CCS specifications under non-interleaving semantics.

The next section presents the formalism of CCS as defined by Milner [8], and its transitional non-interleaving semantics. Section 3 gives a proof of equivalence between interleaving and non-interleaving semantics for a certain class of CCS compositions. Section 4 describes a Petri net based analysis tool for CCS specifications of parallel systems. Section 5 presents some examples of its use. Section 6 concludes the paper.

2 CCS and its transitional non-interleaving semantics

The notation for CCS specifications and its transitional semantics is taken directly from [8]. 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. For the purposes of this paper, only the concepts of a prefix, summation and composition have to be taken into the consideration:

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 ($E_{i_1}|E_{i_2}|\ldots$)

Milner [8] explains why (2) has to be the sum of all expressions E_i. Here, for reasons explained below, (2) allows only choices guarded by input actions (c.f. the next section).

The transition rules are as follows:

$$\textbf{Act}\ \frac{}{a.E \xrightarrow{a} E}$$

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

$$\textbf{Com1}\ \frac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F}$$

$$\textbf{Com2}\ \frac{F \xrightarrow{a} F'}{E|F \xrightarrow{a} E|F'}$$

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

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

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

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

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.

As an example, let us consider $I = \{1..4\}$ and the following composition:

$$C = a.E_1|\overline{a}.E_2|b.E_3|\overline{b}.E_4$$

By the rules **Com3** and **Com4**, we may have:

$$C \xrightarrow{\tau} E_1|E_2|b.E_3|\overline{b}.E_4$$

or

$$C \xrightarrow{\tau} a.E_1|\overline{a}.E_2|E_3|E_4$$

or

$$C \xrightarrow{\tau} E_1|E_2|E_3|E_4$$

τ does not necessarily represent all currently possible handshakes in the whole composition. It only represents some number of them, from 1 to the maximum. We may, however, require that it represent all, if we want maximum parallelism at every step of the composition evolution.

Before other examples are considered, the notion of deadlock should be clarified. In [8] deadlock is said to be possible, if the composition is not ready for all actions that might be required by its environment. On the other hand, in [1] two not identical notions of deadlock are given. One describes deadlock as a state in which the composition cannot perform any actions. In the other, a state of the composition is said to be deadlocked, if it can engage in τ but is never capable of any observable actions. Let us unify the notion of deadlock by taking the environment into the consideration as a part of the composition. Hence, interactions between the environment and the system under analysis take also the form of τ actions. In other words, we shall consider compositions that are always restricted by the set of all actions in all components. Deadlock in such compositions is a state in which no τ action is possible. From the point of view of the environment, absence of deadlock defined that way does not necessarily mean that interaction with the system is always possible. It is always possible, if the system considered without the environment can be proved to guarantee nothing but deadlock.

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 another 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:

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

As this example shows, multiple transitions may exclude certain interleavings of single transitions, thus limiting behaviour of the composition to a fewer possible sequences. Moreover, there may be sequences of single transitions that can not be represented by multiple transitions or their sequences. Generally, if we are interested only in possible effects of a given composition, answers can be provided by sequences of single transitions, i.e. analysis of the composition under interleaving semantics. If, however, we are also interested in possible behaviours of the composition, including questions about transitions that can take place simultaneously, we should analyse the composition under non-interleaving semantics.

Since no timing information is assumed to be provided, no constraints on the degree of parallelism can be imposed. We have to consider all, from 1 to the maximum at every possible step of the composition evolution. With no such information, we can not be sure that possibly simultaneous transitions take place actually simultaneously. Even if we are interested only in how many transitions are possible simultaneously, we can not assume maximum parallelism all the time. A single transition at some step or steps may result in a larger number of simultaneous transitions at some later step, than in the case of maximum parallelism at every step.

However, in the next section, 3 other constraints upon CCS compositions are proposed to make both semantics, interleaving and non-interleaving under maximum parallelism, equivalent as far as possibility of or freedom from deadlock is concerned. The constraints exclude specifications like the example D above.

3 Behavioural equivalence

This section shows equivalence of interleaving and non-interleaving semantics for a certain class of compositions, with respect to the possibility of or freedom from deadlock. To formulate and prove a set of necessary theorems, 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 $a_1, a_2, \ldots, a_n$.

To indicate the maximum degree of parallelism, we shall use:

Definition 1 $\tau_{\{a_1,a_2,\ldots,a_n\}}$

as a multiple handshake over $a_1, a_2, \ldots, a_n$, with which no other handshakes can be performed simultaneously.

The class of the compositions considered here is assumed to satisfy the following constraints. The first has already been mentioned in the previous section:

Assumption 1. Choices are guarded by input actions.

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.

The next constraint concerns the use of action and co-action names in processes:

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.

The next constraint concerns the way processes are defined:

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

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

The purpose of this section is to prove its main theorem stating equivalence between interleaving and non-interleaving semantics for any composition that satisfies Assumptions 1–3. To prove the theorem, a lemma, corollary, and another theorem are needed.

Lemma 1 concerns cycles that may be performed by components of the composition in a sequence of single handshakes, and states their inevitable coincidences.

Lemma 1 For a deadlock free composition E, every possible sequence of single handshakes can be extended so as to include one after which two or more components of E, involved in the handshakes, have completed their respective cycles.

Proof

Let us assume that there is a sequence of single handshakes for $E = P|Q|R$, such that two components of E, say P and Q, can never coincide one with the other when beginning their respective cycles, except for the start of the whole composition. Let

$$\begin{aligned} P &= x \ldots y.P, \quad x, y \in Act \\ Q &= z \ldots u.Q, \quad z, u \in Act \end{aligned}$$

and let R be the rest of the composition. The no coincidence requirement may be presented as a choice of two following sequences of single handshakes: either

$$\tau_x, \ldots, \tau_u, \ldots, \tau_z, \ldots, \tau_y, \ldots$$

or

$$\tau_z, \ldots, \tau_y, \ldots, \tau_x, \ldots, \tau_u, \ldots$$

It is easy to see that neither of the sequences is possible; no component can finish its cycle before it is started.
□

Notice that Lemma 1 does not guarantee the composition to be cyclic. Eg. for the composition $E = P|Q|R|S$, where

$$\begin{aligned} P &= a.P \\ Q &= \overline{a}.\overline{b}.c.Q \\ R &= b.R + d.R \\ S &= \overline{d}.S \end{aligned}$$

in any sequence of handshakes, τ_a and τ_b may occur only once.

The example illustrates the following corollary:

Corollary 1 For a deadlock free composition $E = \prod\{E_i : 1 \leq i \leq n\}$ there is a minimum number $m \leq n$ of components without involvement of which E cannot progress indefinitely.

□

Every composition E can be considered as a combination of 2 compositions:

Definition 2 $E = L_E | D_E$

where L_E (live part of E) includes all components of E that are necessary for its indefinite progress, and D_E (dead part of E) includes all other components of E. For $E = L_E$, i.e. for a composition that has no dead part, we may assume $D_E = \mathbf{0}$. Analogously, for $E = D_E$, i.e. for a composition that has no live part, we assume $L_E = \mathbf{0}$.

Notice that for a deadlock free composition E, handshakes involving components of D_E are irrelevant to the composition progress. They may, but they do not have to occur in any sequences of handshakes.

Theorem 1 establishes correspondence of a multiple handshake to a sequence of single handshakes:

Theorem 1 $\tau_{a_1,a_2,\ldots,a_n}$ is equivalent to single handshakes $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_n}$ performed in any order.

Proof

Let us consider the composition:

$$\prod\{P_i | Q_i : 1 \leq i \leq n\} | R$$

where

$$\begin{aligned} P_i &= a_i.P_i' + b_i.P_i'' + P''' \\ Q_i &= \overline{a_i}.Q_i', \qquad 1 \leq i \leq n \end{aligned}$$

and where R denotes other components of the composition, and $\tau_{a_1,a_2,\ldots,a_n}$ denotes the multiple handshake over $a_1, a_2, \ldots, a_n$. Each of τ_{a_i} is a possible transition for the composition, and it excludes τ_{b_i} and other handshakes, if they are possible as well. Consequently, they can be performed one by one in any order, leading to the same effect as $\tau_{a_1,a_2,\ldots,a_n}$, i.e. to the composition:

$$\prod\{P_i' | Q_i' : 1 \leq i \leq n\} | R$$

□

Theorem 2 establishes equivalence of interleaving and non-interleaving semantics with regard to presence or absence of deadlocks.

Theorem 2 If there is a sequence of single handshakes that leads to deadlock, there is also a sequence of multiple handshakes leading to deadlock.

Proof

In the first 2 steps of the proof, pairs rather than all of simultaneously possible handshakes are considered. Step 3 provides generalization of the previous 2 steps for all simultaneously possible handshakes.

Let $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_n}$ be the sequence leading a composition E to deadlock.

Step 1 Suppose that for some $i : 1 \leq i < n, \tau_{a_i}$ and τ_b are possible together. If there is no mutual exclusion between τ_b and any of $\tau_{a_j}, i < j \leq n$, τ_b would also be possible after τ_{a_n}, thus proving that the sequence $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_n}$ does not lead E to deadlock.

Step 2 Suppose that together with $\tau_{a_i}, 1 \leq i < n$, τ_b becomes possible and it excludes $\tau_{a_j}, i < j \leq n$. After $\tau_{a_{i-1}}$, the composition can be presented as $P|Q|R|S|T$ where

$$\begin{aligned}
P &= \overline{a_i} \ldots \overline{a_j}.P' \\
Q &= \overline{b}.Q' \\
R &= a_j.R' + b.R'' \\
S &= a_i.S'
\end{aligned}$$

and where T denotes the rest of the composition. T may itself be a composition. R and S may have also other alternatives. However, their presence does not change the proof.

Let P^0, Q^0, R^0, S^0 denote the components of E that have evolved into P, Q, R, S by the sequence of handshakes $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_{i-1}}$, correspondingly. Let also E^{a_i} and $E^{a_i,b}$ denote the composition after τ_{a_i} and $\tau_{a_i,b}$:

$$\begin{aligned}
E^{a_i} &= \ldots \overline{a_j}.P'|\overline{b}.Q'|a_j.R' + b.R''|S'|T \\
E^{a_i,b} &= \ldots \overline{a_j}.P'|Q'|R''|S'|T
\end{aligned}$$

Now, it remains to prove that there is a sequence of single handshakes that leads $E^{a_i,b}$ either to deadlock or to E^{a_i} that in turn is lead to deadlock by the sequence $\tau_{a_{i+1}}, \ldots, \tau_{a_j}, \ldots, \tau_{a_n}$. In other words, we have to prove that $E^{a_i,b}$ is not deadlock free.

Step 2.1 Suppose $E^{a_i,b}$ is deadlock free, and assume that it is possible for $E^{a_i,b}$ to proceed indefinitely without R'' completing the current cycle of R^0 (Assumption 3). It means no further involvement of R^0 in progress of E. In other words, R^0 is a component of D_E (cf. Definition 2). According to Assumption 2, P^0 and Q^0 are the only components of E where $\overline{a_j}$ and $\overline{b}$ may occur. So, P^0 and Q^0 are also components of D_E. The same applies to S^0 because it is the only component where a_i may occur.

Therefore, the handshakes τ_{a_i} and τ_{a_j} are irrelevant in the sequence leading E to deadlock. In other words, if $\tau_{a_{i+1}}, \ldots, \tau_{a_{j-1}}, \tau_{a_j}, \tau_{a_{j+1}} \ldots, \tau_{a_n}$ leads E^{a_i} to deadlock, then $\tau_{a_{i+1}}, \ldots, \tau_{a_{j-1}}, \tau_{a_{j+1}} \ldots, \tau_{a_n}$ leads $E^{a_i,b}$ to deadlock as well.

Step 2.2 Suppose R'' completes the current cycle of R^0, and R^0 evolves into R again. If τ_{a_j} was not performed on the way, now it is possible, as required by the original sequence of handshakes leading E to deadlock. However, the new composition is not necessarily the same as E^{a_i}. Some of its other involved components, including those of T, might have not completed their cycles (cf. Corollary 1). If so, the sequence leading the new composition to deadlock may be shorter than $\tau_{a_j}, \tau_{a_{j+1}}, \ldots, \tau_{a_n}$.

If, however, τ_{a_j} was performed and that did not lead $E^{a_i,b}$ to deadlock, it leads to E with either all its components having completed their cycles, or with some components not able to complete their cycles. In the first case, $E^{a_i,b}$ can be led to E^{a_i} by the sequence $\tau_{a_{i+1}}, \ldots, \tau_{a_j}, \ldots, \tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_i}$. In the other case, E has a dead part, D_E (cf. Definition 2), to which components that have not completed their cycles belong. Since handshakes involving components of D_E are irrelevant, we may delete them from the sequence $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_i}$, thus allowing E to be led through it again without involving components of D_E. Therefore, in this case $E^{a_i,b}$ can be led to E^{a_i} by the sequence $\tau_{a_{i+1}} \ldots, \tau_{a_j}, \ldots, \tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_i}$ from which irrelevant handshakes have been deleted.

Step 3 As proven in Steps 1–2, the sequence $\tau_{a_1}, \tau_{a_2}, \ldots, \tau_{a_n}$ may be transformed into a sequence of double handshakes wherever possible, eg. $\tau_{a_1}, \ldots, \tau_{a_i,b}, \ldots$. From Theorem 1 we know that every double handshake corresponds to a pair of single handshakes performed in any order. So, the sequence with double handshakes may be transformed back into another sequence of single handshakes, eg. $\tau_{a_1}, \ldots, \tau_{a_i}, \tau_b, \ldots$. The new sequence may be considered again as in Steps 1–2, resulting in another sequence with double handshakes, eg. $\tau_{a_1}, \ldots, \tau_{a_i,c}, \tau_b, \ldots$. Combined with previous double handshakes, the sequence may now include triple and quadruple handshakes, eg. $\tau_{a_1}, \ldots, \tau_{a_i,c,b}, \ldots$. The process can be repeated until multiple handshakes, $\tau_{\{h_1,h_2,\ldots,h_m\}}$, are formed, i.e. handshakes with which no other handshakes are simultaneously possible.

□

Analysis of any non-trivial compositions requires a tool – program capable of tracking possibly cyclic sequences of transitions, including multiple transitions, and of detecting deadlocks. The next section presents such a tool based on transformation of CCS compositions into Petri nets, and generation of the nets reachability trees [11].

4 Analysis of CCS compositions

The tool described here is a modification and extension of a program developed originally for analysis of CSP[6] specifications of parallel systems under interleaving semantics [10]. Reasons for using Petri net techniques in both cases, CSP and CCS, lie in their ability to generate reachability trees as finite representations of possibly infinite languages of given nets. A CCS or CSP specification can be transformed into a Petri net [13]. Then, the net reachability tree can be generated and interpreted as a representation of all possible sequences of single or multiple transitions that may take place in the life of the specified system.

Confusingly, the word transition is also used in Petri net terminology to mean one of the two possible kinds of nodes, places and transitions out of which Petri nets are built. To avoid any confusion, we shall use the term net-transition for a node of the net, and transition for a change of state of the CCS composition.

A Petri net is a directed graph built out of nodes, places and net-transitions, and arcs that link places with net-transitions and net-transitions with places. Each place can hold a number of tokens, that may change when the net-transition linked to the place is fired. Firing a net-transition is possible when all its input places have at least one token each, and means taking one token from every input place and putting one token into every output place of the net-transition.

The technique used for transformation of CCS compositions into Petri nets is the same as for transformation of CSP specifications. Every composition of k processes is transformed into a net-transition whose outgoing arcs lead to k output places, one for each process, as in Fig. 1.

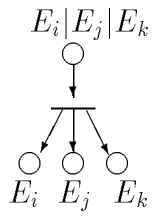

Fig. 2 Composition

Every process is transformed into a thread of arcs leading alternately through places and net-transitions. The net-transitions may also have other incoming and outgoing arcs corresponding to the process input and output actions. For every action and co-action name $a \in A$ and $\overline{a} \in \overline{A}$, there are two places in the net, $a0$ and $a1$, linked by arcs with net-transitions that correspond to the respective input and output actions of the processes. Every action a is transformed into a net-transition with two additional arcs: coming from $a0$, and going to $a1$ (cf. Fig. 2). Every co-action $\overline{a}$ is transformed into a sequence of net-transition – place – net-transition with two additional arcs: from the first net-transition to $a0$, and from $a1$ to the other net-transition, as in the Fig. 2.

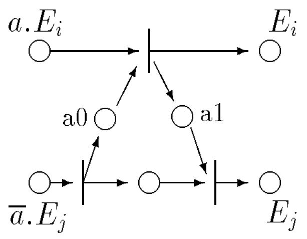

Fig. 2 Communication

Every choice between k processes each of which is guarded by an action, is transformed into a place whose outgoing arcs lead to k net-transitions, one for each action - the guard of the corresponding process, as in Fig. 3.

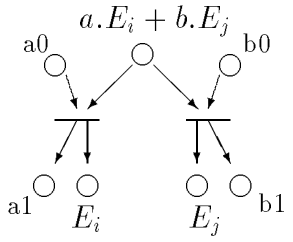

Fig. 3 Choice

Full Petri net representation of a given CCS composition reflects not only the composition structure, but also its naming. Certain places of the net are named after components and actions of the composition, as shown in all 3 figures above. There is one standard place in every net, **0** (stop), from which no arcs lead to any transitions. If not used in the composition definition, **0** remains an isolated place of the corresponding net. Each component other than **0** is transformed into a place with the component name, and an arc leading from that place to the net fragment corresponding to the component definition. References to component names on the right hand side of the definition correspond to arcs leading to places named after the components.

Such a name preserving transformation CCS $\rightarrow$ Petri net facilitates presentation of analysis results in terms of original names used in the composition definition. On the other hand, it may require additional net-transitions to comply with the rule that arcs link places with net-transitions and net-transitions with places only.

Once the transformation of the composition into a Petri net is completed, its reachability tree can be generated. Generation of the tree requires certain initial marking of the net. In our case, it is one token put into the place corresponding to the whole composition. This corresponds to the root of the tree (initial node of the graph). Its children are generated by firing net-transitions that are possible to fire under the initial marking. A node created by firing of a net-transition is labelled by the transition name, and contains the new marking. The new node, then, 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 behaviours of the net, but allows the tree to be finite.

In addition to the normal firing rule, we may also use the rule that all 3 net-transitions representing one handshake of CCS processes, $a.E_i$ and $\overline{a}.E_j$, should be fired in sequence, i.e. with no interleaving with firing of other net-transitions. In effect, such a triple firing may result in creation of one new node of the tree, labelled by the name a.

Upon completion, the tree may be traversed from its root along every possible path. A sequence of labels along one path forms a representation of one or more words of the net language, and consequently, a representation of one or more possible sequences of silent actions in the CCS composition. It corresponds to interleaving semantics for both, the CCS composition and its corresponding Petri net.

As mentioned in the introduction, an analogue to non-interleaving semantics has recently been proposed for Petri nets [15]. Apart from sequences of net-transitions that can be fired one after another, sequences of bags (multisets) of net-transitions are investigated (cf. eg. [9]). Each bag is a collection of net-transitions that can be fired simultaneously. It is a bag rather than a set because of so-called autoconcurrency, i.e. assumption that one net-transition may be fired as many times in one multiple firing as marking of its input places allows.

For our purposes, 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 net-transitions, instead of bags, are objects of our analysis.

The technique of transformation of CCS compositions into Petri nets remains the same. Generation of reachability trees has to be modified. Instead of firing of single net-transitions leading to and labelling new nodes of the tree, sets of net-transitions are fired and used as labels of corresponding nodes of the tree. Depending on the required degree of parallelism, from none (interleaving semantics), through a full range of parallelism, to maximum only at each step (both non-interleaving semantics), the sets are either singular, or they are all possible combinations from one to all net-transitions simultaneously fireable at a given step, or they are just sets of all net-transitions fireable at a given step, correspondingly. Multiple firing means taking one token from every input place, and putting one token into every output place of every net-transition involved. It may mean taking more than one token from one place, and/or putting more than one token into one, possibly different place of the net.

The tool of analysis has been developed as a Miranda [14] program that accepts CCS compositions specified in a notation that requires the words: In for actions, `Out` for co-actions, `Or` for summation, `Par` for composition, and `Proc` for the use of process names on the right hand sides of process definitions. Process and action names are arbitrary strings. Processes – components of compositions – are defined with the use of the function `procdef::[char]->process *` where `process *` is an algebraic type defining possible structures of compositions according to the syntax of CCS shown in the previous section. For possible future uses of the program, `*` stands for a type of values that can be passed between processes in a composition.

The program transforms a given composition into the corresponding Petri net as described above. Then, it builds its reachability tree according to the required degree of parallelism: 1, or full range, or maximum at every step. The resulting tree is reduced by removal, wherever possible, of nodes with labels other than sets of action names. Their removal causes no loss of information about possible behaviour of the composition. Wherever possible means removal that does not change the structure of the tree, i.e. preserves all its branches, cycles, and dead ends. In other words, there may be nodes with no action names in their labels, removal of which would destroy the tree.

Upon completion, the tree is printed one line per node. Each line contains: the node number, the set of action names found in its label, and 'what next'. If the node label does not include any action names, it is printed as an empty set {}. 'What next' may be of the following forms:

`END` denoting a deadlock, or

`go to i,j,...,k` indicating that nodes `i,j,...,k` are children of this node, or

`back to i` indicating that this node is a part of a cycle, and the `i`th node is the next in the cycle.

In addition, the line may contain the symbol `*` (ω in the Petri nets terminology) followed by some process names. This indicates that the named processes may be instantiated uncontrollably many times in the life of the composition. Occurrence of such a line on output indicates that Assumption 2 of section 3 is not satisfied.

The next section presents some examples of CCS compositions and results of their analysis using the program described above.

5 Examples

The first example, analysis of 5 philosophers (cf. eg. [6]), illustrates a very efficient way of deadlock detection. The composition is defined as follows:

$$\begin{aligned} COL &= \prod\{P_i : i \in \{0..4\}\} \mid \prod\{F_i : i \in \{0..4\}\} \\ P_i &= \overline{2i}.\overline{2i+1}.2i.2i+1.P_i \\ F_i &= 2i \ominus 1.\overline{2i \ominus 1}.F_i + 2i.\overline{2i}.F_i \end{aligned}$$

where $\ominus$ means subtraction modulo 10, P_i represents the ith philosopher, and F_i – the ith fork.

There are 10 action and 10 co-action identifiers: $0, 1, \ldots, 9$ and $\overline{0}, \overline{1}, \ldots, \overline{9}$. A pair j and $\overline{j}$ is used by $P_{j \div 2}$ and $F_{(j \oplus 1) \div 2}$ only, thus satisfying Assumption 2 of section 3.

As expected, the analysis result is as follows:

```
0    {}            go to   1
1    {6 8 4 2 0}    END
```

i.e. all five handshakes, 0,2,4,6,8, are possible at once, or in any order, and they lead to a deadlock. The tree building part of the program for this problem has the complexity order 0.

More interesting example is to see how the program copes with a modification of the 5 philosophers problem in which one philosopher behaves differently, i.e. picks up forks in the reverse order. Let it be P_0:

$$P_0 = \overline{1}.\overline{0}.0.1.P_0$$

The rest of the composition remains the same.

The maximum degree of parallelism means the most crowded situations at the philosophers table, i.e. when they get into the way of one another in the most obstructive manner. The result of the analysis is a graph, traversals of which give all sequences of what can happen when all philosophers are present. 65 lines of what the program prints are too many to be reproduced here. The beginning and the end of the printout are as follows:

```
0    {}            go to   1 64
1    {6 8 4 1}     go to   2 63
2    {7}           go to   3
3    {6}           go to   4
4    {7}           go to   5 62
```

```
5    {5}           go to   6
.
.
.
58    {2 4}         back to 27
59    {6}           go to   60 61
60    {1 7}         back to 2
61    {2 7}         back to 40
62    {6}           back to 1
63    {0}           back to 48
64    {6 8 4 2}     back to 39
```

As expected, no deadlock possibility is reported, and the maximum number of simultaneous handshakes is 4. The graph size, 65 nodes, is encouragingly small as for a problem of this complexity. Under interleaving semantics, for much smaller problem of 3 philosophers, the graph has 135 nodes. Under non-interleaving semantics at the full range of parallelism (from none to maximum at every step), it has 240 nodes.

6 Conclusions

The class of CCS compositions for which interleaving and non-interleaving semantics have been proved equivalent is limited by 3 constraints (cf. Assumptions 1–3 of section 3). The first 2 of them are not new. The requirement that choices are guarded by input actions can be found eg. in OCCAM [3] and ADA [12]. One action name in definitions of only 2 composition components is also required by CSP[6] (one channel of communication may be used by 2 processes only). The latter is a serious limitation, since it practically excludes systems where processes are instantiated dynamically. Assumption 3 – processes are cyclic – may seem natural. However, it excludes systems where processes need to interact for the system initialization purposes, before they enter their 'normal' cycles. On the other hand, the reduction of state space under those constraints makes it possible to analyse automatically systems of significantly greater size and intricacy than it has been feasible so far.

The work described in this paper should be continued in 2 directions, theoretical and practical. The theoretical direction should aim at relaxing the constraints imposed upon CCS specifications to which fast deadlock detection techniques can be applied. For instance, it should be investigated whether Assumptions 1–3 could be reformulated and relaxed by an assumption that compositions were connected [7]. Two processes are said to be connected, if the same action or co-action name occurs in their definitions. A composition is said to be connected, if its components are connected with one another. For practical reasons, this assumption should be satisfied notwithstanding. Analysis of a system consisting of 2 unconnected compositions that have n and m possible states respectively, means a system with $n \times m$ states, instead of 2 separate systems with $n + m$ states.

In the practical direction, the analyser described here should be equipped with a CCS syntax checker, and a verifier of compliance with assumptions under which analysis makes sense. Ideally, it should become a part of a software set of tools for verification of parallel systems (cf. eg. [1]).

References

[1] Cleaveland R., Parrow J., Steffen B.: The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems, ACM TOPLAS 15, No. 1, pp. 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] Inmos Limited: *Occam 2 reference manual.* New York: Prentice-Hall 1988

[4] Fidge C.J.: Process Algebra Traces Augmented with Casual Relationships, in (K.R. Parker, G.A. Rose, eds.) *Formal Description Techniques 4*, North-Holland, pp. 527-541, 1992.

[5] Fidge C.J.: A Constraint-Oriented Real-Time Process Calculus, in (M. Diaz, R. Groz, eds.) *Formal Description Techniques 5*, North-Holland, pp. 363-378, 1993.

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

[7] Mazurkiewicz A., Rabinovich A., Trakhtenbrot B.A.: Connectedness and Synchronization, Theor. Comp. Science 90, 171-184, 1991.

[8] Milner R.: *Communication and Concurrency.* Prentice-Hall 1989

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

[10] Olszewski J.: Automatic Analysis of CSP Programs Using Petri Nets, Australian Computer Science Communications 15, No. 1, February 1993

[11] Peterson J.L.: *Petri net theory and the modelling of systems.* Prentice-Hall 1981

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

[13] Taubner D.: *Finite Representation of CCS and TCSP Programs by Automata and Petri Nets.* LNCS 369, Springer 1989

[14] Turner D.: Miranda: a Non-Strict Functional Language with Polymorphic Types. In *Functional Languages and Computer Architectures*, Springer 1985.

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