XCCS diagram to CCS script Conversion Algorithm

Krzysztof Balicki

Interdisciplinary Centre for Computational Modelling
University of Rzeszów
Pigonia 1, 35 - 310 Rzeszów, Poland
kbalicki@ur.edu.pl

Abstract. The XCCS modelling language can be viewed as a tool supporting fast and faultless development of CCS scripts or as an independent process calculus based on CCS process algebra. Having in mind the first of possible applications, we present an algorithm that converts XCCS diagrams into CCS scripts. This algorithm takes into consideration the priority of actions in the XCCS diagram and aims to retain the maximum number of original action's names for the sake of readability.

1 Introduction

The XCCS (eXtended Calculus of Communicating Systems) modelling language was formally defined in the paper [1] as a graphical extension of the Milner CCS ([2], [3], [4], [5]) process calculus. It has has been developed to replace the algebraic coding of concurrent systems with graphical modelling.

XCCS models consist of two layers, a graphical and an algebraic one. In the latter we use the subset of CCS process algebra called CSS (Calculus of Sequential Systems) to describe behaviour of individual agents. CSS scripts may only contain the prefix operator ".", the choice operator "+" and parentheses "( )". Moreover, in the timed version of the calculus, the strong choice operator "++" and the delay operator "$" are allowed. The composition, restriction and relabel operators have graphical representations and provide simple description of the modelled system structure [1], [6], [7].

In the paper [1] we introduced simple Basic Conversion Algorithm which is proper for plain XCCS modelling language. The new Universal Conversion Algorithm presented here can be applied to both plain and full XCCS diagrams. Furthermore, it considers the priority of actions and aims to retain the maximum number of original action's names for the sake of readability. First, we show Trivial Conversion Algorithm that does not perform any relabelling of actions yet works correctly for limited class of models. Then, we thoroughly examine and define the properties of those diagrams that cannot be translated by Trivial Conversion Algorithm. We take into account these properties while applying proper relabeling of actions in the new algorithm.

2 Preliminaries

We recall XCCS modelling language by way of example. Let us consider a simple XCCS diagram.

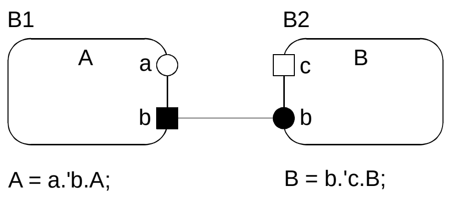

Fig. 1. Example XCCS diagram

The formal definition of the XCCS diagram in Fig. 1 is as follows:

$XCCS = (B, CA, CSS, CR, in, out, v, \rho)$, where:

$\cdot\, B = \{B_1, B_2\}$	- agent blocks
$\cdot\, CA = \{a, b, c\}$	- communication actions, labels
$\cdot\, CSS = \{A = a.\overline{b}.A, B = b.\overline{c}.B\}$	- the set of CSS scripts
$\cdot\, in\colon B \to 2^{CA}$	- input ports assigment
$in(B_1) = \{a\}, in(B_2) = \{b\}$	- labels of input ports
$\cdot\, out\colon B \to 2^{CA}$	- output ports assigment
$out(B_1) = \{b\}, out(B_2) = \{c\}$	- labels of output ports
$OutP = \{(B_1, \overline{b}), (B_2, \overline{c})\} = \{B_1.\overline{b}, B_2.\overline{c}\}$	- output ports
$InP = \{(B_1, a), (B_2, b)\} = \{B_1.a, B_2.b\}$	- input ports
$\cdot\, CR \subseteq OutP \times InP$	- communication relation
$CR = \{((B_1, \overline{b}), (B_2, b))\} = \{B_1.\overline{b} \to B_2.b\}$	- connected ports
$\cdot\, \vartheta\colon OutP \cup InP = P \to \{v, i\}$	- visibility function
$\vartheta(B_1.a) = \vartheta(B_2.\overline{c}) = v$	- visible ports
$\vartheta(B_1.\overline{b}) = \vartheta(B_2.b) = i$	- invisible ports
$\cdot\, \rho\colon B \to CSS, \rho(B_1) = A, \rho(B_2) = B$	- assigment of CSS scripts

The above XCCS diagram can be converted to the following CCS script by the Trivial Conversion Algorithm that does not perform any relabelling of actions.

$$E = (B_1|B_2)\backslash\{b\}$$
$$B_1 = a.\overline{b}.B_1$$
$$B_2 = b.\overline{c}.B_2$$

Please note that the symbols 'b on the diagram and $\overline{b}$ in the script denote the same co-action with the name b. We will use the following notations and definitions:

$\mathcal{A}(CSS_i)$ is the set of all actions of the CSS_i script,	e.g.	$\mathcal{A}(B) = \{b\}$
$\overline{\mathcal{A}}(CSS_i)$ is the set of all co-actions of the CSS_i script,	e.g.	$\overline{\mathcal{A}}(B) = \{\overline{c}\}$
$\mathcal{K}(CSS_i)$ is the set of all agent constants of the CSS_i script,	e.g.	$\mathcal{K}(B) = \{B\}$

Definition 1. *We define the* synchronization relation SR *as follows:*

a) Let CR be a communication relation. For any port $p \in P$ we define the set:

$$CR(p) = \{x \in P \colon (x, p) \in CR \vee (p, x) \in CR\}. \tag{1}$$

If $p \in OutP$, then the set $CR(p)$ contains all input ports connected with the port p.
If $p \in InP$, then the set $CR(p)$ contains all output ports connected with the port p.

b) We define a mapping:

$$SR \colon P \to 2^P, \quad SR(p) = \begin{cases} \{p\} & \text{if } CR(p) = \emptyset \\ CR(p) \cup \bigcup_{x \in CR(p)} CR(x) & \text{if } CR(p) \neq \emptyset \end{cases} \tag{2}$$

c) The synchronization relation SR *is set by SR mapping:*

$$(p_1, p_2) \in SR \subseteq P \times P \overset{\text{def}}{\iff} SR(p_1) = SR(p_2). \tag{3}$$

If an equivalence class contains only one port then this port is isolated and does not communicate with any other port. If an equivalence class contains more then one port then any two complementary ports belonging to the equivalence class are connected one with the other. For the XCCS diagram in Fig.1 the equivalence classes of SR relation are $P/_\sim = \{\{B_1.a\}, \{B_1.b, B_2.\overline{b}\}, \{B_2.\overline{c}\}\}$.

We will use the following symbols:

$V = \{p \in P \colon \vartheta(p) = v\}$	- the set of all visible ports
$I = \{p \in P \colon \vartheta(p) = i\}$	- the set of all invisible ports
$Iso = \{p \in P \colon CR(p) = \emptyset\}$	- the set of all isolated ports
$Int(B_i) = \{a \colon a \in \mathcal{A}(\rho(B_i)) \wedge B_i.a \notin InP\}$	- the set of all inner actions of block B_i
$\overline{Int}(B_i) = \{\overline{a} \colon \overline{a} \in \overline{\mathcal{A}}(\rho(B_i)) \wedge B_i.\overline{a} \notin OutP\}$	- the set of all inner co-actions of block B_i
$Int = \bigcup\limits_{B_i \in B} Int(B_i)$	- the set of all inner actions of the diagram
$\overline{Int} = \bigcup\limits_{B_i \in B} \overline{Int}(B_i)$	- the set of all inner co-actions of the diagram

3 Conflicts and Shadows

We require that the resulting CCS script for a given XCCS diagram is in a Standard Concurrent Form and preserves properties of the diagram, that means:

1. Synchronization may occur only between those pairs of actions and co-actions which correspond to the labels of the ports connected on the diagram.
2. Actions and co-actions which correspond to inner actions are executable.
3. Actions and co-actions which correspond to visible isolated ports are executable.
4. Actions and co-actions which correspond to invisible isolated ports are not executable.

Now we roughly describe the Trivial Conversion Algorithm:

1. The names of CSS scripts become the same as names of corresponding blocks.
2. The ambiguities of names of the agent constants in CSS scripts are eliminated.
3. Output CCS script is a Standard Concurrent Form of all modified CSS scripts.
4. In the restriction set of the main equation of CCS script we place labels of all invisible ports.

If we apply the Trivial Conversion Algorithm to a given XCCS diagram, then it can output a CCS script that does not satisfy requirements. We are able to precisely identify the properties of those XCCS diagrams for which the trivial algorithm does not produce a proper output. We call these properties conflicts and shadows. Shadows yield non executable actions. Conflicts entail unwanted synchronizations whereas expected synchronizations will not occur if the labels of connected ports have different names. Let us define the properties in question.

Definition 2. *We define the* conflict *of ports:*

a) We say that ports $B_i.\overline{a} \in \mathit{OutP}$ and $B_j.a \in \mathit{InP}$ are in conflict*, iff*

$$B_i \neq B_j \ \wedge \ (B_i.\overline{a} \rightarrow B_j.a) \notin \mathit{CR}. \tag{4}$$

We define the set of all conflicting output ports *with the label a:*

$$k.\mathit{OutP}(a) = \{B_i.\overline{a}\colon \exists B_j.a \in \mathit{InP}\colon B_i \neq B_j \wedge (B_i.\overline{a} \rightarrow B_j.a) \notin \mathit{CR}\} \tag{5}$$

We define the set of all conflicting input ports *with the label a:*

$$k.\mathit{InP}(a) = \{B_i.a\colon \exists B_j.\overline{a} \in \mathit{OutP}\colon B_i \neq B_j \wedge (B_j.\overline{a} \rightarrow B_i.a) \notin \mathit{CR}\} \tag{6}$$

Complementary ports are in conflict if they have the same labels and are placed on different blocks and are not connected.

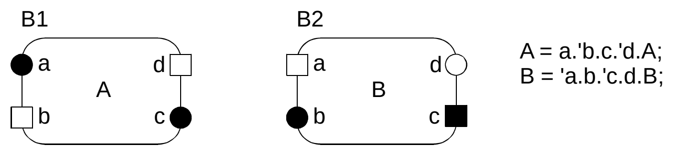

Fig. 2. Conflicts of ports

The conflicts of ports presented in Fig. 2 are as follows: $i.B_1.a$ with $v.B_2.\overline{a}$ and $v.B_1.\overline{b}$ with $i.B_2.b$ and $i.B_1.c$ with $i.B_2.\overline{c}$ and $v.B_1.\overline{d}$ oraz $v.B_2.d$. The letters i and v indicate invisible and visible ports, respectively.

Definition 3. *We define conflicts of ports and inner actions as follows:*

a) We say that port $B_i.\overline{a} \in \mathit{OutP}$ and action a of the block B_j are in conflict, iff

$$B_i \neq B_j \ \wedge \ a \in \mathit{Int}(B_j). \tag{7}$$

b) We say that port $B_i.a \in InP$ and co-action $\overline{a}$ of the block B_j are in conflict, iff

$$B_i \neq B_j \ \wedge \ \overline{a} \in \overline{Int}(B_j). \tag{8}$$

The output port of a given block and the action of another block are in conflict if the label of the output port and the name of the inner action are the same. The input port of a given block and the co-action of another block are in conflict, if the label of the input port and the name of the inner co-action are the same.

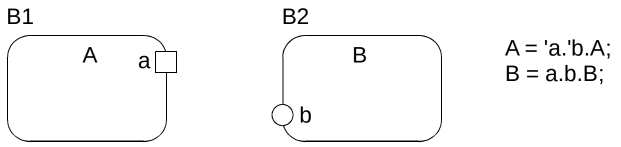

Fig. 3. Port $B_1.\overline{a}$ and action a as well as port $B_2.b$ and co-action $\overline{b}$ are in conflict

Definition 4. *The conflicts among inner action are defined as follows:*

a) The action a of the block B_i and co-action $\overline{a}$ of the block B_j are in conflict, iff:

$$B_i \neq B_j \wedge a \in Int(B_i) \wedge \overline{a} \in \overline{Int}(B_j). \tag{9}$$

We define the set of all blocks for which the action a is conflicting:

$$k.Int(a) = \{B_i \in B \colon a \in Int(B_i) \wedge \overline{a} \in \overline{Int}(B_j) \wedge B_i \neq B_j\} \tag{10}$$

We define the set of all blocks for which the co-action $\overline{a}$ is conflicting:

$$k.Int(\overline{a}) = \{B_i \in B \colon \overline{a} \in \overline{Int}(B_i) \wedge a \in Int(B_j) \wedge B_i \neq B_j\} \tag{11}$$

Inner action and co-action of different blocks are in conflict if they have the same names.

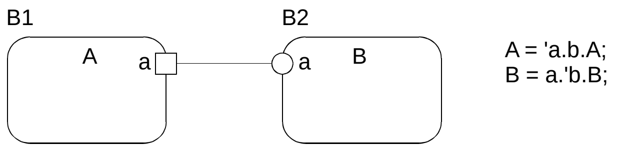

Fig. 4. Action b of the block B_1 and co-action $\overline{b}$ of the block B_2 are in conflict

Definition 5. *The shadows of ports are defined as follows:*

a) We say that port $B_i.\overline{a} \in OutP$ shadows port $B_j.\overline{a} \in OutP$, iff

$$B_i \neq B_j \wedge \vartheta(B_i.\overline{a}) = i \wedge \vartheta(B_j.\overline{a}) = v \tag{12}$$

b) We say that port $B_i.a \in InP$ shadows port $B_j.a \in InP$, iff

$$B_i \neq B_j \wedge \vartheta(B_i.a) = i \wedge \vartheta(B_j.a) = v \tag{13}$$

c) We say that port $B_i.\overline{a} \in OutP$ shadows port $B_i.a \in InP$, iff

$$\vartheta(B_i.\overline{a}) = i \wedge \vartheta(B_i.a) = v \tag{14}$$

d) We say that port $B_i.a \in InP$ shadows port $B_i.\overline{a} \in OutP$, iff

$$\vartheta(B_i.a) = i \wedge \vartheta(B_i.\overline{a}) = v \tag{15}$$

The invisible port of the given block shadows the visible port of another block if both ports have the same label and are not complementary.

The invisible port shadows the visible port on the same block if both have the same label and are complementary.

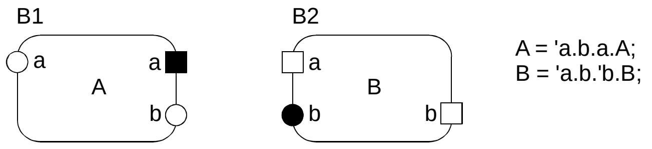

Fig. 5. The shadows of ports

The shadows of ports presented in Fig. 5 are as follows: $i.B_1.\overline{a}$ shadows $v.B_2.\overline{a}$ and $i.B_2.b$ shadows $v.B_1.b$ and $i.B_1.\overline{a}$ shadows $v.B_1.a$ and $i.B_2.b$ shadows $v.B_2.\overline{b}$.

Definition 6. *The shadows related to inner actions are defined as follows:*

a) We say that port $B_i.\overline{a} \in OutP$ shadows co-action $\overline{a}$ of the block B_j, iff

$$B_i \neq B_j \ \wedge \ \vartheta(B_i.\overline{a}) = i \ \wedge \ \overline{a} \in \overline{Int}(B_j). \tag{16}$$

b) We say that port $B_i.a \in InP$ shadows action a of the block B_j, iff

$$B_i \neq B_j \ \wedge \ \vartheta(B_i.a) = i \ \wedge \ a \in Int(B_j). \tag{17}$$

c) We say that port $B_i.\overline{a} \in OutP$ shadows action a of the block B_i, iff

$$\vartheta(B_i.\overline{a}) = i \ \wedge \ a \in Int(B_i). \tag{18}$$

d) We say that port $B_i.a \in InP$ shadows co-action $\overline{a}$ of the block B_i, iff

$$\vartheta(B_i.a) = i \ \wedge \ \overline{a} \in \overline{Int}(B_i). \tag{19}$$

The invisible output port of a given block shadows an inner co-action of another block if this co-action has the same name as the label of that port. The invisible input port of a given block shadows an inner action of another block if this action has the same name as the label of that port.

The invisible output port shadows his own inner action if this action has the same name as the label of the port. The invisible input port shadows his own inner co-action if this co-action has the same name as the label of the port.

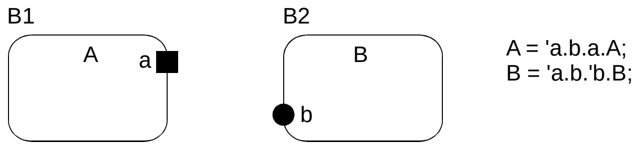

Fig. 6. The shadows related to inner actions

The shadows presented in Fig. 6 are as follows: $i.B_1.\overline{a}$ shadows co-action $\overline{a}$ of the block B_2 and $i.B_2.b$ shadows action b of the block B_1 and $i.B_1.\overline{a}$ shadows action a of the block B_1 and $i.B_2.b$ shadows $\overline{b}$ of the block B_2.

4 Universal Conversion Algorithm

The priority of the relabelling is as follows: labels of visible isolated ports, inner actions and co-actions, labels of visible connected ports, labels of invisible connected ports, labels of invisible isolated ports.

We wish to retain the maximal number of names of visible isolated ports as they correspond to observable actions and co-actions responsible for communication with environment. On the other hand, we are the least interested in retaining names of invisible isolated ports as they correspond to non executable actions and co-actions.

We introduce the following symbols:

$$Actions = \bigcup_{CSS_i \in CSS} \mathcal{A}(CSS_i) \quad \text{the set of all actions contained in CSS scripts}$$

$$\overline{Actions} = \bigcup_{CSS_i \in CSS} \overline{\mathcal{A}}(CSS_i) \quad \text{the set of all co-action contained in CSS scripts}$$

$$Agents = \bigcup_{CSS_i \in CSS} \mathcal{K}(CSS_i) \quad \text{the set of all agent constants contained in CSS scripts}$$

E is the infinite set of labels, where $E \cap Actions = \emptyset \wedge \overline{E} \cap \overline{Actions} = \emptyset$
$\mathcal{C}$ is the infinite set of agent constants, where $\mathcal{C} \cap Agents = \emptyset$.

Please note, the set $\overline{E}$ contains co-labels with the same names as the labels in E. We will consider some finite subset $E' \subset E$ of the set of labels E and some finite set $\mathcal{C}' \subset \mathcal{C}$ of the set of agent constants $\mathcal{C}$. We will use the following notations:

- The ordered pair (B_i, CSS_i) represents the block $B_i \in B$ and the CSS script $CSS_i = \rho(B_i) \in CSS$ assigned to that block.
- The symbol B_i^* denotes the name of the block B_i and the symbol $CSS_i{}^*$ denotes the name of CSS_i script.
- The symbol $\#A$ denotes the cardinality of the set A, where $\#$ binds weaker than operations in the field of sets.

The conversion algorithm is presented here:

1. The names of CSS scripts become the same as names of corresponding blocks. We create a new set CSS' of CSS scripts, i.e. the set of sequences of algebraic equations, in the following manner. For each agent block $B_i \in B$:
   a) If for a pair (B_i, CSS_i) the names of the agent block and the CSS script are the same, i.e. $B_i^* = CSS_i{}^*$, then we add the CSS_i script to the CSS' set.
   b) If for a pair (B_i, CSS_i) the agent constant $B_i^* \notin \mathcal{K}(CSS_i)$, then we replace the name of the CSS_i script and all the $CSS_i{}^*$ agent constants belonging to this script with B_i^* and then we add the B_i^* script to the CSS' set.
   c) If for a pair (B_i, CSS_i) the names of the agent block and CSS script are unlike, i.e. $B_i^* \neq CSS_i{}^*$, but the agent constant $B_i^* \in \mathcal{K}(CSS_i)$, then we replace all the B_i^* agent constants belonging to the CSS_i script with the next agent constant from $\mathcal{C}$ and then we replace the CSS_i script name and all the $CSS_i{}^*$ agent constants belonging to this script with B_i^* and then we add the B_i^* script to the CSS' set.
2. The ambiguities of names of the agent constants in CSS scripts are eliminated. We create a new CSS'' set of CSS scripts in the following manner. For each CSS script $CSS_i \in CSS'$:
   For each agent constant $C \in \mathcal{K}(CSS_i)$ that is different from the name of the script, i.e. $C \neq CSS_i{}^*$, if there is another $CSS_j \in CSS'$ script such that $C \in \mathcal{K}(CSS_j)$, then we replace all the C agent constants in the CSS_i script with the next constant from $\mathcal{C}$ and then we add CSS_i to the CSS'' set.
3. If the XCCS diagram is plain, then for each port $B_i.\overline{a} \in OutP$ we apply: all actions $a \in \mathcal{A}(B_i^*)$ in CSS script B_i^* are changed to co-actions $\overline{a}$.
4. We create the quotient set $P/_\sim$ of all possible equivalence classes of the set of ports P by synchronization relation SR. To the equivalence classes of SR we will assign consecutive labels from the set E' or labels from the set CA. We assume that ports from equivalence classes that have been assigned labels from the set E' are removed from the set P.
   a) We eliminate conflicts among visible isolated ports. For all labels $a \in CA$ of visible isolated input and output ports we apply:
   If $0 < \#V \cap Iso \cap k.InP(a) < \#V \cap Iso \cap k.OutP(a)$, then to the equivalence classes of ports from the set $V \cap Iso \cap k.InP(a)$ we assign the next label from the set E'.
   If $0 < \#V \cap Iso \cap k.OutP(a) \leq \#V \cap Iso \cap k.InP(a)$, then to the equivalence classes of ports from the set $V \cap Iso \cap k.OutP(a)$ we assign the next label from the set E'.

b) We eliminate conflicts of inner actions and co-actions with visible isolated ports. For every inner action $a \in Int$ and co-action $\overline{a} \in \overline{Int}$ we apply:
If the action a is in conflict with visible isolated port $B_i.\overline{a} \in OutP$, then in all CSS scripts that are assigned to blocks from the set $k.Int(a)$ all actions a assume the name of the next label from the set E'.
If the co-action $\overline{a}$ is in conflict with visible isolated port $B_i.\overline{a} \in OutP$, then in all CSS scripts that are assigned to blocks from the set $k.Int(\overline{a})$ all co-actions $\overline{a}$ assume the name of the next label from the set E'.

c) We eliminated conflicts among inner actions and co-actions. For every inner action $a \in Int$ we apply:
If $0 < \#k.Int(a) < \#k.Int(\overline{a})$, then in all CSS scripts assigned to blocks from the set $k.Int(a)$ all actions a assume the name of the next label from the set E'.
If $0 < \#k.Int(\overline{a}) \leq \#k.Int(a)$ then in all CSS scripts assigned to blocks from the set $k.Int(\overline{a})$ all co-actions $\overline{a}$ assume the name of the next label from the set E'.

d) We relabel visible connected ports. In this step we begin from the least numerous equivalent classes. For every equivalence class of synchronization relation SR we apply:
If in the equivalence class exists a port with some label $a \in CA$ and all other ports in this class when relabelled with this label a will not be conflicting with other visible ports and inner actions, then to this class we assign the label a, otherwise we assign the next label from the set E'.

e) We relabel invisible connected ports. In this step we begin from the least numerous equivalent classes. For every equivalence class of synchronization relation SR we apply:
If in the equivalence class exists a port with some label $a \in CA$ and all other ports in this class when relabelled with this label a will not conflict and shadow with: visible ports, inner actions, other invisible connected ports and previously „relabelled ports", then to this class we assign the label a, otherwise we assign the next label from the set E'.

f) We relabel invisible isolated ports. For every one element equivalence class of SR we apply:
If the port from this class is in conflict with or shadows: visible port, inner action or co-action, previously „relabelled ports", then to this port we assign the next label from the set E'.

g) We eliminate conflicts among invisible isolated ports. For all labels $a \in CA$ of invisible isolated input and output ports we apply:
If $0 < \#I \cap Iso \cap k.InP(a) < \#I \cap Iso \cap k.OutP(a)$, then to the equivalence classes of ports from the set $I \cap Iso \cap k.InP(a)$ we assign the next label from the set E'.
If $0 < \#I \cap Iso \cap k.OutP(a) \leq \#I \cap Iso \cap k.InP(a)$, then to the equivalence classes of ports from the set $I \cap Iso \cap k.OutP(a)$ we assign the next label from the set E'.

5. We relabel actions and co-actions in CSS scripts from the CSS'' set. For every port $B_i.a \in InP$ and for every port $B_i.\overline{a} \in OutP$ we apply:

If the equivalence class $[B_i.a]_{SR}$ has been assigned to the label e, then in CSS script B_i^* all a actions are relabelled to e.
If the equivalence class $[B_i.\overline{a}]_{SR}$ has been assigned to the label e, then in CSS script B_i^* all co-actions $\overline{a}$ are relabelled to $\overline{e}$.

6. We create the output CCS script. The main agent equation takes the form:

$$S = (B_1|B_2|\dots|B_n)\backslash\{Res\} \tag{20}$$

where $B_1, B_2, \dots, B_n \in CSS''$, and the set Res contains all labels which has been assigned to the equivalence classes of SR with invisible ports and labels of remaining invisible ports. Under the main expression we place all definitions of CSS scripts.

4.1 Example Conversion

CCS process algebra can be used to verify modular asynchronous circuits [8]. Let us then examine a simple XCCS model of NAND gate presented in Fig. 7.

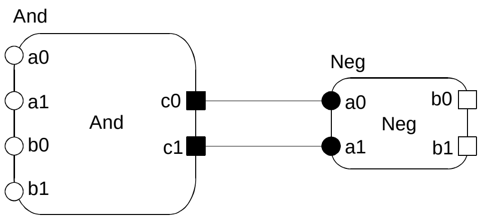

Fig. 7. XCCS model of NAND gate

Definitions of CSS scripts assigned to the blocks are as follows:

```
And = a0.(b0.'c0.And + b1.'c0.And) + a1.(b0.'c0.And + b1.'c1.And);
Neg = a0.'b1.Neg + a1.'b0.Neg;
```

The algorithm executes as follows:

1. The names of CSS scripts are already the same as names of corresponding blocks.
2. There are no any ambiguities of names of agent constants in CSS scripts.
3. Skipped as it is full XCCS diagram.
4. Assume that we have the subset $E' = \{b0\_1, b1\_1\} \subset E$ of labels E. We partition the set of ports P into equivalence classes $P/_\sim$ of synchronization relation SR.
   a) For every label $a \in CA$ of visible isolated input and output port:

$$V \cap Iso \cap k.InP(b0) = \{And.b0\} \quad V \cap Iso \cap k.OutP(b0) = \{Neg.\overline{b0}\}$$
$$V \cap Iso \cap k.InP(b1) = \{And.b1\} \quad V \cap Iso \cap k.OutP(b0) = \{Neg.\overline{b1}\}$$

The following conditions hold:

$$0 < \#V \cap Iso \cap k.OutP(b0) = 1 \leq \#V \cap Iso \cap k.InP(b0) = 1$$
$$0 < \#V \cap Iso \cap k.OutP(b1) = 1 \leq \#V \cap Iso \cap k.InP(b1) = 1$$

Thus, to the equivalence classes of ports from set $V \cap Iso \cap k.OutP(b0)$ and $V \cap Iso \cap k.OutP(b1)$ we assign consecutive labels from the set E':

- $[Neg.\overline{b0}]_{SR} = \{Neg.\overline{b0}\} \mapsto b0\_1$
- $[Neg.\overline{b1}]_{SR} = \{Neg.\overline{b1}\} \mapsto b1\_1$

b) Skipped as the diagram does not have inner actions.
c) Skipped as the diagram does not have inner actions and co-actions.
d) Skipped as the diagram does not have visible connected ports.
e) We consider equivalence classes of invisible connected ports. In both of these classes we find a label that will not cause conflicts and shadows with visible ports and invisible connected ports when the relabelling is applied, thus:
   - $[And.\overline{c0}]_{SR} = \{And.\overline{c0}, Neg.a0\} \mapsto c0 \in CA$
   - $[And.\overline{c1}]_{SR} = \{And.\overline{c1}, Neg.a1\} \mapsto c1 \in CA$
f) Skipped as the diagram does not have invisible isolated ports.
g) Skipped as the diagram does not have invisible isolated ports.

5. We consider equivalence classes that have been assigned the labels:
   - $[Neg.\overline{b0}]_{SR} = \{Neg.\overline{b0}\} \mapsto b0\_1$
   - $[Neg.\overline{b1}]_{SR} = \{Neg.\overline{b1}\} \mapsto b1\_1$
   - $[And.\overline{c0}]_{SR} = \{And.\overline{c0}, Neg.a0\} \mapsto c0$
   - $[And.\overline{c1}]_{SR} = \{And.\overline{c1}, Neg.a1\} \mapsto c1$

   After the relabelling the CSS scripts look like these:
   - $And = a0.(b0.\overline{c0}.And + b1.\overline{c0}.And) + a1.(b0.\overline{c0}.And + b1.\overline{c1}.And)$
   - $Neg = c0.\overline{b1\_1}.Neg + c1.\overline{b0\_1}.Neg$
6. We specify the output CCS script:

$$NAND = (And|Neg)\backslash\{c0, c1\}$$
$$And = a0.(b0.\overline{c0}.And + b1.\overline{c0}.And) + a1.(b0.\overline{c0}.And + b1.\overline{c1}.And)$$
$$Neg = c0.\overline{b1\_1}.Neg + c1.\overline{b0\_1}.Neg$$

5 Conclusions and Future plans

We presented the Universal Conversion Algorithm that is helpful when we use XCCS modelling language as a tool supporting fast and flawless development of CCS process algebra scripts. This algorithm takes into consideration the priority of actions in XCCS diagram and strife to minimize the number of relabelled actions for the sake of clarity.

We plan to elaborate analogous conversion algorithm for a hierarchical version of XCCS modelling language. This would allow easy development and management of the libraries of CCS scripts. In this approach, any CCS script can be represented as a hierarchical agent block. Two or more hierarchical blocks can be used to compile a new model. We do not need to worry about unwanted synchronizations, non-executable actions and absence of desired synchronizations. The conversion algorithm will care to produce a proper resulting CCS script.

Until now, the creation and management of CCS script libraries was hardly possible due to the nature of synchronization in CCS process calculus and lack of appropriate tools.

References

1. Balicki, K., Szpyrka, M.: Formal definition of XCCS modelling language. Fundamenta Informaticae **93** (2009) 1–15
2. Milner, R.: A Calculus of Communicating Systems. Volume 92 of LNCS. Springer-Verlag (1980)
3. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
4. Fencott, C.: Formal Methods for Concurrency. International Thomson Computer Press, Boston, MA, USA (1995)
5. Aceto, L., Ingófsdóttir, A., Larsen, K., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, Cambridge, UK (2007)
6. Szpyrka, M., Balicki, K.: XCCS – graphical extension of CCS language. In: Proc. of Mixdes 2006, the 14th International Conference Mixed Design of Integrated Circuits and Systems, Ciechocinek, Poland (2007) 688–693
7. Szpyrka, M., Matyasik, P.: Formal modelling and verification of concurrent systems with XCCS. In: Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), Krakow, Poland (2008) 454–458
8. Joeli, M., Kol, R., eds.: Verification of Systems and Circuits Using LOTOS, Petri Nets, and CCS. John Wiley and Sons, Chichester, UK (2008)