Interpreting Broadcast Communication in CCS with Priority Choice

Claus Torp Jensen *ctjensen@cs.chalmers.se*
Computer Science Department
Chalmers University of Technology

November 29, 1994

1 Introduction

CBS (Calculus of Broadcasting Systems) is a process calculus in which *broadcast* is the fundamental communication paradigm. Communication between processes is performed through unbuffered broadcasts. One of the processes broadcasts an output and the remaining processes either receive or ignore the broadcast according to whether they are listening or not. Consequently any output is autonomous and may be performed in any environment. Reception on the other hand is controlled by the environment in the sense that a process has to continue listening until something is being broadcast.

Clearly CBS is a synchronous calculus. It has been shown in [1] that broadcasting communication may be interpreted in a different kind of synchronous calculus, namely $SCCS$. An interesting question is whether it is also possible to interpret broadcast communication in an asynchronous handshake model like CCS. In [1] it is argued that broadcast communication cannot be interpreted in pure CCS. We believe this is true. On the other hand extensions of CCS may provide the additional control of communication that is needed in order to simulate the "one-to-many" synchronization in CBS.

In this paper we show that by adding a priority choice operator to CCS, any CBS system may be translated to a term in CCS with priority choice. A CBS synchronization must necessarily correspond to a *sequence* of transitions in the CCS model, and the priority choice operator enables us to ensure that such sequences terminate correctly. The translation is correct in the sense that two CBS terms are strongly bisimilar iff their translations are weakly bisimilar. Due to the multiway synchronizations in CBS this is the simplest relation between a term and its translation that we can hope for.

The reader is expected to have some basic knowledge of process calculi like CCS and of operational semantics in the style of Plotkin [6]. We give brief descriptions of the semantics of CBS and CCS with priority choice, but it is probably preferable to consult the references for better descriptions of these two calculi.

Operator	Transmit	Read	Discard
nil			$nil \xrightarrow{\mu:} nil$
Monitor		$a?p \xrightarrow{a?} p$	$a?p \xrightarrow{\mu:} a?p \ (\mu \neq a)$
Transmit	$\mu!p \xrightarrow{\mu!} p$		$\mu!p \xrightarrow{\nu:} \mu!p$
Choice[1]	$\dfrac{p \xrightarrow{\mu\natural} p'}{p+q \xrightarrow{\mu\natural} p'}$		$\dfrac{p \xrightarrow{\mu:} p' \quad q \xrightarrow{\mu:} q'}{p+q \xrightarrow{\mu:} p'+q'}$
Recursion	$\dfrac{e\{rec\,x.e/x\} \xrightarrow{\mu\natural} p'}{rec\,x.e \xrightarrow{\mu\natural} p'}$		$\dfrac{e\{rec\,x.e/x\} \xrightarrow{\mu:} p'}{rec\,x.e \xrightarrow{\mu:} rec\,x.e}$
Parallel	$\dfrac{p \xrightarrow{\mu\sharp_1} p' \quad q \xrightarrow{\mu\sharp_2} q'}{p\vert q \xrightarrow{\mu(\sharp_1 \bullet \sharp_2)} p'\vert q'} \quad (\sharp_1 \bullet \sharp_2 \neq \perp)$		$\begin{array}{c\|ccc} \bullet & ! & ? & : \\ \hline ! & \perp & ! & ! \\ ? & ! & ? & ? \\ : & ! & ? & : \end{array}$
Translate	$\dfrac{p \xrightarrow{\mu!} p'}{\phi p \xrightarrow{\phi^{\uparrow}\mu!} \phi p'}$	$\dfrac{p \xrightarrow{\phi_{\downarrow}\mu\flat} p'}{\phi p \xrightarrow{\mu\flat} \phi p'}$	

[1] There is a symmetric rule for q.

Table 1: The operational semantics of CBS

2 CBS summary

In this section we present a variant of CBS with channels but without value passing [7] and with the scoping and renaming operators replaced by the new translation operator of [8]. Details on different versions of CBS may be found in [7], [8] and [9].

We define the following sets ranged over by the variables indicated in parenthesis.

A set of names: $\Delta = \{a, b, c, \ldots\}$. A set of messages: $\Delta^{\tau} = \Delta \cup \{\tau\} \quad (\mu, \nu)$.

Let $\sharp$ range over $\{!, ?, :\}$, $\natural$ over $\{!, ?\}$, $\flat$ over $\{?, :\}$, and for $S \subseteq \Delta$ let $S\sharp = \{a\sharp | a \in S\}$.

Let $A = \Delta^{\tau}! \cup \Delta? \cup \Delta^{\tau}: \quad (a, b, c, \ldots)$.

Let ϕ be a pair of functions $\phi^{\uparrow}, \phi_{\downarrow} : \Delta^{\tau} \to \Delta^{\tau}$ satisfying $\phi^{\uparrow}(\tau) = \phi_{\downarrow}(\tau) = \tau$. Note that $\phi^{\uparrow}\mu\sharp$ and $\phi_{\downarrow}\mu\sharp$ means unambiguously $(\phi^{\uparrow}\mu)\sharp$ and $(\phi_{\downarrow}\mu)\sharp$. Assume a set of process variables $\mathcal{V}_{\mathrm{CBS}} = \{x, y, \ldots\}$. The syntax for CBS process expressions is defined by the grammar:

$$e ::= nil \mid x \mid \mu!e \mid a?e \mid e+e \mid rec\,x.e \mid e|e \mid \phi e$$

The operational semantics of the language is defined in table 2. Strong bisimulation is defined in the standard way.

3 CCS with prisum summary

The reader is assumed to be familiar with CCS so we limit ourselves to a brief description of the priority choice operator and the operational semantics for CCS with prisum. Details on adding priority operators to CCS may be found in [2], [3] and [4].

The priority choice constructor $+\rangle$ named *prisum* gives the left agent preference to the right but otherwise behaves like a sum. Intuitively we have that

$$A{+}\rangle B$$

means

Behave as A if you can; if no transitions for A are possible, then behave as B

Now let $CCS^{>}$ denote the new language with priority choice that has CCS as a subset. Let Act be a set of action labels; Act is divided into the set of input labels In, the dual set of output labels Out and the internal action τ. Note that output (input) labels need not necessarily be physical output (input) labels; the division is needed for technical reasons. In what follows, let $^{-}$ be an injective function from Act to Act taking τ to τ and an input (output) label α to its dual output (input) label $\overline{\alpha}$. In general α, β, γ etc. will range over Act and a, b, c etc. will range over In. $CCS^{>}$ expressions then have the following syntax:

$$E \quad ::= \quad X \mid nil \mid \alpha.E \mid E+E \mid E{+}\rangle E \mid (E|E) \mid fix(X=E) \mid E \setminus L \mid E[f]$$

where the symbols from CCS have their usual meaning. Only *guarded* recursions are allowed. For technical reasons we demand that there never be any output labels as possible actions of terms in a priority choice construct; i.e. that output labels are not allowed in priority choice constructs except when guarded by an input prefix.

Before turning to the definition of the transition rules we need some considerations about environments as well as some technical preliminaries. Since all actions in CCS are basically performed in handshakes, one expects the following:

> If the environment of $A{+}\rangle B$ refuses to match any of A's transitions, then B doing a transition leading to B' can be allowed.

A refusal set is defined to be a set of input actions, for which it is known that the environment can't perform any of the matching output actions. Since no output labels are allowed as possible actions of a priority construct, the information in a refusal set is all that we need to know about the environment of a term.

Given the informal definition of refusal sets above, it is immediately observable that a transition would be possible under a very large (actually infinite) number of different refusal sets. Consider the term $a.nil{+}\rangle b.nil$. The b may occur if the environment refuses $\{a\}$. It may also occur if the environment refuses $\{a, c\}$, etc. Clearly some kind of simplification is needed. The solution is to use transitions of the form

$$\vdash_R A \xrightarrow{\alpha} A'$$

understood as meaning

> *In an environment which refuses* at least R *(and possibly* more *actions),*
> *and is prepared to match* α*,*
> *the agent* A *can perform* α *to become* A'*.*

Thus, the subscripted refusal set should be the *weakest condition* for a given derivation. In order to allow the b in $a.nil{+}\rangle b.nil$ to occur *all* we need to know is that the environment refuses at least $\{a\}$.

In order to define the transition relation, we need the two auxiliary functions *inActs* and *outActs* respectively giving the set of possible input actions and the set of possible output actions of a term. Note that since recursion is guarded these functions are easily defined by structural induction on the term. We then have the following transition rules of which some have symmetric counterparts:

$$\vdash_\emptyset \alpha.A \xrightarrow{\alpha} A \tag{1}$$

$$\frac{\vdash_R A \xrightarrow{\alpha} A'}{\vdash_R A + B \xrightarrow{\alpha} A'} \tag{2}$$

$$\frac{\vdash_R G_1 \xrightarrow{a} G_1'}{\vdash_R G_1{+}\rangle G_2 \xrightarrow{a} G_1'} \tag{3}$$

$$\frac{\vdash_R G_2 \xrightarrow{\alpha} G_2',\ \tau \notin inActs(G_1),\ \alpha \notin inActs(G_1)}{\vdash_{R \cup inActs(G_1)} G_1{+}\rangle G_2 \xrightarrow{\alpha} G_2'} \tag{4}$$

$$\frac{\vdash_R A_0 \xrightarrow{\alpha} A_0',\ R \cap outActs(A_1) = \emptyset}{\vdash_R A_0|A_1 \xrightarrow{\alpha} A_0'|A_1} \tag{5}$$

$$\frac{\vdash_{R_0} A_0 \xrightarrow{\alpha} A_0',\ \vdash_{R_1} A_1 \xrightarrow{\overline{\alpha}} A_1',\ R_1 \cap outActs(A_0) = \emptyset,\ R_0 \cap outActs(A_1) = \emptyset}{\vdash_{R_0 \cup R_1} A_0|A_1 \xrightarrow{\tau} A_0'|A_1'} \tag{6}$$

$$\frac{\vdash_R A\{fix(X = A)/X\} \xrightarrow{\alpha} A'}{\vdash_R fix(X = A) \xrightarrow{\alpha} A'} \tag{7}$$

$$\frac{\vdash_R A \xrightarrow{\alpha} A' \ \ (\alpha \notin L \cup \overline{L})}{\vdash_{R \setminus L} A \setminus L \xrightarrow{\alpha} A' \setminus L} \tag{8}$$

$$\frac{\vdash_R A \xrightarrow{\alpha} A'}{\vdash_{R[f]} A[f] \xrightarrow{f(\alpha)} A'[f]} \tag{9}$$

Below is a short intuitive explanation of the most important rules:

- (3) The highest prioritized agent may always perform its transitions.

- (4) Since *inActs* gives *every* possible action for a guard, the refusal set for G_2 transitions should exactly be extended with $inActs(G_1)$. Since τ is not allowed in refusal sets, $\tau \notin inActs(G_1)$ is a necessary condition. A τ will preempt any lower priority transitions from occurring.
  Furthermore it must be the case that the G_2 action is not itself in the new refusal set. We want that $\vdash_R P \xrightarrow{\alpha} P'$ implies $\alpha \notin R$; an environment cannot both refuse α and be prepared to synchronize on it.

- (5) *outActs* gives exactly *every* possible output action for an agent, and one of the agents in a parallel composition cannot do a transition if the other one violates the refusal set condition. The two sub-agents are part of each other's environments.

- (6) The refusal set for the synchronization is the union of the individual refusal sets; otherwise one of the transitions involved in the synchronization would be impossible.

For transitions with empty refusal sets we will omit the refusal set and merely write $P \xrightarrow{\alpha} P'$. A term is *priority closed* if all transitions in the corresponding transition graph have empty refusal sets. For relabeling of terms that are not priority closed, the relabeling function in general needs to be injective, or else counterintuitive transitions may occur. The definitions of strong and weak bisimulation for $CCS^>$ are slightly nonstandard, but on the set of priority closed terms they coincide with standard definitions. Since below we only consider equivalence between priority closed terms, we omit further details and refer the reader to the references.

4 Translating CBS to CCS with Priority Choice

In this section we present a translation taking any CBS term not containing the translation operator to a term in $CCS^>$. We have deferred considerations concerning the translation operator to Section 7. Due to the fact that only guarded recursions are allowed in $CCS^>$ we do not consider CBS terms that contain unguarded recursions.

4.1 Motivation and preliminaries

As already mentioned, CBS is a synchronous calculus with multiway synchronizations, and CCS with prisum is not. Consequently a single CBS transition necessarily must be translated to a *sequence* of transitions $CCS^>$ transitions. Such a sequence once initiated must be completely finished before another sequence is initiated.

One obvious option is to expand all parallels; then we only have a single parallel component and no synchronization problems. On the other hand, such a solution is not very good. In general we would like the translation into $CCS^>$ to be compositional. Hence we need a control process in parallel with the rest of the translation system, registering each process synchronizing and realizing when all processes in the system have participated. Furthermore, each parallel component of the system, after registrering with the controller must be prevented from continuing execution until all synchronizations have been made. Otherwise for instance $a?b?nil$ may do $a?$ followed by $b?$ before synchronization on $a?$ is really finished; the interleaving and asynchronous nature of the parallel composition in $CCS^>$ gets in the way. Then again, resumption of the execution of each parallel component must be controlled.

Each action of a CBS term, $\tau!$ or otherwise, should correspond to an *observable* action in the translation of that term. Otherwise we cannot distinguish actual CBS occurrences in the translation from the internal synchronizations needed for control. Consequently we want the set of CBS actions A to be a subset of the set of $CCS^>$ actions Act.

Only the controller should perform observable actions, since only the controller knows when a synchronization is complete. All control synchronizations should be internalized. Consequently let i be a function from A to Act such that $i(A) \cap A = \emptyset$.

In view of the above, a natural choice for the interpretation of a prefix construction is something like the following:[1]

$$a\natural.P \rightsquigarrow a\natural.end.P$$

where $end \in Act$ is a distinguished label indicating that this particular subprocess has already participated in the current multiway synchronization, or rather the corresponding sequence of $CCS^>$ transitions. As a motivating example consider:

$$\begin{aligned} a?a?nil &\rightsquigarrow a?.end.a?.end.nil \\ a?nil|a?nil &\rightsquigarrow a?.end.nil|a?.end.nil \end{aligned}$$

Without the *end* label, there would be no way to distinguish the translations of $a?a?nil$ and $a?nil|a?nil$ respectively, although the two CBS terms have different behaviour. The only other alternative is to expand the parallel before translating, which as already mentioned is highly undesirable.

As a motivating example for the conclusion that we need a separate process controlling the synchronization sequences consider the following:

$$\begin{aligned} a!P|a?Q &\rightsquigarrow a!.end.P|a?.end.Q \\ a!P|a?Q|a?nil &\rightsquigarrow a!.end.P|a?.end.Q|a?.end.nil \end{aligned}$$

Clearly without detailed analysis of parallel subprocesses we do not in advance know how many components that participate in a particular CBS transition. Since we want our translation to be as compositional and structural as possible, we need a separate process to register *when no more components have an available* $a?$. The connection to refusal sets should be obvious, and in fact the priority choice operator is excellent for this kind of control. Clearly nondeterministic choice is not sufficient since we either run the risk of deadlock or run the risk of premature termination of a sequence.

[1] In the remaining part of this subsection, we have omitted explicit use of the internalization function i.

As a final motivating example consider a situation where one of the parallel components discards a broadcast:

$$a!P|a?Q|b?R \quad \rightsquigarrow \quad a!.end.P|a?.end.Q|b?.end.R$$

In the synchronous CBS semantics an explicit notion of a discard transition is needed in order to avoid negative premises. But when translating to $CCS^>$ as mentioned above we may simply use the priority choice operator to detect the absence of an available $a?$. Hence we need not concern ourselves with discards at all if we use the observation from [7] that the presence of discard transitions does not change the semantics of the language. To be precise, the strong bisimulation equivalence is the same whether we consider discard transitions in its definition or not. Not needing explicit treatment of discards greatly simplifies the translation.

4.2 The translation

We define the translation $T : CBS \to CCS^>$ by:

$$T(P) \quad \equiv \quad (CON \mid T'(P)) \setminus (i(A) \cup \{end\})$$

CON is the process that we need for controlling the synchronization sequences. It is defined as follows:

$$CON \quad \equiv \quad fix(X = \sum_{a \in A, \natural \in \{!,?\}} i(a\natural).fix(Y = i(a?).Y + \rangle fix(Z = end.Z + \rangle a\natural.X)))$$

When trying to understand this definition it might be a good idea to reconsider the motivating examples from Section 4.1. A brief explanation of the definition of CON is the following. One of the CBS subprocesses initiates a sequence by synchronizing with CON on $i(a\natural)$. CON proceeds to synchronize with *all* subprocesses having an available $i(a?)$. Note that a subprocess that has already participated in the current sequence has synchronization on *end* as the only possible action. Only when no more subprocesses have available $i(a?)$ actions, corresponding to a situation where all CBS components have participated in the current transition, may the low priority term in the definition of the Y fixpoint proceed. Subsequently the control process "collects" all *end*'s, returning the rest of the system to a state that is the translation of a CBS term. Finally the control process externally signals $a\natural$ in agrement with the fact that now a sequence corresponding to a CBS transition with this label is completed. As already mentioned, for obvious reasons only the control process can know when such a sequence is finished.

Note that no $T'(P)$ can synchronize on $\tau?$, hence we do not need to explicitly exclude such (illegal) transitions. Furthermore note that it is *possible* to begin a synchronization sequence with any enabled internal $a!$ action. Hence, since we are only interested in bisimulation equivalences, we need not concern ourselves with internal output's occurring in the middle of a synchronization sequence. Such occurrences would have forced us to change the final external result of the sequence, and would create unnecessary complications. As mentioned above, only input labels are allowed as possible actions of priority constructs, and we have obeyed this restriction in the construction of CON.

At first glance we need an infinite sum for the definition of CON, which would technically take us outside the language $CCS^>$. But since we do not need to consider discards, actually any CBS term has finite sort and consequently a finite sum suffices. Except for perhaps using the sort as described above, the definition of CON is independent of the term that is being translated. Indeed the function $T' : CBS \rightarrow CCS^>$ represents the actual translation of terms. Below this function is defined inductively in a compositional manner. Having already defined the control system CON, the definition of T' is easy.

Nil

The CBS term *nil* has no transitions other than discards. Since we ignore discards, we may model *nil* by *nil*:

$$T'(nil) \equiv nil$$

Variable

We assume that the sets of variables in the two calculi are essentially the same (i.e. in bijection) and model the CBS variable x by the $CCS^>$ variable X:

$$T'(x) \equiv X$$

Receive

We have no discards to consider. The term $a?Q$ is willing to receive on channel a. In the translation this is signalled to the controller by sending an $\overline{i(a?)}$, whereupon the process waits until told by the controller that the current multiway synchronization is ended:

$$T'(a?Q) \equiv \overline{i(a?)}.\overline{end}.T'(Q)$$

Note that since we are forced to use only input labels in the construction of CON, we have to use only output labels in the translation of a receive. Naturally, due to the restriction on internal labels, this has no computational significance.

Transmit

Since we have no discards to consider, the translation of a transmit is analogous to the translation of a receive:

$$T'(a!Q) \equiv \overline{i(a!)}.\overline{end}.T'(Q)$$

Choice

Since we have no discards to consider, we need not be worried about prematurely resolving the choice. We can simply take:

$$T'(P+Q) \equiv T'(P) + T'(Q)$$

Only when the controller is ready to receive one of the possible actions of either P or Q can the sum participate in a $CCS^{>}$ transition. But this corresponds exactly to the process participating in a CBS synchronizations with an action that is not discard. We do not need additional operators such as the "green restriction operator" used in [1].

Parallel

Due to the fact that CON directly controls all multiway synchronizations, parallel may simply be translated directly to parallel:

$$T'(P|Q) \equiv T'(P)|T'(Q)$$

Recursion

Again, since we do not consider discards, we need not be concerned about unfolding the recursion prematurely (or even infinitely many times). We can simply take:

$$T'(rec\, x.P) \equiv fix(X = T'(P))$$

Translation summary

The translation $T : CBS \to CCS^{>}$ is defined by

$$T(P) \;\equiv\; (CON \mid T'(P)) \setminus (i(A) \cup \{end\})$$

where CON is defined by

$$CON \;\equiv\; fix(X = \sum_{a,\natural} a\natural.fix(Y = a?.Y + \rangle \tau.(fix(Z = end.Z + \rangle \tau.a\natural.X)))$$

and where the function $T' : CBS \to CCS^{>}$ is defined inductively by

$$\begin{array}{rcl}
T'(nil) & \equiv & nil \\
T'(x) & \equiv & X \\
T'(a?Q) & \equiv & \overline{i(a?)}.\overline{end}.T'(Q) \\
T'(a!Q) & \equiv & \overline{i(a!)}.\overline{end}.T'(Q) \\
T'(P+Q) & \equiv & T'(P) + T'(Q) \\
T'(P|Q) & \equiv & T'(P)|T'(Q) \\
T'(recx.P) & \equiv & fix(X = T'(P))
\end{array}$$

In Section 7 this translation is extended to cover relabeling, restriction and/or translation.

5 Examples

In this section, we give a few examples of translations of CBS terms, and briefly consider the relation between transitions of CBS terms and sequences of transitions of their translations.

Consider the term $a!nil|a?nil|b?nil$ which has the transition

$$a!nil|a?nil|b?nil \xrightarrow{a!} nil|nil|b?nil$$

The translation to $CCS^{>}$ gives us the term

$$T(a!nil|a?nil|b?nil) \equiv (CON|\overline{i(a!)}.\overline{end}.nil|\overline{i(a?)}.\overline{end}.nil|\overline{i(b?)}.\overline{end}.nil) \setminus L$$

which has the following sequence of transitions:

$$\begin{array}{rl}
(T(a!nil|a?nil|b?nil)) \setminus L & \xrightarrow{\tau} \\
(\ fix(Y = i(a?).Y{+}\rangle fix(Z = end.Z{+}\rangle a!.CON)))| & \\
\overline{end}.nil|\overline{i(a?)}.\overline{end}.nil|\overline{i(b?)}.\overline{end}.nil\ \ ) \setminus L & \xrightarrow{\tau} \\
(\ fix(Y = i(a?).Y{+}\rangle fix(Z = end.Z{+}\rangle a!.CON)))| & \\
\overline{end}.nil|\overline{end}.nil|\overline{i(b?)}.\overline{end}.nil\ \ ) \setminus L & \xrightarrow{\tau} \\
(fix(Z = end.Z{+}\rangle a!.CON)|\overline{end}.nil|\overline{end}.nil|\overline{i(b?)}.\overline{end}.nil) \setminus L & \xrightarrow{\tau} \\
(fix(Z = end.Z{+}\rangle a!.CON)|nil|\overline{end}.nil|\overline{i(b?)}.\overline{end}.nil) \setminus L & \xrightarrow{\tau} \\
(fix(Z = end.Z{+}\rangle a!.CON)|nil|nil|\overline{i(b?)}.\overline{end}.nil) \setminus L & \xrightarrow{\tau} \\
(a!.CON|nil|nil|\overline{i(b?)}.\overline{end}.nil) \setminus L & \xrightarrow{a!} \\
(CON|nil|nil|\overline{i(b?)}.\overline{end}.nil) \setminus L &
\end{array}$$

ending in the state

$$(CON|nil|nil|\overline{i(b?)}.\overline{end}.nil) \setminus L \ \equiv \ T(nil|nil|b?nil)$$

Notice how in this example, we *choose* to let the output process be the first to synchronize with CON, giving us a CBS output as result of the synchronization sequence. The interleaving nature of the CCS parallel saves us some considerations concerning (internalized) CBS outputs occurring in the middle of a sequence. Furthermore notice how the completion of the synchronization sequence is forced by the structure of CON. Even though in general any interleaving of the set of inputs from different parallel components taking part in a CBS transition is possible, each such component is forced to take part in the synchronization sequence exactly once with an internalized input action and exactly once with an *end* action.

As an example of the way CON controls choices, consider the term

$$a!nil|(a?nil + b?nil)$$

which has the transition

$$a!nil|(a?nil + b?nil) \xrightarrow{a!} nil|nil$$

The translation to $CCS^{>}$ gives us the term

$$T(a!nil|(a?nil+b?nil)) \equiv (CON|\overline{i(a!)}.\overline{end}.nil|(\overline{i(a?)}.\overline{end}.nil+\overline{i(b?)}.\overline{end}.nil))\backslash L$$

which has the following sequence of transitions:

$$\begin{array}{rl}
T(a!nil|(a?nil+b?nil)) & \stackrel{\tau}{\rightarrow} \\
(\ \ fix(Y = i(a?).Y+\rangle fix(Z = end.Z+\rangle a!.CON)))| & \\
\overline{end}.nil|(\overline{i(a?)}.\overline{end}.nil + \overline{i(b?)}.\overline{end}.nil)\ \ )\setminus L & \stackrel{\tau}{\rightarrow} \\
(fix(Y = i(a?).Y+\rangle fix(Z = end.Z+\rangle a!.CON)))|\overline{end}.nil|\overline{end}.nil)\setminus L & \stackrel{\tau}{\rightarrow} \\
(fix(Z = end.Z+\rangle a!.CON)|\overline{end}.nil|\overline{end}.nil)\setminus L & \stackrel{\tau}{\rightarrow} \\
(fix(Z = end.Z+\rangle a!.CON)|nil|\overline{end}.nil)\setminus L & \stackrel{\tau}{\rightarrow} \\
(fix(Z = end.Z+\rangle a!.CON)|nil|nil)\setminus L & \stackrel{\tau}{\rightarrow} \\
(a!.CON|nil|nil)\setminus L & \stackrel{a!}{\rightarrow} \\
(CON|nil|nil)\setminus L &
\end{array}$$

ending in the state

$$(CON|nil|nil)\setminus L \equiv T(nil|nil)$$

Notice how in this example, the fact that we do not explicitly consider discard actions saves us the trouble of preventing the choice from being prematurely resolved by a discard.

6 Results

In this section only CBS terms that do not contain the translation operator nor an unguarded recursion are considered. Note that the translation of any CBS term is a priority closed $CCS^{>}$ term.

As the main result we prove that two CBS terms are strongly bisimilar iff the $CCS^{>}$ terms resulting from the translation are weakly bisimilar. This is a nice and very simple relationship between terms and their translations, and indeed it is the best that we can hope for. As proof of this just consider

$$(rec\, x.a?x) \sim (rec\, x.a?x)|(rec\, x.a?x)$$

For the translation we have the following general results:

Proposition 1 *If* $P \stackrel{a\natural}{\rightarrow} P'$ *then* $T(P) \stackrel{\tau}{\Rightarrow}\stackrel{a\natural}{\rightarrow} T(P')$

Sketch of proof: Clearly due to the construction of CON it is sufficient to prove Lemma 2 below. $\Box$

Lemma 2 *If* $P \stackrel{a\natural}{\rightarrow} P'$ *then* $T'(P) \stackrel{i(a\natural)}{\rightarrow} (\stackrel{i(a?)}{\rightarrow})^{*}U(\stackrel{end}{\rightarrow})^{*}T'(P')$ *where* $U \stackrel{i(a?)}{\not\rightarrow}$

Sketch of proof: The proof is done by structural induction on P. Note that every recursion is guarded. The only slightly complicated case is the case for parallel which is treated by dividing into subcases according to the combination of CBS actions for the two parallel components. If an output is one of the components, then internal output synchronization is always chosen as the first transition in the sequence. □

With a proof similar to the proof of Proposition 1, we get:

Proposition 3 *If* $T(P) \stackrel{\tau}{\Rightarrow}\stackrel{a\natural}{\rightarrow} Q$ *then* $\exists P'. Q \equiv T(P') \ \wedge \ P \stackrel{a\natural}{\rightarrow} P'$

We now have the following Corollary:

Corollary 4 $P \stackrel{a\natural}{\rightarrow} P'$ *iff* $T(P) \stackrel{\tau}{\Rightarrow}\stackrel{a\natural}{\rightarrow} T(P')$

Proceeding with the proof of the main result of this Section we have:

Proposition 5 *If* $T(P) \approx T(Q)$ *and* $T(P) \stackrel{\tau}{\Rightarrow}\stackrel{a\natural}{\rightarrow} T(P')$ *then* $\exists Q'. T(Q) \stackrel{\tau}{\Rightarrow}\stackrel{a\natural}{\rightarrow} T(Q') \approx T(P')$

Sketch of proof: The proof uses Lemma 6 below which is a consequence of the construction of CON. It goes as follows. If

$$T(P) \stackrel{\tau}{\Rightarrow}\stackrel{a\natural}{\rightarrow} T(P')$$

then by Lemma 6 we have

$$\exists U. T(P) \stackrel{\tau}{\Rightarrow} U \stackrel{a\natural}{\rightarrow} T(P') \ \wedge \ U \sim a\natural.T(P')$$

By using $T(P) \approx T(Q)$ we get

$$\exists U'. T(P) \stackrel{\tau}{\Rightarrow} U' \approx a\natural.T(P') \approx U$$

Clearly we must have

$$\exists U''. U' \stackrel{\tau}{\Rightarrow} U'' \stackrel{a\natural}{\rightarrow} U'''$$

Where $U'' \approx a\natural.T(P')$. Note that we cannot directly conclude that U''' is weakly bisimilar to $T(P')$. According to Proposition 3 we now have

$$\exists Q'. U''' \equiv T(Q')$$

By Lemma 6 we get $a\natural.T(Q') \approx U'' \approx a\natural.T(P')$ from which we deduce that $T(Q') \approx T(P')$ as desired. □

Lemma 6 *If* $T(P) \stackrel{\tau}{\Rightarrow} U \stackrel{a\natural}{\rightarrow} T(P')$ *then* $U \sim a\natural.T(P')$

We can now prove the following Theorem:

Theorem 7 *The set* $\{\langle P, Q\rangle \mid T(P) \approx T(Q)\}$ *is a* CBS *strong bisimulation.*

Sketch of proof: If $P \stackrel{a\natural}{\rightarrow} P'$ then according to Proposition 1 we get

$$T(P) \stackrel{\tau}{\Rightarrow} \stackrel{a\natural}{\rightarrow} T(P')$$

From Proposition 5 we get

$$\exists Q'.\, T(Q) \stackrel{\tau}{\Rightarrow} \stackrel{a\natural}{\rightarrow} T(Q') \approx T(P')$$

But then according to Proposition 3 we get

$$Q \stackrel{a\natural}{\rightarrow} Q'$$

and we are done. □

As a corollary we get:

Corollary 8 *If* $T(P) \approx T(Q)$ *then* $P \sim Q$

The proof of the oppositite implication, if $P \sim Q$ then $T(P) \approx T(Q)$, is simpler. We clearly have the following two results:

Proposition 9 *For any* P*, if* $T(P)$ *has any transitions then* τ *is the only possible action of* $T(P)$*.*

Proposition 10 *For any* P*,* $T(P)$ *has no infinite sequence of* τ *transitions.*

Now with a proof similar to the proof of Proposition 1, using Proposition 10, we get:

Proposition 11 *If* $T(P) \stackrel{\tau}{\rightarrow} Q$ *then there exists* $a, \natural, U, P'$ *such that*

$$T(P) \stackrel{\tau}{\Rightarrow} U \stackrel{a\natural}{\rightarrow} T(P')$$

and whenever

$$Q \stackrel{\tau}{\Rightarrow} U' \stackrel{\alpha}{\rightarrow} U''$$

then $U' \equiv U$*,* $U'' \equiv T(P')$ *and* $\alpha = a\natural$

The above Proposition tells us that the first transition of $T(P)$ chooses which CBS synchronization we are simulating, and using this result we get:

Theorem 12 *If* $P \sim Q$ *then* $T(P) \approx T(Q)$

Sketch of proof: The proof can be done either by structural induction or by exhibiting suitable weak bisimulations. In both cases the reasoning below implies the desired result.

Assume that $T(P)$ has some transition. According to Proposition 9 this must be a τ transition. Using Proposition 11 we get:

$$T(P) \overset{U_0}{\rightarrow}\overset{\tau}{\Rightarrow} U \overset{a\natural}{\rightarrow} T(P')$$

and whenever

$$U_0 \overset{\tau}{\Rightarrow} U' \overset{\alpha}{\rightarrow} U''$$

then $U' \equiv U$, $U'' \equiv T(P')$ and $\alpha = a\natural$.
We now have that $P \overset{a\natural}{\rightarrow} P'$ and since $P \sim Q$ we know that there exists Q' such that $Q \overset{a\natural}{\rightarrow} Q' \sim P'$. But then there exists S such that

$$T(Q) \overset{\tau}{\Rightarrow} S \overset{a\natural}{\rightarrow} T(Q')$$

We know that $U \sim a\natural.T(P')$ and that $S \sim a\natural.T(Q')$. Clearly we can choose $T(Q) \overset{\tau}{\Rightarrow} S$ as the match of $T(P) \overset{\tau}{\rightarrow} Q$ in the deduction of weak bisimilarity. $\square$

We have now finished the proof of the main result of this Section:

7 Renaming, restriction and hiding

Originally, CBS had a renaming operator and a restriction/hiding operator (see [7]), but later developments (see eg. [8]) introduce a more general translation operator instead.

We will begin by considering the original renaming and restriction/hiding operators. Let $f : \Delta' \rightarrow \Delta'$ be a relabeling function and let $L \subset \Delta$. We then have the following additional transition rules:

$$\frac{P \overset{\mu\sharp}{\rightarrow} P'}{P[f] \overset{f(\mu)\sharp}{\rightarrow} P'[f]}$$

$$\frac{P \overset{\mu\sharp}{\rightarrow} P',\ \mu \notin L}{P \setminus L \overset{\mu\sharp}{\rightarrow} P' \setminus L} \qquad \frac{P \overset{\mu !}{\rightarrow} P',\ \mu \in L}{P \setminus L \overset{\tau !}{\rightarrow} P' \setminus L} \qquad \frac{\mu \in L}{P \setminus L \overset{\mu :}{\rightarrow} P \setminus L}$$

Using the relabeling operator from $CCS^{>}$, the renaming operator from CBS may clearly be translated by extending the definition of T' as follows:

$$T'(P[f]) = T'(P)[f']$$

where f' is f extended to *Act* in the obvious way taking for instance $f'(i(\overline{a\natural}) = i(\overline{f(a)\natural})$.
Note that the relabeling function f' may not be injective, but since $T'(P)$ is priority closed for any P this presents no problems.

Using this extension of the translation from CBS to $CCS^{>}$, the main result of Section 6 clearly still holds. We omit further details.

Restriction is more complicated. Consider the CBS term

$$(a!P|a?Q)\setminus\{a\}$$

which has the transition

$$(a!P|a?Q)\setminus\{a\} \xrightarrow{\tau!} (P|Q)\setminus\{a\}$$

Our problem is that more than one process participates in this internal transition. Hence clearly it is not sufficient just to relabel $a!$ to $\tau!$ in the translation; we need to take care of the $a?$ as well or that action never occurs. Relabeling $a?$ to $\tau?$ is not an option either; just to mention one of the problems there would be no way to distinguish a restricted a input from say a restricted b input.

What we are more or less forced to do is to introduce a local analogue of the control process in order to handle the local multiway synchronization resulting in a $\tau!$ transition. Unfortunately this also forces us to slightly change the definition of CON itself because we need some way of beginning and ending local hidden synchronization sequences. Consider the following version of the control process:

$$CON' \equiv fix(X = H + \sum i(a\natural).fix(Y = i(a?).Y + \rangle fix(Z = end.Z + \rangle a\natural.X)))$$

where the addition of H inside the first fixpoint is the only change compared to CON and where H is defined as follows:[2]

$$H \equiv hi_{begin}.(\overline{hi}.\tau!.X + \overline{nohi}.X)$$

Naturally the outermost restriction in the definition of $T(P)$ should now contain hi_{begin}, hi and $nohi$ as well as all internalized labels and end.

The intuition behind the definition of H is that hi_{begin} signals the beginning of (an attempt at) a local hidden synchronization sequence. $nohi$ then indicates that the attempt failed; no local hidden synchronization was performed. hi indicates that a local hidden synchronization has successfully been completed and consequently an external $\tau!$ is sent.

Given the above, we can now define the translation of $P \setminus L$ as follows:

$$T'(P\setminus L) = (LOCCON(L)|T'(P))\setminus(\{i(a?)|a\in L\}\cup\{i(a!)|a\in L\})$$

where $LOCCON(L)$ is the local analogue of the control process that is used for hidden internal synchronizations on labels in the set L:

$$LOCCON(L)$$
$$\equiv$$
$$fix(X = \overline{hi_{begin}}.((\sum_{a\in L} i(a!).fix(Y = i(a?).Y > fix(Z = end.Z > hi.X))) > nohi.X)$$

[2] "hi" is short for "hide"

Note that inputs are restricted whereas outputs are hidden. Hence $LOCCON(L)$ only needs to consider internal synchronizations that are initiated by an output. Since the restriction operator is static, we need the outermost fixpoint in the definition of $LOCCON(L)$. Clearly after sending a hi_{begin}, $LOCCON(L)$ attempts an internal synchronization on one of the labels in L. If no internal synchronization on a label in L is possible, then $LOCCON(L)$ signals a *nohi*.

Using this extended and slightly changed version of the translation from CBS to $CCS^{>}$, the main result of Section 6 still holds. The proofs need modification due to the new presence of τ cycles in translated terms, but the necessary changes are straighforward. We omit further details.

For the more general CBS translation operator, the relabeling operator of $CCS^{>}$ is inadequate. The way the CBS translation operator treats inputs is really relabeling from the outside and in (contrary to the usual inside-out style). Hence in general using the CBS translation operator corresponds to using a relabeling *relation* and not just a function. To get a simple solution to this problem, we may allow the use of relabeling relations in $CCS^{>}$. Then, using the same techniques as above, we are able to translate any CBS term with the main result of Section 6 still holding.

8 Conclusion

We have presented a translation from the synchronous broadcasting calculus CBS to the asynchronous handshaking calculus CCS with priority choice which "as much as possible" preserves the semantics of terms and which is of a compositional nature.[3]

For the fragment of CBS that does not contain the restriction/hiding operator, the translation is very simple. Even the somewhat lenghty definition of CON should be intuitively clear. The use of priority choice allows us to completely ignore discards in the translation. This avoids many of the problems encountered in [1] and is clearly an advantage.

It is an interesting fact, that the priority choice operator provides the additional control needed to simulate multiway synchronizations. With only nondeterministic choice at hand, such control is not possible; either deadlocks become possible, or we cannot ensure correct termination of sequences. In [4] it is proved that although any priority closed term of $CCS^{>}$ may be translated to a strongly bisimilar term in CCS, this can only be done at the price of expanding all parallels. This strongly supports the claim that no compositional translation from CBS to pure CCS exists. Moreover by noting that the situation does not improve even if it is enough to translate strong bisimilarity to weak bisimilarity, we may in fact prove such a claim.

We see no further problems in generalizations to cope with value passing

[3] Strictly speaking, the presence of the control process CON makes the translation noncompositional, but CON is a fixed expression that is completely independent of the term being translated. It may be viewed as an environment in which the real translation takes place.

and perhaps a prioritized version of CBS. In the future, investigation of the relationship between synchronous and asynchronous calculi may show that priorities indeed provide the missing link between these two different views on concurrency. An important point is that both discards and refusal sets are ways of encoding negative premises.

Currently we are working on using broadcasts to simulate priorities. This is a complex problem, which is solvable only because broadcast may be used for "testing" the capabilities of a terms environment.

References

[1] Uno Holmer. Interpreting Broadcast Communication in SCCS. In *LNCS 715*. Springer Verlag, 1993.

[2] Claus Torp Jensen. CCS with Priorities. Master's thesis, Department of Computer Science, Aarhus University, 1992.

[3] Claus Torp Jensen. The Concurrency Workbench with Priorities. In *LNCS 575*. Springer Verlag, 1992.

[4] Claus Torp Jensen. *Prioritized and Independent Actions in Distributed Computer Systems*. PhD thesis, Department of Computer Science, Aarhus University, 1994.

[5] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[6] G. D. Plotkin. A Structural Approach to Operational Semantics. DAIMI FN 19, Computer Science Department, Aarhus University, 1981.

[7] K. V. S. Prasad. A Calculus of Broadcasting Systems. In *LNCS 493*. Springer Verlag, 1991.

[8] K. V. S. Prasad. A Calculus of Value Broadcasts. In *LNCS 694*. Springer Verlag, 1993.

[9] K. V. S. Prasad. Broadcasting with Priority. Programming Methodology Group Report 75, Chalmers University of Technology, 1993.