Chapter 4

A short note on (modelling with) CCS

4.1 Congruence property of strong bisimilarity w.r.t. choice

We want to prove that for any CCS processes p_1, p_2, q we have that

$$p_1 \simeq p_2 \quad \text{implies} \quad p_1 + q \simeq p_2 + q$$

Let us assume that $p_1 \simeq p_2$, i.e., that there exists a strong bisimulation R such that $p_1 \; R \; p_2$.[1]

We want to find a relation R' such that:

1. R' is a strong bisimulation (i.e., $R' \subseteq \Phi(R')$;
2. $p_1 + q \; R' \; p_2 + q$

Let us define R' as follows and then prove that it is a strong bisimulation:

$$R' = \{ \, (p_1 + q, p_2 + q) \, \} \cup R \cup Id$$

where $Id = \{ \, (p, p) \mid p \text{ is a CCS process} \, \}$ is the identity relation.

The pairs in R' come from either $\{ \, (p_1 + q, p_2 + q) \, \}$, R or Id. Let us consider the various cases.

[1] We recall that a relation R is a strong bisimulation if $R \subseteq \Phi(R)$.

- Take any $(s_1, s_2) \in R$ and μ, s_1' such that $s_1 \xrightarrow{\mu} s_1'$. We want to prove that there exists s_2' with $s_2 \xrightarrow{\mu} s_2'$ and $s_1'\ R'\ s_2'$. But since $(s_1, s_2) \in R$ we know that there exists such a s_2' with $(s_1', s_2') \in R \subseteq R'$.
- Analogously to the previous case, take any $(s_1, s_2) \in R$ and μ, s_2' such that $s_2 \xrightarrow{\mu} s_2'$. We want to prove that there exists s_1' with $s_1 \xrightarrow{\mu} s_1'$ and $s_1'\ R'\ s_2'$. But since $(s_1, s_2) \in R$ we know that there exists such a s_1' with $(s_1', s_2') \in R \subseteq R'$.
- take any $(s, s) \in Id$ and μ, s' such that $s \xrightarrow{\mu} s'$. We want to prove that there exists s'' with $s \xrightarrow{\mu} s''$ and $s'\ R'\ s''$. We take $s'' = s'$ and we are done, because $(s', s') \in Id \subseteq R'$.
- take $(p_1 + q, p_2 + q)$ and μ, p_1' such that $p_1 + q \xrightarrow{\mu} p_1'$. We want to prove that there exists p' with $p_2 + q \xrightarrow{\mu} p'$ and $p_1'\ R'\ p'$. Since $p_1 + q \xrightarrow{\mu} p_1'$, by the operational semantics of CCS it must be the case that either $p_1 \xrightarrow{\mu} p_1'$ or $q \xrightarrow{\mu} p_1'$.
  - If $p_1 \xrightarrow{\mu} p_1'$, since $p_1\ R\ p_2$, there exists p_2' with $p_2 \xrightarrow{\mu} p_2'$ and $p_1'\ R\ p_2'$. Then $p_2 + q \xrightarrow{\mu} p_2'$ and $(p_1', p_2') \in R \subseteq R'$, so we take $p' = p_2'$ and we are done.
  - If $q \xrightarrow{\mu} p_1'$, then $p_2 + q \xrightarrow{\mu} p_1'$ with $(p_1', p_1') \in Id \subseteq R'$, so we take $p' = p_1'$ and we are done.
- take $(p_1 + q, p_2 + q)$ and μ, p_2' such that $p_2 + q \xrightarrow{\mu} p_2'$. We want to prove that there exists p' with $p_1 + q \xrightarrow{\mu} p'$ and $p'\ R'\ p_2'$. The proof is analogous to the previous case: complete it as an exercise.

4.2 From imperative languages to CCS

We sketch some ideas on how to model the constructs of a simple imperative language in CCS.

4.2.1 Modelling (shared) variable

Suppose x is a variable whose possible values range over a finite domain $\{v_1, ..., v_n\}$. Such variables can have n different states $X_1, X_2, ..., X_n$, depending on the current value it stores. In any such state, we can perform a write operation, changing the value stored in the variable, or we can read the current value. We can model this situation by considering (recursively defined processes):

$$\begin{array}{lcl} X_1 & \stackrel{\text{def}}{=} & xw_1.X_1 + xw_2.X_2 + ... + xw_n.X_n + xr1.X1 \\ X_2 & \stackrel{\text{def}}{=} & xw_1.X_1 + xw_2.X_2 + ... + xw_n.X_n + xr2.X2 \\ ... & & \\ X_n & \stackrel{\text{def}}{=} & xw_1.X_1 + xw_2.X_2 + ... + xw_n.X_n + xrn.Xn \end{array}$$

where:

- in any state X_i, a message received over the channel xw_j is used to change the state to X_j;
- in the state X_i, a message on channel xr_j is accepted if and only if $j = i$.

For example, we have

$$X_1 \mid \overline{xr_1}.\overline{xw_2}.\mathbf{nil} \xrightarrow{\tau} X_1 \mid \overline{xw_2}.\mathbf{nil} \xrightarrow{\tau} X_2 \mid \mathbf{nil}$$

4.2.2 Termination

To represent sequential composition of commands, we can use a dedicated channel *done* over which a message is sent when the current command is terminated. The message will be received by the continuation. In the following we let *Done* denote the process

$$Done \stackrel{\text{def}}{=} \overline{done}.\mathbf{nil}$$

4.2.3 Variable allocation

A statement like

$$\text{var } x$$

can be modelled by the allocation of an uninitialized variable, together with the termination message:

$$xw_1.X_1 + xw_2.X_2 + ... + xw_n.X_n \mid Done$$

4.2.4 Assignment

An assignment like

$$x := i$$

can be modelled by sending a message over the channel xw_i to the process that manages the variable x, after which the termination message can be sent:

$$\overline{xw_i}.Done$$

4.2.5 Skip

A skip statement ca be translated directly as $\tau.Done$ or simply $Done$.

4.2.6 Sequential composition

Let P_1 be the CCS process modelling the command c_1 and P_2 the CCS process modelling c_2, then we could try to model the sequential composition

$$c_1; c_2$$

simply as

$$P_1 \mid done.P_2$$

but this solution is unfortunate, because when considering several processes composed sequentially, like $c_1; c_2; c_3$, then the termination signal produced by P_1 could activate P_3 instead of P_2. To amend the situation, we can introduce a restricted channel d, which is used to rename the termination channel used by P_1 (while P_2 will still use channel $done$):

$$(P_1[d/done] \mid d.P_2)\backslash d$$

4.2.7 Conditional statement

Let P_1 be the CCS process modelling the command c_1 and P_2 the CCS process modelling c_2, then we can model the conditional statement

$$\textbf{if } x = i \textbf{ then } c_1 \textbf{ else } c_2$$

as the CCS process that executes P_1 if the value i can be read from x and P_2 if a value different than i can be read from x

$$\overline{xr_1}.P_2 + ... + \overline{xr_{i-1}}.P_2 + \overline{xr_i}.P_1 + \overline{xr_{i+1}}.P_2 + ... + \overline{xr_n}.P_2$$

4.2.8 Iteration

Let P be the CCS process modelling the command c, then we can model the while statement

$$\textbf{while } x = i \textbf{ do } c$$

by using the recursive process

$$\textbf{rec } W.\ \overline{xr_1}.Done + ... + \overline{xr_{i-1}}.Done + \overline{xr_i}.(P[d/done] \mid d.W)\backslash d + \overline{xr_{i+1}}.Done + ... + \overline{xr_n}.Done$$

that, in the case the value i can be read from x, activates the continuation

$$(P[d/done] \mid d.\textbf{rec } W.\ (...))\backslash d$$

and in all the other cases it activates the termination process *Done*.

4.2.9 Concurrent execution

Finally, we can of course allow for concurrent execution of commands. Let P_1 be the CCS process modelling the command c_1 and P_2 the CCS process modelling c_2, then we could try to model the parallel composition

$$c_1 \mid c_2$$

as the process that terminates when both P_1 and P_2 have terminated:

$$(\ P_1[d_1/done] \mid (P_2[d_2/done]) \mid d_1.d_2.Done\ )\backslash d_1 \backslash d_2$$

Note that we can use the simpler process

$$d_1.d_2.Done$$

to wait for the termination of $P_1[d_1/done]$ and $P_2[d_2/done]$ instead of the more complex process

$$d_1.d_2.Done + d_2.d_1.Done$$

because the termination message cannot be released anyway until both P_1 and P_2 have terminated.

4.2.10 Optimization

Of course, all the τ moves resulting from the synchronisation over termination messages can be avoided if we give a different, slightly more involved, translation that plugs in a process its continuation in place of the *Done* process. You can investigate, as an exercise, how this translation can be formalised.