Renaming and linking

Canonical buffer: $B \overset{\text{def}}{=} i(x) \cdot o(x) \cdot B$

Relationship to Cop?

One more operator: action renaming function $f$
Renaming and linking

Canonical buffer: \( B \overset{\text{def}}{=} i(x).\overline{\sigma}(x).B \)

Relationship to Cop?

One more operator: action renaming function \( f \)

1. Respects complements: \( f(\overline{a}) = \overline{f(a)} \)

2. Conserves \( \tau \): \( f(\tau) = \tau \)

\( b_1 / a_1, \ldots, b_n / a_n \) is the \( f \) that

- renames \( a_i \) to \( b_i \) (and \( \overline{a_i} \) to \( \overline{b_i} \))
Renaming and linking

Canonical buffer: $B \overset{\text{def}}{=} i(x).\overline{o(x)}.B$

Relationship to Cop?

One more operator: action renaming function $f$

1. Respects complements: $f(\overline{a}) = \overline{f(a)}$
2. Conserves $\tau$: $f(\tau) = \tau$

$b_1/a_1, \ldots, b_n/a_n$ is the $f$ that

- renames $a_i$ to $b_i$ (and $\overline{a_i}$ to $\overline{b_i}$)
- and leaves any other action $c$ unchanged

Building an $n$-place buffer

$B \overset{\text{def}}{=} i(x).\overline{o(x)}.B$

Transition rule

Associated with $f$ is the renaming operator $[f]$

$$R([f]) \quad \frac{E[f] \overset{b}{\rightarrow} F[f]}{E \overset{a}{\rightarrow} F} \quad b = f(a)$$

Example: Cop is $B[\text{in}/i, \text{out}/o]$

Assuming e.g $\text{in}/i$ maps each action $i(v)$ to $\text{in}(v)$
Building an $n$-place buffer

$$B \overset{\text{def}}{=} i(x) \cdot o(x) \cdot B$$

$B_1 \equiv B[\text{o}_1/o]$  
$B_{j+1} \equiv B[\text{o}_j/i, \text{o}_{j+1}/o] \quad 1 \leq j < n-1$  
$B_n \equiv B[\text{o}_{n-1}/i]$

$$B(n) \equiv (B_1 \mid \ldots \mid B_n) \backslash \{\text{o}_1, \ldots, \text{o}_{n-1}\}$$

A scheduler

Problem: assume $n$ tasks when $n > 1$.  
- $a_i$ initiates the $i$th task

- $b_i$ signals its completion
A scheduler

Problem: assume $n$ tasks when $n > 1$.
  - $a_i$ initiates the $i$th task
  - $b_i$ signals its completion

The scheduler plans the order of task initiation, ensuring

1. actions $a_1 \ldots a_n$ carried out cyclically
2. tasks may terminate in any order
3. but a task cannot be restarted until its previous operation has finished. ($a_i$ and $b_i$ happen alternately for each $i$. )

A scheduler

Problem: assume $n$ tasks when $n > 1$.
  - $a_i$ initiates the $i$th task
  - $b_i$ signals its completion

The scheduler plans the order of task initiation, ensuring

1. actions $a_1 \ldots a_n$ carried out cyclically
2. tasks may terminate in any order
3. ...
A scheduler

Problem: assume n tasks when n > 1.

- \( a_i \) initiates the \( i \)th task
- \( b_i \) signals its completion

The scheduler plans the order of task initiation, ensuring
1. actions \( a_1 \ldots a_n \) carried out cyclically
2. tasks may terminate in any order
3. but a task cannot be restarted until its previous operation has finished. (\( a_i \) and \( b_i \) happen alternately for each \( i \).)

A simple cycler: \( \text{Cy}' \overset{\text{def}}{=} a.c.b.d.\text{Cy}' \)

Solution using \( n \) simple cyclers?

\[
\begin{align*}
\text{Cy}_1' &\equiv \text{Cy}'[a_1/a, c_1/c, b_1/b, \overline{c_n}/d] \\
\text{Cy}_i' &\equiv (d.\text{Cy}')[a_i/a, c_i/c, b_i/b, \overline{c_{i-1}}/d] \quad 1 < i \leq n \\
(Cy_1' | \ldots | Cy_n') \setminus \{c_1, \ldots, c_n\}
\end{align*}
\]

When \( n = 4 \). What is wrong?

A scheduler

Problem: assume \( n \) tasks when \( n > 1 \).

- \( a_i \) initiates the \( i \)th task
- \( b_i \) signals its completion

The scheduler plans the order of task initiation, ensuring
1. actions \( a_1 \ldots a_n \) carried out cyclically
2. tasks may terminate in any order
3. but a task cannot be restarted until its previous operation has finished. (\( a_i \) and \( b_i \) happen alternately for each \( i \).)

A simple cycler: \( \text{Cy}' \overset{\text{def}}{=} a.c.b.d.\text{Cy}' \)
A solution: give up simple cycler

\[ Cy \overset{\text{def}}{=} a.c.(b.d.Cy + d.b.Cy) \]

\[ Cy_1 \equiv Cy[a_1/a, c_1/c, b_1/b, \tau_n/d] \]
\[ Cy_i \equiv (d.Cy)[a_i/a, c_i/c, b_i/b, \tau_{i-1}/d] \quad 1 < i \leq n \]
\[ (Cy_1 | \ldots | Cy_n) \backslash \{c_1, \ldots, c_n\} \]

How do we know it is right?

Summary

1. Introduced syntax of CCS: prefix, sum, parallel composition, restriction, renaming
Summary

1. Introduced syntax of CCS: prefix, sum, parallel composition, restriction, renaming
2. Introduced two types of transition $a \rightarrow$ and $a \Rightarrow$ and rules for their derivation
3. Introduced two types of transition graph that abstracts from derivation of transitions
4. Introduced Flow Graphs

Reading: Chapters 1 and 2, Robin Milner *Communication and Concurrency*, Prentice-Hall, 1989