Renaming and linking

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

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

Relationship to $\text{Cop}$?
Renaming and linking

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

Relationship to Cop ?

One more operator: action renaming function \( f \)
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: \( f(\overline{a}) = \overline{f(a)} \)
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 $\tau$: $f(\tau) = \tau$
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
Renaming and linking

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

Relationship to \( \text{Cop} \)?

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

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
Transition rule

Associated with \( f \) is the renaming operator \([f]\)

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

Example: Cop is \( B[\text{in/i, out/o}] \)
Assuming e.g \( \text{in/i} \) maps each action \( i(v) \) to \( \text{in}(v) \)
Building an $n$-place buffer

\[
B \overset{\text{def}}{=} i(x).\overline{o}(x).B
\]
Building an $n$-place buffer

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

Diagram:

- $B_1 \xrightarrow{i} B \xrightarrow{i} \cdots \xrightarrow{i} B \xrightarrow{i} B_n \xrightarrow{o}$
- $B_1 \xrightarrow{i} B_1 \xrightarrow{i} \cdots \xrightarrow{i} B_1 \xrightarrow{i} B_n \xrightarrow{o}$
Building an $n$-place buffer

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

$$B_1 \equiv B[o_1/o]$$
$$B_{j+1} \equiv B[o_j/i, o_{j+1}/o] \quad 1 \leq j < n-1$$
$$B_n \equiv B[o_{n-1}/i]$$
Building an $n$-place buffer

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

\[ B_1 \equiv B[o_1/o] \]
\[ B_{j+1} \equiv B[o_j/i, o_{j+1}/o] \quad 1 \leq j < n - 1 \]
\[ B_n \equiv B[o_{n-1}/i] \]

\[ B(n) \equiv (B_1 | \ldots | B_n) \setminus \{o_1, \ldots, o_{n-1}\} \]
A scheduler

Problem: assume $n$ tasks when $n > 1$.

- $a_i$ initiates the $i$th task
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
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
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
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. 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: $Cy' \overset{\text{def}}{=} a.c.b.d.Cy'$
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: \( Cy' \overset{\text{def}}{=} a.c.b.d.Cy' \)
Solution using $n$ simple cyclers?

\[ Cy'_i \equiv Cy'[a_i/a, c_i/c, b_i/b, c_{i-1}/d] \quad 1 < i \leq n \]

\[ (Cy'_1 | \ldots | Cy'_n) \setminus \{c_1, \ldots, c_n\} \]
When $n = 4$. What is wrong?
A solution: give up simple cycler

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

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

\[ C_{y_1} \equiv C_y[a_1/a, c_1/c, b_1/b, \overline{c}_n/d] \]
\[ C_{y_i} \equiv (d.C_y)[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d] \quad 1 < i \leq n \]

\[ (C_{y_1} | \ldots | C_{y_n}) \backslash \{c_1, \ldots, c_n\} \]
A solution: give up simple cycler

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

\[ Cy_1 \equiv Cy[a_1/a, c_1/c, b_1/b, \bar{c}_n/d] \]
\[ Cy_i \equiv (d.Cy)[a_i/a, c_i/c, b_i/b, \bar{c}_{i-1}/d] \quad 1 < i \leq n \]

\[ (Cy_1 \mid \ldots \mid Cy_n) \setminus \{c_1, \ldots, c_n\} \]

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 $\xrightarrow{a}$ and $\xrightarrow{a}$ and rules for their derivation
Summary

1. Introduced syntax of CCS: prefix, sum, parallel composition, restriction, renaming
2. Introduced two types of transition $\xrightarrow{a}$ and $\xrightarrow{a}$ and rules for their derivation
3. Introduced two types of transition graph that abstracts from derivation of transitions
Summary

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

1. Introduced syntax of CCS: prefix, sum, parallel composition, restriction, renaming
2. Introduced two types of transition $\xrightarrow{a}$ and $\xrightarrow{a\ast}$ 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