Communication and Concurrency
Lecture 4

Colin Stirling (cps)

School of Informatics

30th September 2013

Renaming and linking

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

Renaming and linking

Canonical buffer: $\mathtt{B} \stackrel{\text{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$
Relationship to Cop ?

Renaming and linking

Canonical buffer: $\mathtt{B} \stackrel{\text{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$
Relationship to Cop ?
One more operator: action renaming function f

Renaming and linking

Canonical buffer: $\mathtt{B} \stackrel{\mathrm{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$
Relationship to Cop ?
One more operator: action renaming function f

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

Renaming and linking

Canonical buffer: $\mathtt{B} \stackrel{\mathrm{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$
Relationship to Cop ?
One more operator: action renaming function f

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

Renaming and linking

Canonical buffer: $\mathtt{B} \stackrel{\mathrm{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$
Relationship to Cop ?
One more operator: action renaming function f

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

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

Renaming and linking

Canonical buffer: $\mathtt{B} \stackrel{\mathrm{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$
Relationship to Cop ?
One more operator: action renaming function f

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

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

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

Renaming and linking

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

Relationship to $\mathtt{Cop}$?

One more operator: action renaming function f

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

$b_1/a_1, \ldots, b_n/a_n$ 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 $[f]$

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

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

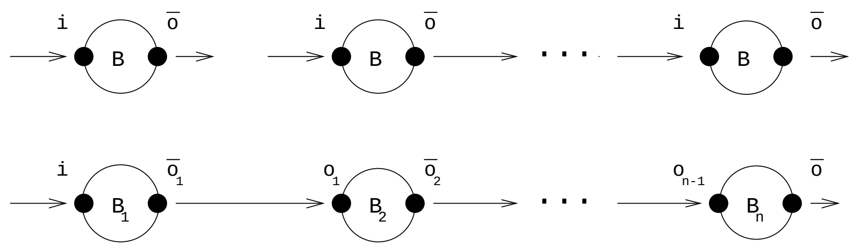
Assuming e.g $\mathtt{in}/\mathtt{i}$ maps each action $\mathtt{i}(v)$ to $\mathtt{in}(v)$

Building an n-place buffer

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

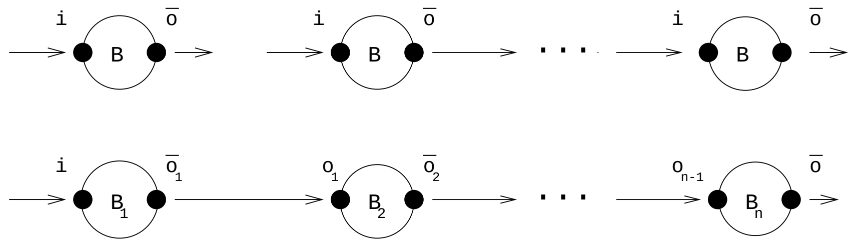
Building an n-place buffer

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



Building an n-place buffer

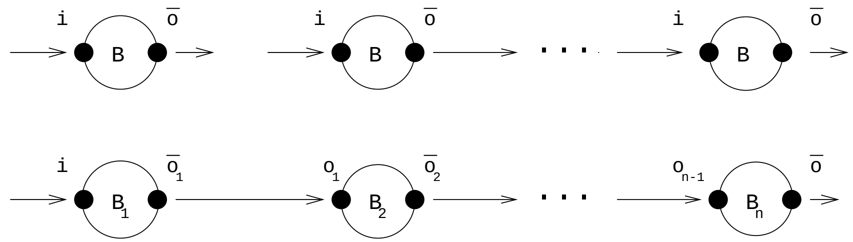
$$\mathtt{B} \stackrel{\text{def}}{=} \mathtt{i}(x).\overline{\mathtt{o}}(x).\mathtt{B}$$



$$\begin{array}{lcll} \mathtt{B}_1 & \equiv & \mathtt{B}[\mathtt{o}_1/\mathtt{o}] & \\ \mathtt{B}_{j+1} & \equiv & \mathtt{B}[\mathtt{o}_j/\mathtt{i}, \mathtt{o}_{j+1}/\mathtt{o}] & 1 \leq j < n-1 \\ \mathtt{B}_n & \equiv & \mathtt{B}[\mathtt{o}_{n-1}/\mathtt{i}] & \end{array}$$

Building an n-place buffer

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



$$\begin{array}{lcll} \mathtt{B}_1 & \equiv & \mathtt{B}[\mathtt{o}_1/\mathtt{o}] & \\ \mathtt{B}_{j+1} & \equiv & \mathtt{B}[\mathtt{o}_j/\mathtt{i}, \mathtt{o}_{j+1}/\mathtt{o}] & 1 \leq j < n-1 \\ \mathtt{B}_n & \equiv & \mathtt{B}[\mathtt{o}_{n-1}/\mathtt{i}] & \end{array}$$

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

A scheduler

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

- a_i initiates the ith task

A scheduler

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

- a_i initiates the ith task
- b_i signals its completion

A scheduler

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

- a_i initiates the ith 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 ith 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 ith 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 ith 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 ith 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: $\mathtt{Cy}' \stackrel{\text{def}}{=} a.c.b.d.\mathtt{Cy}'$

A scheduler

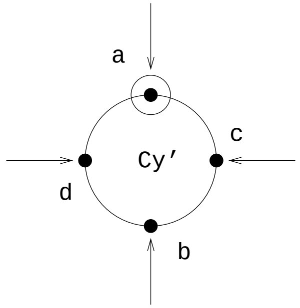
Problem: assume n tasks when $n > 1$.

- a_i initiates the ith task
- b_i signals its completion

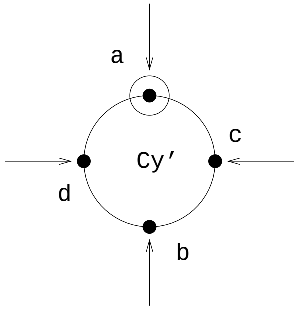
The scheduler plans the order of task initiation, ensuring

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: $\mathtt{Cy}' \stackrel{\text{def}}{=} a.c.b.d.\mathtt{Cy}'$

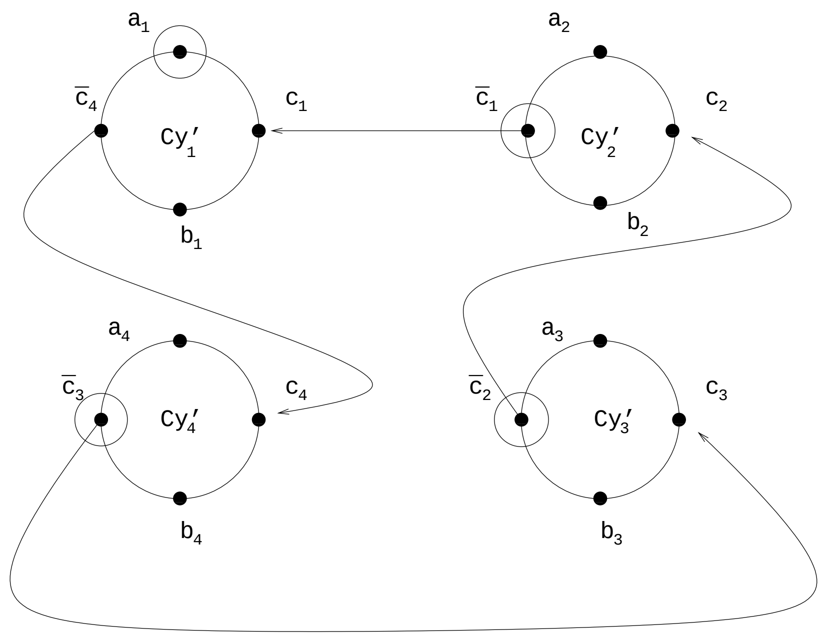


Solution using n simple cyclers ?



$$\begin{array}{lcl} \mathtt{Cy}'_1 & \equiv & \mathtt{Cy}'[a_1/a, c_1/c, b_1/b, \overline{c}_n/d] \\ \mathtt{Cy}'_i & \equiv & (d.\mathtt{Cy}')[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d] \quad 1 < i \leq n \\ & & \\ & & (\mathtt{Cy}'_1 \mid \ldots \mid \mathtt{Cy}'_n)\backslash\{c_1, \ldots, c_n\} \end{array}$$

When $n = 4$. What is wrong ?



A solution: give up simple cycler

$$\texttt{Cy} \stackrel{\text{def}}{=} a.c.(b.d.\texttt{Cy} + d.b.\texttt{Cy})$$

A solution: give up simple cycler

$$\texttt{Cy} \stackrel{\text{def}}{=} a.c.(b.d.\texttt{Cy} + d.b.\texttt{Cy})$$

$$\begin{array}{lcl} \texttt{Cy}_1 & \equiv & \texttt{Cy}[a_1/a, c_1/c, b_1/b, \overline{c}_n/d] \\ \texttt{Cy}_i & \equiv & (d.\texttt{Cy})[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d] \quad 1 < i \leq n \end{array}$$

$$(\texttt{Cy}_1 \mid \ldots \mid \texttt{Cy}_n) \backslash \{c_1, \ldots, c_n\}$$

A solution: give up simple cycler

$$\texttt{Cy} \stackrel{\text{def}}{=} a.c.(b.d.\texttt{Cy} + d.b.\texttt{Cy})$$

$$\begin{array}{lcl} \texttt{Cy}_1 & \equiv & \texttt{Cy}[a_1/a, c_1/c, b_1/b, \overline{c}_n/d] \\ \texttt{Cy}_i & \equiv & (d.\texttt{Cy})[a_i/a, c_i/c, b_i/b, \overline{c}_{i-1}/d] \quad 1 < i \leq n \end{array}$$

$$(\texttt{Cy}_1 \mid \ldots \mid \texttt{Cy}_n) \backslash \{c_1, \ldots, c_n\}$$

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 $\xrightarrow{a}$ 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 $\xRightarrow{a}$ 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 $\xRightarrow{a}$ 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