Topics in Concurrency

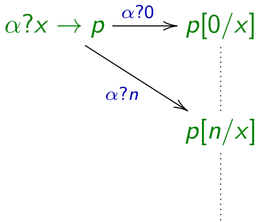
Lecture 3

Glynn Winskel

13 February 2019

Towards a more basic language

- Aim: removal of variables to reveal symmetry of input and output
- Transitions for value-passing carry labels τ, $a?n$, $a!n$



- This suggests introducing prefix $\alpha?n.p$ (as well as $\alpha!n.p$) and view $\alpha?x \to p$ as a sum $\sum_n \alpha?n.p[n/x]$ ↖ infinite sum
- View $\alpha?n$ and $\alpha!n$ as complementary actions
- Synchronization can only occur on complementary actions

Pure CCS

- Actions: a, b, c, ...
- Complementary actions: $\overline{a}$, $\overline{b}$, $\overline{c}$, ...
- Internal action: τ
- Notational convention: $\overline{\overline{a}} = a$
- Processes:

p	$::=$	$\lambda.p$	prefix	λ ranges over τ, a, $\overline{a}$ for any action a
	$\mid$	$\sum_{i \in I} p_i$	sum	I is an indexing set
	$\mid$	$p_0 \parallel p_1$	parallel	
	$\mid$	$p \backslash L$	restriction	L a set of actions
	$\mid$	$p[f]$	relabelling	f a function on actions
	$\mid$	P		process identifier

- Process definitions:

$$P \stackrel{\text{def}}{=} p$$

Transition rules for pure CCS

- **Nil process** no rules
- **Guarded processes**

$$\lambda.p \xrightarrow{\lambda} p$$

- **Sum**

$$\frac{p_j \xrightarrow{\lambda} p' \qquad j \in I}{\sum_{i \in I} p_i \xrightarrow{\lambda} p'_0}$$

- **Parallel composition**

$$\frac{p_0 \xrightarrow{\lambda} p'_0}{p_0 \parallel p_1 \xrightarrow{\lambda} p'_0 \parallel p_1} \qquad \frac{p_1 \xrightarrow{\lambda} p'_1}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0 \parallel p'_1}$$

$$\frac{p_0 \xrightarrow{a} p'_0 \qquad p_1 \xrightarrow{\bar{a}} p'_1}{p_0 \parallel p_1 \xrightarrow{\tau} p'_0 \parallel p'_1}$$

- **Restriction**

$$\frac{p \xrightarrow{\lambda} p' \qquad \lambda \notin L \cup \overline{L}}{p \setminus L \xrightarrow{\lambda} p' \setminus L} \quad \text{where } \overline{L} = \{\overline{a} \mid a \in L\}$$

- **Relabelling**

$$\frac{p \xrightarrow{\lambda} p'}{p[f] \xrightarrow{f(\lambda)} p'[f]}$$

where f is a function such that $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$

- **Identifiers**

$$\frac{p \xrightarrow{\lambda} p' \qquad P \stackrel{\text{def}}{=} p}{P \xrightarrow{\lambda} p'}$$

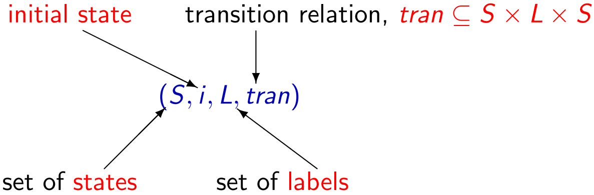
Transition systems

- Given a CCS process p, can construct its *transition system*
- A *transition system* is:

$$(S, i, L, tran)$$

Transition systems

- Given a CCS process p, can construct its *transition system*
- A *transition system* is:



Transition systems

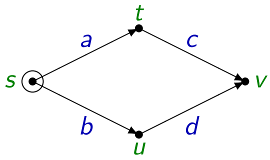
- Given a CCS process p, can construct its *transition system*
- A *transition system* is:

initial state — transition relation, $tran \subseteq S \times L \times S$

$$(S, i, L, tran)$$

set of states — set of labels

- Graphically:



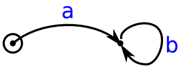
$$\begin{aligned} S &= \{s, t, u, v\} \\ i &= s \\ L &= \{a, b, c, d\} \\ tran &= \{\ (s, a, t), \\ &\quad\ \ (s, b, u), \\ &\quad\ \ (t, c, v), \\ &\quad\ \ (u, d, v)\ \} \end{aligned}$$

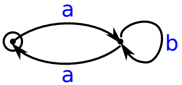
Transition systems from CCS

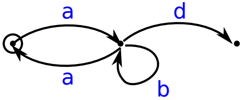
- Example: $(a \parallel \overline{b})[f]$ where $f(a) = w$ and $f(b) = w$
- Example: $a[f] \parallel \overline{b}[f]$ where $f(a) = w$ and $f(b) = w$

Realising transition systems

Give pure CCS terms for:

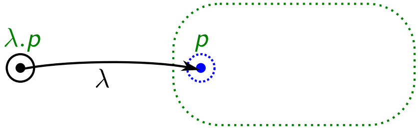






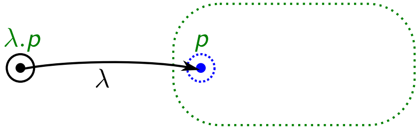
CCS operations on transition systems

- $\lambda.p$:

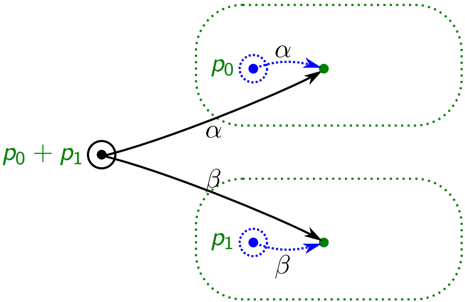


CCS operations on transition systems

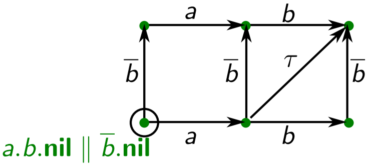
- $\lambda.p$:



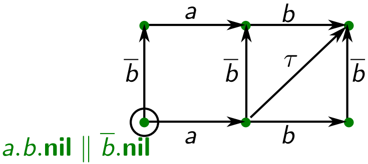
- $p_0 + p_1$:



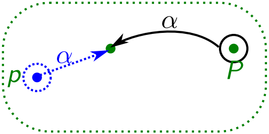
- $a.b \parallel \overline{b}$:



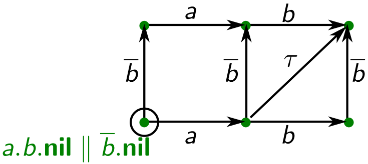
- $a.b \parallel \overline{b}$:



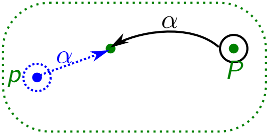
- P where $P \stackrel{\text{def}}{=} p$:



- $a.b \parallel \overline{b}$:



- P where $P \stackrel{\text{def}}{=} p$:



$p \setminus L, p[f]$: ...

A denotational semantics!

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$
nil	**nil**

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$
nil	**nil**
$(\tau \to p)$	$(\tau.\widehat{p})$

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \to p)$	$(\tau.\widehat{p})$	
$(\alpha!a \to p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \to p)$	$(\tau.\widehat{p})$	
$(\alpha!a \to p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m
$(\alpha?x \to p)$	$\sum_{m \in \mathbf{Num}} \alpha m.\widehat{p[m/x]}$	

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \rightarrow p)$	$(\tau.\widehat{p})$	
$(\alpha!a \rightarrow p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m
$(\alpha?x \rightarrow p)$	$\sum_{m \in \mathbf{Num}} \alpha m.\widehat{p[m/x]}$	
$(b \rightarrow p)$	$\widehat{p}$	if b evaluates to true
	nil	if b evaluates to false

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \to p)$	$(\tau.\widehat{p})$	
$(\alpha!a \to p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m
$(\alpha?x \to p)$	$\sum_{m \in \mathbf{Num}} \alpha m.\widehat{p[m/x]}$	
$(b \to p)$	$\widehat{p}$	if b evaluates to true
	nil	if b evaluates to false
$p_0 + p_1$	$\widehat{p_0} + \widehat{p_1}$	

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \to p)$	$(\tau.\widehat{p})$	
$(\alpha!a \to p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m
$(\alpha?x \to p)$	$\sum_{m \in \mathbf{Num}} \alpha m.\widehat{p[m/x]}$	
$(b \to p)$	$\widehat{p}$	if b evaluates to true
	nil	if b evaluates to false
$p_0 + p_1$	$\widehat{p_0} + \widehat{p_1}$	
$p_0 \parallel p_1$	$\widehat{p_0} \parallel \widehat{p_1}$	

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \to p)$	$(\tau.\widehat{p})$	
$(\alpha!a \to p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m
$(\alpha?x \to p)$	$\sum_{m\in\mathbf{Num}} \alpha m.\widehat{p[m/x]}$	
$(b \to p)$	$\widehat{p}$	if b evaluates to true
	nil	if b evaluates to false
$p_0 + p_1$	$\widehat{p_0} + \widehat{p_1}$	
$p_0 \parallel p_1$	$\widehat{p_0} \parallel \widehat{p_1}$	
$p \setminus L$	$\widehat{p} \setminus \{\alpha m \mid \alpha \in L \ \&\ m \in \mathbf{Num}\}$	

From value-passing to pure

A translation giving a pure CCS process $\widehat{p}$ from a value-passing CCS closed term p

p	$\widehat{p}$	
nil	**nil**	
$(\tau \to p)$	$(\tau.\widehat{p})$	
$(\alpha!a \to p)$	$\overline{\alpha m}.\widehat{p}$	where a evaluates to m
$(\alpha?x \to p)$	$\sum_{m \in \mathbf{Num}} \alpha m.\widehat{p[m/x]}$	
$(b \to p)$	$\widehat{p}$	if b evaluates to true
	nil	if b evaluates to false
$p_0 + p_1$	$\widehat{p_0} + \widehat{p_1}$	
$p_0 \parallel p_1$	$\widehat{p_0} \parallel \widehat{p_1}$	
$p \setminus L$	$\widehat{p} \setminus \{\alpha m \mid \alpha \in L \ \&\ m \in \mathbf{Num}\}$	
$P(a_1, \cdots, a_k)$	$P_{m_1,\cdots,m_k}$	where a_i evaluates to m_i

For every definition $P(x_1, \cdots, x_k)$, we have a collection of definitions $P_{m_1,\ldots,m_k}$ indexed by $m_1, \cdots, m_k \in \mathbf{Num}$.

Correspondence

Theorem

$$p \xrightarrow{\lambda} p' \ \textit{iff} \ \widehat{p} \xrightarrow{\widehat{\lambda}} \widehat{p'}$$

Recursion: an alternative

- Instead of a process

$$P \text{ where } P \stackrel{\text{def}}{=} p$$

we can use

$$\text{rec}(P = p)$$

with rule

$$\frac{p[\text{rec}(P = p)/P] \xrightarrow{\lambda} p'}{\text{rec}(P = p) \xrightarrow{\lambda} p'}$$

- Example: $\text{rec}(P = a.\mathbf{nil} + b.P)$

Recursion: an alternative

- Instead of a process

$$P \text{ where } P \stackrel{\text{def}}{=} p \text{ and } Q = q$$

we can use the notation

$$\mathsf{rec}_1(P = p, Q = q)$$

and for Q we can use

$$\mathsf{rec}_2(P = p, Q = q)$$

Recursion: an alternative

- Instead of a process

$$P \text{ where } P \stackrel{\text{def}}{=} p \text{ and } Q = q$$

we can use the notation

$$\mathsf{rec}_1(P = p, Q = q)$$

and for Q we can use

$$\mathsf{rec}_2(P = p, Q = q)$$

- Generally, instead of P_j where $P_i = p_i$ is a collection of definitions indexed by $i \in I$, can use

$$\mathsf{rec}_j(P_i = p_i)_{i \in I}$$

which is also written

$$\mathsf{rec}_j(\vec{P} = \vec{p})$$

Proofs of correctness

- By satisfying formulas in a logic
- By satisfying an equivalence