Topics in Concurrency
Lecture 3

Jonathan Hayman

18 October 2016
Towards a more basic language

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

\[
\alpha?x \rightarrow p \xrightarrow{\alpha?0} p[0/x] \\
\alpha?n \rightarrow p[n/x]
\]

- This suggests introducing prefix $\alpha?n.p$ (as well as $\alpha!n.p$) and view $\alpha?x \rightarrow p$ 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: $a, b, c, \ldots$
- Complementary actions: $\bar{a}, \bar{b}, \bar{c}, \ldots$
- Internal action: $\tau$
- Notational convention: $\bar{a} = a$
- Processes:

$$p ::= \lambda.p \quad \text{prefix} \quad \lambda \text{ ranges over } \tau, a, \bar{a}$$
$$\quad \sum_{i \in I} p_i \quad \text{sum} \quad I \text{ is an indexing set}$$
$$\quad p_0 \parallel p_1 \quad \text{parallel}$$
$$\quad p \setminus L \quad \text{restriction} \quad L \text{ a set of actions}$$
$$\quad p[f] \quad \text{relabelling} \quad f \text{ a function on actions}$$
$$\quad P$$

- Process definitions:

$$P \overset{\text{def}}{=} p$$
Transition rules for pure CCS

- **Nil process** no rules
- **Guarded processes**
  \[
  \lambda.p \xrightarrow{\lambda} p
  \]
- **Sum**
  \[
  \sum_{i \in I} p_i \xrightarrow{\lambda} p_0' \quad j \in I
  \]
- **Parallel composition**
  \[
  \begin{align*}
  p_0 \xrightarrow{\lambda} p_0' & \quad p_1 \xrightarrow{\lambda} p_1' \\
  p_0 \parallel p_1 \xrightarrow{\lambda} p_0' \parallel p_1 & \quad p_0 \parallel p_1 \xrightarrow{\lambda} p_0 \parallel p_1'
  \end{align*}
  \]
  \[
  \begin{align*}
  p_0 \xrightarrow{a} p_0' & \quad p_1 \xrightarrow{\overline{a}} p_1' \\
  p_0 \parallel p_1 \xrightarrow{\tau} p_0' \parallel p_1' & \quad p_0 \parallel p_1 \xrightarrow{\tau} p_0' \parallel p_1'
  \end{align*}
  \]
- **Restriction**
  \[
  p \xrightarrow{\lambda} p' \quad \lambda \notin L \cup \overline{L}
  \]
  where \( \overline{L} = \{ \overline{a} \mid a \in L \} \)

  \[
  p \setminus L \xrightarrow{\lambda} p' \setminus L
  \]

- **Relabelling**
  \[
  p \xrightarrow{\lambda} p'
  \]
  \[
  p[f] \xrightarrow{f(\lambda)} p'[f]
  \]
  where \( f \) is a function such that \( f(\tau) = \tau \) and \( f(\overline{a}) = \overline{f(a)} \)

- **Identifiers**
  \[
  p \xrightarrow{\lambda} p' \quad P \overset{\text{def}}{=} p
  \]
  \[
  P \xrightarrow{\lambda} p'
  \]
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:
  - initial state
  - transition relation, \( \text{tran} \subseteq S \times L \times S \)
  - set of states
  - set of labels

\[ (S, i, L, \text{tran}) \]
Given a CCS process $p$, can construct its transition system.

A transition system is:

- initial state
- transition relation, $tran \subseteq S \times L \times S$
- set of states
- set of labels

**Graphically:**

```latex
S = \{s, t, u, v\}
i = s
L = \{a, b, c, d\}
tran = \{(s, a, t), (s, b, u), (t, c, v), (u, d, v)\}
```
Transition systems from CCS

- Example: \((a ∥ \overline{b})[f]\) where \(f(a) = w\) and \(f(b) = w\)
- Example: \(a[f] ∥ \overline{b}[f]\) where \(f(a) = w\) and \(f(b) = w\)
Realising transition systems

Give pure CCS terms for:

1. 

2. 

3. 

4. 

CCS operations on transition systems

\[ \lambda.p: \]

\[ \lambda.p \]

\[ \lambda \]
CCS operations on transition systems

- $\lambda.p$:

- $p_0 + p_1$:
\[ a.b \parallel \overline{b}: \]

\[ a.b.\text{nil} \parallel \overline{b}.\text{nil} \]
- $a.b \parallel \overline{b}$:

- $a.b.\text{nil} \parallel \overline{b}.\text{nil}$

- $P$ where $P \overset{\text{def}}{=} p$:
\[ a.b \parallel \bar{b} : \]

\[ a.b.\text{nil} \parallel \bar{b}.\text{nil} \]

\( P \) where \( P \overset{\text{def}}{=} p : \)

\[ p \setminus L, p[f] : \ldots \]

A denotational semantics!
A translation giving a pure CCS process $\hat{p}$ from a value-passing CCS closed term $p$
A translation giving a pure CCS process $\hat{p}$ from a value-passing CCS closed term $p$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>$(\tau \to p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
</tbody>
</table>
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>$(\tau \rightarrow p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \rightarrow p)$</td>
<td>$\alpha m.\hat{p}$</td>
</tr>
</tbody>
</table>

where $a$ evaluates to $m$
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{nil}$</td>
<td>$\text{nil}$</td>
</tr>
<tr>
<td>$(\tau \rightarrow p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \rightarrow p)$</td>
<td>$\alpha m.\hat{p}$ where $a$ evaluates to $m$</td>
</tr>
<tr>
<td>$(\alpha?x \rightarrow p)$</td>
<td>$\sum_{m \in \text{Num}} \alpha m.p[m/x]$</td>
</tr>
</tbody>
</table>
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>$(\tau \to p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \to p)$</td>
<td>$\overline{\alpha m.\hat{p}}$ where $a$ evaluates to $m$</td>
</tr>
<tr>
<td>$(\alpha?x \to p)$</td>
<td>$\sum_{m \in \text{Num}} \alpha m.\hat{p}[m/x]$</td>
</tr>
<tr>
<td>$(b \to p)$</td>
<td>$\hat{p}$ if $b$ evaluates to true \nnil if $b$ evaluates to false</td>
</tr>
</tbody>
</table>
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{nil}$</td>
<td>$\text{nil}$</td>
</tr>
<tr>
<td>$(\tau \to p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \to p)$</td>
<td>$\alpha m.\hat{p}$</td>
</tr>
<tr>
<td></td>
<td>where $a$ evaluates to $m$</td>
</tr>
<tr>
<td>$(\alpha?x \to p)$</td>
<td>$\sum_{m \in \text{Num}} \alpha m.p[m/x]$</td>
</tr>
<tr>
<td>$(b \to p)$</td>
<td>$\hat{p}$</td>
</tr>
<tr>
<td></td>
<td>$\text{nil}$</td>
</tr>
<tr>
<td>$p_0 + p_1$</td>
<td>$\hat{p}_0 + \hat{p}_1$</td>
</tr>
</tbody>
</table>
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>$(\tau \to p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \to p)$</td>
<td>$\overline{\alpha m.\hat{p}}$</td>
</tr>
<tr>
<td>$\text{where } a \text{ evaluates to } m$</td>
<td></td>
</tr>
<tr>
<td>$(\alpha?x \to p)$</td>
<td>$\sum_{m \in \text{Num}} \alpha m.\hat{p}[m/x]$</td>
</tr>
<tr>
<td>$(b \to p)$</td>
<td>$\hat{p}$</td>
</tr>
<tr>
<td>$\text{if } b \text{ evaluates to true}$</td>
<td></td>
</tr>
<tr>
<td>nil</td>
<td>$\hat{p}$</td>
</tr>
<tr>
<td>$\text{if } b \text{ evaluates to false}$</td>
<td></td>
</tr>
<tr>
<td>$p_0 + p_1$</td>
<td>$\hat{p}_0 + \hat{p}_1$</td>
</tr>
<tr>
<td>$p_0 \parallel p_1$</td>
<td>$\hat{p}_0 \parallel \hat{p}_1$</td>
</tr>
</tbody>
</table>
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>$(\tau \rightarrow p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \rightarrow p)$</td>
<td>$\alpha m.\hat{p}$</td>
</tr>
<tr>
<td>$(\alpha?x \rightarrow p)$</td>
<td>$\sum_{m \in \text{Num}} \alpha m.p[m/x]$</td>
</tr>
<tr>
<td>$(b \rightarrow p)$</td>
<td>$\hat{p}$ if $b$ evaluates to true  \n nil if $b$ evaluates to false</td>
</tr>
<tr>
<td>$p_0 + p_1$</td>
<td>$\hat{p}_0 + \hat{p}_1$</td>
</tr>
<tr>
<td>$p_0 \parallel p_1$</td>
<td>$\hat{p}_0 \parallel \hat{p}_1$</td>
</tr>
<tr>
<td>$p \setminus L$</td>
<td>$\hat{p} \setminus {\alpha m \mid \alpha \in L &amp; m \in \text{Num}}$</td>
</tr>
</tbody>
</table>
From value-passing to pure

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\hat{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>$(\tau \to p)$</td>
<td>$(\tau.\hat{p})$</td>
</tr>
<tr>
<td>$(\alpha!a \to p)$</td>
<td>$\alpha.m.\hat{p}$ where $a$ evaluates to $m$</td>
</tr>
<tr>
<td>$(\alpha?x \to p)$</td>
<td>$\sum_{m \in \text{Num}} \alpha.m.p[m/x]$</td>
</tr>
<tr>
<td>$(b \to p)$</td>
<td>$\begin{cases} \hat{p} &amp; \text{if } b \text{ evaluates to true} \ \text{nil} &amp; \text{if } b \text{ evaluates to false} \end{cases}$</td>
</tr>
<tr>
<td>$p_0 + p_1$</td>
<td>$\hat{p}_0 + \hat{p}_1$</td>
</tr>
<tr>
<td>$p_0 \parallel p_1$</td>
<td>$\hat{p}_0 \parallel \hat{p}_1$</td>
</tr>
<tr>
<td>$p \setminus L$</td>
<td>$\hat{p} \setminus {\alpha.m \mid \alpha \in L &amp; m \in \text{Num}}$</td>
</tr>
<tr>
<td>$P(a_1, \ldots, a_k)$</td>
<td>$P_{m_1, \ldots, m_k}$ where $a_i$ evaluates to $m_i$</td>
</tr>
</tbody>
</table>

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

Theorem

\[ p \xrightarrow{\lambda} p' \text{ iff } \hat{p} \xrightarrow{\hat{\lambda}} \hat{p}' \]
Recursion: an alternative

Instead of a process

\[ P \text{ where } P \overset{\text{def}}{=} p \]

we can use

\[ \text{rec}(P = p) \]

with rule

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

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

Example: \( \text{rec}(P = a.\text{nil} + b.P) \)
Recursion: an alternative

Instead of a process

\( P \) where \( P \overset{\text{def}}{=} p \) and \( Q = q \)

we can use the notation

\[ \text{rec}_1(P = p, Q = q) \]

and for \( Q \) we can use

\[ \text{rec}_2(P = p, Q = q) \]
Recursion: an alternative

Instead of a process

\[ P \text{ where } P \overset{\text{def}}{=} p \text{ and } Q = q \]

we can use the notation

\[ \text{rec}_1(P = p, Q = q) \]

and for \( Q \) we can use

\[ \text{rec}_2(P = p, Q = q) \]

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

\[ \text{rec}_j(P_i = p_i)_{i \in I} \]

which is also written

\[ \text{rec}_j(\vec{P} = \vec{p}) \]
Proofs of correctness

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