Foundations of Programming

– Concurrency –

Session 11 – April 22, 2002

Uwe Nestmann

EPFL-LAMP

Goals

- Session 11
  - from CCS to π-calculus
    - pragmatic
    - syntactic
    - (semantic)
- Session 12
  - programming in Nomadic Pict

Warming Up / Repetition

Is **inequality** an equivalence relation?

Check $A \approx \mathbf{0}$ for $A \stackrel{\mathrm{def}}{=} \tau.A \ldots$
$\ldots$ and compare to $A \sim \mathbf{0}$.

Unbounded Structures: Stacks (I)

$\mathcal{N} := \{ \text{empty} \} \cup \{ \text{push}_v, \text{pop}_v \}_{v \in \mathbf{V}}$
$\vec{v} \in \mathbf{V}^*$

$\text{Stack}_{\vec{w}}(\, \text{empty}, \overrightarrow{\text{push}_v}, \overrightarrow{\text{pop}_v}\,)$

$$\text{Stack} \stackrel{\mathrm{def}}{=} \sum_v \text{push}_v.\text{Stack}_v + \overline{\text{empty}}.\text{Stack}$$

$$\text{Stack}_{v,\vec{w}} \stackrel{\mathrm{def}}{=} \sum_u \text{push}_u.\text{Stack}_{u,v,\vec{w}} + \overline{\text{pop}_v}.\text{Stack}_{\vec{w}}$$

Unbounded Structures: Stacks (II)

$$\mathcal{N} := \{\ \mathsf{empty}, \mathsf{drop}\ \} \cup \{\ \mathsf{push}_v, \mathsf{pop}_v, \mathsf{not}_v, \mathsf{pull}_v\ \}_{v \in \mathbf{V}}$$

$$E(\ \overrightarrow{\mathcal{N}}\ ) := E(\ \mathsf{empty}, \mathsf{drop}, \overrightarrow{\mathsf{push}_v}, \overrightarrow{\mathsf{not}_v}, \overrightarrow{\mathsf{pop}_v}, \overrightarrow{\mathsf{pull}_v}\ )$$

$$X\langle\ \overrightarrow{\mathcal{N}}\ \rangle \frown Y\langle\ \overrightarrow{\mathcal{N}}\ \rangle := (\boldsymbol{\nu}\vec{a}, b, \vec{c})$$

$$(\ X\langle\ \overrightarrow{\mathcal{N}}\ \rangle[^{\vec{a},b,\vec{c}}/_{\overrightarrow{\mathsf{not}_v},\mathsf{drop},\overrightarrow{\mathsf{pull}_v}}] \mid Y\langle\ \overrightarrow{\mathcal{N}}\ \rangle[^{\vec{a},b,\vec{c}}/_{\overrightarrow{\mathsf{push}_v},\mathsf{empty},\overrightarrow{\mathsf{pop}_v}}]\ )$$

$$\begin{aligned}
E &:= \textstyle\sum_v \mathsf{push}_v.(C_v \frown E) + \overline{\mathsf{empty}}.E \\
C_v &:= \textstyle\sum_u \mathsf{push}_u.(C_u \frown C_v) + \overline{\mathsf{pop}_v}.D \\
D &:= \textstyle\sum_u \mathsf{pull}_u.C_u + \mathsf{drop}.E \\
S_{\vec{v}} &:= C_{v_1} \frown \cdots \frown C_{v_n} \frown E
\end{aligned}$$

$$\mathsf{Stack}_{\vec{v}} \approx S_{\vec{v}}$$

Criticism

Example:

Calculate the states for the transition sequence $\xrightarrow{\mathsf{push}_1}\xrightarrow{\mathsf{push}_2}\xrightarrow{\mathsf{pop}_2}$ and “stabilize” the remainder.

- ☐ D’s cannot be reused for storing *new* values (neither inner nor outer D’s!).
- ☐ E’s are never “used”, pile up and stay around. (Note that, although $E \frown E \sim E$, explicit garbage collection would be required.)

Unbounded Structures: Stacks (III)

$$
\begin{array}{lll}
E & := & \sum_v \mathsf{push}_v.C_v + \overline{\mathsf{empty}}.E \\
C_v & := & \sum_u \mathsf{push}_u.(C_u \frown C_v) + \overline{\mathsf{pop}_v}.D + \overline{\mathsf{not}_v}.D \\
D & := & \sum_u \mathsf{pull}_u.C_u + \mathsf{drop}.E + \sum_u \mathsf{push}_u.C_u \\
S_{\vec{v}} & := & C_{v_1} \frown \cdots \frown C_{v_n} \frown E
\end{array}
$$

- What are the problems of this “implementation”?
- Think about how to derive unbounded buffers from unbounded stacks . . .

Turing Power

A **Turing-machine** consists of:

- a finite alphabet of symbols
- an infinite tape
- a finite control mechanism
- movement or r/w-head to left or right

A Turing-machine can be nicely simulated with concurrent processes by two stacks (the tape). Neither an infinite alphabet nor infinite summation is necessary for this. [Milner 89]

Turing Power (II)

$\Longrightarrow$

1. The language/calculus of concurrent process expressions is Turing powerful.
2. Since the halting problem for the representation TM of some Turing machine can be encoded as TM $\approx \ldots$
   weak bisimulation is ...

Expressiveness

Still, concurrent process expressions are,
in some particular sense, not expressive enough:
it is not possible to *cut out dead cells* E.

If we had the possibility to *dynamically change the interconnection structure* among process components, cells could *drop out* by connecting their left and right neighbors together.

One way to do this is the transmission of "*channels over channels*".

Name-Passing Syntax

negative actions $\overline{a}\langle v\rangle$: *send name v over name a.*

positive actions $a(x)$: *receive any name, say v, over name a and "bind the result" to name x.*

Binding results in *substitution* of the formal parameter x by the actual parameter v.

polyadic communication $\overline{a}\langle\vec{v}\rangle$ and $a(\vec{x})$ ($\vec{x}$ pairwise different) *transmit many values at a time*.

Hand-Over Protocol

("external" slide)

Syntax Conventions

$\mathcal{N}$	names	$a, b, c \ldots, x, y, z$
$\mathcal{A}$	actions	$\pi \ ::= \ x(y) \ \mid \ \overline{x}\langle y\rangle \ \mid \ \tau$

- ☐ finite sequences $\vec{a} \ldots$
- ☐ parametric processes with defining equations are modeled via the more primitive notion of replication and name-passing
- ☐ All values/variables/channels are just names. Parentheses usually indicate bindings. Angled brackets are often omitted.

Syntax / Grammar

Definition: The set $\mathcal{P}$ of π-calculus proc. exp. is defined (precisely) by the following syntax:

$$\begin{array}{rcl} P & ::= & M \ \mid \ P|P \ \mid \ (\boldsymbol{\nu}a)\,P \ \mid \ \boxed{!\,P} \\ M & ::= & \mathbf{0} \ \mid \ \boxed{\pi.P} \ \mid \ M+M \end{array}$$

We use P, Q, P_i to stand for process expressions.

- ☐ $(\boldsymbol{\nu}ab)\,P$ abbreviates $(\boldsymbol{\nu}a)\,(\boldsymbol{\nu}b)\,P$
- ☐ $\sum_{i\in\{1..n\}} \pi_i.P_i$ abbreviates $\pi_1.P_1 + \ldots + \pi_n.P_n$

Bound and Free Names

- $(\boldsymbol{\nu}x)\,P$ $\boxed{\text{and } y(x).P}$ **bind** x in P
- x occurs **bound** in P, if it occurs in a subterm $(\boldsymbol{\nu}x)\,Q$ $\boxed{\text{or } y(x).P}$ of P
- x occurs **free** in P, if it occurs without enclosing $(\boldsymbol{\nu}x)\,Q$ $\boxed{\text{or } y(x).P}$ in P
- Note the use of parentheses (round brackets).
- Define $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$ inductively on $\mathcal{P}$ (sets of free/bound names of P) $\ldots$

Mobility ? "Flowgraphs" !

$P{=}\overline{x}\langle z\rangle.P'$
$Q{=}x(y).Q'$
$R{=}\ldots z\ldots$

Assume that $z \notin \mathrm{fn}(P')$.

Depict the transition

$$(\boldsymbol{\nu}z)\,(\ P|R\ )\mid Q \;\rightarrow\; P' \mid (\boldsymbol{\nu}z)\,(\ R|Q'\ )$$

as a flow graph (with scopes) and verify it using the reaction and congruence rules.

Exercise: Overtaking Cars

A car $C\langle\, n, b, f\,\rangle$ on a road is connected to its back and front neighbor through b and f, respectively, while n just represents its identifier.

The road is assumed to be infinite, so we ignore any boundary problem, and it is static in the sense that no cars may enter or leave the road.

Define $C\langle\, n, b, f\,\rangle$ such that a car may overtake another car. Beware of deadlocks and nested overtake attempts. You are not allowed to change the parameter n of instances of C.

Solution: Overtaking Cars

many implementations might be valid ...
... here's just one proposal

$$\begin{aligned}
\mathsf{Car}\langle\, x, b, f\,\rangle &\stackrel{\text{def}}{=} \\
\mathsf{Fast}\langle\, x, b, f\,\rangle &\stackrel{\text{def}}{=} \\
\mathsf{Slow}\langle\, x, b, f, b'\,\rangle &\stackrel{\text{def}}{=}
\end{aligned}$$

Buffers in New Clothes ...

$$B(\,i,o\,) \stackrel{\text{def}}{=} i(x).C\langle\, x,i,o\,\rangle$$

$$C(\,x,i,o\,) \stackrel{\text{def}}{=} \overline{o}\langle x\rangle.B\langle\, i,o\,\rangle + i(y).(\,C\langle\, y,i,o\,\rangle \frown C\langle\, x,i,o\,\rangle\,)$$

where

$$X\langle\, i,o\,\rangle \frown Y\langle\, i,o\,\rangle \stackrel{\text{def}}{=} (\boldsymbol{\nu} m)\,(\,X\langle\, i,o\,\rangle[{}^{m}\!/\!_{o}] \mid Y\langle\, i,o\,\rangle[{}^{m}\!/\!_{i}]\,)$$

- ☐ Observe how much nicer name/value-passing is :-)
- ☐ Follow the sequence $\xrightarrow{i1}\xrightarrow{i2}\xrightarrow{\overline{o}2}\xrightarrow{\cdots}$

Elastic Buffers

Make the buffer elastic,
i.e., make empty cells disappear!

Several design decisions need to be taken concerning the question *when* an empty cell should cut itself out of a chain and die.

- ☐ if empty cell is next to a full/empty cell?
- ☐ if empty cell is left/right to a cell?
- ☐ should it be *allowed* (suicide)
  or *forced* (murder) to die?