TMC Session 10 @ 09/01/2002

U. Nestmann

EPFL-LAMP

Warming Up

Is **inequality** an equivalence relation?

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

Unbounded Structures: Stacks (I)

$$\mathcal{N} := \{\ \mathsf{empty}\ \} \cup \{\ \mathsf{push}_v, \mathsf{pop}_v\ \}_{v \in \mathbf{V}}$$

$$\vec{v} \in \mathbf{V}^*$$

$$\mathsf{Stack}_{\vec{w}}(\ \mathsf{empty}, \widetilde{\mathsf{push}_v}, \widetilde{\mathsf{pop}_v}\ )$$

$$\begin{array}{rcl} \mathsf{Stack} & \stackrel{\text{def}}{=} & \sum_v \mathsf{push}_v.\mathsf{Stack}_v + \overline{\mathsf{empty}}.\mathsf{Stack} \\ \mathsf{Stack}_{v,\vec{w}} & \stackrel{\text{def}}{=} & \sum_u \mathsf{push}_u.\mathsf{Stack}_{u,v,\vec{w}} + \overline{\mathsf{pop}_v}.\mathsf{Stack}_{\vec{w}} \end{array}$$

Unbounded Structures: Stacks (II)

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

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

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

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

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

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

Criticism

Example:
Calculate the states for the transition sequence $\xrightarrow{\text{push}_1}\xrightarrow{\text{push}_2}\xrightarrow{\text{pop}_2}$ 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)

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

- 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 $\approx$ . . .
   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* 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.

All values/variables/channels are just names.
Parentheses usually indicate bindings.
Angled brackets are often omitted.

Hand-Over Protocol

(external slide)

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.