Concurrency: Theory, Languages and Programming – Pi Calculus – Session 6 – November 27, 2002

Uwe Nestmann

EPFL-LAMP

Concurrency: Theory, Languages and Programming - Pi Calculus - Session 6 - November 27, 2002 - (produced on March 4, 2004, 18:38) - p.1/2

Redexes

There are (at least) two reasons for studying LTSs (as opposed to mere reductions as in the -calculus):

the emphasis in on *interaction* with other programs redexes in a concurrent program are usually distributed over terms, not juxtaposed as in -calculus.

redexes are the "source" of reductions or internal transitions, visible as the pattern in the conclusion of either the -rule (in) or the COMM-rule (in [VP]CCS).

Unbounded Structures: Stacks (I)

"specification":

stored values are encoded in the index of the identifier needs an unbounded number of process identifiers does not exploit "concurrency inside"

empty push pop

Stack				
Stack	push	Stack	empty	Stack
Stack	push	Stack	pop	Stack

Unbounded Structures: Stacks (II)

"implementation":

using a **chain** of individual cells for the stored values

cells can have one of the following states:

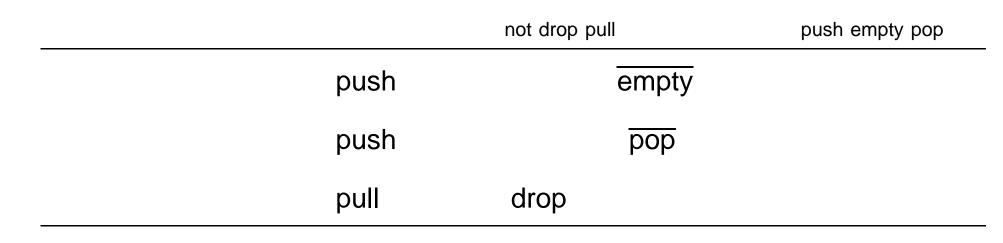
: nothing is left in the stack accessible through this cell

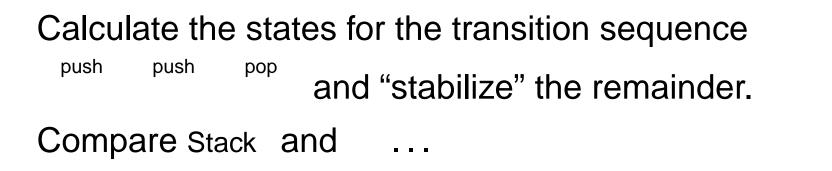
: a cell containing value

: nothing left in this particular cell, but maybe beyond on "the right of it"

Unbounded Structures: Stacks (III)

push empty pop not drop pull





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]

- 1. The calculus of concurrent process expressions is *Turing-powerful*.
- 2. The halting problem for some "Turing machine" TM can be encoded as the *existence* of an infinite sequence TM .

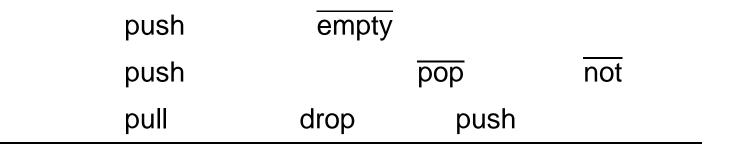
Unbounded Structures: Stacks (IV)

Some criticism:

's cannot be reused for storing *new* values (neither inner nor outer 's!).

's are never "used", pile up and stay around. (Note that, although "=", explicit garbage collection would be required.)

Unbounded Structures: Stacks (V)



What are the problems of this "implementation"?

Expressiveness

Although Turing-powerful, concurrent process expressions are, in some particular sense, not expressive enough: it is not possible to *cut out unusable (=dead) cells*.

If we had the possibility to *dynamically change the interconnection structure* among process components, then 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 — : <u>send name</u> over name .

positive actions : <u>receive</u> any name, say , over name and "bind the result" to name .

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

polyadic communication — and (pairwise different) transmit many values at a time.

Syntax Conventions

- \mathcal{N} names
- ${\cal A}$ actions

finite sequences ...

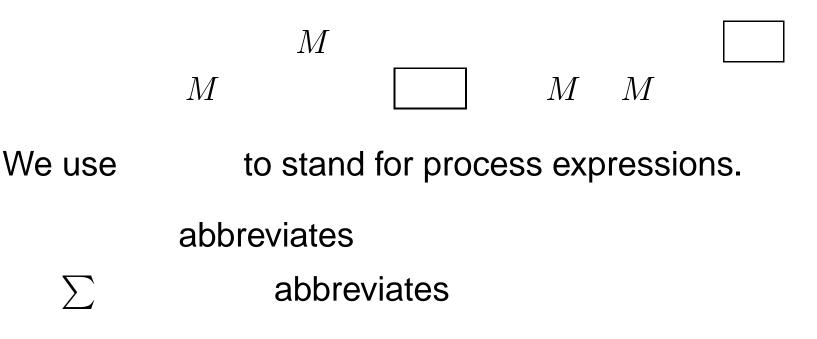
All values/variables/channels are just names.

Parentheses usually indicate bindings. Angled brackets are often omitted.

parametric processes with defining equations are modeled via the more primitive notion of replication and name-passing

Pi Calculus

<u>Definition</u>: The set of -calculus proc. exp. is defined (precisely) by the following syntax:



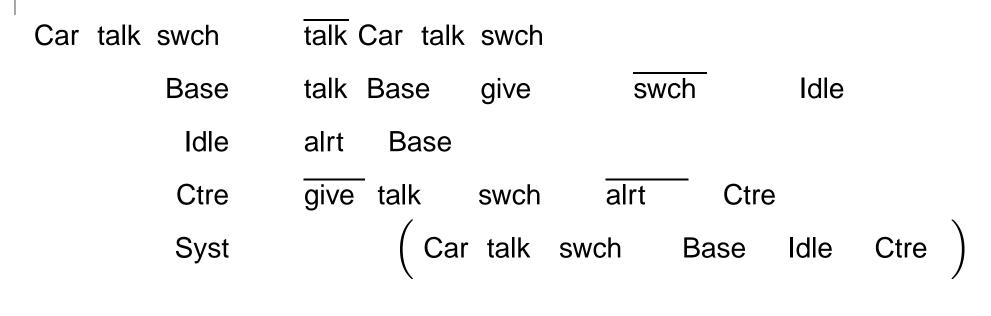
Mobility ? "Flowgraphs" !

Assume that

Depict the transition

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

Example: Hand-Over Protocol



Exercise: Observe that Syst Syst

Exercise: Overtaking Cars

A car on a road is connected to its back and front neighbor through and , respectively, while 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.

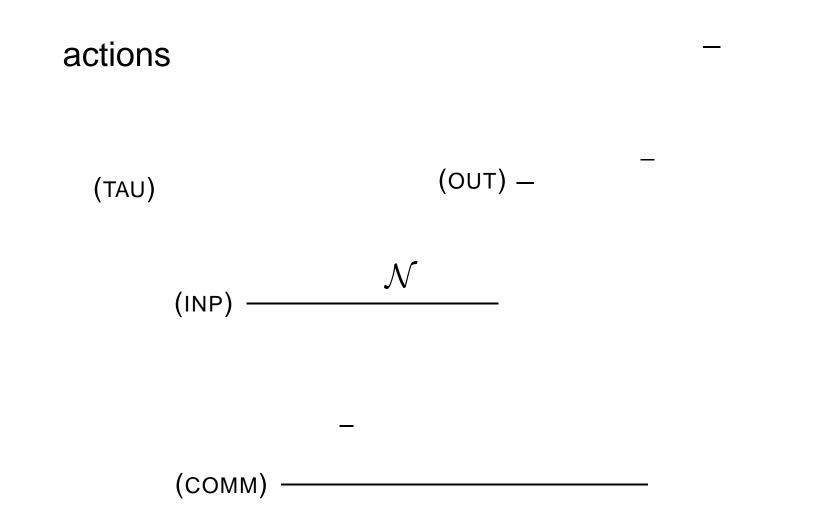
Define such that a car may overtake another car. Beware of deadlocks and nested overtake attempts. You are not allowed to change the parameter of instances of .

Car

Fast

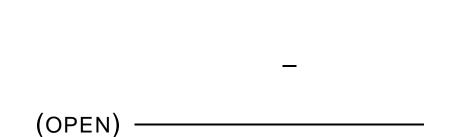
Slow

LTS: Prefixes



Concurrency: Theory, Languages and Programming - Pi Calculus - Session 6 - November 27, 2002 - (produced on March 4, 2004, 18:38) - p.16/20

LTS: Restriction



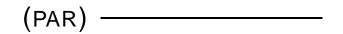
.)

(RES) ———

The label on transition (Invariant:

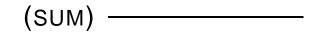
is called *bound output*.

LTS: Parallel Composition



(CLOSE) —

LTS: Miscellaneous



(ALP) _____

Recursion

, where

can be used in:

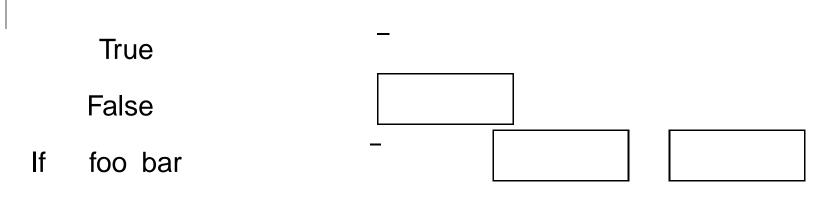
can be modeled through:

- 1. invent to stand for
- 2. for any ,

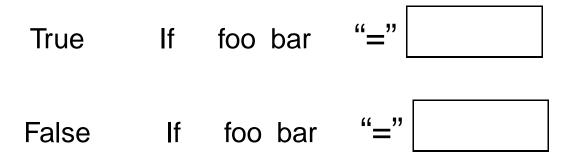
let denote the result of replacing any call by -

3. replace by

Booleans



Check that for all



Encoding Tuples

Think about:

Encoding Synchrony

-

Encoding Summation

 \sum

 \sum

Encoding Lambda-Calculus



Try to evaluate/encode M

Concurrency: Theory, Languages and Programming - Pi Calculus - Session 6 - November 27, 2002 - (produced on March 4, 2004, 18:38) - p.25/20

"Final Words"

name-passing vs. value-pasing

better pragmatics natural programming idioms semantic foundations for all (?) major programming styles

security protocols via (s)pi-calculus (research at LAMP2 ...)