Concurrency: Theory, Languages and Programming

– CCS –

Session 4 – November 13, 2002

Uwe Nestmann

EPFL-LAMP

Plan

Session 4

- from -calculus to CCS: towards concurrency
- Structural Operational Semantics (SOS)

Session 5

- examples . . .
- . . . using the Scala-library

Session 6

- from CCS to -calculus: pragmatics, syntax, semantics
- more SOS

Session 7

- examples . . .
- . . . using the Scala-library

Foundational Calculi ?

We are interested in the foundations of programming.
We use “foundational” mini-languages as vehicles that guide our intuition and style of expression. When does such a mini-language deserve to be called a “calculus”?

- few primitives
- mathematically tractable
  - *calculate* computational steps
  - notion of equivalence
- *computationally complete* (Turing, URM, GOTO, . . .)
- “*naturally complete*”: design of programming languages
  - easily “extensible” via *encodings*
  - *higher-order* principles

Concurrency?

parallelism: independent threads of control

distribution:

- logical concurrency
- physical concurrency
- failures

synchronization / cooperation / coordination
communication

foundational calculus for (just) concurrency ?

λ-Calculus

Syntax (for example)
a BNF-grammar generates the set of expressions . . .

$$M \qquad\qquad\qquad\qquad M$$

Semantics (for example)
a set of inference rules generates (and controls) the possible reductions of terms

$$(\quad)\ \frac{}{\qquad M \qquad M \qquad}$$

$$(\text{FUN})\ \frac{M \qquad M}{M \qquad\qquad M} \qquad\qquad (\text{ARG})\ \frac{}{M \qquad\qquad M}$$

Functional vs Concurrent

<u>functional / reduction systems</u>:

reduce a term to value form

only the resulting value is interesting

observation after termination

<u>concurrent / reactive systems</u>:

describe the possible interactions *during* evaluation

the resulting value is *not* (necessarily) interesting

observation through and during interaction

The notion of *interaction* (*communication*) is important !

Hoare (CSP) and Milner (CCS) proposed
handshake-communication as <u>the</u> primitive form of interaction.

Functional vs Concurrent

	functional	concurrent
determinism	possible	?
confluence	wanted/needed	?
termination	?	?
foundation		CCS, , (Petri nets, . . .)
ff-language	ML, Scala, . . .	Pict, Join, Scala, . . .

CCS

	process identifiers		
$\mathcal{N}$	names		
$\overline{\mathcal{N}}$	co-names	_ ‾ _	
	labels (buttons)	metavariables	$\mathcal{N}$ $\overline{\mathcal{N}}$
$\mathcal{A}$	actions	metavariables	

visible/external actions: labels

invisible/internal actions:

finite sequences for *names* (*not co*-names!)

parametric processes with
name parameters (neither *co*-names, nor labels, . . .)

Sequential Process Expressions (I)

Definition: The sets and $\mathcal{M}$ of sequential process expressions is defined (precisely) by the following BNF-syntax:

$$M$$

$$M \qquad M \quad M$$

We use to stand for *process expressions*, while M M always stand for *choices* or *summations*.
We also use the abbreviation

$$\sum$$

where is the finite indexing set .
Note that then the order of summands is not fixed.

Sequential Process Expressions (II)

each process identifier is assumed
to have a **defining equation** (note the brackets)

$$M$$

where M is a summation, is (or: includes) M .
Note: does only include *names*!

: the set of all of the **(free) names** of

means the same as M

substitution (for matching and)
replaces *all* occurrences of in by .

Free Names, Inductively

Definition: The set is defined inductively by:

$$\left\{ \begin{array}{l} \text{if} \\ \text{if} \\ \text{if} \end{array} \right.$$

$M \quad M \quad M \quad M$

Substitution, Inductively

Definition:

$$\left\{ \begin{array}{ll} \overline{} & \text{if} \\ & \text{if} \quad \overline{} \\ & \text{otherwise} \end{array} \right.$$

$$M \quad M \qquad\qquad M \qquad\qquad M$$

Simultaneous Substitution, Inductively

Definition:

Let and

$\Big\{$ $-$ if with

if with $-$

otherwise

M M M M

Example: 1-Place Binary Buffer

$\mathcal{N}$ in out

in in out out

Buff 1-place buffer containing

Buff $\sum$ in Buff

Buff $\overline{\text{out}}$ Buff

Example: 2-Place Binary Buffer

$\mathcal{N}$ in out

in in out out

Buff 2-place buffer containing

Buff $\sum$ in Buff

Buff $\overline{\text{out}}$ Buff $\sum$ in Buff

Buff $\overline{\text{out}}$ Buff

---

modify Buff to release values in either order

write an analogous definition for Buff ...

Labeled Transition Systems

Definition:
An **LTS** $\mathcal{T}$ over an **action alphabet** $\mathcal{A}$:

- a set of **states**
- a ternary **transition relation** $\mathcal{T}$ $\mathcal{A}$

A transition $\mathcal{T}$ is also written .

If we call a **derivative** of .

LTSs are automata, but ignoring starting and accepting states.
Transition Graphs are useful . . .

LTS - Sequential Expressions

Definition: The LTS $\mathcal{T}$ of sequential process expressions over $\mathcal{A}$ has as states, and its transitions $\mathcal{T}$ are precisely generated by the following rules:

PRE:

$$\text{SUM}\ :\ \frac{M \qquad\qquad M}{M \quad M \qquad\qquad M} \qquad\qquad \text{SUM}\ :\ \frac{M \qquad\qquad M}{M \quad M \qquad\qquad M}$$

$$\text{DEF}:\ \frac{M}{\qquad\qquad\qquad\qquad} \quad \text{IF} \qquad\qquad M$$

Note that transition under prefix is not allowed/included.

Concurrent Process Expressions (I)

Definition: The set of concurrent process expressions is defined (precisely) by the following BNF-syntax:

$$M$$

$$M \qquad M \quad M$$

We use to stand for process expressions.

restricts the scope of to

abbreviates

Concurrent Process Expressions (II)

precedence: unary binds tighter than binary

what about:

Bound and Free Names

binds in

occurs **bound** in ,
if it occurs in a subterm of

occurs **free** in ,
if it occurs without enclosing in

Define and inductively on
(sets of free/bound names of):

α-Conversion & Substitution

substitution (for matching and)
replaces *all* **free** occurrences of in by .

-conversion, written :
conflict-free **renaming of bound names**
(no new name-bindings shall be generated)

substitution (for matching and , where p.w.d.)
replaces *all* **free** occurrences of in by ,
possibly enforcing -conversion.

Examples

LTS — Concurrent Expressions

PAR : $\overline{}$ PAR : $\overline{}$

–

REACT: $\overline{}$

RES: $\overline{}$ IF –

ALPHA: $\overline{}$ IF AND

Buffers, revisited . . .

$\mathcal{N}$ in out x

in in out out

Bluff x x Buff in in x x

Buff x x out out

compare the behavior (= LTSs) of Buff and Bluff

regard both as lack boxes with “buttons” . . .