Concurrency: Theory, Languages and Programming

– Equivalences for π-Calculus –

Session 13 – January 29, 2003

Uwe Nestmann

EPFL-LAMP

Derivation of Transitions (Repetition)

What is Operational Semantics about?

It provides us with a *formal* (=mechanizable) way to find out which *computations steps* (=transitions) are possible for the current state of a system.

It provides a *compiler* with a *precise specification* of what to do!

It provides the basis for the definition of *program equivalences* (and congruences!) like bisimularities.

A tool like the ABC should (=must) be able to:
(1) derive transitions according to the operational semantics,
(2) play the bisimulation game based on this information,
(3) allow us to simulate system behaviors using this information.

Derivation of Transitions: Example

Towards Bisimulation in π-Calculus

"standard" definition is based on *labeled transitions*

PROBLEM: *infinite branching*
due to infinitely many input transitions
late input transitions

PROBLEM: lack of congruence properties!

when should substitution take place?

how to keep track of freshness of names?

PROBLEM: four (!) different (!!) styles of bisimulation
ground — early — late — open
which bisimulation is the "best"?

Input Transitions

for all $\mathcal{N}$

generates ***infinitely many*** transitions
for each enabled input prefix.

collapses all of them in one
by ***not yet instantiating*** the received variable.
The input is called ***late*** (or *symbolic*).
(The . . . -rule should then take care of substitutions.)

(PRE)

replaces the former (TAU), (OUT), and (INP).

Other Transitions ?

Now, we have transition labels

$\bar{}$

where and . (Note that there are no more labels of the form as we had in Session 6.)

If we change the rule for input transitions, then what is the precise effect on the other transitions?

Note that the names in an input label arose from an input binding, and that we still need to substitute them . . .

Let us defined the bound names of a label by:

$\bar{}$

and, of course .

Output Transitions

No input transitions involved.
No change needed, here.

(RES)

(OPEN)

“Uniform” Transitions

No change required.
Only non-critical access to bound names of transitions . . .

(SUM) ―――――――――――――

(REP) ―――――――――――――

(ALP) ―――――――――

Transitions of Parallel Compositions

Some change & care required.
(PAR) must respect the bound input names.

(PAR)

(CLOSE)

(CLOSE) must deal with the proper label and perform the substitution . . . quite at a quite late stage.

Simulating Input Transitions (I)

Definition: ("standard")

. . . whenever , if then

there is such that with

Compare the following terms:

So, this kind of input simulation does not yield a congruence !

Closure under input prefix means *closure under substitutions* !

Simulating Input Transitions (II)

. . . whenever , if then

ground

there is

such that with

early

for all there is

such that with

late

there is

such that for all with

Simulating Input Transitions

Compare again the following terms:

So, neither early nor late input simulation yield congruences !

Open Input Simulation

. . . whenever ,

for all , if then

there is such that with .

Note:

Substitution-closure is required ***before*** *each step*.

Open simulation provides substitution-closure "by definition".

However, it is going a bit too far . . .

Example

Compare the following terms:

–   –

–

What happens after the output transition ?

If we forget that was freshly generated, then it might accidentally be confused with when open-simulating the next ( ) transition.

Simulating Output Transitions

Under *open simulation* the approach:

. . . whenever ,

‾

if then

‾

there is such that with .

is too naïve !!

Distinction

Definition:
A *distinction* is a
finite symmetric *ir*reflexive relation on names.

A substitution *respects* a distinction
if implies .

A *D-congruence* is . . . w.r.t. only those contexts
that do not use the names in as bound names.

Open Bisimilarity

Definition:

is a distinction is the largest family of symmetric relations such that if and respects , then

- if and is not a bound output, then there is such that with .
- if , then there is such that with where .

The weak version is defined as usual.
Both the strong and weak bisimilarities are *-congruences*.

Relation to the ABC

The bisimulation relation generated by the ABC are open bisimulations.

Each element of such a relation is a triple, consisting of two terms and a *distinction* ...

Some more interesting examples next week ...