Reasoning about mobility in theorem provers

Daniel Hirschkoff - ENS Lyon

Subject of this talk

$\longrightarrow$ describe formalisations of the π-calculus in a general-purpose theorem prover (Isabelle/HOL or Coq)

- proofs *about the π-calculus*

(not proofs about systems specified in the π-calculus)

- no long-term effort yet

Motivations

- Provide a framework for formal reasoning on mobility
  - ▷ validate complex proofs
  - ▷ help in writing them
  - ▷ experiment with proof techniques

NB: *the proof should preexist on paper*

- Feedback from the tool
  - ▷ better insight on the theory
  - ▷ evidenciate difficult/questionable points

- Hopefully: from formalised mathematics to tools

Outline

- Syntax: the manipulation of binders

  many difficulties

- Studying the behaviour of processes

- Lessons learned

Implementing the syntax

Specifying the calculus

Questions to address:

- what are names?
- treatment of the two binders
  - ▷ α-conversion
  - ▷ different properties

input	$a(x).P \mid \overline{a}\langle v\rangle.Q \rightarrow P_{\{x:=v\}} \mid Q$
restriction	$(\nu x)\,(P \vert Q) \equiv P \vert (\nu x)\,Q \quad x \notin \mathsf{fn}(P)$

- issues
  - ▷ be able to reason on the syntax of terms (induction)
  - ▷ have a syntax that looks like on paper
  - ▷ consistency (!)

First-order approaches

Consider the term $a(x).(\nu y)\,\overline{x}\langle y\rangle$

first solution: have variables $\leftrightarrow$ *first-order encoding*

- McKinna/Pollack variables and variables [HG98]
  - ▷ substitution without α-conversion
  - ▷ an extra mechanism
- De Bruijn indices $a\lambda.\nu\,\overline{1}\langle 0\rangle$ [Hir97]
  - ▷ work on α-equivalences classes
  - ▷ technical (2 binders, 600 lemmas) – hard to read

FO approaches – comments

- "the rules are respected"
- adequacy of the encoding: obvious
- Coq and Isabelle/HOL automatically provide induction principles to reason over processes
- proofs about substitutions are carried along

“Higher-order” approach [HMS99] [Des00] [RHB01]

```
Inductive pi := ... | Out : name->name->pi->pi          ā⟨b⟩.P
                    | Inp : name->(name->pi)->pi        a(b).P
                    | Res : (name->pi)->pi.             (νa) P
```

- one uses the underlying binding mechanism of the prover *(“shallow embedding”)*
- α-conversion and substitution come for free

But having put our hands *within* the prover, difficulties arise

- reason on the syntax
- ensuring adequacy
- preserving consistence

Shallow embedding: difficulties

`Res : (name->pi)->pi`

- usual induction principle over the structure of terms is lost

- reasoning on the syntax
  - ▷ `free_names(Res(`$\lambda n.$`P))?`
  - ▷ no bound variable at object level

- adequacy of the encoding
  - ▷ less easy than in a deep embedding
  - ▷ exotic terms

Shallow embedding: *exotic terms*

`Res : (name->pi)->pi`

Suppose `name = nat`; what is the π-calculus term associated to

$$E \stackrel{def}{=} \texttt{Res}(\lambda n.\underline{\texttt{if}}\ n = 51\ \underline{\texttt{then}}\ \mathsf{P}\ \underline{\texttt{else}}\ \mathsf{Q})\ ?$$

`Res` should not take a "true" function, the λ is just there to bind

(cf. D. Miller: *"Abstract syntax for variable binders: a perspective"*)

- in Isabelle/HOL, we are in a classical setting
- ▷ one can always decide equality on names to define E
- ▷ restrict the class of functions used for `Res`

Shallow embedding: *exotic terms*

`Res : (name->pi)->pi`

Suppose `name = nat`; what is the π-calculus term associated to

$$E \stackrel{def}{=} \texttt{Res}(\lambda n.\underline{\texttt{if}}\ n = 51\ \underline{\texttt{then}}\ \mathsf{P}\ \underline{\texttt{else}}\ \mathsf{Q})\ ?$$

`Res` should not take a function, the λ is just there to bind.

- Coq's logics is intuitionnistic [HMS00]
  - ▷ as long as you don't take `name = nat`, things are safe
  - ▷ but one needs to compare names to reason over processes:

$\forall n, n' : \texttt{name}.\ (n = n') \vee (n \neq n')$ in `Prop` (no computational content)

The theory of contexts [HMS00]

- monotonicity: *if a is fresh in $f(b)$, then a is fresh in f*

- extensionality of contexts: *two contexts (functions from names to processes) f and g are equal if for some fresh name a, $f(a) = g(a)$*

- β-expansion: *given a process P, one can construct a context f_P by abstracting over some free variable a of P*

▷ in Coq: axioms [HMS01]
meta-level justification [Hof99]

Theory of contexts in Isabelle/HOL

(EXT) *two contexts (functions from names to processes) f and g are equal if for some fresh name a, $f(a) = g(a)$*

(EXP) *given a process P, one can construct a context f_P by abstracting over some free variable a of P*

- equality is extensional: (EXT) can be derived

BUT exotic terms lead to inconsistencies

- adapt an idea from [DH94]: well-formedness predicates

- then, you have induction principles to derive (EXP)

and (inernal) adequacy w.r.t. a deep embedding of π

Gordon and Melham approach [Gay00] [Gil01]

- Two layers:
- ▷ an implementation of the λ-calculus,

including formalisation of α-conversion and substitution

- ▷ use HOAS on top of it to represent your language

  combining deep embedding and Higher-Order

- technical work brought by a deep embedding is still needed

- but the treatment of binding is done once for all
- ▷ generality (cf Isabelle's architecture)
- ▷ no problem with *negative occurrences* in types

Other approaches

- Gabbay-Pitts' operator

Isabelle/FM; the "fresh" operator captures α-conversion (and fits to the modelling of restriction)

[Roe01]: π-calculus with *permutations of names* (cf. fusions)
$\rightsquigarrow$ what about substitutions?

- "Smaller" frameworks

Twelf [Pfenning Schürmann], FO$\lambda^{\Delta IN}$ [McDowell Miller]

A few words more about the syntax

- recursive definitions

$$A \quad = \quad rec\ X.P(X)$$

Yet another binder: if possible, use replication.

- polyadicity

$$\overline{a}\langle\vec{v}\rangle.P \text{ and } a(\vec{v}).P$$

interaction between restriction and emission: $(\nu\tilde{x})\,\overline{a}\langle\vec{v}\rangle$, $\tilde{x} \subseteq \vec{v}$

moreover, the polyadic π-calculus is typed

Reasoning about the behaviour of processes

Transitions

All works except one define a Labelled Transition System (LTS) (mostly in early style):

$$P \xrightarrow{a(b)} P' \qquad P \xrightarrow{\overline{a}\langle b\rangle} P' \qquad P \xrightarrow{\overline{a}(b)} P' \qquad P \xrightarrow{\tau} P'$$

- automatically get an induction principle
- smoothly define bisimulation on top of it (see later)

- some works quadruplicate rules (ease automation)
- [Des00]: original approach using “higher-order actions”

Reductions [Gay 00]

Operational semantics:

$$\longrightarrow \quad \text{modulo} \quad \equiv$$

- ▷ two nested inductions
- ▷ theorem provers are not well designed to reason *modulo*

Reductions, continued [Gay 00]

- you cannot plug $\equiv$ in the equality of the prover,

and use rewriting tactics
(putting $\equiv$ in the equality leads to inconsistencies)

- a bisimulation relation becomes infinite
- work in a systematical way:
  - ▷ normal forms for $\equiv$ [EG96], [Hir99], [DZ00]
  - ▷ enhance your prover with AC rewriting
  - ▷ refine the definition of $\longrightarrow$ (F. Pottier's $\langle\pi\rangle$)

Bisimilarity and bisimulation

- [Hir97]: check bisimilarity laws
  - ▷ bisimulation
  - ▷ *up-to proof techniques* [San95]
  - ▷ structural congruence laws
  - ▷ theorems about private replications

- [HMS00]: idem
  - ▷ coinduction
  - ▷ congruence results play the role of up-to techniques
  - ▷ structural congruence laws

Establishing bisimilarity laws – comments

- *"induction vs coinduction"* does not seem to be an issue *in these works*

- smooth integration of equational reasoning within bisimulation proofs

- no formalisation uses quantification over all contexts to introduce behavioural equivalence

Type systems [HG98] [Des00] [Gay00] [Gil01]

- definition of typing judgments of the form

$$\Gamma \vdash P$$

meaning that the usage of channels obeys a type discipline (I/O types)

- subject reduction: if P is well typed, then any evolution of P is

- it is a property of the *type system*

- several subject reduction proofs have a common structure [Gay00]: linear and non linear types

- typed bisimulation?

Lessons learned

Lessons learned

- reasoning on the π-calculus in a theorem prover is not an easy task, *especially regarding specification*

- knowing how to deal with e.g. binders and Associative-Commutative rewriting is necessary for proofs about mobility

- no *algorithms* have been mechanised in a theorem prover (type inference, bisimulation verification, model checking)

Extensions

- *can we handle other languages?*

e.g. Spi (E. Tuosto), Join, Mobile Ambients
Fusion, explicit substitution calculi

- *can we enrich the language?*

e.g. PICT

- *can we explore other methods?*

e.g. tools for equational reasoning,
Modal Logics, Spatial Logics

Comparing Coq and Isabelle/HOL

- differences in the logics:

Isabelle/HOL	Coq
classical	intuitionnistic
extensional equality	C. Paulin's equality

- the practice of proofs: more automation in Isabelle, more powerful tactics
- Coq has proof objects:
  - ▷ extraction of programs from proofs
  - ▷ reflection to build new tactics

  $\longrightarrow$ not used yet in this context
- a higher-order encoding fits better to Isabelle/HOL

Using the prover to reason on concrete processes

- some reasonable systems cannot be handled purely automatically: need for interaction

- adopt a different point of view – axioms
  - ▷ proof techniques
  - ▷ equational laws
  - ▷ subject reduction
  - ▷ but, still, be careful about consistency!

- interaction with verification tools: two approaches
  - ▷ run the tool, and extract some proof traces
  - ▷ *“PVS approach”*: black boxes