Reasoning about mobility in theorem provers

Daniel Hirschkoff - ENS Lyon
Subject of this talk

→ 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

• Studying the behaviour of processes

• Lessons learned
Implementing the syntax
Specifying the calculus

Questions to address:

- what are names?
- treatment of the two binders
  - $\alpha$-conversion
  - different properties

input $a(x).P \mid \bar{a}⟨v⟩.Q \rightarrow P\{x:=v\} \mid Q$

restriction $(\nu x)(P|Q) \equiv P|(\nu x)Q \quad x \notin \text{fn}(P)$

- 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 $a(x) . (\nu y) \bar{x} \langle y \rangle$

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

- McKinna/Pollack variables and variables \[ \text{[HG98]} \]
- substitution without $\alpha$-conversion
- an extra mechanism
- De Bruijn indices $a \lambda. \nu \bar{1} \langle 0 \rangle$ \[ \text{[Hir97]} \]
- work on $\alpha$-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 \( \alpha(b).P \)
| Inp: name -> (name -> pi) -> pi \( a(b).P \)
| Res: (name -> pi) -> pi \( (\nu a) P \)

- one uses the underlying binding mechanism of the prover ("shallow embedding")
- \( \alpha \)-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

- usual induction principle over the structure of terms is lost
- reasoning on the syntax
  - $\text{free_names}(\text{Res}(\lambda n. P))$?
  - no bound variable at object level
- adequacy of the encoding
  - less easy than in a deep embedding
  - exotic terms

Res: $(\text{name-}\to\text{pi}-)\to\text{pi}$
Shallow embedding: exotic terms

Res : (name->pi)->pi

Suppose \texttt{name = nat}; what is the \pi-calculus term associated to

\[
E \overset{\text{def}}{=} \text{Res}(\lambda n.\text{if } n = 51 \text{ then } P \text{ else } Q)
\]

\texttt{Res} should not take a “true” function, the \lambda is just there to bind


- in Isabelle/HOL, we are in a classical setting
  - one can always decide equality on names to define \textit{E}
  - restrict the class of functions used for \texttt{Res}
Shallow embedding: exotic terms

Suppose \texttt{name = nat}; what is the \(\pi\)-calculus term associated to

\[
E \overset{\text{def}}{=} \text{Res}(\lambda n. \text{if } n = 51 \text{ then } P \text{ else } Q)
\]

\text{Res} should not take a function, the \(\lambda\) is just there to bind.

- Coq’s logics is intuitionnistic \cite{HMS00}
  - as long as you don’t take \texttt{name = nat}, things are safe
  - but one needs to compare names to reason over processes:
    \[
    \forall n, n' : \text{name.} \, (n = n') \lor (n \neq n') \in \text{Prop} \text{ (no computational content)}
    \]
The theory of contexts \[\text{[HMS00]}\]

- 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 \(a\), \(f(a) = g(a)\)

- \(\beta\)-expansion: given a process \(P\), one can construct a context \(f_P\) by abstracting over some free variable \(a\) of \(P\)

▷ in Coq: axioms in Coq: meta-level justification \[\text{[HMS01]}\] \[\text{[Hof99]}\]
Theory of contexts in Isabelle/HOL

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

(ESP) 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 (internal) adequacy w.r.t. a deep embedding of $\pi$
Gordon and Melham approach [Gay00] [Gil01]

- Two layers:
  - an implementation of the $\lambda$-calculus, including formalisation of $\alpha$-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 $\alpha$-conversion (and fits to the modelling of restriction)

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

- “Smaller” frameworks
  Twelf [Pfenning Schürmann], $\text{FOL}\Delta^{IN}$ [McDowell Miller]
A few words more about the syntax

- recursive definitions

\[ A = \text{rec } X.P(X) \]

Yet another binder: if possible, use replication.

- polyadicity

\[ \overline{a} \langle \overline{v} \rangle . P \text{ and } a(\overline{v}) . P \]

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

moreover, the polyadic \(\pi\)-calculus is typed
Reasoning about the behaviour of processes
Transitions

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

\[ P \xrightarrow{a(b)} P' \quad P \xrightarrow{\overline{a}(b)} P' \quad P \xrightarrow{\overline{a}(b)} P' \quad P \xrightarrow{\tau} P' \]

- 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

Operational semantics:

\[ \rightarrow \text{ modulo } \equiv \]

- two nested inductions
- theorem provers are not well designed to reason \textit{modulo}
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 $\rightarrow$ (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

- 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 $\pi$-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
    - 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