MPRI C.2.3 - Concurrency

Probabilistic models and applications
Lecture 3

Kostas Chatzikokolakis

Jan 10, 2012
Outline of the lectures

- Dec 13
- Dec 20
- Jan 10
- Jan 17
- Jan 24
Outline of the lectures

- The need for randomization
- Probabilistic automata
- Probabilistic bisimulation
- Probabilistic calculi
- Encoding of the pi-calculus into the asynchronous fragment
- Introduction to probabilistic model checking and PRISM
- Verification of anonymity protocols: Dining Cryptographers, Crowds
Exercises from the last lecture

Exercise 1: Show that probabilistic bisimulation is a generalization of traditional bisimulation
Puzzle from the last lecture

- I select two real numbers in some arbitrary way.
- I put them in two envelopes, you select one of them (in any way you want).
- You see the number and you have 2 options: keep it, or exchange it with the other envelope.
- Your goal is to select the bigger number.
- Is there any strategy that guarantees winning this game with pb higher than 1/2?
Outline

Encoding of $\pi$-calculus in the asynchronous fragment

Encoding of $\pi$-calculus in the probabilistic asynchronous $\pi$

Non-deterministic transition systems and CTL
Probabilistic asynchronous $\pi$-calculus

The input guarded choice is probabilistic.

The prefixes

$$\alpha ::= x(y) \mid \tau$$  \hspace{1cm} input \mid silent \ action

The processes

$$P ::= 0 \hspace{1cm} inaction$$
$$\sum_i p_i \alpha_i . P_i \hspace{1cm} probabilistic \ choice$$
$$\overline{xy} \hspace{1cm} output$$
$$P \mid P \hspace{1cm} parallel$$
$$\nu x P \hspace{1cm} new \ name$$
$$! P \hspace{1cm} replication$$

where $\sum_i p_i = 1$
Expressive power of $\pi_a$ wrt $\pi$

- Clearly $\pi$ is at least as expressive as $\pi_a$

- The latter is practically a subset of the former: $\tilde{x}y$ (in $\pi_a$) can be seen as $\tilde{x}y.0$ (in $\pi$)

- What about the opposite direction? We need to encode:
  - the output prefix
  - the choice operator
    - Three types of choice: internal, separate, mixed

- In general, in order to compare the expressive power of two languages, we look for the existence/non-existence of an encoding with certain properties among these languages

- What is a good notion of encoding to be used as basis to measure the relative expressive power?
A “good” notion of encoding

In general we would be happy with an encoding $\llbracket \cdot \rrbracket : \pi \rightarrow \pi_a$ being:

- Compositional wrt the operators $\llbracket P \ op \ Q \rrbracket = C_{op}[\llbracket P \rrbracket, \llbracket Q \rrbracket]$

- (Preferably) homomorphic wrt | (distribution-preserving) $\llbracket P | Q \rrbracket = \llbracket P \rrbracket | \llbracket Q \rrbracket$

- Preserving some kind of semantics. Here there are several possibilities
  
  - Preserving observables $\text{Obs}(P) = \text{Obs}(\llbracket P \rrbracket)$
  
  - Preserving equivalence

    $\llbracket P \rrbracket \ \text{equiv} \ \llbracket Q \rrbracket \Rightarrow P \ \text{equiv}' \ Q$ (soundness)

    $\llbracket P \rrbracket \ \text{equiv} \ \llbracket Q \rrbracket \ \Leftarrow \ P \ \text{equiv}' \ Q$ (completeness)

    $\llbracket P \rrbracket \ \text{equiv} \ \llbracket Q \rrbracket \ \Leftrightarrow \ P \ \text{equiv}' \ Q$ (full abstraction, correctness)
Testing semantics

- A test $O$ is a process with a distinct success action $\omega$

- A process $P$ may pass $O$ iff there is a computation of $[P|O]$ where $\omega$ is enabled
  eg. $a.b + a$ may pass $\bar{a}.\bar{b}.\omega$

- A process $P$ must pass $O$ iff all computations of $[P|O]$ reach a state where $\omega$ is enabled
  eg. $a.b + a$ must pass $\bar{a}.\omega$
Testing semantics

- \( P \sqsubseteq_{\text{may}} Q \iff \forall O : P \text{ may } O \Rightarrow Q \text{ may } O \)
- \( P \sqsubseteq_{\text{must}} Q \iff \forall O : P \text{ must } O \Rightarrow Q \text{ must } O \)
- Exercise: are \( \sqsubseteq_{\text{may}}, \sqsubseteq_{\text{must}} \) pre-congruences for CCS, \( \pi \)?
- We would like the encodings to satisfy:
  - \( P \text{ may pass } O \iff \llbracket P \rrbracket \text{ may pass } \llbracket O \rrbracket \)
  - \( P \text{ must pass } O \iff \llbracket P \rrbracket \text{ must pass } \llbracket O \rrbracket \)
The encoding of Boudol

Encodes the output prefix (but without choice). Idea: we proceed only when it is sure that the communication can take place, by using a sort of rendez-vous protocol.

\[
\begin{align*}
\text{[0]} & = 0 \\
\text{[P \mid Q]} & = \text{[P]} \mid \text{[Q]} \\
\text{[(\nu x)P]} & = (\nu x)\text{[P]} \\
\text{[! P]} & = ! \text{[P]}
\end{align*}
\]

The encoding satisfies P may pass O iff [P] may pass [O]
Encoding of Honda-Tokoro

A more compact encoding, it takes two steps instead than three. The idea is to let the receiver take the initiative.

\[
\begin{align*}
\bullet \ [\bar{x}y.P] &= x(z).\bar{z}y \mid [P]) \\
\bullet \ [x(y).Q] &= (\nu z)(\bar{x}z \mid z(y).[Q])
\end{align*}
\]

\text{[·] is homomorphic for all the other operators}

\[
\begin{align*}
\bullet \ [0] &= 0 \\
\bullet \ [P \mid Q] &= [P] \mid [Q] \\
\bullet \ [(\nu x)P] &= (\nu x)[P] \\
\bullet \ [!] P &= ! [P]
\end{align*}
\]

The encoding satisfies $P$ may pass $O$ iff $[P]$ may pass $[O]$
Encoding of the output prefix

- The encodings of Boudol and Honda-Tokoro do not satisfy $P$ must pass $O$ iff $\overline{P}$ must pass $\overline{O}$

- This is a problem of fairness

- Must testing is preserved if we restrict to fair computations only

- The encodings preserve a version of testing called “fair must testing”
Encoding of internal choice

The blind choice (or internal choice) construct $P \oplus Q$ has the following semantics

$$
\begin{align*}
P \oplus Q & \xrightarrow{\tau} P \\
Q \oplus Q & \xrightarrow{\tau} Q
\end{align*}
$$

In $\pi$ this operator can be represented by the construct $\tau.P + \tau.Q$

Exercise: Let $\pi'$ be $\pi$ where the $+$ operator can only occur as a blind choice.

Give an encoding $\llbracket \cdot \rrbracket : \pi^\oplus \rightarrow \pi_a^*$ such that $\forall P \, \llbracket P \rrbracket \sim P$
Encoding of input-guarded choice

Input-guarded choice is a construct of the form: \[ \sum_{i \in I} x_i(y_i).P_i \]

Let \( \pi^i \) be \( \pi \) where \( + \) can only occur in an input-guarded choice. The following encoding of \( \pi^i \) into \( \pi_a \) was defined by Nestmann and Pierce [1996]:

\[
\llbracket \sum_{i \in I} x_i(y_i)P_i \rrbracket = (v \ell)(\ell \text{ true} \mid \prod_{i \in I} \text{Branch}_{\ell i})
\]

\[
\text{Branch}_{\ell i} = x_i(z_i).\ell(w).(if \ w
\begin{align*}
& \text{then } (\ell \text{ false} \mid \llbracket P_i \rrbracket) \\
& \text{else } (\ell \text{ false} \mid x_i z_i) )
\end{align*}
\]

Nestmann and Pierce proved that his encoding is fully abstract wrt a notion of equivalence called coupled bisimulation, and it does not introduce divergences.
The $\pi$-calculus hierarchy

$\pi_a$: asynchronous $\pi$
$\pi_{ic}$: asynchronous $\pi$ + input-guarded choice
$\pi_{op}$: asynchronous $\pi$ + output prefix
$\pi_s$: asynchronous $\pi$ + separate choice
$\pi_I$: $\pi$ with internal mobility (Sangiorgi)
$ccs_{vp}$: value-passing ccs

$\rightarrow$: Language inclusion
$\rightarrow$: Encoding
$\rightarrow$: Non-encoding

Diagram:
- $\pi_a$ to $\pi_{ic}$
- $\pi_a$ to $\pi_{op}$
- $\pi_{ic}$ to $\pi_s$
- $\pi_{op}$ to $\pi_s$
- $\pi_s$ to $\pi$
- $ccs_{vp}$ to $\pi_s$
The separation between $\pi$ and $\pi_s$

This separation result is based on the fact that it is not possible
to solve the symmetric leader election problem in $\pi_s$, while it is
possible in $\pi$

Leader Election Problem (LEP): All the nodes of a distributed
system must agree on who is the leader. This means that in every
possible computation, all the nodes must eventually output the
name of the leader on a special channel

- No deadlock

- No livelock

- No conflict (only one leader must be elected, every process
  outputs its name and only its name)
The separation between $\pi$ and $\pi_s$

**Theorem**

It is impossible to write in $\pi_s$ a symmetric (having an automorphism with a single orbit) solution to the LEP.

**Crucial point:** Diamond lemma: when a node $P_i$ performs an action, any other node $P_j$ can perform the same action returning to a symmetric state. (Note: this does not hold in $\pi$)

**Corollary:** in a symmetric $\pi_s$ network trying to solve the LEP, there is at least one diverging computation.
The separation between $\pi$ and $\pi_s$

Remark: In $\pi$ (in $\pi$ with mixed choice) we can easily write a symmetric solution for the LEP in a network of two nodes:

\[ P_0 = x.\text{out} \, 0 + \bar{y}.\text{out} \, 1 \]
\[ P_1 = y.\text{out} \, 1 + \bar{x}.\text{out} \, 0 \]
The separation between $\pi$ and $\pi_s$

**Corollary:** there does not exists an encoding of $\pi$ in $\pi_s$ which is homomorphic wrt $|$ and renaming, and preserves the observables on every computation.

**Proof (sketch):** An encoding homomorphic wrt $|$ and renaming transforms a symmetric solutions to the LEP in the source language into a symmetric solution to the LEP in the target language.
Outline

Encoding of $\pi$-calculus in the asynchronous fragment

Encoding of $\pi$-calculus in the probabilistic asynchronous $\pi$

Non-deterministic transition systems and CTL
Probabilistic testing semantics

- Test $O$: same as before (but can be probabilistic)

- $sexec([P|O])$ the set of successful executions of $[P|O]$ (those containing $\omega$)

- Note: $sexec([P|O])$ can be obtained as a countable union of disjoint cones

- $\mu_\sigma(sexec([P|O]))$ the probability of success under scheduler $\sigma$
Probabilistic testing semantics

- $P$ may pass $O$ iff $\exists \sigma: \mu_\sigma(\text{sexec}([P|O])) > 0$
- $P$ must pass $O$ iff $\forall \sigma: \mu_\sigma(\text{sexec}([P|O])) = 1$
- $\subseteq_{\text{may}}, \subseteq_{\text{must}}$: same as before
  - $P \subseteq_{\text{may}} Q$ iff $\forall O: P$ may $O \Rightarrow Q$ may $O$
  - $P \subseteq_{\text{must}} Q$ iff $\forall O: P$ must $O \Rightarrow Q$ must $O$
Probabilistic testing semantics

Exercises:

○ Give alternative definitions that consider the exact probabilities of success. Are they equivalent to the ones of the previous slide?

○ Show that for all probabilistic CCS processes $P, Q$:

  $\cdot P +_p Q \sqsubseteq_{\text{may}} \tau.P + \tau.Q$

  $\cdot \tau.P + \tau.Q \sqsubseteq_{\text{must}} P +_p Q$
Encoding of $\pi$ into $\pi_{ap}$ (high level idea)

- Similarly to the encoding of Nestmann-Pierce, the branches of the choice are put in parallel together with a lock.

- An input process will try to acquire both its own lock and its partner’s lock.

- If a lock is not available it aborts and tries again.

- Crucial point: the locks are tried in random order, similarly to the solution of the Dining Philosophers problem.

- This ensures that divergences will have probability 0.
Encoding of $\pi$ into $\pi_{ap}$ (high level idea)

- This encoding satisfies:
  - $P$ may pass $O$ iff $\llbracket P \rrbracket$ may pass $\llbracket O \rrbracket$
  - $P$ must pass $O$ iff $\llbracket P \rrbracket$ must pass $\llbracket O \rrbracket$

- Under a weak assumption on the schedulers (weaker than fairness)

Outline

Encoding of $\pi$-calculus in the asynchronous fragment

Encoding of $\pi$-calculus in the probabilistic asynchronous $\pi$

Non-deterministic transition systems and CTL
Model Checking

- Techniques to automatically verify systems (software, hardware, protocols, ...) using automata and temporal logics
- We give a formal model $M$ of the system (typically using some kind of automaton)
- We give a formal property $\varphi$ to verify (typically a formula in some temporal logic)
- An algorithm decides whether $M$ satisfies the formula $\varphi$, written

$$M \models \varphi$$
Non-deterministic Transition Systems

A tuple \((S, s_0, \rightarrow)\) where

- \(S\) is a **finite set of states**
- \(s_0 \in S\) is the **initial state**
- \(\rightarrow \subseteq S \times S\) is a **transition relation**.

We write \(s_1 \rightarrow s_2\) when there \((s_1, s_2) \in \rightarrow)
Non-deterministic Transition Systems

Example: a coffee machine

We want to verify properties like “the machine always goes back to its initial state”.
Computation Tree Logic (CTL)

- Formulas are **evaluated** on a transition system $M$
- On each state $s$ we assign a set $L(s)$ of **atomic propositions**. These are the propositions that we consider to be **true** on this state.
- Two types of formulas:
  - **state** formulas are evaluated on states
  - **path** formulas are evaluated on infinite sequences of states
Computation Tree Logic (CTL)

Syntax:

\[ \varphi ::= p \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \neg \varphi \mid A\psi \mid E\psi \]  
\text{state formulas}

\[ \psi ::= \circ \varphi \mid \square \varphi \mid \Diamond \varphi \mid \varphi U \varphi' \]  
\text{path formulas}

Path quantifiers:

- \( A\psi \) for all paths starting from this state \( \psi \) holds
- \( E\psi \) there exists a path starting from this state s.t. \( \psi \) holds

Temporal operators:

- \( \circ \varphi \) in the next state \( \varphi \) holds \((X\varphi)\)
- \( \square \varphi \) (always) in all states \( \varphi \) holds \((G\varphi)\)
- \( \Diamond \varphi \) (eventually) in some future state \( \varphi \) holds \((F\varphi)\)
- \( \varphi U \varphi' \) \( \varphi \) holds in all states until a state where \( \varphi' \) holds
Computation Tree Logic (CTL)

finally $P$

globally $P$

next $P$

$P$ until $q$

$AF_P$

$AG_P$

$AX_P$

$A[P \cup q]$

$EF_P$

$EG_P$

$EXP$

$E[P \cup q]$