MPRI – Course on Concurrency

Probabilistic methods in Concurrency

Catuscia Palamidessi
INRIA Futurs and LIX
catuscia@lix.polytechnique.fr
www.lix.polytechnique.fr/~catuscia

Page of the course:
http://pauillac.inria.fr/~leifer/teaching/mpri-concurrency-2004/
Filling the gap between the pi-calculus and the asynchronous pi-calculus

- The pi-calculus is strictly more expressive than the asynchronous pi-calculus

- On the other hand, the asynchronous pi-calculus can be implemented in a distributed way, while the (synchronous) pi-calculus cannot

- It is possible to enrich the asynchronous pi-calculus with a probabilistic (internal) choice so to achieve the same expressive power as the pi-calculus

- The idea is to implement the mixed choice construct via a randomized algorithm which is based on (an extension of) the randomized solution of Lehmann and Rabin to the dining philosophers
The dining philosophers

- Each philosopher needs exactly two forks
- Each fork is shared by exactly two philosophers
- A philosopher can access only one fork at the time
Intended properties of solution

- **Deadlock freedom** (aka progress): if there is a hungry philosopher, a philosopher will eventually eat.
- **Starvation freedom**: every hungry philosopher will eventually eat (but we won't consider this property here).
- **Robustness wrt a large class of adversaries**: Adversaries decide who does the next move (schedulers).
- **Fully distributed**: no centralized control or memory.
- **Symmetric**:
  - All philosophers run the same code and are in the same initial state.
  - The same holds for the forks.
Non-existence of a “deterministic” solution

- Lehman and Rabin have shown that there does not exist a “deterministic” (i.e. non-probabilistic) solution to the dining philosophers, satisfying all properties listed in previous slide.

- The proof proceeds by proving that for every possible program we can define an adversary (scheduler) which preserves the initial symmetry.

- **Note:** Francez and Rodeh did propose a “deterministic” solution using CSP, similar to the solution in pi-calculus given in Lecture 6. The solution to this apparent contradiction is that CSP and the pi-calculus cannot be implemented in a fully distributed way because of the (mixed) guarded choice construct.
The algorithm of Lehmann and Rabin

1. Think
2. randomly choose fork in \{left, right\} \hspace{1mm} %commit
3. if taken(fork) then goto 3
   else take(fork)
4. if taken(other(fork)) then \{release(fork); goto 2\}
   else take(other(fork))
5. eat
6. release(other(fork))
7. release(fork)
8. goto 1
Correctness of the algorithm of Lehmann and Rabin

- **Theorem**: for every *fair* adversary, if a philosopher becomes hungry, then a philosopher (not necessarily the same) will eventually eat with probability 1.

- **Proof**: the original proof is not fully formalized it is difficult to follow. There is a proof by Segala and Lynch, using *Progress Statements*, which is easier.

- **Question**: why the fairness requirement? Can we write a variant of the algorithm which does not require fairness?
Progress statements

• Progress statements
  - Proposed by Lynch and Segala
  - A formal method to analyze probabilistic algorithms

• Definition (progress statements)
  - Given sets of states $S$, $T$, and a class of adversaries $A$, we write
    $$S \rightarrow^A,p \rightarrow T$$
    if, under any adversary in $A$, from any state in $S$, we eventually reach a state in $T$ with probability at least $p$

  - Furthermore, we write
    $$S \text{ unless } T$$
    if, whenever from a state in $S$ we do not reach a state in $T$, we remain in $S$ (possibly in a different state of $S$)
History insensitivity

- **Definition:** A class of adversaries $A$ is history-insensitive if: for every $\alpha \in A$, and for every fragment of execution $e$, there exists $\alpha' \in A$ such that for every fragment of execution $e'$, $\alpha'(e') = \alpha(ee')$

- **Proposition:** The class of fair adversaries is history-insensitive

  **Proof:** Given $\alpha$ and $e$, define $\alpha'(e') = \alpha(ee')$. Clearly $\alpha'$ is still fair
Progress statements

- Some useful properties

  - If $A$ is history-insensitive, $S \rightarrow_{A,p} T$, and $T \rightarrow_{A,q} U$, then $S \rightarrow_{A,pq} U$

  - If $S_1 \rightarrow_{A,p_1} T_1$, and $S_2 \rightarrow_{A,p_2} T_2$, then $S_1 \cup S_2 \rightarrow_{A,p} T_1 \cup T_2$

    where $p = \min\{p_1, p_2\}$

  - $S \rightarrow_{A,1} S$

  - If $A$ is history-insensitive and $S \rightarrow_{A,p} T$ and $S$ unless $T$, and $p > 0$, then $S \rightarrow_{A,1} T$
Proof of d-f for the dining philosophers

- Proof of deadlock-freedom for the algorithm of Lehmann and Rabin for the Dining Philosophers
  - We will show that under a fair adversary scheduler we have deadlock-freedom (and livelock-freedom), i.e. if a philosopher gets hungry, then with probability 1 some philosopher (not necessarily the same) will eventually eat.
Proof of d-f: the algorithm

<table>
<thead>
<tr>
<th>State</th>
<th>action</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>think or get hungry</td>
<td>reminder region</td>
</tr>
<tr>
<td>F</td>
<td>flip</td>
<td>ready to toss</td>
</tr>
<tr>
<td>W</td>
<td>wait</td>
<td>waiting for first fork</td>
</tr>
<tr>
<td>S</td>
<td>second</td>
<td>checking second resource</td>
</tr>
<tr>
<td>D</td>
<td>drop</td>
<td>dropping first resource</td>
</tr>
<tr>
<td>P</td>
<td>eat</td>
<td>pre-critical region</td>
</tr>
<tr>
<td>C</td>
<td>exit</td>
<td>critical region</td>
</tr>
<tr>
<td>E_F</td>
<td>dropF</td>
<td>drop first fork</td>
</tr>
<tr>
<td>E_S</td>
<td>dropS</td>
<td>drop second fork</td>
</tr>
<tr>
<td>E_R</td>
<td>rem</td>
<td>move to reminder region</td>
</tr>
</tbody>
</table>
Example of verification: The dining philosophers

- Let us introduce the following global (sets of) states
  - Try: at least one phil is in T={F,W,S,D,P}
  - Eat: at least one phil is in C
  - RT: at least one phil is in T, all the others are in T, R or ER
  - Flip: at least one phil is in F
  - Pre: at least one phil is in P
  - Good: at least one process is in a "good state", i.e. in \{W,S\}
    while his second fork f is not the first fork for the neighbor (i.e. the neighbor is not committed to f)

- We want to show that Try → A,1→ Eat for A = fair adv
Example of verification: The dining philosophers

• We can prove that, for the class of fair adversaries $A$ (omitted in the following notation):
  - $\text{Try} \xrightarrow{-1} \text{RT} \cup \text{Eat}$
  - $\text{RT} \xrightarrow{-1} \text{Flip} \cup \text{Good} \cup \text{Pre}$
  - $\text{Flip} \xrightarrow{-1/2} \text{Good} \cup \text{Pre}$
  - $\text{Good} \xrightarrow{-1/4} \text{Pre}$
  - $\text{Pre} \xrightarrow{-1} \text{Eat}$

• Using the properties of progress statements we derive $\text{Try} \xrightarrow{-1/8} \text{Eat}$

• Since we also have $\text{Try unless Eat}$, we can conclude
Encoding $\pi$ into $\pi_{pa}$

- $[[\ ]]: \pi \rightarrow \pi_{pa}$

- **Fully distributed**
  \[
  [[P \mid Q]] = [[P]] \mid [[Q]]
  \]

- **Uniform**
  \[
  [[P \sigma]] = [[P]] \sigma
  \]

- **Correct wrt a notion of probabilistic testing semantics**
  \[
  P \text{ must } O \iff [[P]] \text{ must } [[O]] \text{ with prob } 1
  \]
Encoding $\pi$ into $\pi_{pa}$

- **Idea:**
  - Every mixed choice is translated into a parallel comp. of processes corresponding to the branches, plus a lock $f$
  - The input processes compete for acquiring both its own lock and the lock of the partner
  - The input process which succeeds first, establishes the communication. The other alternatives are discarded

The problem is reduced to a generalized dining philosophers problem where each fork (lock) can be adjacent to more than two philosophers
Problems

• Wrt to our encoding goal, the algorithm of Lehmann and Rabin has two problems:

  1. It only works for certain kinds of graphs

  2. It works only for fair schedulers

• Problem 2 however can be solved by replacing the busy waiting in step 3 with suspension.
  [Duflot, Friburg, Picaronny 2002] - see also [Herescu’s PhD thesis]
The algorithm of Lehmann and Rabin

1. Think
2. randomly choose fork in \{left, right\} %commit
3. if taken(fork) then goto 3
   else take(fork)
4. if taken(other(fork)) then \{release(fork); goto 2\}
   else take(other(fork))
5. eat
6. release(other(fork))
7. release(fork)
8. goto 1
The algorithm of Lehmann and Rabin modified so to eliminate the need of fairness

1. Think
2. randomly choose fork in \{left, right\} %commit
3. if taken(fork) then wait;
4. take(fork)

5. if taken(other(fork)) then \{release(fork); goto 2\}
   else take(other(fork))
6. eat
7. release(other(fork))
8. release(fork)
9. goto 1
Dining Phils: generalized case

Each fork can be shared by more than two philosophers
Dining Phils: generalized case

• **Theorem:** The algorithm of Lehmann and Rabin is deadlock-free if and only if all cycles are pairwise disconnected.

• There are essentially three ways in which two cycles can be connected:

\[\text{Diagram:} \quad \text{two interconnected hexagons, two overlapping hexagons, one hexagon with a diamond.}\]
Proof of the theorem

• If part) Each cycle can be considered separately. On each of them the classic algorithm is deadlock-free. Some additional care must be taken for the arcs that are not part of the cycle.

• Only if part) By analysis of the three possible cases. Actually they are all similar. We illustrate the first case.
Proof of the theorem

• The initial situation has probability $p > 0$
• The scheduler forces the processes to loop
• Hence the system has a deadlock (livelock) with probability $p$

• Note that this scheduler is not fair. However we can define even a fair scheduler which induces an infinite loop with probability $> 0$. The idea is to have a scheduler that “gives up” after $n$ attempts when the process keep choosing the “wrong” fork, but that increases (by $f$) its “stubbornness” at every round.

• With a suitable choice of $n$ and $f$ we have that the probability of a loop is $p/4$
Solution for the Generalized DP

• As we have seen, the algorithm of Lehmann and Rabin does not work on general graphs

• However, it is easy to modify the algorithm so that it works in general

• The idea is to reduce the problem to the pairwise disconnected cycles case:
  Each fork is initially associated with one token. Each phil needs to acquire a token in order to participate to the competition. After this initial phase, the algorithm is the same as the Lehmann & Rabin

• **Theorem:** The competing philks determine a graph in which all cycles are pairwise disconnected
  Proof: By case analysis. To have a situation with two connected cycles we would need a node with two tokens.
Dining Phils: generalized case

Reduction to the classic case: each fork is initially associated with a token. Each phil needs to acquire a token in order to participate to the competition. The competing phils determine a set of subgraphs in which each subgraph contains at most one cycle.