

# A Behavioral Analysis Tool for Models of Software Systems

Ricardo Silva

Advisor: Prof. Doutor Luis Caires

Departamento de Informática  
Faculdade de Ciências e Tecnologia  
Universidade Nova de Lisboa

13 de Fevereiro de 2009

# Motivation

- It is hard to analyze concurrent systems in order to verify their correctness
  - Non-determinism plays a crucial role in this difficulty
- Formal methods of testing seem more appropriate as simulation and extensive testing may not be able to cover all possible interleavings, and thus test all possible scenarios

## Goals of the Work

- We intend to provide additional means to help verify concurrent systems
- In our work we address the implementation of behavioral equivalence checking between systems described in a process calculus
- The key idea is to use SLMC to check a characteristic formula
- We will incorporate this means of equivalence checking into the SLMC tool

# Outline

- 1 Preliminaries
  - Process Calculi
  - CCS
  - Bisimulation
  - Hennessy-Milner's Logic
  - $\mu$ -calculus
- 2 Thesis Description
  - Simple Example
  - Semaphores
  - Work Plan

# Process Calculi

- Calculi are simple languages that can be manipulated algebraically
- Process Calculi are calculi to represent concurrent computations
- There are several ways to reason about the correctness of a system described in a process calculus
  - comparison using some sort of behavioral equivalence
  - definition of higher order properties using a logic

# Calculus of Communicating Systems

- Proposed by Robin Milner
- Calculus to capture the notion of concurrent programming
  - A process is the central entity
  - Processes evolve through synchronous message exchange
  - Complex systems are created by composing simpler processes

# The language

$P, Q$	$::=$	$0$	(inaction)
		$\alpha.P$	(prefix)
		$P + Q$	(choice)
		$P \mid Q$	(composition)
		$P \setminus L$	(restriction)
		$P[f]$	(relabelling)

# Semantics

$$\alpha.P \xrightarrow{\alpha} P \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'} \quad \frac{Q \xrightarrow{\alpha} Q' \quad P \xrightarrow{\bar{\alpha}} P'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \quad \alpha, \bar{\alpha} \notin L \quad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

# Behavioral Equivalence

- Define a process SPEC which models the intended behavior of the system
- Define a process IMPL which models how the system is implemented
- Compare SPEC and IMPL using some behavioral equivalence
- Behavioral equivalence is defined formally by the notion of bisimilarity

$$\text{SPEC} \sim \text{IMPL}$$

# Bisimulation

- Two processes are bisimilar if they can “simulate” each other's transitions
- This is captured by two processes,  $P$  and  $Q$ , being in a bisimulation iff:
  - If  $P \xrightarrow{\alpha} P'$ , then there exists  $Q'$  such that  $Q \xrightarrow{\alpha} Q'$  and  $Q'$  is bisimilar to  $P'$ .
  - If  $Q \xrightarrow{\alpha} Q'$ , then there exists  $P'$  such that  $P \xrightarrow{\alpha} P'$  and  $P'$  is bisimilar to  $Q'$ .

# Process Logics

- Bisimulation allows us to relate the behavior of two processes
- To specify general properties of a process, so-called process logics were introduced

$$\begin{array}{ccc} P & \models & A \\ \text{a process} & \text{satisfies} & \text{a property} \end{array}$$
$$\text{Sem} \models \langle \text{get} \rangle \top \wedge$$
$$[\text{get}](\langle \text{put} \rangle \top \wedge [\text{get}] \perp)$$

- One such logics is the one proposed by Hennessy and Milner
  - Has the usual logical connectives
  - Plus operators to reason about a process' behavior

# Process Logics

- Bisimulation allows us to relate the behavior of two processes
- To specify general properties of a process, so-called process logics were introduced

$$\begin{array}{ccc} P & \models & A \\ \text{a process} & \text{satisfies} & \text{a property} \\ \text{Sem} & \models & \langle \text{get} \rangle \top \wedge \\ & & [\text{get}](\langle \text{put} \rangle \top \wedge [\text{get}] \perp) \end{array}$$

- One such logics is the one proposed by Hennessy and Milner
  - Has the usual logical connectives
  - Plus operators to reason about a process' behavior

# Hennessy-Milner's Logic

$\Phi, \Psi$	::=	$\top$	(true)
		$\perp$	(false)
		$\Phi \vee \Psi$	(and)
		$\Phi \wedge \Psi$	(or)
		$\neg\Phi$	(negation)
		$\langle\alpha\rangle\Phi$	(diamond)
		$[\alpha]\Phi$	(box)

# The $\mu$ -calculus

A more expressive logic

- Kozen devised an extension to HML with fixed point operators
  - $\mu.X(\Phi)$  – minimum fixed point
  - $\nu.X(\Phi)$  – maximum fixed point
- These operators allow for the definition of recursive properties
- an example property:
  - $\nu.X(\langle a \rangle T \wedge [-]X)$
  - it is always possible to perform action  $a$

## Approach of the Work

- Model checking deals with the problem of verifying the truth value of a formula for a process
- Our approach will consist of building a logical formula describing the behavior of the process, and model checking that the other process satisfies the formula
- This provides a general approach which can be implemented in every model checker and which takes advantage of the optimizations in model checking algorithms

# Approach of the Work

## Characteristic formula

We will build a characteristic formula for a process' behavior. A characteristic formula

- is true for all processes which are bisimilar to the one which originated it
- is false to all others
- fully characterizes the behavior of the process

# A Simple Example

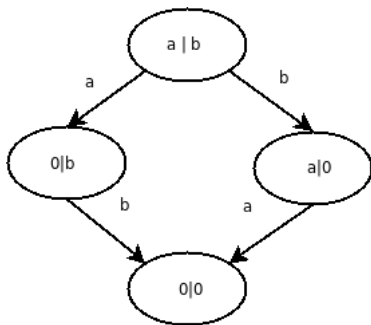
## The processes

```
defproc A = a?() | b?() ;
```

```
defproc B =  
  select{ a?().b?() ;  
         b?().a?() } ;
```

# A Simple Example

## The Labelled Transition System



# A Simple Example

## Characteristic Formula

```
defprop s =  
  <a?>s1 and <b?>s2 and  
  [a?]s1 and [b?]s2 and  
  [a!]false and  
  [b!]false and  
  [c] false;
```

```
defprop s1 =  
  <b?>s3 and [b?]s3 and  
  [a]false and  
  [b!]false and  
  [c]false;
```

```
defprop s2 =  
  <a?>s3 and  
  [a?]s3 and  
  [a!]false and  
  [b]false and  
  [c] false;
```

```
defprop s3 =  
  [a]false and  
  [b]false and  
  [c]false ;
```

# A Simple Example

## Verification in SLMC

```
> check A |= s;
```

```
Processing...
```

```
* Process A satisfies the formula s *
```

```
> check B |= s;
```

```
Processing...
```

```
* Process B satisfies the formula s *
```

# A Simple Example

## Building the formula

For each state in the LTS we have a logical formula. The formula for each state is a logical conjunction containing:

- $\langle \alpha \rangle s$  for each transition  $\alpha$  leading to the state satisfying  $s$ .
- $[\alpha](s_1 \vee s_2 \vee \dots \vee s_n)$  for all states ( $s_1$  to  $s_n$ ) lead to by label  $\alpha$ , for every different label  $\alpha$ .
- $[\alpha]\perp$  for all transitions  $\alpha$  not possible in that state.

# Semaphores

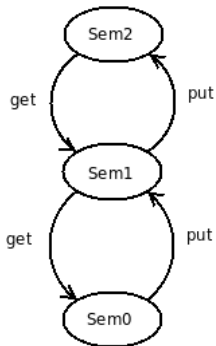
## The Processes

```
defproc   Sem2 = get?().Sem1
and      Sem1 = select { get?().Sem0 ; put?().Sem2 }
and      Sem0 = put?().Sem1 ;

defproc   Sem = get?().put?().Sem ;
defproc SemSem = Sem | Sem ;
```

# Semaphores

## The Labelled Transition System



# Semaphores

## The Properties

```
defprop bisim =
```

```
  maxfix X.(
```

```
    <get?>s2(X) and
```

```
    [get?]s2(X) and
```

```
    [put]false and
```

```
    [get!]false);
```

```
defprop s3(Y) =
```

```
  <put?>Y and
```

```
  [put?]Y and
```

```
  [get]false and
```

```
  [put!]false;
```

```
defprop s2(X) =
```

```
  maxfix Y.(
```

```
    <put?>X and
```

```
    [put?]X and
```

```
    <get?>s3(Y) and
```

```
    [get?]s3(Y) and
```

```
    [put!]false and
```

```
    [get!]false);
```

# Semaphores

## Building the formula

To build this new formula we extend the rules presented above with a rule for states where a transition from a cycle leads to:

- Create a maximum fixed-point variable and pass it along as a parameter for the other formulas to loop back when appropriate

# Work Plan

- The  $\pi$ -calculus is an extension of CCS which allows names of communicating channels to be sent in messages
  - We will extend this technique to the  $\pi$ -calculus
  - We will implement the verification procedure in the SLMC tool
- 1 Define the rules to build the characterizing formula for  $\pi$ -calculus processes (6-8 weeks)
  - 2 Implement the checker in SLMC (4-5 weeks)
  - 3 Writing of the dissertation (8-10 weeks)
  - 4 Example production will undergo through most stages, however the examples used in comparison with other tools will be done during the third stage