Tool support for MSOS and CBabel using Maude

Christiano Braga
cbraga@ic.uff.br

http://www.ic.uff.br/~cbraga

Universidade Federal Fluminense (UFF)

Objective of this talk . . .

. . . is to give details about our research projects using Maude, once we have already given the context and motivated them in the department talk.

Talk Plan

For each project we:

1. Recapitulate its motivation and the source language syntax.
2. Present the mapping from the source language into rewriting logic.
3. Present the general implementation architecture as an extension to Full Maude.
4. Present some examples of the generated theories.
5. Point out pros and cons.
6. Propose possible collaborations with the UCMaude group.

Maude MSOS Tool (MMT)

Recapitulating motivation:

- Mosses' modular structural operational semantics solves SOS modularity problem, which means that specifications in MSOS can be written as conservative extensions of existing ones. In practical terms, once a new semantic component is added, there is no need to change existing rules.
- This is accomplished by means of structuring the labels as records: when a new semantic element is added, a new index should be added to the label to refer to that new semantic component. The new rules only mention the newly added semantic component and therefore existing rules remain the same.

Maude MSOS Tool (MMT)

Recapitulating motivation:

- MMT gives tool support for MSOS with syntax features that allows concise specifications to be written.

Maude MSOS Tool (MMT)

Recapitulating syntax:

```
(msos PROCESS is
  Process .
  Process ::= 0 .
  Process ::= Action ; Process [prec 20] .
  Process ::= Process + Process [assoc comm prec 30] .
  Process ::= Process || Process [assoc comm prec 25] .
  Process ::= rel (Process, Label, Label) [prec 20] .
  Process ::= Process \ Label [prec 25] .

  Label = {trace' : Action*, ...} .

  [prefix] (Action ; Process) : Process -{trace' = Action,-}-> Process .

             Process1 -{trace' = Action,-}-> Process1' ,
             Process2 -{trace' = ~ Action,-}-> Process2'
  [par2] -- ---------------------------------------------------------
             (Process1 || Process2) : Process -{trace' = tau,-}->
                                                  Process1' || Process2' .
...
)
```

Maude MSOS Tool (MMT)

The mapping from MSOS to RWL:

- The syntactic features of MSDF are transformed into a signature of a rewrite theory. In this talk we will focus on the transformation of label declarations and the transition rules.
- The first step is the choice of representing MSOS transition rules as conditional rewrite rules in RWL, between the proposals of Martí-Oliet and Meseguer.
- On top of this representation, we structure the configuration being rewritten as a *pair*, composed by the *program syntax* and a *record* that holds the semantic components, such as the environment, the store or synchronization signals.

Maude MSOS Tool (MMT)

The mapping from MSOS to RWL:

- The technique of organizing the configuration like above, together with the very idea of representing the semantic datatypes by means of *abstract datatypes*, we call *modular* rewriting semantics (MRS).
- What is left for us to think about is how a label in a transition should be represented in a rewrite.

Maude MSOS Tool (MMT)

The mapping from MSOS to RWL:

- First we should define the *pre* and *post* projections of a label in a transition. These projections are defined by cases on each possible index:
  - read-only: *pre* and *post* just project (both) the value bound to a read-only index.
  - read-write: *pre* projects the first component of the pair bound to the read-write index, and *post* projects the second component.

Maude MSOS Tool (MMT)

The mapping from MSOS to RWL:

  - write-only: *pre* projects the prefix of the monoid bound to the write-only component and the *post* projection produces the *pre* value appended to value in the current transition. If the label is in a premise, the *pre* projection produces the empty value of the monoid.

- Now, if we consider the *pre* and *post* projections and the representation of transition rules as MRS conditional rewrite rules, it should be easy to see how a label in transition is represented as *records* in a rewrite.

Maude MSOS Tool (MMT)

The mapping from MSOS to RWL:

- That is, we look at the labels in a computation as a *pre-order*.
- Given a MSOS specification $\mathcal{S}$, this transformation is *semantics-preserving* in the precise sense of taking the form of a strong bissimulation of the labeled transition system associated with the initial reachability preorder (restricted to the sort for MRS configurations) that models the rewrite theory generated from $\mathcal{S}$ and the category of finite computations defined by $\mathcal{S}$.

Maude MSOS Tool (MMT)

MMT as an extension to Full Maude:

- The implementation of MMT followed the usual process of extending Full Maude, by extending the grammar, the database handling, adding new commands, writing the parser, bubble solver, and finally connecting this “front end” with the transformation function.
- One issue in the case of MMT is the fact that a MSOS unit does not have explicit includes, but implicit inclusions and “sees”. This had to be adapted to whenever Full Maude wanted to “know” about MSOS unit inclusions, the proper modules were returned, that is, a sort of intermediate representation for the meta-representation of a MSOS unit had to be created in order to Full Maude work properly.

Maude MSOS Tool (MMT)

An example of a generated rewrite theory: CCS

- Translating label indices: a label index declaration adds a new index to the MRS `RECORD` theory by means of a membership equation.
- MSDF:

```
Label = {trace' : Action*, ...} .
```

- Maude:

```
op trace' : -> WO-Index
mb trace' = V@:Seq`(Action`) : WOField .
```

Maude MSOS Tool (MMT)

An example of a generated rewrite theory: CCS

- The translation of a transition rule gives rise to a conditional rewrite rule such that the configuration being rewritten requires additional constructors to cope with the `step` rule: a trick to control the numbers of rewrites, necessary to represent a one-step MSOS transition as a rewrite.
- MSDF:

```
         Process1 -{trace' = Action,-}-> Process1' ,
         Process2 -{trace' = ~ Action,-}-> Process2'
[par2] -- ------------------------------------------------------------
         (Process1 || Process2) : Process -{trace' = tau,-}->
                                                 Process1' || Process2' .
```

Maude MSOS Tool (MMT)

An example of a generated rewrite theory: CCS

- Maude:

```
crl {Process1:Process || Process2:Process ::: 'Process,
                                    {trace' = @C:Seq`(Action`),-:PreRecord}}
    => [Process1':Process || Process2':Process ::: 'Process,
                                {trace' = @C:Seq`(Action`),tau,-:PreRecord}]
if {Process1:Process ::: 'Process,{trace' =(),-:PreRecord}} =>
   [Process1':Process ::: 'Process,{trace' = Action:Action,-:PreRecord}] /\
   {Process2:Process ::: 'Process,{trace' =(),-:PreRecord}}=>
   [Process2':Process ::: 'Process,{trace' = ~ Action:Action,-:PreRecord}]
[label par2] .
```

Maude MSOS Tool (MMT)

Pros and cons

- Alberto Verdejo has shown several forms of representing SOS is Maude. There are also other tools to execute operational semantics such as LETHOS, for SOS, or Typol, for Natural Semantics. We believe we contribute to that effort by devising a tool that allows for the specification of concise and modular operational semantics allowing the possibility of the application of Maude analyzes tools to such specifications.

Maude MSOS Tool (MMT)

Pros and cons

- It is also true that the same conciseness may be error prone, such as forgetting to write a quote in a label. Maybe it's the price to pay...
- It is true that the techniques applied here, namely record inheritance and abstract datatypes, can be directly used in a Maude specification without the MSDF syntax. However with MMT this is already implicit in the tool, that is, is just a matter of *abstraction* just like object-oriented concepts can be added to imperative, functional or logical functions.

Maude MSOS Tool (MMT)

Collaborations with the UCMaude group:

- We are currently working with Alberto Verdejo on the integration of MMT and the strategy language interpreter (SLI). Actually the integration was already done, just by writing a Maude module that integrates both grammars and interfaces under Full Maude. An example using CCS has been written. MMT now has a command that turns on and off the generation of the step rule therefore allowing a proper combination with SLI.

CBabel Tool (CBT)

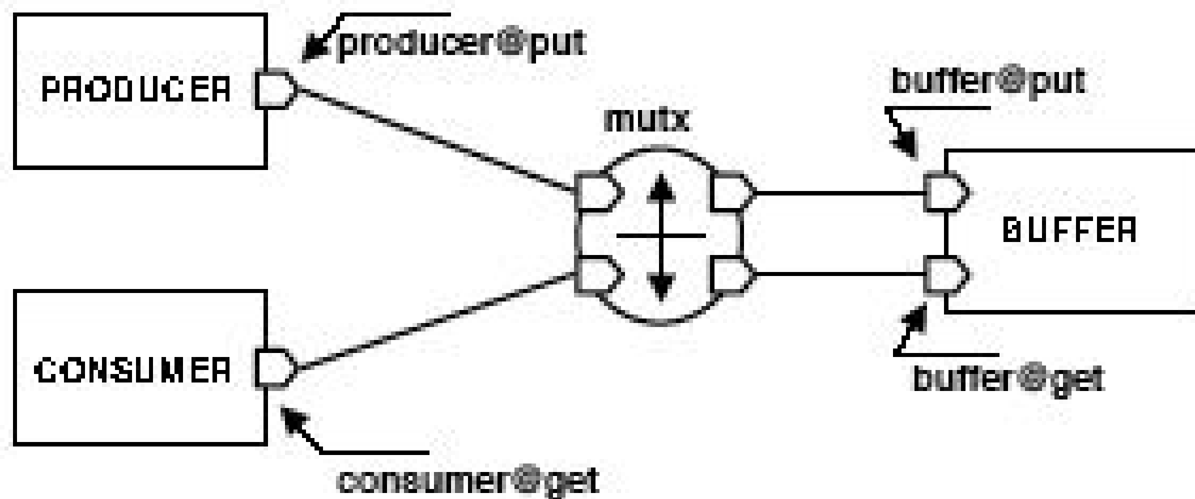
Recapitulating motivation:

- Software architecture description languages are used to glue together components by specifying the so-called non-functional aspects such as coordination, interaction and quality-of-service.
- The CBabel language is a software architecture description language (ADL) that allows such descriptions by means of contracts, that is, such non-functional aspects are made first-class citizens in the language.
- CBabel tool gives tool support for the analyzes of such descriptions by implementing a formally defined mapping from CBabel to RWL, therefore making available the Maude tools to CBabel descriptions.

CBabel Tool (CBT)

Recapitulating syntax: Producers, consumers, a buffer and a mutex connector.

- The following architecture can be drawn using our (under development) Eclipse environment, called FormArch...



CBabel Tool (CBT)

Recapitulating syntax: Producers, consumers, a buffer and a mutex connector.

- Generating the following textual description: (The modules PRODUCER, CONSUMER and BUFFER are not shown.)

```
connector MUTEX {
 in port mutex@in1 ;
 in port mutex@in2 ;
 out port mutex@out1 ;
 out port mutex@out2 ;
 exclusive{
  mutex@in1 > mutex@out1 ;
  mutex@in2 > mutex@out2 ;
 }
}
```

```
application PC-MUTEX {
 instantiate BUFFER as buff ;
 instantiate PRODUCER as prod ;
 instantiate CONSUMER as cons ;
 instantiate MUTEX as mutx ;
 link prod.producer@put to mutx.mutex@in1 ;
 link mutx.mutex@out1 to buff.buffer@put ;
 link cons.consumer@get to mutx.mutex@in2 ;
 link mutx.mutex@out2 to buff.buffer@get ;
}
```

CBabel Tool (CBT)

The mapping from CBabel to RWL:

CBabel		RWL
component	$\longrightarrow$	object-oriented theory
component instance	$\longrightarrow$	object
application state	$\longrightarrow$	multiset of objects
port	$\longrightarrow$	message declaration
port stimulus	$\longrightarrow$	message passing (rewrite rules)
link	$\longrightarrow$	unconditional equation
coordination contract	$\longrightarrow$	rewrite rules
local or state required variable	$\longrightarrow$	class attribute
bind of variables	$\longrightarrow$	equations

CBabel Tool (CBT)

The mapping from CBabel to RWL:

There are two main principles we tried to follow while devising the mapping:

1. keep the modularity of the original description and
2. give semantics to the *main* concepts of the language, such that other constructions could be seen as syntax sugar or "macros".

CBabel Tool (CBT)

The mapping from CBabel to RWL:

- Item 1, in the previous slide, was enforced because we assumed that would be an important matter while discussing dynamic reconfiguration. Intuitively, if the representation was to be monolithic dynamic reconfiguration could become a problem.
- Also because of item 1, synchronous communication is not done in the traditional RWL way of having two objects in the left-hand side of a rule. (We do not know before hand which are the synchronizing objects.) Synchronization then is realized through message passing, that is, there are generic `send` and `ack` messages.

CBabel Tool (CBT)

The mapping from CBabel to RWL:

- Since it may be possible to have more then one instance of a module connected to a port, we keep track of the actual path the message takes inside the message: the interaction. This allows the proper acknowledgment in such a configuration.
- A consequence of item 2 is that we only allow one contract per connector. Actually there are some open issues regarding the semantics of more then one contract in one connector and their composition.

CBabel Tool (CBT)

CBT as an extension to Full Maude:

- The implementation of CBT also followed the usual process of extending Full Maude, by extending the grammar, the database handling, adding new commands, writing the parser, bubble solver, and finally connecting this “front end” with the transformation function.
- CBT shares with MMT the fact its unit does not have explicit includes. The inclusion is managed by the applications. So it also needed to be adapted to whenever Full Maude wanted to “know” about unit inclusions.

CBabel Tool (CBT)

CBT as an extension to Full Maude:

- Perhaps a document explaining how to extend Full Maude, or even to define a (meta)language and a tool to help in that matter could be an interesting piece of work.

CBabel Tool (CBT)

An example of the generated rewrite theory: Mutex

- Each component, that is, either a module or a connector, gives rise to an object oriented theory. A connector gives rise to a class declaration and a `instantiate` “constructor” equation. The port declarations are declared as constants to be used as parameters of generic `send` and `ack` messages.

CBabel Tool (CBT)

An example of the generated rewrite theory: Mutex

```
Maude> (show module MUTEX .)
omod MUTEX is
  including CBABEL-CONFIGURATION .

  class MUTEX | status : PortStatus .
  eq instantiate(O:Oid,MUTEX) = < O:Oid : MUTEX | status : unlocked > .

  op mutex-in1 : -> PortInId   [ctor] .
  op mutex-out1 : -> PortOutId [ctor] .
  op mutex-in2 : -> PortInId   [ctor] .
  op mutex-out2 : -> PortOutId [ctor] .
```

CBabel Tool (CBT)

An example of the generated rewrite theory: Mutex

- The mutual exclusive contract is given by a single rule that chooses a message to be rewritten, if the status of the connector is `unlocked`, thus turning it to `locked`. When the proper `ack` message arrives, the connector is `unlocked` and another message can be chosen.

```
rl < O:Oid : MUTEX | status : unlocked >
   send(O:Oid,mutex-in1,IT:Interaction)
   => < O:Oid : MUTEX | status : locked >
   send(O:Oid,mutex-out1,[O:Oid,mutex-out1] :: IT:Interaction)
 [label MUTEX-sending-mutex-in1] .
rl < O:Oid : MUTEX | status : locked >
   ack([O:Oid,mutex-out1] :: IT:Interaction) =>
   < O:Oid : MUTEX | status : unlocked > ack(IT:Interaction)
 [label MUTEX-acking-mutex-out1] .
[...]
endom
```

CBabel Tool (CBT)

An example of the generated rewrite theory: PCBM Application

- Each instantiate command gives rise to an object identifier named after the instance name in the command and to a call to the appropriate instantiate function. A constant topology is also created holding all the objects in the instantiated architecture. Each link command is mapped to an equation that simply rewrites a message `send` to another, according to the ports in the link command.

CBabel Tool (CBT)

An example of the generated rewrite theory: PCBM Application

```
omod PC-MUTEX is
  including CBABEL-CONFIGURATION . including BUFFER .
  including CONSUMER . including MUTEX . including PRODUCER .
  op buff : -> Oid .    op cons : -> Oid .
  op mutx : -> Oid .    op prod : -> Oid .
  op topology : -> Configuration .
  eq topology = instantiate(buff,BUFFER)instantiate(cons,CONSUMER)
     instantiate(mutx, MUTEX)instantiate(prod,PRODUCER)none .
  eq send(cons,consumer@get,IT:Interaction)
    = send(mutx,mutex@in2,IT:Interaction) [label ...] .
  eq send(mutx,mutex@out1,IT:Interaction)
    = send(buff,buffer@put,IT:Interaction) [label ...] .
  eq send(mutx,mutex@out2,IT:Interaction)
    = send(buff,buffer@get,IT:Interaction) [label ...] .
  eq send(prod,producer@put,IT:Interaction)
    = send(mutx,mutex@in1,IT:Interaction) [label ...] .
endom
```

CBabel Tool (CBT)

The mapping of other constructs:

- Sequential interaction between two ports: represented by an unconditional rule.
- Guarded interaction between two ports: given that the condition is true, a `before` equation, representing its before clause, is applied to the object that represents the connector and the message is rewritten by a rule. When the acknowledgment message arrives the `after` equation, representing the connectors after clause, is applied to the connector object and the acknowledgment message is forwarded. If the guard condition does not hold, the message representing the input port is simply not rewritten.

CBabel Tool (CBT)

The mapping of other constructs:

- Binding between `state required` variables: state required variables are shared variables. When two state required variables are bound by the application module, they should be synchronized. This is done by means of equations that update the variables accordingly. Since equations are applied before rules, communication will always occur with synchronized variables.

CBabel Tool (CBT)

Pros and cons

- With ADLs we one can focus either on the component or on non-functional aspects of the application keeping the so-called separation of concerns proper of component based programming. With our tool we support this paradigm by providing analyzes tools for descriptions that follow this approach.
- However, there are some drawbacks. Component based programming yields quite verbose descriptions since everything is organized by interfaces that have to be connected afterwords. On the other hand, that's where visual programming comes into place.

CBabel Tool (CBT)

Pros and cons

- Our tool currently implements a mapping that produces a huge state system, even for simple applications. This is due to a literal representation of the components and their communication as objects and messages.
- We are currently researching on more concise representations perhaps paying the price of modularity in order to produce specifications more suitable for analyzes. Another approach that is also being investigated is the use of abstractions, that would have the interesting property of not requiring the change of the current representation that may be seen as a natural view of architectural concepts in terms of objects and messages.

CBabel Tool (CBT)

Collaborations with the UCMaude group:

- Miguel Palomino has been kind enough to explain us how to use the abstraction generation prototype. We are currently investigating how the integration between CBT and the prototype may occur.

Tool support for MSOS and CBabel

Thank you!