Language-oriented Software Development and Rewriting Logic

Christiano Braga
cbraga@ic.uff.br

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

Universidade Federal Fluminense (UFF)

Acknowledgments

- Prof. Narciso Martí-Oliet and the UCMaude Group.
- The Advanced Systems Research Group of the Computer Science Institute at UFF.
- The Brazilian Research Council, CNPq.

C.S. Graduate Program at UFF

- UFF: located in Niterói, RJ, Brazil; aprox. 22.000 undergraduate students, 3.000 graduate students, and aprox. 2.200 professors.
- PGC: 134 graduate students and 20 professors, in the following research areas:
  - Parallel and distributed processing.
  - Computer graphics and human-computer interaction.
  - Combinatorial optimization and artificial intelligence.
  - Mathematical modeling and power systems.

Advanced Systems Research Group

- Recently formed, with 4 staff and 10 graduate students.
- Our group has the following interests:
  - Formal methods
  - Mobile computing
  - Real-time systems
  - Software architecture
  - Software adaptability
  - Power management in computer systems
- (Under construction) web page: `http://asr.ic.uff.br`

Objective of this talk . . .

. . . is to present our research initiatives, and in particular, our results while looking at rewriting logic as a *semantic framework*.

Talk Plan

1. First a software engineering approach that we call *language-oriented software development* (LoSD) is *informally* introduced. The LoSD approach concisely represents our line of research.
2. Next, LoSD is related with rewriting logic and the way formal tools are developed in the Maude system.

Talk Plan

3. Finally we motivate, exemplify and informally discuss how tool support has been given to the specification and analysis of modular operational semantics specifications and software architecture descriptions.

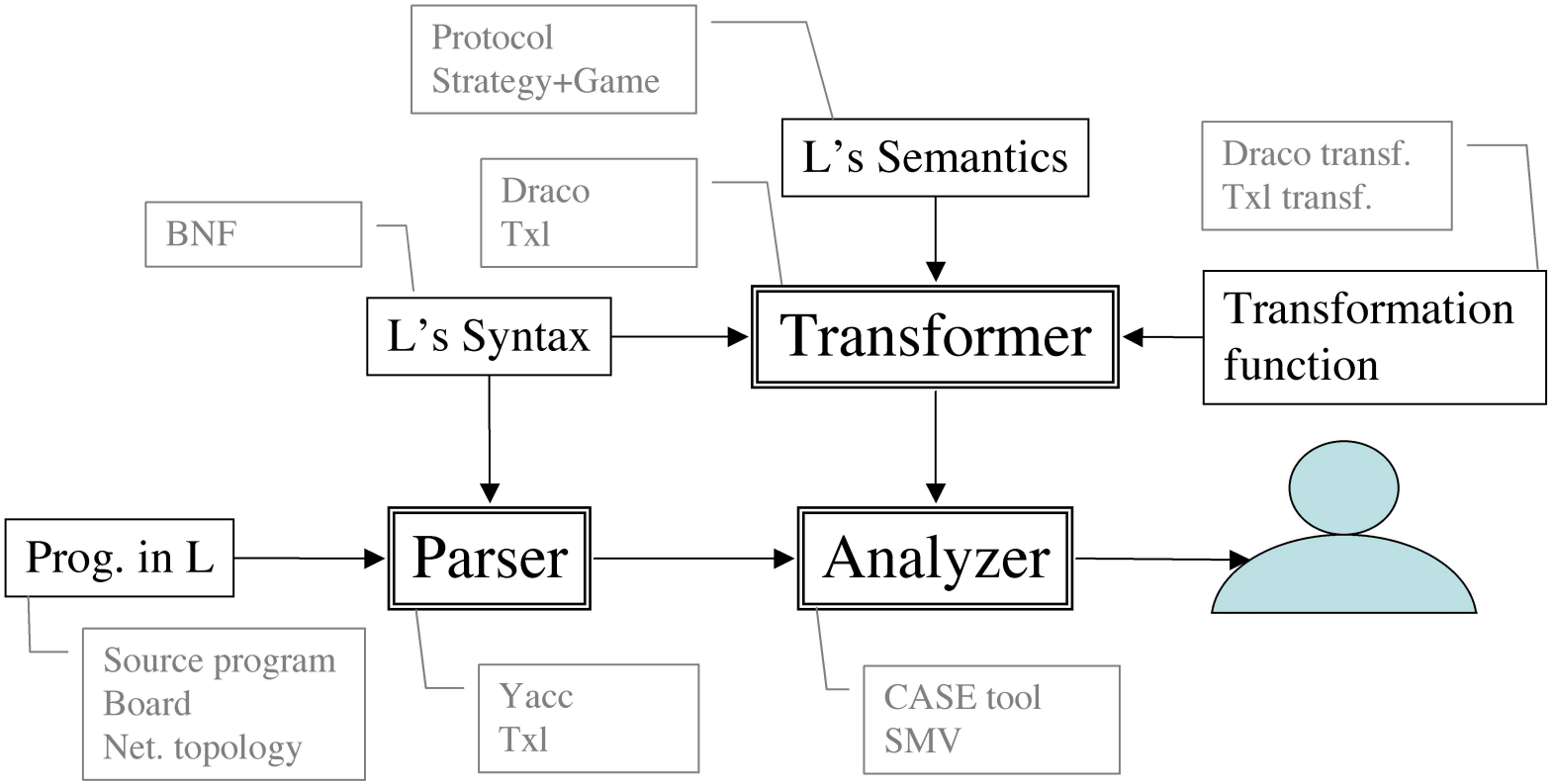
Language-oriented software development (LoSD)

- Is a software engineering approach that relies on formal software tools built using the syntax and semantics of the language from the problem domain.
- Examples of such formal tools are language transformers (such as W3 XSLT and Cordy's TXL), model checkers (such as CMU's SMV or Birmingham's PRISM), and theorem provers (such as SRI's PVS and (Cambridge and TU Munich)'s Isabelle).

Some projects using LoSD

- A language-independent software documentation generation tool, combining Draco transformation tool and a CASE (Computer Aided Software Engineering) tool. (My own M.Sc dissertation at PUC-Rio, directed by Arndt von Staa.)
- Analysis of game strategies combining TXL and SMV. (M.Sc dissertation of Davi Romero at PUC-Rio, directed by Hermann Haeusler.)
- Analysis of communication protocols using TXL and SMV. (Ph.D. thesis of Carlos Bazilio at PUC-Rio, being directed by Hermann Haeusler.)

Language-oriented software development (LoSD)

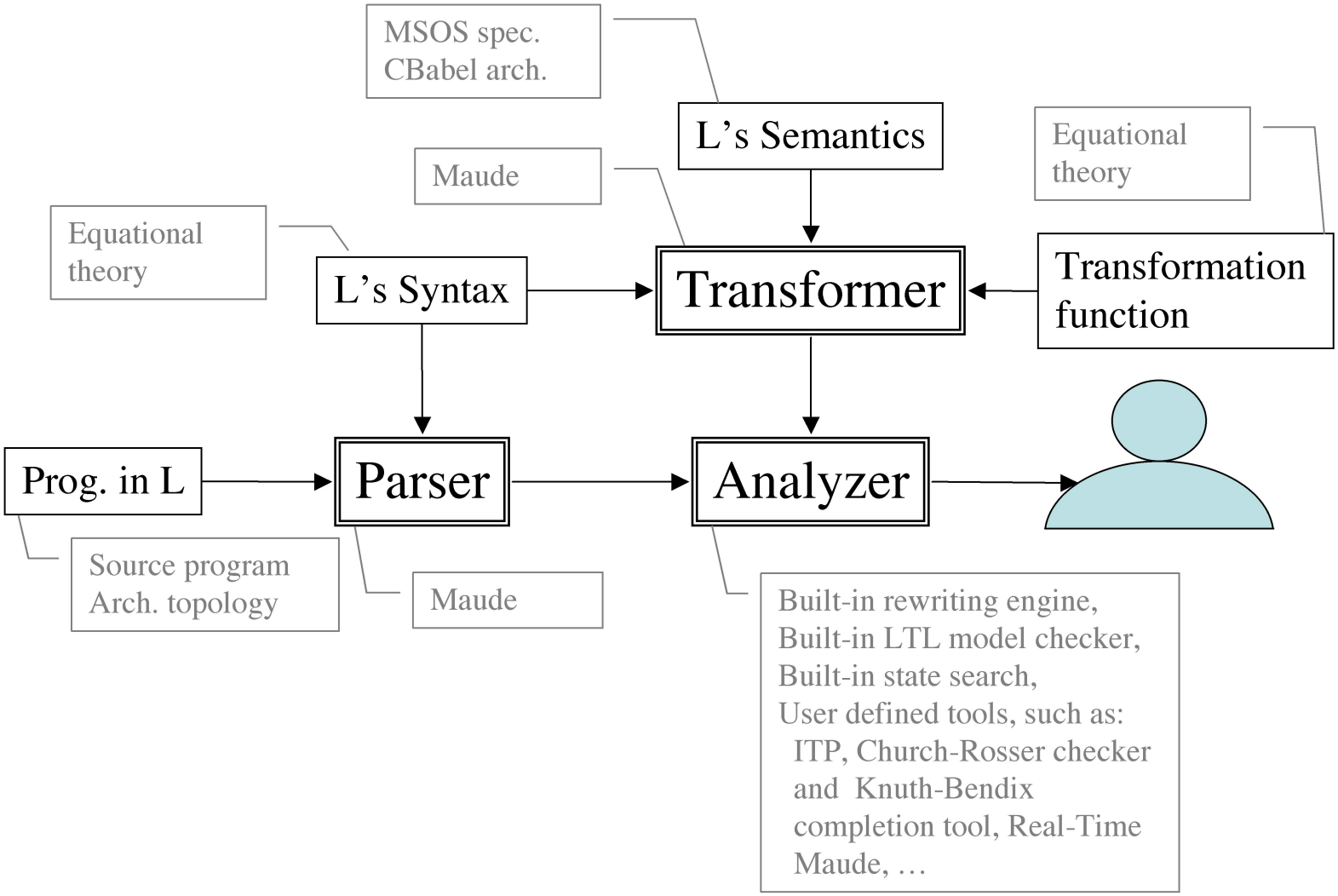


Formal tools in rewriting logic and Maude

In a “nut shell”:

- Rewriting logic (RWL) is a *reflective logic*: one can reason about RWL within RWL.
- RWL is a *logic* and a *semantic framework*: many logics, specification languages and models of computation can be represented in RWL.
- Maude is a fast implementation of RWL that allows *meta-programming* as a realization of RWL reflective features.

LoSD with Maude



Current projects

We instantiate ’L’, with two different languages:

- Case ’L = MSOS’ : Maude MSOS Tool, which gives tool support for the specification and analysis Mosses’ modular structural operational semantics (MSOS) specifications.
- Case ’L = CBabel’ : CBabel Tool, which gives tool support for the specification and analysis of software architecture descriptions.

Maude MSOS Tool (MMT)

What is MMT?

- MMT gives tool support to modular structural operational semantics (MSOS).
- MSOS is a variation of Plotkin's structural operational semantics (SOS) with support to modular specifications, that is, specifications in MSOS may be defined as *conservative extensions* of existing MSOS specifications.

Why MSOS?

- To illustrate the modularity problem of SOS, let us consider the following specification for the semantics of the binary operation $\bullet$ with an environment (ρ).

$$\frac{\rho \vdash e_0 \rightarrow e_0'}{\rho \vdash e_0 \bullet e_1 \rightarrow e_0' \bullet e_1}$$

Why MSOS?

- Now if one wants to *extend* the semantics of $\bullet$ with an environment (ρ) to add a store (σ, σ'), the previous rule would have to be modified.

$$\frac{\rho \vdash \langle e_0, \sigma \rangle \rightarrow \langle e_0', \sigma' \rangle}{\rho \vdash \langle e_0 \bullet e_1, \sigma \rangle \rightarrow \langle e_0' \bullet e_1, \sigma' \rangle}$$

Maude MSOS Tool (MMT)

Why MSOS?

- Mosses' MSOS solves the modularity problem in structural operational semantics.
- Transition *labels* carry the semantic information associated with computations and configurations are only value-added abstract syntax trees.

$$\frac{e_0 \; -X\rightarrow \; e_0'}{e_0 \bullet e_1 \; -X\rightarrow \; e_0' \bullet e_1}$$

Why MSOS?

- Components and are accessed through indices

$$\frac{e \; -\{\rho = \rho_1[\rho_0], \ldots\}\rightarrow \; e'}{\mathbf{let}\ \rho_0\ \mathbf{in}\ e\ \mathbf{end} \; -\{\rho = \rho_1, \ldots\}\rightarrow \mathbf{let}\ \rho_0\ \mathbf{in}\ e'\ \mathbf{end}}$$

Maude MSOS Tool (MMT)

Why MSOS?

- Indexed components in labels are of three different types. (More can be defined, if necessary.)
  - read only (e.g. environments of bindings),
  - read write (e.g. stores),
  - write only (e.g. output).

Maude MSOS Tool (MMT)

How is MMT implemented?

- MMT implements a formally defined and proven correct mapping from MSOS to RWL.
- MMT is implemented as an extension to Full Maude, a tool that endows Maude with an extensible module algebra.
- MMT supports a specification language called MSDF, also developed by Mosses. MSDF has interesting characteristics to allow concise language semantics specifications.

Maude MSOS Tool (MMT)

How is MMT implemented?

- MSDF features:
  - BNF style for grammar specification.
  - Concise syntax for label declaration.
  - Concise syntax for transition rule declaration.
  - Implicit variable declaration.
  - Implicit (and explicit) module inclusion.
  - Implicit data type declaration for sequences and non-empty sequences.
  - Built-in parameterized types: Map, Set and List.
  - Typed abstract syntax trees.

Maude MSOS Tool (MMT)

Example: CCS in MSDF

- Grammar declaration: BNF + Maude operator attributes.

```
(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] .
```

Maude MSOS Tool (MMT)

Example: CCS in MSDF

- Label declaration with implicit sequence sort.

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

Maude MSOS Tool (MMT)

Example: CCS in MSDF

- Rule declaration with implicit variables.

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

         Process1 -{...}-> Process1'
[sum] -- ------------------------------------------------
         (Process1 + Process2) : Process -{...}-> Process1' .

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

...
sosm)
```

Maude MSOS Tool (MMT)

What can one do with MMT?

- MSDF specifications may be run and verified using Maude and the tools developed for the analysis of Maude specifications.
- Some experiments:
  - Specified and model-checked several distributed algorithms described in Lynch's 'Distributed Algorithms' book.
  - Specified and model-checked simple algorithms written in CML, a concurrent functional language.

Maude MSOS Tool (MMT)

What can one do with MMT?

- Some experiments:
  - Specified and executed ’Incremental MSOS’ (IMSOS), which is an MSOS semantics for a set of abstract constructs for the specification of programming language semantics. With IMSOS the semantics of a programming language is given in terms of a mapping from its concrete syntax to the IMSOS abstract constructs.

Maude MSOS Tool (MMT)

What can one do with MMT?

- Some experiments:
  - An integration between MMT and the strategy language interpreter prototype developed by Martí-Oliet, Meseguer and Verdejo. (The MSDF specification of CCS presented in this talk was written together with Verdejo during this work.)

Maude MSOS Tool (MMT)

MMT exists due to the collaboration of several people:

- Fabricio Chalub, E. Hermann Hæusler, José Meseguer, and Peter Mosses.

CBabel Tool (CBT)

What is CBT?

- CBT gives tool support to the specification and analysis of software architecture descriptions written in the CBabel software architecture description language (ADL).
- CBabel was designed by Alexandre Sztajnberg and is a simple ADL that concisely captures relevant aspects of software architectures by means of *contracts*, such as sequential interaction, mutual exclusion, guarded interaction and quality-of-service (QoS).

CBabel Tool (CBT)

Why CBabel?

- The purpose of architecture description Languages (ADLs) is to keep *separated* the description of how distributed components are *connected* from the descriptions of the *internal behavior* of each component.

CBabel Tool (CBT)

Why CBabel?

- By using the concept of contracts, CBabel allows the treatment of coordination aspects in a more flexible manner when compared to other ADLs. CBabel also provides QoS contracts that cater for other non-functional aspects. (QoS contracts are not yet handled by the tool.)

CBabel Tool (CBT)

Why CBT?

- A formal semantics for a architecture description language $\mathcal{L}$ provides:
  - An unambiguous definition of what $\mathcal{L}$ *means*.
  - The ability to formally reason about $\mathcal{L}$ and *prove* properties about architectures, such as coordination related properties.
  - Moreover, if the specification is *executable*, the formal reasoning can be *computer aided*.

CBabel Tool (CBT)

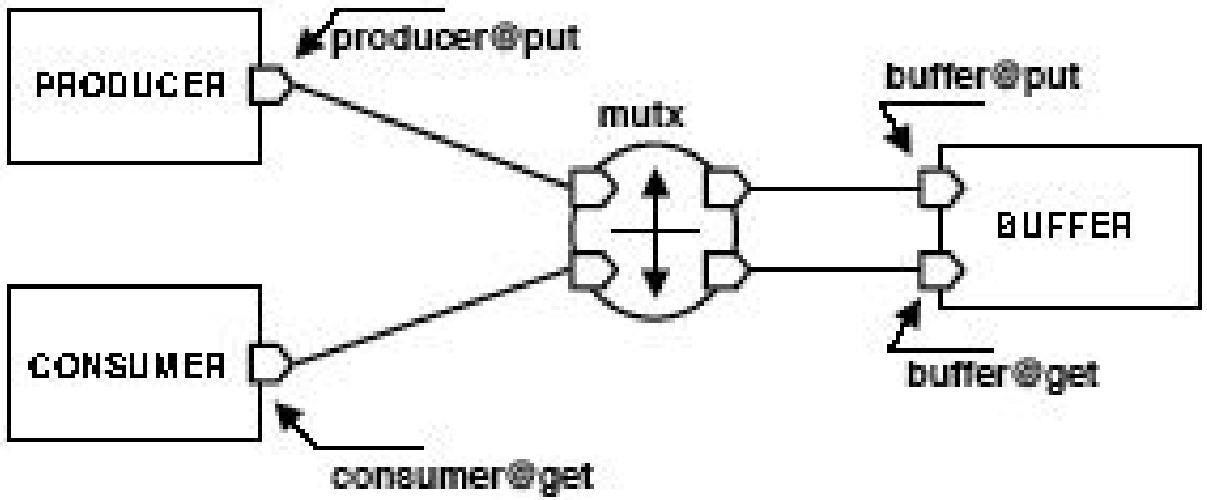
How is CBT implemented?

- CBT implements a formally defined transformation from CBabel to RWL. (Before our formalization, CBabel had no formal specification of its semantics. To prove our transformation correct, an abstract semantics would need to be specified.)
- CBT is implemented, like MMT and other Maude tools, as an extension to Full Maude.

CBabel Tool (CBT)

Example: 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)

Example: 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)

What can one do with CBT?

- CBabel descriptions may be run and verified using Maude and the tools developed for the analysis of Maude specifications. Several toy examples have been described and model checked.
- Unfortunately, the state space produced by our mapping from CBabel to RWL has a huge number of states, even for simple applications as the 5 philosophers. This has prevented us from developing more significant examples.

CBabel Tool (CBT)

What can one do with CBT?

- We are currently exploring ways to make the mapping more efficient, perhaps paying the price of modularity and the “natural” representation of architectural concepts as objects and messages. Another research direction is the application of *abstraction* techniques.

CBabel Tool (CBT)

CBT exists due to the collaboration of :

- Alexandre Rademaker and Alexandre Sztajnberg.

Thank you!