Implementing Modular SOS in Maude

Fabricio Chalub Barbosa do Rosário

May 26, 2005

Why modular specifications?

- essential in large-scale specification projects (ease of extension);
- leads to a better design and easier understanding;
- education.

Why tool support for operational semantics?

- “interpreter for free”;
- verification;
- prettyprinting (documentation).

Why a formal tool?

- precise axiomatization of MSOS;
- why Rewriting Logic and Maude?

SOS is not modular

(Structural Operational Semantics.) Lack of modularity: earlier rules must be replaced if new components are added to the specification.

$$\begin{aligned} f(t_0, \ldots, t_n) &\rightarrow t \\ \rho \vdash f(t_0, \ldots, t_n) &\rightarrow t \\ \rho \vdash f(t_0, \ldots, t_n), \sigma &\rightarrow t, \sigma \\ \rho \vdash f(t_0, \ldots, t_n), \sigma &\xrightarrow{\tau} t, \sigma \end{aligned}$$

MSOS is modular

(Modular SOS.) Components moved from configurations to labels. Labels only need to explicit the components needed for a certain transition.
Modularity in MSOS:

- Semantically: generalized transition systems where labels are morphisms of a category;
- Syntactically: labels are structured as records.

Transitions have labels with "unspecified components".

Modular SOS Specification Formalism, MSDF

Mosses's specification language for MSOS. Three main parts:

- abstract syntax declaration (using sets and functions);
- label declaration;
- transitions.

```
Id .                   Env = (Id, Int)Map .
Exp .                  St = (Loc, Int)Map .
Exp ::= Id | if Exp then Exp else Exp | tup Exp* .
Label = { env : Env, st : St, st' : St, ... }
      St' := f(St), Exp -{ st = St, st = St', ...}-> Exp'
-- -----------------------------------------------------------------
tup(Exp*,Exp,Exp*) -{ st = St, st = St', ...}-> tup(Exp*,Exp',Exp*) .
```

Modular SOS Specification Formalism, MSDF

Mosses's specification language for MSOS. Three main parts:

- abstract syntax declaration (using sets and functions);
- label declaration;
- transitions.

```
Id .                  Env = (Id, Int)Map .
Exp .                 St = (Loc, Int)Map .
Exp ::= Id | if Exp then Exp else Exp | tup Exp* .
Label = { env : Env, st : St, st' : St, ... }
      St' := f(St), Exp -{ st = St, st = St', ...}-> Exp'
-- -----------------------------------------------------------------
tup(Exp*,Exp,Exp*) -{ st = St, st = St', ...}-> tup(Exp*,Exp',Exp*) .
```

Modular SOS Specification Formalism, MSDF

Mosses's specification language for MSOS. Three main parts:

- abstract syntax declaration (using sets and functions);
- label declaration;
- transitions.

```
Id .                   Env = (Id, Int)Map .
Exp .                  St = (Loc, Int)Map .
Exp ::= Id | if Exp then Exp else Exp | tup Exp* .
Label = { env : Env, st : St, st' : St, ... }
      St' := f(St), Exp -{ st = St, st = St', ...}-> Exp'
-- -----------------------------------------------------------------
tup(Exp*,Exp,Exp*) -{ st = St, st = St', ...}-> tup(Exp*,Exp',Exp*) .
```

Modular SOS Specification Formalism, MSDF

Mosses's specification language for MSOS. Three main parts:

- abstract syntax declaration (using sets and functions);
- label declaration;
- transitions.

```
Id .                   Env = (Id, Int)Map .
Exp .                  St = (Loc, Int)Map .
Exp ::= Id | if Exp then Exp else Exp | tup Exp* .
Label = { env : Env, st : St, st' : St, ... }
      St' := f(St), Exp -{ st = St, st = St', ...}-> Exp'
-- -----------------------------------------------------------------
tup(Exp*,Exp,Exp*) -{ st = St, st = St', ...}-> tup(Exp*,Exp',Exp*) .
```

Why rewriting logic?

RWL unifies different computational systems and logics.

- Reflection: a mapping from a logic $\mathcal{L}$ to RWL is reified as metafunction from $D_{\mathcal{L}}$ (the data type representing $\mathcal{L}$-programs) to $\mathcal{R}$ (rewriting logic theories).
- Executable environments: created with the extension and composition of built-in metafunctions such as `metaParse`, `metaReduce`, etc.

Why Maude?

High-performance implementation of RWL ($O(10^6)$ rewrites/sec.) with several formal tools available:

- built-in: breadth-first search, LTL model checker
- available through reflection: Inductive Theorem Prover, Completeness Checker, Real Time Theories, etc.

Consistent support for the construction and interoperability of formal tools

MSOS and Rewriting Logic

Braga/Meseguer defined Modular Rewriting Semantics (MRS) as an extension of the work of Braga/Haeusler/Mosses/Meseguer for defining MSOS in terms of Rewriting Logic.
Modularity techniques in MRS:

- record inheritance: '`RECORD`' theory capture the structure of MSOS labels. Configurations in MRS are pairs (P, R) of program text and records;
- abstract interfaces: semantic components in specifications are defined abstractly and need a "concrete" implementation for the specification to be executable. (Useful when a series of extensions to the data types are created.)

From MSOS to Rewriting Logic

- We only needed the record inheritance technique for the implementation of MSOS;
- Transitions in MSOS are conditional rewrite rules where the label is separated in *pre* and *post*;
- The contents of *pre* and *post* depends on the types of components on the label;

MMT as an extension of Full Maude

- "Maude way" of constructing formal tools: metafunction '`convertMSOS`' over terms that represent MSDF specifications to terms that represent MRS theories.
- A front-end "plugs" '`convertMSOS`' into Full Maude. Full Maude's module algebra was extended to support MSDF modules.
- Removal of the '`Program`' and '`Component`' sorts from the '`RECORD`' theory in MRS: avoid collapsing all program and component sorts.

High-level view of the compilation process

From a MSDF module $\mathcal{M} = (D, L, T)$ to a system module $\mathcal{R} = (\Sigma, E, R)$

- Parsing user input (two-phase parsing with "bubbles");
- Extracting Σ from D;
- Solving bubbles with Σ;
- Compilation
  - Σ obtained from D (again);
  - E obtained from D;
  - R obtained from T.

More details

- implicit module inclusion: straightforward by looking up Full Maude's database;
- implicit metavariables: naive approach;
- source-dependency check: straightforward (after compilation is done), requires attention to Maude's "matching equations."

Compilation example

```
msos Id is
 Exp .
 Id .
 Exp ::= Id | Int .
 Env = (Id, Int) Map .
 Label = { r : Env, ... } .

    Int := lookup (Id, Env)
 -- ----------------------------
 Id : Id -{ r = Env, - }-> Int .
sosm
```

Compilation example

```
mod Id is
 including QID .
 including MSOS-RUNTIME .
 including SEQUENCE < Exp > .
 including SEQUENCE < Id > .
 including MAP < Id | Int > .
 including @@INTEGER .

 sorts Env ; Exp ; Id .
 subsort Id < Exp .
 subsort Int < Exp .
 subsort eSort(Map, Id | Int) < Env .
 subsort eSort(NeSeq, Id) < eSort(NeSeq, Exp) .
 subsort eSort(NeSeq, Int) < eSort(NeSeq, Exp) .
 subsort eSort(Seq, Id) < eSort(Seq, Exp) .
 subsort eSort(Seq, Int) < eSort(Seq, Exp) .
```

Compilation example

```
op <_,_> : Exp Record -> Conf [ctor] .
op _:::_ : Exp Qid -> Exp [none] .
op _=_ : [Index] [Env] -> [Field] [ctor prec(50)] .
op [_,_] : [Exp] [Record] -> [Conf] [ctor] .
op {_,_} : [Exp] [Record] -> [Conf] [ctor] .
op r : nil -> RO-Index [ctor] .
```

Compilation example

```
mb r = E : ROField [none] .

eq I = C, PR1 ; I = C, PR2 = I = C, (PR1 ; PR2) .
eq duplicated (I = C, I = C', PR) = true .
```

Compilation example

```
 vars Exp Exp' : Exp .    vars Id Id' : Id .
 var  - : PreRecord .      var  Env : Env .
 var  Int : Int

 crl < Exp ::: 'Exp, R > => < Exp' ::: 'Exp, R' >
 if  { Exp ::: 'Exp, R } => [ Exp' ::: 'Exp, R' ] .

 crl < Id ::: 'Id, R > => < Id' ::: 'Id, R' >
 if  { Id ::: 'Id, R } => [ Id' ::: 'Id, R' ] .

 crl { Id ::: 'Id, { r = Env, - } } =>
     [ Int ::: 'Id, { r = Env, - } ]
 if Int := lookup (Id, Env) .
endm
```

Incremental SOS

A set of minimal modules that define a language neutral construction each. Used on Mosses's lecture notes and MSOS Tool. We implemented 74 modules (800 LoC).

- ML to IMSOS (based on Mosses's lecture notes): 315 LoC of flex/bison;
- MiniJava to IMSOS: 1812 LoC of SableCC/Java. A non-trivial mapping to IMSOS.

Mini-Freja

Lazy functional language with a complete pattern matcher. Implemented to compare performance with LETOS and RML (to be discussed later). Two specifications:

- recursion via unfolding (233 LoC) — LETOS/RML
- recursion via fixed point operator (224 LoC)

Distributed algorithms

- Several algorithms from Lynch: bakery, dining philosophers (several variants), leader election on asynchronous ring, Peterson's shared variable concurrency, semaphores, thread Game: total 1737 LoC.
- Developed to assess the use of MMT in the context of distributed systems.

User interface

- parsing errors occurs at three levels: MSDF, Full Maude, and Maude: confusing;
- almost no static analysis of MSDF modules: can be frustrating for a new user;
- prettyprinting of analysis results currently only at Maude level;
- analysis modules must be defined at the Maude level;
- a more robust user interface at the MSOS domain for querying properties of specifications.

Implementation

- parsing MSDF specifications is hard with Maude's parser;
- removal of the '`Program`' and '`Component`' sorts;
- the step flag and strategies;
- implementation of automatic metavariables;
- preregularity and MSDF: should warn about preregular MSDF modules;
- long compilation time: due to the use of flat modules in Full Maude

Analysis of MSDF specifications

- cannot query about terms that happen on conditions; evaluation contexts may help?
- fairness problems in distributed algorithms;
- state explosion; use of abstractions?

Much has been done since the original (2001) version of the prototype. MMT's usability has been assessed empirically with several different specifications.

As for future work, we envision two different approaches:

- enhancing the tool: (i) much "Maude" bleeding into the MSOS domain; (ii) better performance: evaluation contexts and the work of Roșu and Meseguer?; partial evaluation?
- extending MMT: integration with available tools: Verdejo's Strategy Language; Palomino's abstraction generator; Clavel's ITP (when it supports rewrite rules)

Thank you! Obrigado!

Source-dependent and Maude's "matching equations".

```
vars x y z w : Foo .

crl f(x, y) => w
if w := g (x, y) .
```

Querying in conditions. In the main path of computation, ρ is always empty!

$$\frac{e_0 \xrightarrow{X} e_0'}{\texttt{let } x = e_0 \texttt{ in } e_1 \texttt{ end } \xrightarrow{X} \texttt{let } x = e_0' \texttt{ in } e_1 \texttt{ end}}$$

$$\frac{e_1 \xrightarrow{\{env = \rho[m/x], \ldots\}} e_1'}{\texttt{let } x = m \texttt{ in } e_1 \texttt{ end } \xrightarrow{\{env = \rho, \ldots\}} \texttt{let } x{=}m \texttt{ in } e_1' \texttt{ end}}$$

$$\texttt{let } x{=}m \texttt{ in } n \texttt{ end } \xrightarrow{U} n$$