Modular Structural Operational Semantics $\mapsto$ Modular Rewriting Semantics

Fabricio Chalub Barbosa do Rosário
frosario@ic.uff.br

October 15, 2003

Outline

- Structural Operational Semantics (SOS)
- Modular Structural Operational Semantics (MSOS)
- Translating MSOS rules to a "concrete" language
- Modular Rewriting Semantics (MRS)
- Implementing an interpreter for the MSOS "language"

SOS: Labelled Terminal Transition System

$$\langle \Gamma, A, \rightarrow, T \rangle$$

$$\begin{aligned}
\Gamma &= \{\langle \rho, e, \sigma \rangle\} \cup \{\langle \rho, c, \sigma \rangle\} \cup \{\langle \rho, d, \sigma \rangle\} \\
T \subseteq \Gamma &= \{\langle \rho, con, \sigma \rangle\} \cup \{\langle \rho, \textbf{nil}, \sigma \rangle\} \cup \{\langle \rho, \rho', \sigma \rangle\} \\
\rightarrow &\subseteq \Gamma \times A \times \Gamma \\
\langle \gamma, \alpha, \gamma' \rangle \in \rightarrow &= \gamma \xrightarrow{\alpha} \gamma'
\end{aligned}$$

"configurations are states of transitions systems, and computations consist of sequences of transitions between configurations"

SOS: environment

$$\frac{\langle \rho, e_0 \rangle \to \langle \rho, e_0' \rangle}{\langle \rho, e_0 \bullet e_1 \rangle \to \langle \rho, e_0' \bullet e_1 \rangle} \quad \equiv \quad \frac{\rho \vdash e_0 \to e_0'}{\rho \vdash e_0 \bullet e_1 \to e_0' \bullet e_1}$$

SOS: environment + store

$$\frac{\langle \rho, e_0, \sigma \rangle \longrightarrow \langle \rho, e_0', \sigma \rangle}{\langle \rho, e_0 \bullet e_1, \sigma \rangle \longrightarrow \langle \rho, e_0' \bullet e_1, \sigma \rangle} \quad \equiv \quad \frac{\rho \vdash \langle e_0, \sigma \rangle \longrightarrow \langle e_0', \sigma \rangle}{\rho \vdash \langle e_0 \bullet e_1, \sigma \rangle \longrightarrow \langle e_0' \bullet e_1, \sigma \rangle}$$

MSOS: Generalized Transition System

$$\langle \Gamma, \mathbb{A}, \rightarrow, T \rangle$$

$$\begin{aligned} \Gamma &= e \cup c \cup d \\ T \subseteq \Gamma &= com \cup \{\textbf{nil}\} \cup \rho \end{aligned}$$

$$\frac{v_1 \xrightarrow{u_1} v_1' \quad \cdots \quad v_n \xrightarrow{u_n} v_n' \quad C}{f(t_1, \ldots, t_n) \xrightarrow{u} t'}$$

MSOS: label components

- read-only (environments), read-write (stores), write-only (exceptions, traces, logging)

- being a ternary relation, read-write indices must be primed as indication of change

$$\frac{\sigma_1 = f(\sigma_0, t)}{t \xrightarrow{\{\sigma = \sigma_0, \sigma' = \sigma_1, \ldots\}} t'}$$

MSOS: concrete language

- MSOS has some implicit assumptions: which indices are RO, RW, WO; indices are functions in some $A \times B$ relation.

$$\frac{e \xrightarrow{\{\rho = \rho_1[\rho_0], \ldots\}} e'}{\textbf{let } \rho_0 \textbf{ in } e \textbf{ end} \xrightarrow{\{\rho = \rho_1, \ldots\}} \textbf{let } \rho_0 \textbf{ in } e' \textbf{ end}}$$

index ρ: $(i, v) \in Id \times DVal$, read-only

operation $\rho = \rho_n[\rho_m]$

index σ: $(l, v) \in Loc \times SVal$, read-write

MSOS concrete language: design questions

- do we need to declare what labels are in use (what about modulartiy?)

- how to implement the functionality of the components?

```
declare rho is BC-ENVIRONMENT
```

- how to relate the abstract interface with the expected functionality of the component?

$$\{\sigma = \sigma_0[l \mapsto v]\}$$

```
sigma = update (sigma0, l, v)
```

- idea: let the user specify the component directly in equational logic

MSOS concrete language: conditional rules

- how to describe MSOS rules (triples)?

$$\frac{v \xrightarrow{u} v'}{\mathrm{f(t)} \xrightarrow{u} \mathrm{t}'}$$

```
1) \RULE {v \OTRANS{u}{v'}} {f(t) \OTRANS{u}{t'}}

2) msos [f] : < f(t), u, t' > if < v, u, v' >

3) mr [f]   : { u } f(t) -> t' if { u } v -> v'
```

MSOS concrete language: language phrases

- how to specify language phrases?

$$\textbf{let } \rho_0 \textbf{ in } e \textbf{ end}$$

- idea: use Maude's algebraic capabilities:

```
sorts Decl Exp .
op let_in_end : Decl Exp -> Exp .
```

MSOS concrete language: preliminary conclusion

$$\mathcal{R} \equiv \langle \text{algebraic structure} \rangle + \langle \text{rewriting rules} \rangle$$

$$\mathcal{MSOS} \equiv \langle \text{algebraic structure*} \rangle + \langle \text{msos rules} \rangle$$

we may consider $\langle$record components$\rangle \subset \langle$algebraic structure*$\rangle$

MSOS concrete language: example

- two options:

1) declare labels and “bind” them to functional modules in Maude

2) let the user declare and use the modules to her will

MSOS concrete language: example

$$\frac{v = \rho_0(x)}{x \xrightarrow{\{\rho = \rho_0, \mathrm{PR}\}} v}$$

$$\frac{\sigma_1 = \sigma_0[x \mapsto v]}{x := v \xrightarrow{\{\sigma = \sigma_0, \sigma' = \sigma_1, \mathrm{PR}\}} \mathrm{noop}}$$

MSOS concrete language: example

- user is bound to the previously written components

```
op _:=_ : Exp Exp -> Exp .

declare rho read-only, sigma read-write .
declare rho is ['SML-ENVIRONMENT],
        sigma is ['SML-STORE] .

msos < x, { rho = rho0, ... }, v > if v := f (rho0, x) .

msos < x := v, { sigma = sigma0, sigma' = sigma1, ... },
       noop > if sigma1 := update (sigma0, x, v) .
```

MSOS concrete language: example

- user is free to define her own label components

```
op _:=_ : Exp Exp -> Exp .

msos < x, { rho = rho0, PR }, v > if v := lookup (rho0, x) .

msos < x := v, { sigma = sigma0, sigma' = sigma1, PR },
       noop > if sigma1 := update-store (sigma0, x, v) .
```

Modular Rewriting Semantics

$$\mathcal{R} = (\Sigma, E, \Phi, R)$$

- configuration

```
fmod PROGRAM-RECORD is
 op <_,_> = Program Record -> ProgramRecord [ctor] .
endfm
```

- record inheritance ({ `(st: sigma), (env: rho), PR` } a special case of { `PR` })

```
crl < f(t1,...,tn), u > => < t', u' > if C .
```

MRS: example

```
crl < e1 +' e2, { PR }> => < e'1 +' e2, { PR' }
    if < e1, { PR } > => < e'1, { PR' } > /\ e1 =/= e'1 .
```

$e_1 \neq e'_1$ added due to the **reflexivity** deduction rule

Mapping MSOS to MRS

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

$$\mathcal{R} \vdash \langle v, w \rangle \longrightarrow \langle v', w' \rangle$$

$$\langle v, w \rangle = \langle v_0, w_0 \rangle \longrightarrow \cdots \longrightarrow \langle v_{n-1}, w_{n-1} \rangle \longrightarrow \langle v_n, w_n \rangle = \langle v', w' \rangle$$

- equality

- nested replacement

Mapping MSOS to MRS: example

```
crl : < P, R > => < P', R' > if { P, R } => [ P', R' ] .
rl { 'a, 'b } => [ 'c, 'd ] .
rl { 'c, 'd } => [ 'e, 'f ] .

rew < 'a, 'b > .
```

$$\langle 'a, 'b \rangle \longrightarrow \langle 'c, 'd \rangle \qquad (\{'a, 'b\} \longrightarrow ['c, 'd])$$

$$\langle 'c, 'd \rangle \longrightarrow \langle 'e, 'f \rangle \qquad (\{'c, 'd\} \longrightarrow ['e, 'f])$$

$$\langle 'a, 'b \rangle \longrightarrow \langle 'e, 'f \rangle$$

Mapping MSOS to MRS

$$\frac{v_1 \xrightarrow{u_1} v_1' \quad \cdots \quad v_n \xrightarrow{u_n} v_n' \quad C}{f(t_1, \ldots, t_n) \xrightarrow{u} t'}$$

$$\{f(t_1, \ldots, t_n)\} \Rightarrow [t', u']$$

$$\mathsf{if}\,\{v_1, w_1\} \Rightarrow [v_1', w_1'] \wedge \ldots \wedge \{v_n, w_n\} \Rightarrow [v_n', w_n'] \wedge C$$

Orthogonal vs. non-orthogonal changes

orthogonal: expressions, abstractions, imperatives

non-orthogonal extensions: modified behaviour of stores and environments

Confluence

$(\text{MSOS} + \text{MRS}) \rightarrow$ Standard ML $\rightarrow$ Mini-Java