Modular SOS: Differences from SOS*

Peter D. Mosses

BRICS** & Department of Computer Science, University of Aarhus
Ny Munkegade bldg. 540, DK-8000 Aarhus C, Denmark
E-mail: pdmosses@brics.dk, Home page: http://www.brics.dk/~pdm

Abstract. Modular SOS (MSOS) is a simple variant of conventional Structural Operational Semantics (SOS). Using MSOS, the transition rules for each construct of a programming language can be given definitively, once and for all, and never need reformulation when further constructs are added to the language. MSOS thus provides an exceptionally high degree of modularity in language descriptions, thereby removing a shortcoming of the original SOS framework.
This succinct paper assumes familiarity with the main features of SOS. It explains and motivates the crucial differences between SOS and MSOS, and gives illustrative examples of MSOS rules, showing how straightforward it is to convert from SOS to MSOS. The illustrations include a novel and completely modular treatment of exception-handling.

1 Introduction

MSOS appears to solve the modularity problem for SOS as effectively as monad transformers do for denotational semantics. Moreover, although the foundations of MSOS involve concepts from Category Theory, MSOS descriptions can be understood just as easily as ordinary SOS, and MSOS has been class-tested successfully at Aarhus in undergraduate courses.

Previous papers have presented the foundations of MSOS [1], discussed its pragmatic aspects [5], and demonstrated its usefulness in modular operational descriptions of action notation [2] and the core of Concurrent ML [4]. The present paper focuses on the differences between SOS and MSOS, giving illustrations of MSOS that can be compared directly with the corresponding SOS descriptions in Plotkin's Aarhus notes [6].

Readers are assumed to be familiar with the basic notions of operational semantics, and with the standard conceptual analysis of common constructs of high-level programming languages. The notation used here generally follows Plotkin (op. cit.) regarding common features of SOS and MSOS, to facilitate comparison, although this gives rise to some stylistic differences from the notation used in the previous papers on MSOS.

* Presented at the First APPSEM-II Workshop, Nottingham, March 2003
** Basic Research in Computer Science (http://www.brics.dk), funded by the Danish National Research Foundation

2 The Main Differences

Semantic descriptions in both SOS and MSOS involve specification of a labelled transition system LTS, which we take to be a quadruple $\langle \Gamma, A, \longrightarrow, T \rangle$ consisting of a set Γ of configurations γ, a set A of labels α, a ternary relation $\longrightarrow \subseteq \Gamma \times A \times \Gamma$ (written $\gamma \xrightarrow{\alpha} \gamma'$) of transition with label α between configurations γ, γ', and a set $T \subseteq \Gamma$ of terminal configurations, such that $\gamma \longrightarrow \gamma'$ implies $\gamma \notin T$. A computation in LTS from γ_0 is a finite or infinite sequence of successive transitions $\gamma_i \xrightarrow{\alpha_i} \gamma_{i+1}$ (written $\gamma_0 \xrightarrow{\alpha_1} \gamma_1 \xrightarrow{\alpha_2} \cdots$), such that when the sequence terminates with γ_n we have $\gamma_n \in T$.

Let's consider the main differences between SOS and MSOS in the various parts of semantic descriptions: abstract syntax, the sets of computed values, configurations, labels, and the transition rules. Readers who are interested in the description of process algebras, rather than conventional programming languages, may prefer to skip to Section 2.4.

2.1 Abstract Syntax

There are no essential differences between SOS and MSOS regarding abstract syntax. For the illustrations of MSOS in this paper, we'll use the same notation for syntactic sets and constructors as in Plotkin's notes, to facilitate direct comparison. For example, abstract syntax for a conditional command is specified thus:

$$\begin{array}{ll} \text{Expressions} & e \in \mathit{Exp} \\ \text{Commands} & c \in \mathit{Com} \\ & \qquad c ::= \mathbf{if}\, e\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1 \end{array}$$

When giving truly definitive descriptions of common constructs in MSOS, however, it is preferable to avoid bias toward the concrete syntax of particular languages, and use a neutral notation for constructors, such as $c ::= \mathit{cond}(e, c_0, c_1)$. Moreover, it is necessary to develop a standard nomenclature for both syntactic sets and common constructors (not switching between *Com* and *Stm* in descriptions of different languages, for instance).

2.2 Computed Values

In SOS the set of values computed by expressions might be numbers $\mathbb{N}$ and truth-values $\mathbb{T}$. For declarations, it would be *Env*, the set of environments. However, commands aren't usually regarded as computing any values at all.

MSOS requires a set of computed values to be specified for each sort of syntax, so as to give a systematic treatment of configurations (see below). Thus commands are regarded as computing a fixed, null value—as in the monadic variant of denotational semantics.

2.3 Configurations

One of the main differences between SOS and MSOS concerns the configurations $\gamma \in \Gamma$. In SOS descriptions of programming languages, configurations typically involve not only abstract syntax and computed values,[1] but also auxiliary entities such as environments and stores. However, such auxiliary entities are generally not used in SOS descriptions of process algebras.

In MSOS, configurations are always simply abstract syntax and computed values. Any auxiliary entities that are needed have to be incorporated in the labels.

2.4 Labels

In SOS, labels on transitions are optional. When absent, the set of labels may be assumed to be a singleton, or one may regard the SOS as specifying an unlabelled transition system. A label α on a transition $\gamma \xrightarrow{\alpha} \gamma'$ usually represents only the information relevant for interactions between separate processes, e.g. communication possibilities on channels.

In MSOS, labels are obligatory. A label α on a transition represents all the information associated with that transition, including the current bindings, and the state of the store both before and after the transition, as well as any interaction possibilities.

In SOS, the set of labels has little or no algebraic structure, whereas in MSOS, the set of labels is the set of morphisms of a category, and thus equipped with a composition operation (regarded as a partial operation applicable to arbitrary labels, the result being undefined when the labels aren't composable) and with a distinguished subset of identity morphisms. In fact the category of labels is generally a product category, and notation is provided for accessing and changing particular components independently of the presence of other components.

In SOS, traces of computations are arbitrary sequences of labels. So as to be able to define weak semantic equivalences that ignore unobservable transitions, a special label, usually written τ, is added to the set of labels.

In MSOS, in contrast, the labels on successive transitions in a computation are required to trace a path through the label category, i.e. adjacent labels must be composable. Moreover, transitions are regarded as unobservable precisely when their labels are identity morphisms.

2.5 Rules

Transition rules are specified in exactly the same way in SOS and MSOS. In SOS, however, rules may need to be (systematically) reformulated, perhaps several times, when a described programming language is extended with further constructs. In MSOS, the transition rules for a particular construct are given definitively, once and for all, and *never* need reformulation.

[1] Abstract syntax trees where some nodes have been replaced by their computed values are needed for intermediate configurations.

The appearance of rules in SOS and MSOS descriptions of programming languages is often quite different, since transitions in SOS may well involve 5 or more components, e.g. $\rho \vdash \langle \gamma, \sigma \rangle \longrightarrow \langle \gamma, \sigma \rangle$, whereas in MSOS, transitions are of the form $\gamma \xrightarrow{\alpha} \gamma'$ (written $\gamma \ -\alpha\rightarrow \ \gamma'$ below for notational convenience) and particular components of α are mentioned only when needed in the individual rules. Thus rules in MSOS descriptions of programming languages have comparable simplicity to those in SOS (or MSOS) descriptions of process algebras.

Finally, when preparing an SOS for extension of the described language with further constructs, it isn't immediate that the reformulations are conservative regarding the computations (and equivalences) for the original constructs. In MSOS, it can be proved that adding a new component to labels is indeed a conservative extension. Of course, this doesn't say anything about what might happen when the further constructs and their rules are subsequently added.

3 Essential Notation for MSOS

The notation used for labels in the illustrative examples below is a special case of that used for record patterns in Standard ML. For instance:

- $\{\rho{=}\rho_0, \ldots\}$ specifies labels whose ρ-component is ρ_0;
- $\{\sigma{=}\sigma_0, \sigma'{=}\sigma_1, \ldots\}$ specifies labels where the σ-component is σ_0, and the σ'-component is σ_1;
- $\{\sigma, \ldots\}$ abbreviates $\{\sigma{=}\sigma, \ldots\}$, letting the variable σ refer to the σ-component of the label.

The explicit '...' in the notation above is obligatory, and ensures that unmentioned components of labels are not excluded. Different occurrences of '...' in the same transition rule stand for the same set of unmentioned components. The order in which components are written is, of course, insignificant.

The set of labels is specified by declaring the indices (such as ρ, σ, σ') used for referring to components, together with the corresponding sets for the comoponents themselves. The meta-variable X ranges over arbitrary labels, whereas U is restricted to labels that are identity morphisms. We'll generally write $\gamma \ -U\rightarrow \ \gamma'$ as $\gamma \longrightarrow \gamma'$ when U isn't needed elsewhere in the rule.

Label composition is written $X_1 \,;\, X_2$, and its result is defined only when X_1 and X_2 have identical sets of indices for components. For a fixed set I of indices, call an index *read-only* if it occurs in I only without a prime '′', *read-write* if it occurs both with and without primes, and *write-only* if it occurs only with a prime.

Then $X_1;X_2$ is defined iff each read-only component is the same in X_1 and X_2, and each primed read-write component of X_1 is the same as the corresponding unprimed component of X_2. The sets for the write-only components are assumed to be monoids, and each write-only component of $X_1 \,;\, X_2$ is obtained by applying the monoid composition to the respective components of X_1 and X_2. Moreover, X is an identity morphism iff each primed read-write component is the same as

the corresponding unprimed component, and each write-only component is the unit of its monoid. This determines a category from the set of labels.

Regarding notation for finite mappings such as environments and stores, let's follow Plotkin's notes, using $\{x{=}y\}$ for a singleton mapping x to y, $\mu_0[\mu_1]$ for the mapping where μ_1 overrides μ_0 (writing just $\mu_0[x{=}y]$ when μ_1 is the singleton $\{x{=}y\}$), and $\mu_0 \dot{\cup} \mu_1$ for the union of mappings with disjoint domains.

4 Illustrative Examples in MSOS

The examples given below are the definitive MSOS rules (modulo notation for abstract syntax) for many of the simpler constructs covered in Plotkin's notes. The only exception is the command **recover** c, which is added so as to illustrate a novel technique (discovered by Klin, a PhD student at Aarhus) that allows a completely modular treatment of exception-handling.

Perhaps the best test of MSOS is whether the reader can understand the transition rules below as easily as the corresponding SOS rules given by Plotkin. Rather than giving further explanations, let's leave the rules to speak for themselves.

4.1 Abstract Syntax

Truth-values	$t \in \mathbb{T}$	$= \{\mathbf{tt}, \mathbf{ff}\}$
Numbers	$n \in \mathbb{N}$	$= \{0, 1, 2, \ldots\}$
Variables	$x \in \mathit{Var}$	$= \{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \ldots\}$
Binary Ops.	$\mathit{bop} \in \mathit{Bop}$	$= \{+, -, *, \ldots\}$
Constants	$\mathit{con} \in \mathit{Con}$	
	con	$::= t \mid n$
Expressions	$e \in \mathit{Exp}$	
	e	$::= \mathit{con} \mid x \mid e_0\ \mathit{bop}\ e_1 \mid {\sim} e \mid \mathbf{let}\, d\, \mathbf{in}\, e$
Commands	$c \in \mathit{Com}$	
	c	$::= \mathbf{nil} \mid x := e \mid c_0\,;c_1 \mid d\,;c \mid$
		$\mathbf{if}\, e\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1 \mid \mathbf{while}\, e\, \mathbf{do}\, c \mid$
		$\mathbf{recover}\, c$
Declarations	$d \in \mathit{Dec}$	
	d	$::= \mathbf{nil} \mid \mathbf{const}\, x = e \mid \mathbf{var}\, x := e \mid$
		$d_0\,;d_1 \mid d_0\, \mathbf{and}\, d_1 \mid d_0\, \mathbf{in}\, d_1$

4.2 Computed Values

$$\begin{array}{ll} \text{Computed Values} & T = \mathbb{N} \cup \mathbb{T} \cup \{\mathbf{nil}\} \cup \mathit{Env} \\ \text{Environments} & \rho \in \mathit{Env} = \mathit{Var} \rightarrow_f \mathit{BV} \end{array}$$

4.3 Configurations

$$\begin{array}{ll} \text{Expressions} & e \in \mathit{Exp} \\ & \quad e ::= \mathit{error} \\ \text{Declarations} & d \in \mathit{Dec} \\ & \quad d ::= \rho \\ \text{Configurations} & \Gamma \;=\; \mathit{Exp} \cup \mathit{Com} \cup \mathit{Dec} \end{array}$$

4.4 Label Components

$$\begin{array}{r} \rho : \mathit{Env} \\ \sigma, \sigma' : S \\ \varepsilon' : \{\mathit{err}\}^* \end{array}$$

4.5 Expression Rules

$$\frac{U = \{\rho, \ldots\}, \quad \rho(x) = \mathit{con}}{x \;\; -U\rightarrow \;\; \mathit{con}} \tag{1}$$

$$\frac{U = \{\rho, \sigma, \ldots\}, \quad \rho(x) = l, \quad \sigma(l) = \mathit{con}}{x \;\; -U\rightarrow \;\; \mathit{con}} \tag{2}$$

$$\frac{e_0 \;\; -X\rightarrow \;\; e_0'}{e_0 \;\; \mathit{bop} \;\; e_1 \;\; -X\rightarrow \;\; e_0' \;\; \mathit{bop} \;\; e_1} \tag{3}$$

$$\frac{e_1 \;\; -X\rightarrow \;\; e_1'}{\mathit{con}_0 \;\; \mathit{bop} \;\; e_1 \;\; -X\rightarrow \;\; \mathit{con}_0 \;\; \mathit{bop} \;\; e_1'} \tag{4}$$

$$\frac{\mathit{bop} = +, \quad n = n_0 + n_1}{n_0 \;\; \mathit{bop} \;\; n_1 \longrightarrow n} \tag{5}$$

$$\frac{\mathit{bop} = -, \quad n_0 < n_1, \quad U = \{\varepsilon, \ldots\}}{n_0 \;\; \mathit{bop} \;\; n_1 \;\; -\{\varepsilon{=}\mathit{err}, \ldots\}\rightarrow \;\; \mathit{error}} \tag{6}$$

$$\frac{e \;\; -X\rightarrow \;\; e'}{\sim e \;\; -X\rightarrow \;\; \sim e'} \tag{7}$$

$$\frac{t' = \neg t}{\sim t \longrightarrow t'} \tag{8}$$

$$\frac{d \ -X\rightarrow \ d'}{\mathbf{let}\, d\, \mathbf{in}\, e \ -X\rightarrow \ \mathbf{let}\, d'\, \mathbf{in}\, e} \tag{9}$$

$$\frac{e \ -\{\rho{=}\rho[\rho_0],\ldots\}\rightarrow \ e'}{\mathbf{let}\, \rho_0\, \mathbf{in}\, e \ -\{\rho,\ldots\}\rightarrow \ \mathbf{let}\, \rho_0\, \mathbf{in}\, e} \tag{10}$$

$$\mathbf{let}\, \rho_0\, \mathbf{in}\, con \longrightarrow con \tag{11}$$

4.6 Command Rules

$$\frac{e \ -X\rightarrow \ e'}{x := e \ -X\rightarrow \ x := e'} \tag{12}$$

$$\frac{U = \{\rho, \sigma, \sigma', \ldots\}, \quad \rho(x) = l}{x := con \ -\{\rho, \sigma, \sigma'{=}\sigma[l{=}con], \ldots\}\rightarrow \ \mathbf{nil}} \tag{13}$$

$$\frac{c_0 \ -X\rightarrow \ c_0'}{c_0\,;c_1 \ -X\rightarrow \ c_0'\,;c_1} \tag{14}$$

$$\mathbf{nil}\,;c_1 \longrightarrow c_1 \tag{15}$$

$$\frac{d \ -X\rightarrow \ d'}{d\,;c \ -X\rightarrow \ d'\,;c} \tag{16}$$

$$\frac{c \ -\{\rho{=}\rho[\rho_0],\ldots\}\rightarrow \ c'}{\rho_0\,;c \ -\{\rho,\ldots\}\rightarrow \ \rho_0\,;c'} \tag{17}$$

$$\rho_0\,;\mathbf{nil} \longrightarrow \mathbf{nil} \tag{18}$$

$$\frac{e \ -X\rightarrow \ e'}{\mathbf{if}\, e\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1 \ -X\rightarrow \ \mathbf{if}\, e\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1} \tag{19}$$

$$\mathbf{if}\, \mathbf{tt}\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1 \longrightarrow c_0 \tag{20}$$

$$\mathbf{if}\, \mathbf{ff}\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1 \longrightarrow c_1 \tag{21}$$

$$\mathbf{while}\, e\, \mathbf{do}\, c \longrightarrow \mathbf{if}\, e\, \mathbf{then}\, c\,;\mathbf{while}\, e\, \mathbf{do}\, c\, \mathbf{else}\, \mathbf{nil} \tag{22}$$

$$\frac{c \ -\{\varepsilon{=}(\,),\ldots\}\rightarrow \ c'}{\mathbf{recover}\, c \ -\{\varepsilon{=}(\,),\ldots\}\rightarrow \ \mathbf{recover}\, c'} \tag{23}$$

$$\frac{c \ -\{\varepsilon{=}err,\ldots\}\rightarrow \ c'}{\mathbf{recover}\, c \ -\{\varepsilon{=}(\,),\ldots\}\rightarrow \ \mathbf{nil}} \tag{24}$$

$$\mathbf{recover}\, \mathbf{nil} \longrightarrow \mathbf{nil} \tag{25}$$

4.7 Declaration Rules

$$\mathbf{nil} \longrightarrow \emptyset \tag{26}$$

$$\frac{e \ -X\rightarrow \ e'}{\mathbf{const}\, x = e \ -X\rightarrow \ \mathbf{const}\, x = e'} \tag{27}$$

$$\frac{\rho = \{x{=}con\}}{\mathbf{const}\, x = con \longrightarrow \rho} \tag{28}$$

$$\frac{e \ -X\rightarrow \ e'}{\mathbf{var}\, x := e \ -X\rightarrow \ \mathbf{var}\, x := e'} \tag{29}$$

$$\frac{U = \{\sigma, \sigma', \ldots\}, \quad l \notin \mathrm{dom}(\sigma), \quad \rho = \{x{=}l\}}{\mathbf{var}\, x := con \ -\{\sigma, \sigma'{=}\sigma[l{=}con], \ldots\}\rightarrow \ \rho} \tag{30}$$

$$\frac{d_0 \ -X\rightarrow \ d_0'}{d_0\,;d_1 \ -X\rightarrow \ d_0'\,;d_1} \tag{31}$$

$$\frac{d_1 \ -\{\rho{=}\rho[\rho_0], \ldots\}\rightarrow \ d_1'}{\rho_0\,;d_1 \ -\{\rho, \ldots\}\rightarrow \ \rho_0\,;d_1'} \tag{32}$$

$$\rho_0\,;\rho_1 \longrightarrow \rho_0[\rho_1] \tag{33}$$

$$\frac{d_0 \ -X\rightarrow \ d_0'}{d_0\,\mathbf{and}\,d_1 \ -X\rightarrow \ d_0'\,\mathbf{and}\,d_1} \tag{34}$$

$$\frac{d_1 \ -X\rightarrow \ d_1'}{\rho_0\,\mathbf{and}\,d_1 \ -X\rightarrow \ \rho_0\,\mathbf{and}\,d_1'} \tag{35}$$

$$\frac{\rho = \rho_0 \,\dot{\cup}\, \rho_1}{\rho_0\,\mathbf{and}\,\rho_1 \longrightarrow \rho} \tag{36}$$

$$\frac{d_0 \ -X\rightarrow \ d_0'}{d_0\,\mathbf{in}\,d_1 \ -X\rightarrow \ d_0'\,\mathbf{in}\,d_1} \tag{37}$$

$$\frac{d_1 \ -\{\rho{=}\rho[\rho_0], \ldots\}\rightarrow \ d_1'}{\rho_0\,\mathbf{in}\,d_1 \ -\{\rho, \ldots\}\rightarrow \ \rho_0\,\mathbf{in}\,d_1'} \tag{38}$$

$$\rho_0\,\mathbf{in}\,\rho_1 \longrightarrow \rho_1 \tag{39}$$

References

1. P. D. Mosses. Foundations of Modular SOS (extended abstract). In *MFCS'99*, volume 1672 of *LNCS*, pages 70–80. Springer-Verlag, 1999. Full version available at `http://www.brics.dk/RS/99/54/`.
2. P. D. Mosses. A modular SOS for Action Notation. Research Series RS-99-56, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1999. `http://www.brics.dk/RS/99/56`. Full version of [3].
3. P. D. Mosses. A modular SOS for Action Notation (extended abstract). In *AS'99*, number NS-99-3 in BRICS Notes Series, pages 131–142, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1999. Full version available at `http://www.brics.dk/RS/99/56/`.
4. P. D. Mosses. A modular SOS for ML concurrency primitives. Research Series RS-99-57, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1999. `http://www.brics.dk/RS/99/57/`.
5. P. D. Mosses. Pragmatics of modular SOS. In *AMAST'02, Proc. 9th International Conference on Algebraic Methods and Software Technology, Reunion Island, France, Sept. 2002, Proceedings*, volume 2422 of *LNCS*, pages 21–40. Springer-Verlag, 2002.
6. G. D. Plotkin. A structural approach to operational semantics. Lecture Notes DAIMI FN–19, Dept. of Computer Science, Univ. of Aarhus, 1981.