Structural Operational Semantics

> [Specifications of operational semantics via abstract machines] "have a tendancy to pull the syntax to pieces or at any rate to wander around the syntax creating various complex symbolic structures which do not seem particularly forced by the demands of the language itself"
>
> *Gordon Plotkin*, "A Structural Approach to Operational Semantics" (1981)

Structural Operational Semantics

[Specifications of operational semantics via abstract machines] "have a tendancy to pull the syntax to pieces or at any rate to wander around the syntax creating various complex symbolic structures which do not seem particularly forced by the demands of the language itself"

Gordon Plotkin, "A Structural Approach to Operational Semantics" (1981)

popularised use of rule-based inductive definitions (of various kinds of relation), where the rules are "syntax-directed"

Structural Operational Semantics

Contrasting, but related styles of SOS:

- Milner, Kahn: evaluation relations (“big-step” SOS)
- Plotkin: transition relations (“small step” SOS)
- Felleisen: transitions using evaluation-contexts (“frame stacks”)

Structural Operational Semantics

Contrasting, but related styles of SOS:

- Milner, Kahn: evaluation relations ("big-step" SOS)
- Plotkin: transition relations ("small step" SOS)
- Felleisen: transitions using evaluation-contexts ("frame stacks")

We will use a fragment of ML to illustrate this [OS&PE, Appendix A] — it features:
recursively defined, higher-order, call-by-value functions
+ dynamically created mutable state.

Syntax of types (A1) & expressions (A2)

Types

$ty ::=$	bool	booleans
	int	integers
	unit	unit
	int ref	integer storage locations
	$ty * ty$	pairs
	$ty \to ty$	functions

Syntax of types (A1) & expressions (A2)

Expressions $e ::=$

$x \quad f$ value identifiers ($x, f \in Var$)

true false booleans

if e then e else e conditional

n integer constant ($n \in \mathbb{Z}$)

$e \; op \; e$ arithmetic ($op \in \{=, +, -, ...\}$)

$()$ unit value

e, e pair

fst e snd e projections

fun $(x : ty) \to e$ function

fun $f = (x : ty) \to e$ recursive fn

$e \; e$ function application

let $x = e$ in e local definition

$!e$ look-up

$e := e$ assignment

ref e storage creation

$e == e$ location equality

ℓ storage locations ($\ell \in Loc$)

$e ; e$ sequencing

Syntax of types (A1) & expressions (A2)

- abstract syntax trees modulo α-equivalence
  — binding forms & free variables

$$\boxed{\begin{array}{rcl} fv(\,\mathrm{fun}(x:ty)\to e) & \triangleq & fv(e) - \{x\} \\ fv(\mathrm{fun}\ f = (x:ty)\to e) & \triangleq & fv(e) - \{f, x\} \\ fv(\mathrm{let}\ x = e\ \mathrm{in}\ e'\ ) & \triangleq & fv(e) \cup (fv(e') - \{x\}) \end{array}}$$

- environment-free formulation
  — so storage locations (aka "addresses") occur explicitly in expressions

$$\boxed{loc(e) \triangleq \text{locations occurring in } e}$$

Evaluation to canonical form

Canonical forms $\subseteq$ **expressions** :

$$
\begin{array}{lll}
v ::= & x \quad f & (x, f \in Var) \\
& \mathrm{true} & \\
& \mathrm{false} & \\
& n & (n \in \mathbb{Z}) \\
& () & \\
& v, v & \\
& \mathrm{fun}\ (x : ty) \to e & \\
& \mathrm{fun}\ f = (x : ty) \to e & \\
& \ell & (\ell \in Loc)
\end{array}
$$

ML Evaluation Semantics (simplified, environment-free form)

Evaluation relation

$$\boxed{s, e \Rightarrow v, s'}$$

$$\begin{cases} s & = \text{initial state} \\ e & = \text{closed expression to be evaluated} \\ v & = \text{resulting closed canonical form} \\ s' & = \text{final state} \end{cases}$$

is inductively generated by rules *following the structure* of e, for example:

$$\frac{s, e_1 \Rightarrow v_1, s' \quad s', e_2[v_1/x] \Rightarrow v_2, s''}{s, \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Rightarrow v_2, s''}$$

Evaluation semantics is also known as *big-step* (anon), *natural* (Kahn 1987), or *relational* (Milner) semantics.

finite function from locations to integers

ML Evaluation Semantics (simplified, environment-free form)

$fv(e) = \emptyset = fv(v)$

Evaluation relation

$$\boxed{s, e \Rightarrow v, s'}$$

$$\begin{cases} s & = \text{initial state} \\ e & = \text{closed expression to be evaluated} \\ v & = \text{resulting closed canonical form} \\ s' & = \text{final state} \end{cases}$$

is inductively generated by rules *following the structure* of e, for example:

$$\frac{s, e_1 \Rightarrow v_1, s' \quad s', e_2[v_1/x] \Rightarrow v_2, s''}{s, \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Rightarrow v_2, s''}$$

Evaluation semantics is also known as *big-step* (anon), *natural* (Kahn 1987), or *relational* (Milner) semantics.

invariant: $loc(e) \subseteq dom(s)$ & $loc(v) \subseteq dom(s')$

look-up:

$$\frac{s, e \Rightarrow \ell, s' \qquad (\ell \mapsto n) \in s'}{s, !e \Rightarrow n, s'}$$

assignment:

$$\frac{s, e_1 \Rightarrow \ell, s' \qquad s', e_2 \Rightarrow n, s''}{s, e_1 := e_2 \Rightarrow (), s''[\ell \mapsto n]}$$

storage creation:

$$\frac{s, e \Rightarrow n, s' \qquad \ell \notin dom(s')}{s, \text{ref}\, e \Rightarrow \ell, s'[\ell \mapsto n]}$$

function application (I) :

$$\frac{s, e_1 \Rightarrow v_1, s' \quad s', e_2 \Rightarrow v_2, s'' \quad v_1 = \text{fun}\ (x : ty) \to e \quad s'', e[v_2/x] \Rightarrow v_3, s'''}{s, e_1\, e_2 \Rightarrow v_3\ s'''}$$

function application (II):

$$\frac{s, e_1 \Rightarrow v_1, s' \qquad s', e_2 \Rightarrow v_2, s'' \qquad v_1 = \text{fun } f = (x : ty) \to e \qquad s'', e[v_1/f, v_2/x] \Rightarrow v_3, s'''}{s, e_1\, e_2 \Rightarrow v_3\ s'''}$$

E.g. for $\text{fact} \triangleq \text{fun } f = (x:\text{int}) \to \text{if } x=0 \text{ then } 1 \text{ else } x * f(x-1)$

have:

$$s, \text{fact } 3 \Rightarrow 6, s$$

'cos $s, \text{if } 3=0 \text{ then } 1 \text{ else } 3 * \text{fact}(3-1) \Rightarrow 6, s$

'cos $s,\ 3 * \text{fact}(3-1) \Rightarrow 6, s$

'cos ... (etc)

Properties of evaluation relation

- if $s, e \Rightarrow v, s'$, then $\mathrm{dom}(s) \subseteq \mathrm{dom}(s')$
- [essentially] deterministic:
  if $s, e \Rightarrow v_1, s_1$ and $s, e \Rightarrow v_2, s_2$,
  then v_1, s_1 and v_2, s_2 only differ up to permutation of the freshly created locations,
  i.e. there is a permutation $\pi : \mathrm{Loc} \cong \mathrm{Loc}$ fixing $\mathrm{dom}(s)$ [$\pi(\ell) = \ell$ for $\ell \in \mathrm{dom}(s)$]
  and with

$$\pi \cdot v_1 = v_2 \qquad \pi \cdot s_1 = s_2$$

[proof: . . .]

Proof strategy for showing $\Rightarrow$ is deterministic:

Consider

$$H \triangleq \{(s, e, v_1, s_1) \mid s, e \Rightarrow v_1, s_1 \ \& \ \forall v_2, s_2.\ s, e \Rightarrow v_2, s_2 \supset \exists \pi \in Perm(Loc).\ \forall \ell \in dom(s).\ \pi(\ell) = \ell \ \& \ \pi \cdot v_1 = v_2 \ \& \ \pi \cdot s_1 = s_2\}$$

and show ~~that~~ H is closed under the axioms & rules inductively defining $\Rightarrow$.

(E.g. show

$$(s, e, n_1, s_1) \in H \ \& \ \ell_1 \notin dom(s) \supset (s, ref\,e, \ell_1, s[\ell_1 \mapsto n_1]) \in H$$

etc.)

Properties of evaluation relation

- non-termination : given s, e (with $loc(e) \subseteq dom(s)$), there can be <u>no</u> v, s' with $s, e \Rightarrow v, s'$.

The non-termination of

$(\mathrm{fun}\ f = (x:\mathrm{unit}) \rightarrow f\ x)()$ [divergence]

is qualitatively different from that for

$3()$ [type error]

— we use a type system to statically check for the latter kind of non-termination...