Operational Semantics Using the Partiality Monad

Nils Anders Danielsson (Nottingham)

AIM XI, 2010-03-24

Introduction

Operational semantics are often specified as *relations*:

- Small-step.
- Big-step.

This talk:

- Operational semantics as total functions.
- Using the partiality monad.
- Small-step or big-step.

A language which allows loops and crashes

$$
\begin{aligned}
&\textbf{data}\ \mathit{Tm}\,(n : \mathbb{N}) : \mathit{Set}\ \textbf{where}\\
&\quad \mathsf{con} : \mathbb{N} \to \mathit{Tm}\ n\\
&\quad \mathsf{var} : \mathit{Fin}\ n \to \mathit{Tm}\ n\\
&\quad \lambda : \mathit{Tm}\,(\mathit{suc}\ n) \to \mathit{Tm}\ n\\
&\quad \_\cdot\_ : \mathit{Tm}\ n \to \mathit{Tm}\ n \to \mathit{Tm}\ n
\end{aligned}
$$

Values

Closures:

```
mutual
  data Value : Set where
    con : ℕ → Value
    λ   : ∀ {n} → Tm (suc n) → Env n → Value
  Env : ℕ → Set
  Env n = Vec Value n
```

Relational, big-step semantics

$$
\begin{array}{l}
\textbf{data}\ \_\vdash\_\Downarrow\_\ \{n\}\ (\rho\ :\ \mathit{Env}\ n)\ :\\
\qquad \mathit{Tm}\ n\ \to\ \mathit{Value}\ \to\ \mathit{Set}\ \textbf{where}\\
\quad \mathsf{var}\ :\ \rho \vdash \mathsf{var}\ x\ \Downarrow\ \mathit{lookup}\ x\ \rho\\
\quad \mathsf{con}\ :\ \rho \vdash \mathsf{con}\ i\ \Downarrow\ \mathsf{con}\ i\\
\quad \lambda\ :\ \rho \vdash \lambda\ t\ \Downarrow\ \lambda\ t\ \rho\\
\quad \mathsf{app}\ :\ \rho \vdash t_1\ \Downarrow\ \lambda\ t\ \rho'\ \to\ \rho \vdash t_2\ \Downarrow\ v'\ \to\\
\qquad\quad v' :: \rho' \vdash t\ \Downarrow\ v\ \to\ \rho \vdash t_1 \cdot t_2\ \Downarrow\ v
\end{array}
$$

Relational, big-step semantics

$$\textbf{data}\ \_\vdash\_\Uparrow\ \{n\}\ (\rho : \mathit{Env}\ n) : \mathit{Tm}\ n \to \mathit{Set}\ \textbf{where}$$

$$\begin{aligned}
\mathsf{app}^{\mathrm{l}} &: \infty\,(\rho \vdash t_1 \Uparrow) \to \rho \vdash t_1 \cdot t_2 \Uparrow \\
\mathsf{app}^{\mathrm{r}} &: \rho \vdash t_1 \Downarrow v \to \infty\,(\rho \vdash t_2 \Uparrow) \to \\
&\quad\ \rho \vdash t_1 \cdot t_2 \Uparrow \\
\mathsf{app} &: \rho \vdash t_1 \Downarrow \lambda\ t\ \rho' \to \rho \vdash t_2 \Downarrow v' \to \\
&\quad\ \infty\,(v' :: \rho' \vdash t \Uparrow) \to \rho \vdash t_1 \cdot t_2 \Uparrow
\end{aligned}$$

$$\_\vdash\_\lightning : \forall\ \{n\} \to \mathit{Env}\ n \to \mathit{Tm}\ n \to \mathit{Set}$$

$$\rho \vdash t \lightning = (\nexists\ \lambda\ v \to \rho \vdash t \Downarrow v) \times \neg\,(\rho \vdash t \Uparrow)$$

Relational, big-step semantics

$$
\begin{array}{llllllll}
\_\vdash\_\Downarrow\_ & : & Env\ n & \to & Tm\ n & \to & Value & \to & Set \\
\_\vdash\_\Uparrow & : & Env\ n & \to & Tm\ n & & & \to & Set \\
\_\vdash\_\lightning & : & Env\ n & \to & Tm\ n & & & \to & Set
\end{array}
$$

- Code duplication.
- Risk of forgetting rules.
- Deterministic?
- Executable?
- Awkward interface:

$$
\begin{array}{l}
exec : \forall\ \rho\ t \ \to\ (\exists\ \lambda\ v\ \to\ \rho \vdash t \Downarrow v)\ \uplus \\
\qquad \rho \vdash t \Uparrow\ \uplus\ \rho \vdash t\ \lightning
\end{array}
$$

Outline

- Codata.
- Partiality monad.
- Semantics using the partiality monad.
- Virtual machine with small-step semantics.
- Compiler correctness.

Codata

Codata

$\infty\, A$: Suspended computations of type A.

$\sharp\_$: Suspends a computation.

$\flat$: Forces a suspended computation.

$$
\begin{array}{llcll}
\sharp\_ & : \{A : Set\} & \rightarrow & A & \rightarrow \infty\, A \\
\flat & : \{A : Set\} & \rightarrow & \infty\, A & \rightarrow A
\end{array}
$$

Partiality monad

Partiality monad

$A_\perp \approx \nu X.\, A + X$:

```
data _⊥ (A : Set) : Set where
  now   : A → A ⊥
  later : ∞ (A ⊥) → A ⊥
```

```
never : ∀ {A} → A ⊥
never = later (♯ never)
```

```
_>>=_ : ∀ {A B} → A ⊥ → (A → B ⊥) → B ⊥
now x   >>= f = f x
later x >>= f = later (♯ (♭ x >>= f))
```

Equality

Weak bisimilarity:

$$
\begin{array}{llrll}
\multicolumn{5}{l}{\textbf{data}\ \_\approx\_\ \{A : \mathit{Set}\} : A_{\perp} \to A_{\perp} \to \mathit{Set}\ \textbf{where}} \\
\text{now} & : & & \to & \text{now}\ v \approx \text{now}\ v \\
\text{later} & : & \infty\,(\flat\, x \approx \flat\, y) & \to & \text{later}\ x \approx \text{later}\ y \\
\text{later}^{\mathrm{r}} & : & x \approx \flat\, y & \to & x \approx \text{later}\ y \\
\text{later}^{\mathrm{l}} & : & \flat\, x \approx y & \to & \text{later}\ x \approx y
\end{array}
$$

$\_\gg\!=\_$ preserves equality.

Functional semantics

Functional, big-step semantics

$$
\begin{aligned}
&[\![\_]\!] : \forall \{n\} \to Tm\ n \to Env\ n \to (Maybe\ Value)_{\bot} \\
&[\![\ \mathsf{con}\ i\ ]\!]\ \rho = return\ (\mathsf{con}\ i) \\
&[\![\ \mathsf{var}\ x\ ]\!]\ \rho = return\ (lookup\ x\ \rho) \\
&[\![\ \lambda\ t\ ]\!]\ \rho = return\ (\lambda\ t\ \rho) \\
&[\![\ t_1 \cdot t_2\ ]\!]\ \rho = [\![\ t_1\ ]\!]\ \rho \gg= \lambda\ v_1 \to \\
&\qquad [\![\ t_2\ ]\!]\ \rho \gg= \lambda\ v_2 \to \\
&\qquad v_1 \bullet v_2
\end{aligned}
$$

$$
\begin{aligned}
&\_\bullet\_ : Value \to Value \to (Maybe\ Value)_{\bot} \\
&\mathsf{con}\ i \bullet v_2 = fail \\
&\lambda\ t_1\ \rho \bullet v_2 = \mathsf{later}\ (\sharp\ ([\![\ t_1\ ]\!]\ (v_2 :: \rho)))
\end{aligned}
$$

Functional, big-step semantics

- Maybe monad transformer applied to partiality monad.
- Not guarded. Easy to work around (AIM IX).

Functional, big-step semantics

- Deterministic.
- Can be executed directly (inefficiently).
- Equivalent (classically) to relational big-step semantics:

$$
\begin{aligned}
\rho \vdash t \Downarrow v \quad &\Leftrightarrow \quad [\![t]\!] \rho \approx \mathit{return}\ v \\
\rho \vdash t \Uparrow \quad &\Leftrightarrow \quad [\![t]\!] \rho \approx \mathit{never} \\
\rho \vdash t \lightning \quad &\Leftrightarrow \quad [\![t]\!] \rho \approx \mathit{fail}
\end{aligned}
$$

Functional, big-step semantics

- *Operational* semantics:
  $\_\bullet\_$ defined in terms of $[\![\_]\!]$.
- Can define denotational semantics:

$$
\begin{aligned}
&[\![\_]\!]' : \forall \{n\} \rightarrow Tm\ n \rightarrow Env\ n \rightarrow \\
&\qquad (Maybe\ Value)_{\bot} \ / \ \_\approx\_ \\
&[\![\ t_1 \cdot t_2\ ]\!]'\ \rho = f\ ([\![\ t_1\ ]\!]'\ \rho)\ ([\![\ t_2\ ]\!]'\ \rho) \\
&\quad \textbf{where} \\
&\quad f\ x\ y = x \gg\!= \lambda\ v_1 \rightarrow \\
&\qquad\qquad\ \ y \gg\!= \lambda\ v_2 \rightarrow \\
&\qquad\qquad\ \ v_1 \bullet v_2
\end{aligned}
$$

f preserves equality.

Virtual machine

Virtual machine

- Instruction set: $\mathit{Instr} : \mathbb{N} \to \mathit{Set}$
- Code: $\mathit{Code}\ n = \mathit{List}\ (\mathit{Instr}\ n)$
- States: $\mathit{State} : \mathit{Set}$
- Initial state: $\mathit{init} : \mathit{Code}\ \mathit{0} \to \mathit{State}$
- Values: $\mathit{Value}_{\mathsf{VM}} : \mathit{Set}$
- Compiler:

$$\begin{aligned} &\mathit{comp} &&: \forall\ \{n\} \to \mathit{Tm}\ n \to \mathit{Code}\ n \\ &\mathit{comp}_{\mathsf{v}} &&: \mathit{Value} \to \mathit{Value}_{\mathsf{VM}} \end{aligned}$$

Relational, small-step semantics

$$
\begin{array}{lcl}
\_\longrightarrow\_ & : & \mathit{State} \to \mathit{State} \to \mathit{Set} \\
\_\mathit{Is}\_ & : & \mathit{State} \to \mathit{Value}_{\mathsf{VM}} \to \mathit{Set}
\end{array}
$$

$$
\begin{array}{lcl}
s \Downarrow v & = & \exists\, s'.\ s \longrightarrow^{\star} s' \ \wedge\ s' \not\longrightarrow \ \wedge\ s'\ \mathit{Is}\ v \\
s \Uparrow & = & s \longrightarrow^{\infty} \\
s \lightning & = & \exists\, s'.\ s \longrightarrow^{\star} s' \ \wedge\ s' \not\longrightarrow \ \wedge\ \nexists\, v.\ s'\ \mathit{Is}\ v
\end{array}
$$

- Avoids rule duplication.
- Exhaustive?
- Deterministic?
- Executable?

Functional, small-step semantics

$$
\begin{array}{l}
\textbf{data}\ \mathit{Result}\ :\ \mathit{Set}\ \textbf{where} \\
\quad \mathsf{continue}\ :\ \mathit{State} \to \mathit{Result} \\
\quad \mathsf{done}\ :\ \mathit{Value}_{\mathsf{VM}} \to \mathit{Result} \\
\quad \mathsf{crash}\ :\ \mathit{Result}
\end{array}
$$

$$\mathit{step}\ :\ \mathit{State} \to \mathit{Result}$$

$$
\begin{array}{l}
\mathit{exec}\ :\ \mathit{State} \to (\mathit{Maybe}\ \mathit{Value}_{\mathsf{VM}})_{\perp} \\
\mathit{exec}\ s\ \textbf{with}\ \mathit{step}\ s \\
\ldots \mid \mathsf{continue}\ s' = \mathsf{later}\ (\sharp\ \mathit{exec}\ s') \\
\ldots \mid \mathsf{done}\ v = \mathit{return}\ v \\
\ldots \mid \mathsf{crash} = \mathit{fail}
\end{array}
$$

Functional, small-step semantics

- Equivalent to relational semantics:

$$
\begin{array}{lcl}
s \Downarrow v & \Leftrightarrow & \textit{exec}\ s \approx \textit{return}\ v \\
s \Uparrow & \Leftrightarrow & \textit{exec}\ s \approx \textit{never} \\
s \lightning & \Leftrightarrow & \textit{exec}\ s \approx \textit{fail}
\end{array}
$$

- Still possible to forget a case in *step*:

$$\textit{step}\ \_ \;=\; \text{crash}$$

- Deterministic.
- Executable.

Compiler correctness

Compiler correctness statement

"The compiler preserves the semantics."

For relational semantics:

$$
\begin{array}{lcl}
[\,] \vdash t \Downarrow v & \Leftrightarrow & \mathit{init}\ (\mathit{comp}\ t) \Downarrow \mathit{comp}_{\mathsf{v}}\ v \\
[\,] \vdash t \Uparrow & \Leftrightarrow & \mathit{init}\ (\mathit{comp}\ t) \Uparrow \\
[\,] \vdash t \lightning & \Leftrightarrow & \mathit{init}\ (\mathit{comp}\ t) \lightning
\end{array}
$$

$\Leftarrow$ often omitted.

Compiler correctness statement

"The compiler preserves the semantics."

For relational semantics:

$$
\begin{array}{lcl}
[\,] \vdash t \Downarrow v & \Leftrightarrow & \mathit{init}\ (\mathit{comp}\ t) \Downarrow \mathit{comp}_{\mathsf{v}}\ v \\
[\,] \vdash t \Uparrow & \Leftrightarrow & \mathit{init}\ (\mathit{comp}\ t) \Uparrow \\
[\,] \vdash t \lightning & \Leftrightarrow & \mathit{init}\ (\mathit{comp}\ t) \lightning
\end{array}
$$

$\Leftarrow$ often omitted.

For functional semantics:

$$
\begin{array}{l}
\mathit{exec}\ (\mathit{init}\ (\mathit{comp}\ t)) \approx \\
\quad [\![\, t \,]\!]\ [\,] \ggg= \lambda\ v \rightarrow \mathit{return}\ (\mathit{comp}_{\mathsf{v}}\ v)
\end{array}
$$

Easy to reason about?

$\_\approx\_$ not "infinitely transitive"

$\_\approx\_$ is an equivalence relation.
Let us postulate transitivity:

$$
\begin{array}{lcl}
\textbf{data}\ \_\approx\_\ \{A : \mathit{Set}\} : A_\bot \to A_\bot \to \mathit{Set}\ \textbf{where} \\
\quad \mathsf{now} : \qquad \to \mathsf{now}\ v \approx \mathsf{now}\ v \\
\quad \mathsf{later} : \infty\,(\flat\, x \approx \flat\, y) \to \mathsf{later}\ x \approx \mathsf{later}\ y \\
\quad \mathsf{later}^{\mathrm{r}} : x \approx \flat\, y \to x \approx \mathsf{later}\ y \\
\quad \mathsf{later}^{\mathrm{l}} : \flat\, x \approx y \to \mathsf{later}\ x \approx y \\
\quad \_\approx\langle\_\rangle\_ : \forall\, x \to x \approx y \to y \approx z \to x \approx z
\end{array}
$$

_≈_ not "infinitely transitive"

_□: Proof of reflexivity.

$$
\begin{array}{l}
\textit{trivial} \;:\; \{A \;:\; \textit{Set}\}\,(x\;y \;:\; A\,_\bot) \;\to\; x \approx y \\
\textit{trivial}\;x\;y \;=\; \\
\quad \begin{array}{ll}
x & \approx\langle\; \mathsf{later}^{\mathrm{r}}\;(x\;\Box)\;\rangle \\
\mathsf{later}\;(\sharp\;x) & \approx\langle\; \mathsf{later}\;(\sharp\;\textit{trivial}\;x\;y)\;\rangle \\
\mathsf{later}\;(\sharp\;y) & \approx\langle\; \mathsf{later}^{\mathrm{l}}\;(y\;\Box)\;\rangle \\
y & \Box
\end{array}
\end{array}
$$

Compare the problem of "weak bisimulation up to".

Only a problem for infinite proofs.

$\_\approx\_$ not “infinitely transitive”

One possible workaround:

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\textbf{data}\ \_\approx\_\ \{A : \mathit{Set}\} : A_\bot \to A_\bot \to \mathit{Set}\ \textbf{where}} \\
\quad \mathsf{now} & : & \to \mathsf{now}\ v \approx \mathsf{now}\ v \\
\quad \mathsf{later} & : & \infty\,(\flat\, x \approx \flat\, y) \to \mathsf{later}\ x \approx \mathsf{later}\ y \\
\quad \mathsf{later}^{\mathrm{r}} & : & x \approx \flat\, y \to x \approx \mathsf{later}\ y \\
\quad \mathsf{later}^{\mathrm{l}} & : & \flat\, x \approx y \to \mathsf{later}\ x \approx y \\
\quad \_\gtrsim\langle\_\rangle^{\mathrm{l}}\_ & : & \forall\, x \to x \gtrsim y \to y \approx z \to x \approx z \\
\quad \_\gtrsim\langle\_\rangle^{\mathrm{r}}\_ & : & \forall\, x \to x \approx y \to y \lesssim z \to x \approx z
\end{array}
$$

$x \gtrsim y$: y terminates faster than x, or both loop.

Similar to $\_\longrightarrow^\infty$: $x \longrightarrow^\star y \longrightarrow^\infty \Rightarrow x \longrightarrow^\infty$.

Wrapping up

Related work

- Rutten, A note on Coinduction and Weak Bisimilarity for While Programs.
- Capretta, General Recursion via Coinductive Types.
- Nakata and Uustalu, Trace-Based Coinductive Operational Semantics for While.

Conclusions

- Exhaustive pattern matching $\Rightarrow$ harder to forget rules.
- Deterministic monad $\Rightarrow$ deterministic semantics.
- Executable semantics.
- Small-step or big-step.
- Less scope for abstraction.
- Other drawbacks?
- Future work: Non-determinism, concurrency.

?

$\_\approx\_$ not "infinitely transitive"

Can reduce need for transitivity by using continuation-passing style.

Goal ($comp'\ t\ c \equiv comp\ t\ +\!\!+\ c$):

$$exec\ (init\ (comp'\ t\ [\,]))\approx$$
$$[\![\ t\ ]\!]\ [\,]\ \ggg\ \lambda\ v\ \rightarrow\ return\ (comp_{\mathsf{v}}\ v)$$

Generalisation:

$$(\forall\ v\ \rightarrow\ exec\ (\ldots\ c\ \ldots\ v\ \ldots\ \rho\ \ldots)\approx f\ v)\ \rightarrow$$
$$exec\ (\ldots\ comp'\ t\ c\ \ldots\ \rho\ \ldots)\approx$$
$$[\![\ t\ ]\!]\ \rho\ \ggg\ f$$