Operational Semantics Using the Partiality Monad

Nils Anders Danielsson (Nottingham)

DTP 2010, Edinburgh, 2010-07-09

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:

$$
\begin{array}{l}
\textbf{mutual} \\
\quad \textbf{data}\ \mathit{Value}\ :\ \mathit{Set}\ \textbf{where} \\
\quad\quad \mathsf{con}\ :\ \mathbb{N} \to \mathit{Value} \\
\quad\quad \lambda\ \ \ :\ \forall \{n\} \to \mathit{Tm}\,(\mathit{suc}\ n) \to \mathit{Env}\ n \to \mathit{Value} \\
\quad \mathit{Env}\ :\ \mathbb{N} \to \mathit{Set} \\
\quad \mathit{Env}\ n\ =\ \mathit{Vec}\ \mathit{Value}\ n
\end{array}
$$

Relational, big-step semantics

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

Relational, big-step semantics

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{data}\ \_\vdash\_\Downarrow\_\ \{n\}\ (\rho\ :\ \mathit{Env}\ n)\ :} \\
\multicolumn{3}{l}{\qquad \mathit{Tm}\ n\ \rightarrow\ \mathit{Value}\ \rightarrow\ \mathit{Set}\ \textbf{where}} \\
\mathsf{var} & : & \rho \vdash \mathsf{var}\ x\ \Downarrow\ \mathit{lookup}\ x\ \rho \\
\mathsf{con} & : & \rho \vdash \mathsf{con}\ i\ \Downarrow\ \mathsf{con}\ i \\
\lambda & : & \rho \vdash \lambda\ t\ \Downarrow\ \lambda\ t\ \rho \\
\mathsf{app} & : & \rho \vdash t_1\ \Downarrow\ \lambda\ t\ \rho'\ \rightarrow\ \rho \vdash t_2\ \Downarrow\ v'\ \rightarrow \\
 & & v' :: \rho' \vdash t\ \Downarrow\ v\ \rightarrow\ \rho \vdash t_1 \cdot t_2\ \Downarrow\ v
\end{array}
$$

We are not done. Assume $\nexists\, v.\ \rho \vdash t\ \Downarrow\ v$.
Does the program crash or run forever?

Relational, big-step semantics

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

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

Relational, big-step semantics

Coinductive:

$$
\begin{array}{l}
\textbf{data}\ \_\vdash\_\Uparrow\ \{n\}\ (\rho\ :\ \mathit{Env}\ n)\ :\ \mathit{Tm}\ n\ \rightarrow\ \mathit{Set}\ \textbf{where} \\
\quad \mathsf{app}^{\mathrm{l}}\ :\ \infty\,(\rho \vdash t_1\ \Uparrow)\ \rightarrow\ \rho \vdash t_1 \cdot t_2\ \Uparrow \\
\quad \mathsf{app}^{\mathrm{r}}\ :\ \rho \vdash t_1\ \Downarrow\ v\ \rightarrow\ \infty\,(\rho \vdash t_2\ \Uparrow)\ \rightarrow \\
\qquad\qquad \rho \vdash t_1 \cdot t_2\ \Uparrow \\
\quad \mathsf{app}\ \ :\ \rho \vdash t_1\ \Downarrow\ \lambda\ t\ \rho'\ \rightarrow\ \rho \vdash t_2\ \Downarrow\ v'\ \rightarrow \\
\qquad\qquad \infty\,(v' :: \rho' \vdash t\ \Uparrow)\ \rightarrow\ \rho \vdash t_1 \cdot t_2\ \Uparrow
\end{array}
$$

Relational, big-step semantics

Coinductive:

$$
\begin{aligned}
&\textbf{data } \_\vdash\_\Uparrow \{n\}\ (\rho : \mathit{Env}\ n) : \mathit{Tm}\ n \to \mathit{Set}\ \textbf{where} \\
&\quad \mathsf{app}^{\mathrm{l}} : \infty\,(\rho \vdash t_1 \Uparrow) \to \rho \vdash t_1 \cdot t_2 \Uparrow \\
&\quad \mathsf{app}^{\mathrm{r}} : \rho \vdash t_1 \Downarrow v \to \infty\,(\rho \vdash t_2 \Uparrow) \to \\
&\qquad\qquad \rho \vdash t_1 \cdot t_2 \Uparrow \\
&\quad \mathsf{app} : \rho \vdash t_1 \Downarrow \lambda\ t\ \rho' \to \rho \vdash t_2 \Downarrow v' \to \\
&\qquad\qquad \infty\,(v' :: \rho' \vdash t \Uparrow) \to \rho \vdash t_1 \cdot t_2 \Uparrow
\end{aligned}
$$

$$
\begin{aligned}
&\_\vdash\_\lightning : \forall\ \{n\} \to \mathit{Env}\ n \to \mathit{Tm}\ n \to \mathit{Set} \\
&\rho \vdash t \lightning = (\nexists\ \lambda\ v \to \rho \vdash t \Downarrow v) \times \neg\,(\rho \vdash t \Uparrow)
\end{aligned}
$$

Relational, big-step semantics

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

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

$$
\begin{array}{lll}
\mathit{eval} & : \forall\, \rho\ t\ \to & (\exists\, \lambda\, v\ \to\ \rho \vdash t \Downarrow v)\ \uplus \\
& & \rho \vdash t \Uparrow\ \uplus\ \rho \vdash t\ \lightning
\end{array}
$$

Outline

- Partiality monad.
- Semantics using the partiality monad.
- Compiler correctness statement.

Partiality monad

Partiality monad

data $\_{}_\bot$ $(A : Set) : Set$ **where**
  now : $A \to A_\bot$
  later : $\infty (A_\bot) \to A_\bot$

- ∞ makes the definition coinductive.
- $A_\bot \approx \nu C.\, A + C$.
- Delay and force:

$\sharp : A \to \infty A$
$\flat : \infty A \to A$

Partiality monad

$$
\begin{array}{l}
\textbf{data}\ \_{}_\bot\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where} \\
\quad \mathsf{now} \ : A \to A_\bot \\
\quad \mathsf{later} : \infty\,(A_\bot) \to A_\bot
\end{array}
$$

$$
\begin{array}{l}
\mathit{never} : \forall \{A\} \to A_\bot \\
\mathit{never} = \mathsf{later}\ (\sharp\ \mathit{never})
\end{array}
$$

$$
\begin{array}{l}
\_{\gg=}\_ : \forall \{A\ B\} \to A_\bot \to (A \to B_\bot) \to B_\bot \\
\mathsf{now}\ x \gg= f = f\ x \\
\mathsf{later}\ x \gg= f = \mathsf{later}\ (\sharp\ (\flat\ x \gg= f))
\end{array}
$$

Functional semantics

Functional, big-step semantics

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

$$
\begin{array}{l}
\_\bullet\_ : \mathit{Value} \to \mathit{Value} \to (\mathit{Maybe}\ \mathit{Value})_{\perp} \\
\mathsf{con}\ i \bullet v_2 = \mathit{fail} \\
\lambda\ t_1\ \rho \bullet v_2 = \mathsf{later}\,(\sharp\,([\![\, t_1\,]\!]\,(v_2 :: \rho)))
\end{array}
$$

Functional, big-step semantics

- No code duplication.
- Exhaustive pattern matching.
- Can be executed directly (inefficiently).
- Deterministic?
- Equivalent to relational big-step semantics?

Equality for the partiality monad

Weak bisimilarity:

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

Equality for the partiality monad

Weak bisimilarity:

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

$$
\begin{array}{l}
\_\approx\_ \approx \nu C.\ \mu I.\ \lambda\, x\, y. \\
\qquad (\exists\, v.\ \ x \equiv \mathsf{now}\ v \times y \equiv \mathsf{now}\ v) \\
\quad + (\exists\, x', y'.\ x \equiv \mathsf{later}\ x' \times y \equiv \mathsf{later}\ y' \times \\
\qquad\qquad C\ (\flat\, x')\ (\flat\, y')) \\
\quad + (\exists\, y'.\ \ y \equiv \mathsf{later}\ y' \times I\ x\ (\flat\, y')) \\
\quad + (\exists\, x'.\ \ x \equiv \mathsf{later}\ x' \times I\ (\flat\, x')\ y)
\end{array}
$$

Equality for the partiality monad

Weak bisimilarity:

$$\textbf{data}\ \_{\approx}\_\ \{A\ :\ Set\}\ :\ A_{\bot}\ \rightarrow\ A_{\bot}\ \rightarrow\ Set\ \textbf{where}$$

$$\begin{array}{lll}
\text{now} & : & \rightarrow\ \text{now}\ v \approx \text{now}\ v \\
\text{later} & : \infty\,(\flat\, x \approx \flat\, y) & \rightarrow\ \text{later}\ x \approx \text{later}\ y \\
\text{later}^{\mathrm{r}} & : x \approx \flat\, y & \rightarrow\ x \approx \text{later}\ y \\
\text{later}^{\mathrm{l}} & : \flat\, x \approx y & \rightarrow\ \text{later}\ x \approx y
\end{array}$$

$$\begin{aligned}
{\approx} \approx\ & \nu C.\ \mu I.\ \lambda\, x\ y. \\
& (\exists\, v.\quad x \equiv \text{now}\ v \times y \equiv \text{now}\ v) \\
& + (\exists\, x', y'.\ x \equiv \text{later}\ x' \times y \equiv \text{later}\ y' \times \\
& \qquad C\ (\flat\, x')\ (\flat\, y')) \\
& + (\exists\, y'.\quad y \equiv \text{later}\ y' \times I\ x\ (\flat\, y')) \\
& + (\exists\, x'.\quad x \equiv \text{later}\ x' \times I\ (\flat\, x')\ y)
\end{aligned}$$

Functional, big-step semantics

$[\![\_]\!]$ is equivalent (classically) to the relational big-step semantics:

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

Functional, big-step semantics

Operational semantics:

- $\_ \bullet \_$ defined in terms of $[\![\_]\!]$
  (not compositional).
- $[\![\, \lambda \, x. \, x \,]\!] \, [\,] \not\approx [\![\, \lambda \, x. \, (\lambda \, x. \, x) \, x \,]\!] \, [\,]$.
  (Can define more interesting equalities.)

Compiler correctness

Virtual machine semantics

- Relations defined in terms of relational small-step semantics:

$$
\begin{array}{lcl}
\_\Downarrow\_ & : & \mathit{State} \rightarrow \mathit{Value}_{\mathsf{VM}} \rightarrow \mathit{Set} \\
\_\Uparrow & : & \mathit{State} \rightarrow \mathit{Set} \\
\_\lightning & : & \mathit{State} \rightarrow \mathit{Set}
\end{array}
$$

- Functional small-step semantics:

$$
\mathit{exec} : \mathit{State} \rightarrow (\mathit{Maybe}\ \mathit{Value}_{\mathsf{VM}})_{\bot}
$$

Compilers

$$\begin{aligned} comp &: \forall \{n\} \rightarrow Tm\ n \rightarrow State \\ comp_{\mathsf{v}} &: Value \rightarrow Value_{\mathsf{VM}} \end{aligned}$$

Compiler correctness statement

"The compiler preserves the semantics."

For relational semantics:

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

Compiler correctness statement

"The compiler preserves the semantics."

For relational semantics:

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

For functional semantics:

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

Wrapping up

Conclusions

- Exhaustive pattern matching $\Rightarrow$ harder to forget rules.
- Deterministic monad $\Rightarrow$ deterministic semantics.
- Executable semantics.
- Small-step or big-step.

Conclusions

- Less scope for abstraction.
- Other drawbacks?
- Future work: Non-determinism, concurrency.
- Related work:
  Rutten, Capretta, Nakata and Uustalu.

?

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.

Virtual machine

Virtual machine

- States: $\mathit{State} : \mathit{Set}$
- Values: $\mathit{Value}_{\mathsf{VM}} : \mathit{Set}$
- Compiler:

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

Relational, small-step semantics

$$
\begin{array}{lllllll}
\_ \longrightarrow \_ & : & \mathit{State} & \rightarrow & \mathit{State} & \rightarrow & \mathit{Set} \\
\_ \sim \_ & : & \mathit{State} & \rightarrow & \mathit{Value}_{\mathsf{VM}} & \rightarrow & \mathit{Set}
\end{array}
$$

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

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

Functional, small-step semantics

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

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

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

Functional, small-step semantics

- Equivalent to relational semantics:

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

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

$$\mathit{step}\ \_ \ = \ \mathsf{crash}$$

- Deterministic.
- Executable.

Easy to reason about?

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

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

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

_≈_ not "infinitely transitive"

_□: Proof of reflexivity.

$$
\begin{array}{l}
\mathit{trivial} : \{A : \mathit{Set}\}\ (x\ y : A_{\bot}) \to x \approx y \\
\mathit{trivial}\ x\ y = \\
\quad \begin{array}{ll}
x & \approx\langle\ \mathsf{later}^{\mathrm{r}}\ (x\ \square)\ \rangle \\
\mathsf{later}\ (\sharp\ x) & \approx\langle\ \mathsf{later}\ (\sharp\ \mathit{trivial}\ x\ y)\ \rangle \\
\mathsf{later}\ (\sharp\ y) & \approx\langle\ \mathsf{later}^{\mathrm{l}}\ (y\ \square)\ \rangle \\
y & \square
\end{array}
\end{array}
$$

Compare the problem of "weak bisimulation up to".

Only a problem for infinite proofs.

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

One possible workaround:

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

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

Similar to $\_\to^{\infty}$: $x \to^{\star} y \to^{\infty} \;\Rightarrow\; x \to^{\infty}$.

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

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

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

$$\begin{aligned}&exec\ (comp'\ t\ [\,])\approx\\&\quad [\![\ t\ ]\!]\ [\,]\ \gg\!=\ \lambda\ v\ \rightarrow\ return\ (comp_{\mathsf{v}}\ v)\end{aligned}$$

Generalisation:

$$\begin{aligned}&(\forall\ v\ \rightarrow\ exec\ (\ldots\ c\ \ldots\ v\ \ldots\ \rho\ \ldots)\approx f\ v)\ \rightarrow\\&\quad exec\ (\ldots\ comp'\ t\ c\ \ldots\ \rho\ \ldots)\approx [\![\ t\ ]\!]\ \rho\ \gg\!=\ f\end{aligned}$$

Deterministic monad

Deterministic monad

Is $[\![\_]\!]$ deterministic?

- If the monad is "deterministic":

$$
\begin{aligned}
&\_\in\_ \;:\; \mathit{Result}\ A \;\rightarrow\; M\ A \;\rightarrow\; \mathit{Set} \\
&r \;\in\; m \;=\; \ldots
\end{aligned}
$$

$$
r \;\in\; m \quad \wedge \quad r' \;\in\; m \quad \Rightarrow \quad r \equiv r'
$$

- Example of non-deterministic monad:
  List monad.