Pretty-big-step semantics

Arthur Charguéraud

INRIA

October 2012

Motivation

Formalization of JavaScript

$\rightarrow$ with Sergio Maffeis, Daniele Filaretti, Alan Schmitt, Martin Bodin.

Previous work:

- Semi-formal small-step semantics for the entire language (jssec.net)
- Informal big-step semantics for the core language (POPL'12)

Current work:

- Formal big-step semantics for the entire language
- Interpreter proved correct w.r.t. the semantics

Motivation for big-step

Big-step semantics:

- more faithful to the reference manual
- easier than small-step for proving an interpreter
- easier than small-step for proving a program logic

Small-step semantics considered better-suited for:

- machine-code semantics $\rightarrow$ not the case of JS
- type soundness proofs $\rightarrow$ no types in JS
- concurrent languages $\rightarrow$ no concurrency in JS

Big-step semantics for loops

Semantics of a C-style loop "$\mathsf{for}\,(\ ;\, t_1\,;\, t_2)\ \{\, t_3\, \}$", written "$\mathsf{for}\, t_1\, t_2\, t_3$",
in terms of the evaluation judgment $t_{/m} \ \Rightarrow\ v_{/m'}$.

$$\frac{t_{1/m_1} \ \Rightarrow\ \mathsf{false}_{/m_2}}{\mathsf{for}\, t_1\, t_2\, t_{3/m_1} \ \Rightarrow\ tt_{/m_2}}$$

$$\frac{t_{1/m_1} \ \Rightarrow\ \mathsf{true}_{/m_2} \quad t_{3/m_2} \ \Rightarrow\ tt_{/m_3} \quad t_{2/m_3} \ \Rightarrow\ tt_{/m_4} \quad \mathsf{for}\, t_1\, t_2\, t_{3/m_4} \ \Rightarrow\ tt_{/m_5}}{\mathsf{for}\, t_1\, t_2\, t_{3/m_1} \ \Rightarrow\ tt_{/m_5}}$$

Big-step semantics for loops: exceptions

Exceptions in terms of the judgment $t_{/m} \Rightarrow^{\mathsf{exn}}{}_{/m'}$.

$$\frac{t_{1/m_1} \Rightarrow^{\mathsf{exn}}{}_{/m_2}}{\mathsf{for}\; t_1\; t_2\; t_{3/m_1} \Rightarrow^{\mathsf{exn}}{}_{/m_2}}$$

$$\frac{t_{1/m_1} \;\Rightarrow\; \mathsf{true}_{/m_2} \qquad t_{3/m_2} \Rightarrow^{\mathsf{exn}}{}_{/m_3}}{\mathsf{for}\; t_1\; t_2\; t_{3/m_1} \Rightarrow^{\mathsf{exn}}{}_{/m_3}}$$

$$\frac{t_{1/m_1} \;\Rightarrow\; \mathsf{true}_{/m_2} \qquad t_{3/m_2} \;\Rightarrow\; tt_{/m_3} \qquad t_{2/m_3} \Rightarrow^{\mathsf{exn}}{}_{/m_4}}{\mathsf{for}\; t_1\; t_2\; t_{3/m_1} \Rightarrow^{\mathsf{exn}}{}_{/m_4}}$$

$$\frac{t_{1/m_1} \;\Rightarrow\; \mathsf{true}_{/m_2} \qquad t_{3/m_2} \;\Rightarrow\; tt_{/m_3} \qquad t_{2/m_3} \;\Rightarrow\; tt_{/m_4} \qquad \mathsf{for}\; t_1\; t_2\; t_{3/m_4} \Rightarrow^{\mathsf{exn}}{}_{/m_5}}{\mathsf{for}\; t_1\; t_2\; t_{3/m_1} \Rightarrow^{\mathsf{exn}}{}_{/m_5}}$$

Big-step semantics for loops: divergence

Divergence in terms of the coinductive judgment $t_{/m} \Rightarrow^\infty$ (Leroy 2006).

$$\frac{t_{1/m_1} \Rightarrow^\infty}{\mathsf{for}\ t_1\ t_2\ t_{3/m_1} \Rightarrow^\infty}$$

$$\frac{t_{1/m_1} \Rightarrow \mathsf{true}_{/m_2} \quad t_{3/m_2} \Rightarrow^\infty}{\mathsf{for}\ t_1\ t_2\ t_{3/m_1} \Rightarrow^\infty}$$

$$\frac{t_{1/m_1} \Rightarrow \mathsf{true}_{/m_2} \quad t_{3/m_2} \Rightarrow tt_{/m_3} \quad t_{2/m_3} \Rightarrow^\infty}{\mathsf{for}\ t_1\ t_2\ t_{3/m_1} \Rightarrow^\infty}$$

$$\frac{t_{1/m_1} \Rightarrow \mathsf{true}_{/m_2} \quad t_{3/m_2} \Rightarrow tt_{/m_3} \quad t_{2/m_3} \Rightarrow tt_{/m_4} \quad \mathsf{for}\ t_1\ t_2\ t_{3/m_4} \Rightarrow^\infty}{\mathsf{for}\ t_1\ t_2\ t_{3/m_1} \Rightarrow^\infty}$$

Big-step semantics for loops: summary

$$\frac{t_1/m_1 \Rightarrow \textsf{false}/m_2}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow tt/m_2}$$

$$\frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow tt/m_3 \quad t_2/m_3 \Rightarrow tt/m_4 \quad \textsf{for}\ t_1\ t_2\ t_3/m_4 \Rightarrow tt/m_5}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow tt/m_5}$$

$$\frac{t_1/m_1 \Rightarrow^{\textsf{exn}}/m_2}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\textsf{exn}}/m_2} \qquad \frac{t_1/m_1 \Rightarrow^{\infty}}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\infty}}$$

$$\frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow^{\textsf{exn}}/m_3}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\textsf{exn}}/m_3} \qquad \frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow^{\infty}}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\infty}}$$

$$\frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow tt/m_3 \quad t_2/m_3 \Rightarrow^{\textsf{exn}}/m_4}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\textsf{exn}}/m_4} \qquad \frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow tt/m_3 \quad t_2/m_3 \Rightarrow^{\infty}}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\infty}}$$

$$\frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow tt/m_3 \quad t_2/m_3 \Rightarrow tt/m_4 \quad \textsf{for}\ t_1\ t_2\ t_3/m_4 \Rightarrow^{\textsf{exn}}/m_5}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\textsf{exn}}/m_5} \qquad \frac{t_1/m_1 \Rightarrow \textsf{true}/m_2 \quad t_3/m_2 \Rightarrow tt/m_3 \quad t_2/m_3 \Rightarrow tt/m_4 \quad \textsf{for}\ t_1\ t_2\ t_3/m_4 \Rightarrow^{\infty}}{\textsf{for}\ t_1\ t_2\ t_3/m_1 \Rightarrow^{\infty}}$$

→ Even with factorization: 9 rules, 21 premises.

→ With pretty-big-step: 6 rules, 7 premises.

In this talk

Pretty-big-step semantics:

- construction
- extension to traces
- application to core-Caml
- ~~type soundness proofs~~

Pretty-big-step

Big-step semantics

Grammar of λ-terms

$$\begin{array}{rcl} v & := & \mathsf{int}\,n \mid \mathsf{abs}\,x\,t \\ t & := & \mathsf{val}\,v \mid \mathsf{var}\,x \mid \mathsf{app}\,t\,t \end{array}$$

Call-by-value big-step semantics ($t \Rightarrow v$)

$$\frac{}{v \Rightarrow v} \qquad \frac{t_1 \Rightarrow \mathsf{abs}\,x\,t \qquad t_2 \Rightarrow v \qquad [x \to v]\,t \Rightarrow v'}{\mathsf{app}\,t_1\,t_2 \Rightarrow v'}$$

A first attempt

Big-step rule for applications:

$$\frac{t_1 \;\Rightarrow\; \mathsf{abs}\, x\, t \qquad t_2 \;\Rightarrow\; v \qquad [x \to v]\, t \;\Rightarrow\; v'}{\mathsf{app}\, t_1\, t_2 \;\Rightarrow\; v'}$$

A first attempt at pretty-big-step rules:

$$\frac{t_1 \;\Rightarrow\; v_1 \quad \mathsf{app}\, v_1\, t_2 \;\Rightarrow\; v'}{\mathsf{app}\, t_1\, t_2 \;\Rightarrow\; v'} \qquad \frac{t_2 \;\Rightarrow\; v_2 \quad \mathsf{app}\, v_1\, v_2 \;\Rightarrow\; v'}{\mathsf{app}\, v_1\, t_2 \;\Rightarrow\; v'} \qquad \frac{[x \to v]\, t \;\Rightarrow\; v'}{\mathsf{app}\, (\mathsf{abs}\, x\, t)\, v \;\Rightarrow\; v'}$$

→ Similar idea in Cousot and Cousot's bi-inductive semantics (2007)

Intermediate terms

To prevent overlap between the rules, we use intermediate terms.

$$e \quad := \quad \mathsf{trm}\, t \mid \mathsf{app1}\, v\, t \mid \mathsf{app2}\, v\, v$$

Definition of the judgment $e \Downarrow v$, with trm implicit

$$\frac{}{v \Downarrow v} \qquad \frac{t_1 \Downarrow v_1 \qquad \mathsf{app1}\, v_1\, t_2 \Downarrow v'}{\mathsf{app}\, t_1\, t_2 \Downarrow v'}$$

$$\frac{t_2 \Downarrow v_2 \qquad \mathsf{app2}\, v_1\, v_2 \Downarrow v'}{\mathsf{app1}\, v_1\, t_2 \Downarrow v'} \qquad \frac{[x \to v]\, t \Downarrow v'}{\mathsf{app2}\, (\mathsf{abs}\, x\, t)\, v \Downarrow v'}$$

Adding exceptions

Value-carrying exceptions and exception handlers

$$t \quad := \quad \ldots \mid \mathsf{raise}\, t \mid \mathsf{try}\, t\, t$$

Two behaviors: return a value or throw an exception.

$$e \Downarrow b \qquad\qquad b \quad := \quad \mathsf{ret}\, v \mid \mathsf{exn}\, v$$

Generalization of intermediate terms

Need to generalize intermediate terms

$$\frac{t_1 \Downarrow b_1 \qquad \mathsf{app1}\, b_1\, t_2 \Downarrow b}{\mathsf{app}\, t_1\, t_2 \Downarrow b}$$

$$\frac{}{\mathsf{app1}\,(\mathsf{exn}\, v)\, t_2 \Downarrow \mathsf{exn}\, v} \qquad \frac{t_2 \Downarrow b_2 \qquad \mathsf{app2}\, v_1\, b_2 \Downarrow b}{\mathsf{app1}\,(\mathsf{ret}\, v_1)\, t_2 \Downarrow b}$$

New grammar of intermediate terms

$$e \quad := \quad \mathsf{trm}\, t \mid \mathsf{app1}\, b\, t \mid \mathsf{app2}\, v\, b \mid \mathsf{raise1}\, b \mid \mathsf{try1}\, b\, t$$

Pretty-big-step rules for exceptions

$$\frac{}{v \Downarrow v} \qquad \frac{t_1 \Downarrow b_1 \qquad \mathsf{app1}\, b_1\, t_2 \Downarrow b}{\mathsf{app}\, t_1\, t_2 \Downarrow b} \qquad \frac{}{\mathsf{app1}\, (\mathsf{exn}\, v)\, t \Downarrow \mathsf{exn}\, v}$$

$$\frac{t_2 \Downarrow b_2 \qquad \mathsf{app2}\, v_1\, b_2 \Downarrow b}{\mathsf{app1}\, v_1\, t_2 \Downarrow b} \qquad \frac{}{\mathsf{app2}\, v\, (\mathsf{exn}\, v) \Downarrow \mathsf{exn}\, v}$$

$$\frac{[x \to v]\, t \Downarrow b}{\mathsf{app2}\, (\mathsf{abs}\, x\, t)\, v \Downarrow b}$$

$$\frac{t_1 \Downarrow b_1 \qquad \mathsf{try1}\, b_1\, t_2 \Downarrow b}{\mathsf{try}\, t_1\, t_2 \Downarrow b} \qquad \frac{}{\mathsf{try1}\, v\, t \Downarrow v} \qquad \frac{\mathsf{app}\, t\, v \Downarrow b}{\mathsf{try1}\, (\mathsf{exn}\, v)\, t \Downarrow b}$$

$$\frac{t \Downarrow b_1 \qquad \mathsf{raise1}\, b_1 \Downarrow b}{\mathsf{raise}\, t \Downarrow b} \qquad \frac{}{\mathsf{raise1}\, v \Downarrow \mathsf{exn}\, v} \qquad \frac{}{\mathsf{raise1}\, (\mathsf{exn}\, v) \Downarrow \mathsf{exn}\, v}$$

Adding divergence

Outcome of an evaluation: termination or divergence

$$o \;:=\; \mathsf{ter}\, b \;\mid\; \mathsf{div}$$

New grammar of intermediate terms

$$e := \mathsf{trm}\, t \;\mid\; \mathsf{app1}\, o\, t \;\mid\; \mathsf{app2}\, v\, o \;\mid\; \mathsf{raise1}\, o \;\mid\; \mathsf{try1}\, o\, t$$

Evaluation rules

$$\frac{t_1 \Downarrow o_1 \qquad \mathsf{app1}\, o_1\, t_2 \Downarrow b}{\mathsf{app}\, t_1\, t_2 \Downarrow b} \qquad\qquad \frac{}{\mathsf{app1}\, \mathsf{div}\, t \Downarrow \mathsf{div}}$$

The abort predicate

We want to factorize pairs of similar rules, such as:

$$\frac{}{\mathsf{app1}\,(\mathsf{exn}\,v)\,t \Downarrow (\mathsf{exn}\,v)} \qquad \frac{}{\mathsf{app1}\,\mathsf{div}\,t \Downarrow \mathsf{div}}$$

Solution:

$$\frac{\mathsf{abort}\,o}{\mathsf{app1}\,o\,t \Downarrow o}$$

where "abort" characterizes exceptions ($\mathsf{exn}\,v$) and divergence (div).

Summary: grammars and judgments

Grammars:

$$
\begin{array}{lcl}
b & := & \mathsf{ret}\, v \mid \mathsf{exn}\, v \\
o & := & \mathsf{ter}\, b \mid \mathsf{div} \\
e & := & \mathsf{trm}\, t \mid \mathsf{app1}\, o\, t \mid \mathsf{app2}\, v\, o \mid \mathsf{raise1}\, o \mid \mathsf{try1}\, o\, t
\end{array}
$$

Judgments:

$$\mathsf{abort}\, o \qquad\qquad e \Downarrow o \qquad\qquad e \Downarrow^{\mathsf{co}} o$$

Theorem (equivalence with big-step)

$$t \Downarrow \mathit{ter}\, b \quad \Leftrightarrow \quad t \Rightarrow b$$

$$t \Downarrow^{co} \mathit{div} \quad \Leftrightarrow \quad t \Rightarrow^{\infty}$$

Summary: rules

$$\frac{}{v \Downarrow v} \qquad \frac{t_1 \Downarrow o_1 \qquad \mathsf{app1}\, o_1\, t_2 \Downarrow o}{\mathsf{app}\, t_1\, t_2 \Downarrow o} \qquad \frac{\mathsf{abort}\, o}{\mathsf{app1}\, o\, t \Downarrow o}$$

$$\frac{t_2 \Downarrow o_2 \qquad \mathsf{app2}\, v_1\, o_2 \Downarrow o}{\mathsf{app1}\, v_1\, t_2 \Downarrow o} \qquad \frac{\mathsf{abort}\, o}{\mathsf{app2}\, v\, o \Downarrow o} \qquad \frac{[x \to v]\, t \Downarrow o}{\mathsf{app2}\, (\mathsf{abs}\, x\, t)\, v \Downarrow o}$$

$$\frac{t \Downarrow o_1 \qquad \mathsf{raise1}\, o_1 \Downarrow o}{\mathsf{raise}\, t \Downarrow o} \qquad \frac{\mathsf{abort}\, o}{\mathsf{raise1}\, o \Downarrow o} \qquad \frac{}{\mathsf{raise1}\, v \Downarrow \mathsf{exn}\, v}$$

$$\frac{t_1 \Downarrow o_1 \qquad \mathsf{try1}\, o_1\, t_2 \Downarrow o}{\mathsf{try}\, t_1\, t_2 \Downarrow o} \qquad \frac{}{\mathsf{try1}\, v\, t \Downarrow v} \qquad \frac{\mathsf{app}\, t\, v \Downarrow o}{\mathsf{try1}\, (\mathsf{exn}\, v)\, t \Downarrow o}$$

$$\frac{}{\mathsf{try1}\, \mathsf{div}\, t \Downarrow \mathsf{div}}$$

Traces

Definition of traces

A trace records I/O interactions and ϵ-transitions.

A terminating program has a finite trace.
A diverging program has an infinite trace.

$$
\begin{array}{lcl}
\alpha & := & \epsilon \mid \mathsf{in}\, n \mid \mathsf{out}\, n \\
\tau & := & \mathsf{list}\, \alpha \\
\sigma & := & \mathsf{stream}\, \alpha \\
o & := & \mathsf{ter}\, \tau\, b \mid \mathsf{div}\, \sigma
\end{array}
$$

$\rightarrow$ We are not using possibly-infinite traces (coinductive lists) like Nakata and Uustalu (2009) and Danielsson (2012).

Operation on traces

Concatenation of a finite trace τ to the front

$$\tau \cdot \tau' \qquad \tau \cdot \sigma \qquad \tau \cdot o$$

Equivalence of two traces up to finite consecutive insertions of ϵ-transitions

$$\frac{o \approx o'}{\epsilon^n \cdot [\alpha] \cdot o \;\approx\; \epsilon^m \cdot [\alpha] \cdot o'}$$

Trace semantics in pretty-big-step

Every rule appends an ϵ-transtion in order to be *productive*.

$$\frac{}{v \Downarrow \mathsf{ter}\,[\epsilon]\,v} \qquad \frac{t_1 \Downarrow o_1 \qquad \mathsf{app1}\,o_1\,t_2 \Downarrow o}{\mathsf{app}\,t_1\,t_2 \Downarrow [\epsilon] \cdot o} \qquad \frac{\mathsf{abort}\,o}{\mathsf{app1}\,o\,t \Downarrow o}$$

$$\frac{t_2 \Downarrow o_2 \qquad \mathsf{app2}\,v_1\,o_2 \Downarrow o}{\mathsf{app1}\,(\mathsf{ter}\,\tau\,v_1)\,t_2 \Downarrow \tau \cdot o} \qquad \frac{\mathsf{abort}\,o}{\mathsf{app2}\,v\,o \Downarrow o}$$

$$\frac{[x \to v]\,t \Downarrow o}{\mathsf{app2}\,(\mathsf{abs}\,x\,t)\,(\mathsf{ter}\,\tau\,v) \Downarrow \tau \cdot o}$$

Trace semantics in pretty-big-step, cont.

I/O operations are recorded in the trace.

$$\frac{t \Downarrow o_1 \qquad \mathsf{write1}\, o_1 \Downarrow o}{\mathsf{write}\, t \Downarrow [\epsilon] \cdot o}$$

$$\frac{}{\mathsf{write1}\; (\mathsf{ter}\, \tau\, n) \Downarrow \mathsf{ter}\, \tau \cdot [\mathsf{out}\, n]\; tt} \qquad \frac{t \Downarrow o_1 \qquad \mathsf{read1}\, o_1 \Downarrow o}{\mathsf{read}\, t \Downarrow [\epsilon] \cdot o}$$

$$\frac{}{\mathsf{read1}\; (\mathsf{ter}\, \tau\; tt) \Downarrow \mathsf{ter}\, \tau \cdot [\mathsf{in}\, n]\; n} \qquad \frac{\mathsf{abort}\, o}{\mathsf{read1}\, o \Downarrow o} \qquad \frac{\mathsf{abort}\, o}{\mathsf{write1}\, o \Downarrow o}$$

Benefits of trace semantics

Theorem (finite traces can only be produced by finite derivations)

$$e \Downarrow^{co} \textit{ter}\, \tau\, v \qquad \Leftrightarrow \qquad e \Downarrow \textit{ter}\, \tau\, v$$

So, we do not need the inductive judgment: the coinductive one suffices.

Theorem (equivalence with big-step)

$$t \Downarrow^{co} \textit{ter}\, \tau\, b \qquad \Leftrightarrow \qquad t \Rightarrow b/\tau$$

$$t \Downarrow^{co} \textit{div}\, \sigma \qquad \Leftrightarrow \qquad t \Rightarrow^{\infty} /\sigma$$

Proofs with trace semantics: problems

A typical simulation theorem

$$[\![e]\!] \Downarrow^{\mathsf{co}} o \qquad \rightarrow \qquad \exists o'. \quad o' \approx o \quad \wedge \quad e \Downarrow^{\mathsf{co}} o'$$

Coinductive proof? No luck!

1. $\exists$ is not coinductive
2. $\wedge$ is not coinductive
3. o' is not coinductive

$\rightarrow$ Yet, coinductive reasoning is morally correct.

Pretty-big-step: scaling up

Scaling up to real languages

What's next:

- the generic abort rule
- semantics of side-effects
- semantics of loops
- formalization of core-Caml

Abort rules

Many similar abort rules: can we factorize them?

$$\frac{\mathsf{abort}\, o}{\mathsf{app1}\, o\, t \Downarrow o} \qquad \frac{\mathsf{abort}\, o}{\mathsf{app2}\, v\, o \Downarrow o} \qquad \frac{\mathsf{abort}\, o}{\mathsf{raise1}\, o \Downarrow o}$$

The generic abort rule

The auxiliary function "getout"

getout (app1 $o\,t$)	$\equiv$	Some o
getout (app2 $v\,o$)	$\equiv$	Some o
getout (raise1 o)	$\equiv$	Some o

getout (trm t)	$\equiv$	None
getout (try1 $o\,t$)	$\equiv$	None

The generic abort rule, which replaces the rules from the previous slide

$$\frac{\text{getout}\, e = \text{Some}\, o \qquad \text{abort}\, o}{e \Downarrow o}$$

Side-effects

Generalization of terminating outcomes to carry a memory store:

$$o \;:=\; \mathsf{ter}\, m\, b \;\mid\; \mathsf{div}$$

Evaluation judgment in the form $e\;_{/m} \Downarrow o$. Example rules:

$$\frac{t_1\;_{/m} \Downarrow o_1 \qquad \mathsf{app1}\, o_1\, t_2\;_{/m} \Downarrow o}{\mathsf{app}\, t_1\, t_2\;_{/m} \Downarrow o}$$

$$\frac{t_2\;_{/m'} \Downarrow o_2 \qquad \mathsf{app2}\, v_1\, o_2\;_{/m'} \Downarrow o}{\mathsf{app1}\, (\mathsf{ter}\, m'\, v_1)\, t_2\;_{/m} \Downarrow o}$$

Pretty-big-step semantics for loops

A single intermediate term "for $i\, o\, t_1\, t_2\, t_3$", where $i \in \{1, 2, 3\}$.

Evaluation rules, with the judgment $e\ _{/m} \Downarrow o$.

$$\frac{t_1\ _{/m} \Downarrow o_1 \qquad \mathsf{for}\ 1\ o_1\ t_1\ t_2\ t_3\ _{/m} \Downarrow o}{\mathsf{for}\ t_1\ t_2\ t_3\ _{/m} \Downarrow o}$$

$$\frac{}{\mathsf{for}\ 1\ (\mathsf{ret}\ m\ \mathsf{false})\ t_1\ t_2\ t_3\ _{/m'} \Downarrow \mathsf{ret}\ m\ tt}$$

$$\frac{t_3\ _{/m} \Downarrow o_3 \qquad \mathsf{for}\ 2\ o_3\ t_1\ t_2\ t_3\ _{/m} \Downarrow o}{\mathsf{for}\ 1\ (\mathsf{ret}\ m\ \mathsf{true})\ t_1\ t_2\ t_3\ _{/m'} \Downarrow o}$$

$$\frac{t_2\ _{/m} \Downarrow o_2 \qquad \mathsf{for}\ 3\ o_2\ t_1\ t_2\ t_3\ _{/m} \Downarrow o}{\mathsf{for}\ 2\ (\mathsf{ret}\ m\ tt)\ t_1\ t_2\ t_3\ _{/m'} \Downarrow o} \qquad \frac{\mathsf{for}\ t_1\ t_2\ t_3\ _{/m} \Downarrow o}{\mathsf{for}\ 3\ (\mathsf{ret}\ m\ tt)\ t_1\ t_2\ t_3\ _{/m'} \Downarrow o}$$

$$\frac{\mathsf{abort}\ o}{\mathsf{for}\ i\ o\ t_1\ t_2\ t_3\ _{/m} \Downarrow o}$$

Pretty-big-step semantics for core-Caml

Formalization of core-Caml:

booleans, integers, tuples, algebraic data types, mutable records, boolean operators (lazy and, lazy or, negation), integer operators (negation, addition, subtraction, multiplication, division), comparison operator, functions, recursive functions, applications, sequences, let-bindings, conditionals (with optional *else* branch), *for* loops and *while* loops, pattern matching (with nested patterns, *as* patterns, *or* patterns, and *when* clauses), *raise* construct, *try-with* construct with pattern matching, and assertions.

	rules	premises	tokens
Big-step without divergence	71	83	1540
Big-step with divergence	113	143	2263
Pretty-big-step	70	60	1361

→ Pretty-big-step reduces the size of the definition by 40%.
→ Pretty-big-step reduces the number of premises by more than a factor 2.

Thanks!