Divergence as State in Coinductive Big-Step Semantics

Casper Bach Poulsen and Peter D. Mosses

Swansea University, Swansea, UK
{cscbp,p.d.mosses}@swansea.ac.uk

Abstract

The coinductive interpretation of a big-step relation for a call-by-value functional language is insufficient for expressing all divergent computations. A commonly adopted alternative is to use a divergence predicate that suffers from a serious duplication problem. We consider divergence as state in coinductive big-step semantics, and show that this avoids the duplication problem. Big-step rules with divergence as state are slightly less expressive than using a divergence predicate or pretty-big-step rules, but are more concise than both.

1 Introduction and Background

Big-step semantics (also called *natural semantics* [5]) relates programs to their final results of evaluation. The standard inductive interpretation of a big-step semantics only describes the behaviour of terminating programs. In contrast, small-step rules relate intermediate configurations, allowing non-terminating behaviour to be described as infinite sequences of reduction steps. Leroy and Grall [6] observed that the coinductive interpretation of big-step rules can express some but not all diverging computations. To express diverging computations on a par with small-step semantics, Leroy and Grall used a separate divergence predicate $\overset{\infty}{\Rightarrow}$.

Instead of a separate divergence predicate, we consider a simple extension of big-step rules that augments rules to propagate a divergence state, such that once a computation enters a divergent state, all subsequent computations are also in a divergent state. A straightforward modification of Leroy and Grall's Coq proofs shows that this suffices to express diverging computations comparable to small-step semantics.[1] This allows one to use a single relation for all proofs, which should minimise the proof-burden of working with a big-step semantics.

We consider the same language as Leroy and Grall [6]: call-by-value λ-calculus extended with constants. Its syntax is:

$$\textit{Constants} \ni c ::= 0 \mid 1 \mid \ldots \qquad \textit{Variables} \ni x, y, z \qquad \textit{Terms} \ni a, b, v ::= x \mid c \mid \lambda x.a \mid ab$$

where c and $\lambda x.a$ are values. The call-by-value big-step semantics is inductively given by:

$$\frac{}{c \Rightarrow c}\ \text{Const} \qquad \frac{}{\lambda x.a \Rightarrow \lambda x.a}\ \text{Fun} \qquad \frac{a_1 \Rightarrow \lambda x.b \quad a_2 \Rightarrow v_2 \quad b[x \leftarrow v_2] \Rightarrow v}{a_1 a_2 \Rightarrow v}\ \text{App}$$

Here, $b[x \leftarrow v_2]$ denotes the capture-avoiding substitution of v_2 for all free occurrences of x in b. These rules capture only terminating computations. For example, for $\omega = (\lambda x.xx)(\lambda x.xx)$, there is no a such that $\omega \Rightarrow a$. Following Cousot and Cousot [3], Leroy and Grall define divergence by the coinductive interpretation of the rules:

$$\frac{a_1 \overset{\infty}{\Rightarrow}}{a_1 a_2 \overset{\infty}{\Rightarrow}}\ \text{App-l} \qquad \frac{a_1 \Rightarrow v \quad a_2 \overset{\infty}{\Rightarrow}}{a_1 a_2 \overset{\infty}{\Rightarrow}}\ \text{App-r} \qquad \frac{a_1 \Rightarrow \lambda x.b \quad a_2 \Rightarrow v_2 \quad b[x \leftarrow v_2] \overset{\infty}{\Rightarrow}}{a_1 a_2 \overset{\infty}{\Rightarrow}}\ \text{App-f}$$

[1] The modified Coq code is at: `http://cs.swansea.ac.uk/~cscbp/nwpt14-coq.zip`

Let $\overset{\mathsf{co}}{\Rightarrow}$ denote the relation given by the coinductive interpretation of Const, Fun, and App. If $a \overset{\mathsf{co}}{\Rightarrow} v$, then either $a \Rightarrow v$ or $a \overset{\infty}{\Rightarrow}$, but the converse does not hold, as shown by the counter-example $\omega(0\ 0)$: the coinductive interpretation of App requires that $(0\ 0)$ coevaluates to a value, which is not the case.

2 Divergence as State

Following our previous work [1] that shows how to avoid propagation of exceptions by encoding them in a stateful manner, which in turn is analogous to Charguéraud's abort rules [2], we introduce a 'divergence flag', ranged over by $\delta ::= \downarrow \mid \uparrow$. Here, $\downarrow$ denotes convergence and $\uparrow$ divergence. Consider the following rules:

$$\frac{}{c_{/\downarrow} \overset{\mathsf{d}}{\Rightarrow} c_{/\downarrow}}\ \delta\text{-Const} \qquad \frac{}{\lambda x.a_{/\downarrow} \overset{\mathsf{d}}{\Rightarrow} \lambda x.a_{/\downarrow}}\ \delta\text{-Fun} \qquad \frac{}{a_{/\uparrow} \overset{\mathsf{d}}{\Rightarrow} b_{/\uparrow}}\ \delta\text{-Div}$$

$$\frac{a_{1/\downarrow} \overset{\mathsf{d}}{\Rightarrow} \lambda x.b_{/\delta} \quad a_{2/\delta} \overset{\mathsf{d}}{\Rightarrow} v_{2/\delta'} \quad b[x \leftarrow v_2]_{/\delta'} \overset{\mathsf{d}}{\Rightarrow} v_{/\delta''}}{a_1 a_{2/\downarrow} \overset{\mathsf{d}}{\Rightarrow} v_{/\delta''}}\ \delta\text{-App}$$

Each rule except δ-Div can be automatically derived from Const, Fun, and App by letting the conclusion source be in a $\downarrow$ state, and threading the δ flag through the premises to the conclusion target in the order of evaluation (in this case, left-to-right, as illustrated by the rule δ-App). The intuition behind the δ-Div rule is that, if we are diverging, no value is produced, so we may choose to any term b as result. Under an inductive interpretation of these rules, a computation starting in a convergent state never results in divergence:

Theorem 1. $a_{/\downarrow} \overset{\mathsf{d}}{\Rightarrow} v_{/\downarrow}$ *iff* $a \Rightarrow v$.

Let $\overset{\mathsf{dco}}{\Rightarrow}$ be the coinductive counterpart to $\overset{\mathsf{d}}{\Rightarrow}$. $\overset{\mathsf{dco}}{\Rightarrow}$ is expressive enough that we can prove for any v that $\omega(0\ 0)_{/\downarrow} \overset{\mathsf{dco}}{\Rightarrow} v_{/\uparrow}$. However, while $0\ \omega \overset{\infty}{\Rightarrow}$ is provable, there is no v such that $0\ \omega_{/\downarrow} \overset{\mathsf{dco}}{\Rightarrow} v_{/\uparrow}$. The rule App-r allows the left-hand side of an application to be an arbitrary value when the right-hand side of the application diverges. $\overset{\infty}{\Rightarrow}$ reflects the behaviour specified by the following small-step rules, given by Leroy and Grall [6]:

$$\frac{v \in \mathit{Values}}{(\lambda x.a)v \to a[x \leftarrow v]}\ \beta \qquad \frac{a_1 \to a_2}{a_1 b \to a_2 b}\ \text{app-l} \qquad \frac{a \in \mathit{Values} \quad b_1 \to b_2}{a b_1 \to a b_2}\ \text{app-r}$$

In contrast to App-r and app-r, δ-App requires that the left-hand side of an application evaluates to a function. One solution is to use Charguéraud's *pretty-big-step* [2] style. Another is to modify $\overset{\infty}{\Rightarrow}$ to insist that the converging left-hand terms always give a function, thereby disallowing terms such as $0\ \omega$. Opting for this restriction, we replace app-r and App-r by:

$$\frac{b_1 \to b_2}{(\lambda x.a)b_1 \to (\lambda x.a)b_2}\ \text{app-r}' \qquad \frac{a_1 \Rightarrow \lambda x.a \quad a_2 \overset{\infty}{\Rightarrow}}{a_1 a_2 \overset{\infty}{\Rightarrow}}\ \text{App-r}'$$

Let $\to'$ and $\overset{\infty}{\Rightarrow}'$ be the results of this replacement. Using the law of excluded middle:

Theorem 2. *For any* b, $a \overset{\infty}{\Rightarrow}'$ *iff* $a_{/\downarrow} \overset{\mathsf{dco}}{\Rightarrow} b_{/\uparrow}$.

This shows that divergence as state suffices to use a standard big-step relation to express divergent computations. The straightforward extension does not introduce new rules nor require existing rules to be factored into multiple rules. Whereas Leroy and Grall use 6 rules with 9 premises (3 of which are duplicates), divergence as state uses 4 rules with 3 premises. The corresponding pretty-big-step rules use 6 rules with 5 premises.[2]

To test the applicability of divergence as state, we proved type soundness of the simply-typed typing rules of Leroy and Grall [6] relative to $\overset{\mathsf{dco}}{\Rightarrow}$. Letting $\emptyset \vdash a : T$ denote that a has type T in the empty context, we proved the following:

Theorem 3. *If $\emptyset \vdash a : T$ then there exist v and δ such that $a_{/\downarrow} \overset{\mathsf{dco}}{\Rightarrow} v_{/\delta}$.*

We leave to future work simplifying Leroy and Grall's proof structure, which uses big-step progress and preservation lemmas [6, Lemma 48 and 50], and uses the law of excluded middle. We conjecture that a constructive and simpler proof exists that exploits that $\overset{\mathsf{dco}}{\Rightarrow}$ subsumes both converging and diverging computations. Sources of inspiration for constructing such proofs include Nakata and Uustalu's work [7] on reasoning about execution traces and divergence constructively, and Hur et al.'s work [4] on parameterised coinduction.

3 Concluding Remarks

Theorem 2 shows that straightforwardly extending a big-step semantics by divergence as state suffices to prove properties about divergence. Big-step rules with divergence as state are slightly less expressive than using a divergence predicate or pretty-big-step rules, but are more concise than both. Based on these observations, we propose divergence as state as an attractive and original alternative to expressing divergence in big-step semantics.

We expect that it is possible to extend our approach to include traces in the divergence flag to obtain big-step rules similar to (but more concise than) Charguéraud's pretty-big-step rules with traces [2]. We also expect that it is possible to further augment such extended rules to obtain a semantics similar to Nakata and Uustalu's trace-based coinductive operational semantics [7]. Deciding whether these expectations hold is left to future work.

References

[1] C. Bach Poulsen & P.D. Mosses (2014): *Deriving Pretty-Big-Step Semantics from Small-Step Semantics.* In: *ESOP'14*, *LNCS* 8410, Springer, pp. 270–289.

[2] A. Charguéraud (2013): *Pretty-Big-Step Semantics.* In: *ESOP'13*, *LNCS* 7792, Springer, pp. 41–60.

[3] P. Cousot & R. Cousot (1992): *Inductive Definitions, Semantics and Abstract Interpretations.* In: *POPL'92*, ACM, pp. 83–94.

[4] C. Hur, G. Neis, D. Dreyer & V. Vafeiadis (2013): *The Power of Parameterization in Coinductive Proof.* In: *POPL'13*, ACM, pp. 193–206.

[5] G. Kahn (1987): *Natural Semantics.* In: *STACS'87*, *LNCS* 247, Springer, pp. 22–39.

[6] X. Leroy & H. Grall (2009): *Coinductive Big-Step Operational Semantics. Inf. Comput.* 207(2), pp. 284–304.

[7] K. Nakata & T. Uustalu (2009): *Trace-Based Coinductive Operational Semantics for While.* In: *TPHOLs'09*, *LNCS* 5674, Springer, pp. 375–390.

[2] An appendix with pretty-big-step rules is at: `http://cs.swansea.ac.uk/~cscbp/nwpt14-appendix.pdf`

A Divergence in Pretty-Big-Step Semantics

A pretty-big-step evaluates a single sub-term at a time. If implemented naïvely, pretty-big-step rules evaluating a single subterm at a time will be self-applicative. Inhibiting self-applicative rules requires extra structure at the syntax level. Following Charguéraud [2], we can add term constructors for distinguishing which sub-terms have been evaluated already. Alternatively, we can follow our previous work [1] and introduce term and expression constructors that make terms and values syntactically distinguishable.

A.1 Charguéraud's Pretty-Big-Step Semantics

We recall Charguéraud's definition of call-by-value λ-calculus, but where we follow Leroy and Grall in not using a separate constructor for values:

$$\mathit{Expressions} \ni e ::= o \mid \mathsf{app1}(e,e) \mid \mathsf{app2}(e,e) \qquad \mathit{Outcomes} \ni o ::= a \mid \mathsf{div}$$

Here, $a \in \mathit{Terms}$ as defined above. The pretty-big-step rules equivalent to Charguéraud's are:

$$\frac{}{c \Rightarrow c}\ \text{Const} \qquad \frac{}{\lambda x.a \Rightarrow \lambda x.a}\ \text{Fun} \qquad \frac{a_1 \Rightarrow v_1 \quad \mathsf{app1}(v_1,a_2) \Rightarrow v}{a_1 a_2 \Rightarrow v}\ \text{App}$$

$$\frac{\begin{array}{c} v_1 \in \mathit{Values} \\ a_2 \Rightarrow v_2 \quad \mathsf{app2}(v_1,v_2) \Rightarrow v \end{array}}{\mathsf{app1}(v_1,a_2) \Rightarrow v}\ \text{App1} \qquad \frac{v_2 \in \mathit{Values} \quad b[x \leftarrow v_2] \Rightarrow v}{\mathsf{app2}(\lambda x.b, v_2) \Rightarrow v}\ \text{App2}$$

$$\frac{}{\mathsf{app1}(\mathsf{div},a_2) \Rightarrow \mathsf{div}}\ \text{App1-Div} \qquad \frac{v_1 \in \mathit{Values}}{\mathsf{app1}(v_1,\mathsf{div}) \Rightarrow \mathsf{div}}\ \text{App2-Div}$$

A.2 Divergence as State in Pretty-Big-Step Semantics

Rather than augment the final results by a div term, we can use divergence as state. This avoids the App1-Div and App2-Div rules.

$$\mathit{Expressions} \ni e ::= a \mid \mathsf{app1}(e,e) \mid \mathsf{app2}(e,e)$$

$$\frac{}{c_{/\downarrow} \Rightarrow c_{/\downarrow}}\ \text{Const} \qquad \frac{}{\lambda x.a_{/\downarrow} \Rightarrow \lambda x.a_{/\downarrow}}\ \text{Fun} \qquad \frac{a_{1/\downarrow} \Rightarrow v_{1/\delta} \quad \mathsf{app1}(v_1,a_2)_{/\delta} \Rightarrow v_{/\delta'}}{a_1 a_{2/\downarrow} \Rightarrow v_{/\delta'}}\ \text{App}$$

$$\frac{\begin{array}{c} v_1 \in \mathit{Values} \\ a_{2/\downarrow} \Rightarrow v_{2/\delta} \quad \mathsf{app2}(v_1,v_2)_{/\delta} \Rightarrow v_{/\delta'} \end{array}}{\mathsf{app1}(v_1,a_2)_{/\downarrow} \Rightarrow v_{/\delta'}}\ \text{App1} \qquad \frac{v_2 \in \mathit{Values} \quad b[x \leftarrow v_2]_{/\downarrow} \Rightarrow v_{/\delta}}{\mathsf{app2}(\lambda x.b, v_2)_{/\downarrow} \Rightarrow v_{/\delta}}\ \text{App2}$$

$$\frac{}{a_{/\uparrow} \Rightarrow b_{/\uparrow}}\ \text{Div}$$