Deriving Pretty-Big-Step Semantics from Small-Step Semantics

Casper Bach Poulsen and Peter D. Mosses

Department of Computer Science, Swansea University, Swansea, UK
{cscbp,p.d.mosses}@swansea.ac.uk

Abstract. Big-step semantics for languages with abrupt termination and/or divergence suffer from a serious duplication problem, addressed by the novel 'pretty-big-step' style presented by Charguéraud at ESOP'13. Such rules are less concise than corresponding small-step rules, but they have the same advantages as big-step rules for program correctness proofs. Here, we show how to automatically derive pretty-big-step rules directly from small-step rules by 'refocusing'. This gives the best of both worlds: we only need to write the relatively concise small-step specifications, but our reasoning can be big-step as well as small-step. The use of strictness annotations to derive small-step congruence rules gives further conciseness.

Keywords: structural operational semantics, SOS, Modular SOS, pretty-big-step semantics, small-step semantics, big-step semantics, natural semantics, refocusing.

1 Introduction

Structural operational semantics (SOS) are typically given in either small-step (Plotkin 2004) or big-step (Kahn 1987) style. Big-step rules evaluate terms by relating them to their computed values, whereas small-step evaluation involves partly evaluated terms. Both styles are powerful frameworks for formalizing operational semantics, and each has its own merits and limitations. For example, small-step semantics is usually preferred for process algebras (Milner 1980), interleaving, and type soundness proofs (Pierce 2002; Wright and Felleisen 1994), whereas the big-step style is more suitable for proving correctness of program transformations (Charguéraud 2013; Leroy and Grall 2009). An equally important concern is the effort involved in specifying the semantics: rules should be concise, but comprehensible. But which style requires less effort?

The answer to this question depends not only on conciseness, but also on the application, i.e., on features of the specified language and properties that the semantics will be used to reason about. When the language involves abrupt termination, Charguéraud (2013) recently noted that big-step semantics (also called *natural semantics*) duplicate premises and rules to propagate abrupt termination and/or divergence. In contrast, the small-step style allows for more concise

Z. Shao (Ed.): ESOP 2014, LNCS 8410, pp. 270–289, 2014.

specifications involving abrupt termination, and there is no need to specify propagation of divergence. However, this would seem of little consolation if the use of the semantics requires big-step reasoning. Charguéraud provides an alternative by showing how to decompose big-step rules into simpler *pretty-big-step* rules. Such rules allow for more concise specifications without sacrificing the ability to do big-step reasoning. The style also incorporates coinductive reasoning similar to *coinductive big-step semantics* (Leroy and Grall 2009).

Table 1 illustrates the difference between big-step, pretty-big-step, and small-step SOS rules for subtracting natural numbers. In the small-step SOS rules we use a state variable a to propagate exceptions. By matching on the exception

Table 1. Comparison of big-step, pretty-big-step, and small-step SOS rules for partially defined subtraction of natural numbers ($\mathbb{N}$)

Big-step SOS $\boxed{t \Rightarrow b}$

$$t ::= \mathsf{minus}(t,t) \mid n \qquad n \in \mathbb{N} \qquad b ::= n \mid \mathsf{exc}(n)$$

$$\frac{t_1 \Rightarrow n_1 \qquad t_2 \Rightarrow n_2 \qquad n_1 \geq n_2 \qquad n = n_1 - n_2}{\mathsf{minus}(t_1,t_2) \Rightarrow n}\ [\textsc{Big1}]$$

$$\frac{t_1 \Rightarrow n_1 \qquad t_2 \Rightarrow \mathsf{exc}(n')}{\mathsf{minus}(t_1,t_2) \Rightarrow \mathsf{exc}(n')}\ [\textsc{Big2}]$$

$$\frac{t_1 \Rightarrow n_1 \qquad t_2 \Rightarrow n_2 \qquad n_1 < n_2}{\mathsf{minus}(t_1,t_2) \Rightarrow \mathsf{exc}(0)}\ [\textsc{Big3}] \qquad \frac{t_1 \Rightarrow \mathsf{exc}(n')}{\mathsf{minus}(t_1,t_2) \Rightarrow \mathsf{exc}(n')}\ [\textsc{Big4}]$$

Pretty-big-step SOS $\boxed{e \Downarrow o}$

$$t ::= \mathsf{minus}(t,t) \mid n \qquad n \in \mathbb{N} \qquad e ::= t \mid \mathsf{minus1}(o,t) \mid \mathsf{minus2}(n,o) \qquad o ::= n \mid \mathsf{exc}(n)$$

$$\frac{t_1 \Downarrow o_1 \qquad \mathsf{minus1}(o_1,t_2) \Downarrow o}{\mathsf{minus}(t_1,t_2) \Downarrow o}\ [\textsc{Pretty1}] \qquad \frac{t_2 \Downarrow o_2 \qquad \mathsf{minus2}(n_1,o_2) \Downarrow o}{\mathsf{minus1}(n_1,t_2) \Downarrow o}\ [\textsc{Pretty2}]$$

$$\frac{n_1 \geq n_2 \qquad n = n_1 - n_2}{\mathsf{minus2}(n_1,n_2) \Downarrow n}\ [\textsc{Pretty3}] \qquad \frac{n_1 < n_2}{\mathsf{minus2}(n_1,n_2) \Downarrow \mathsf{exc}(0)}\ [\textsc{Pretty4}]$$

$$\frac{\mathsf{abort}(o_1)}{\mathsf{minus1}(o_1,t_2) \Downarrow o_1}\ [\textsc{Pretty5}] \qquad \frac{\mathsf{abort}(o_2)}{\mathsf{minus2}(n_1,o_2) \Downarrow o_2}\ [\textsc{Pretty6}]$$

$$\frac{}{\mathsf{abort}(\mathsf{exc}(n))}\ [\textsc{Abort}]$$

Small-step SOS $\boxed{\langle t,a\rangle \rightarrow \langle t',a'\rangle}$

$$t ::= \mathsf{minus}(t,t) \mid n \qquad n \in \mathbb{N} \qquad a ::= \tau \mid \mathsf{exc}(n)$$

$$\frac{\langle t_1,\tau\rangle \rightarrow \langle t_1',a'\rangle}{\langle \mathsf{minus}(t_1,t_2),\tau\rangle \rightarrow \langle \mathsf{minus}(t_1',t_2),a'\rangle}\ [\textsc{Small1}]$$

$$\frac{\langle t_2,\tau\rangle \rightarrow \langle t_2',a'\rangle}{\langle \mathsf{minus}(n_1,t_2),\tau\rangle \rightarrow \langle \mathsf{minus}(n_1,t_2'),a'\rangle}\ [\textsc{Small2}]$$

$$\frac{n_1 \geq n_2 \qquad n = n_1 - n_2}{\langle \mathsf{minus}(n_1,n_2),\tau\rangle \rightarrow \langle n,\tau\rangle}\ [\textsc{Small3}] \qquad \frac{n_1 < n_2}{\langle \mathsf{minus}(n_1,n_2),\tau\rangle \rightarrow \langle 0,\mathsf{exc}(0)\rangle}\ [\textsc{Small4}]$$

state, rather than explicit exception terms, we can abruptly terminate as soon as an exception state is entered.

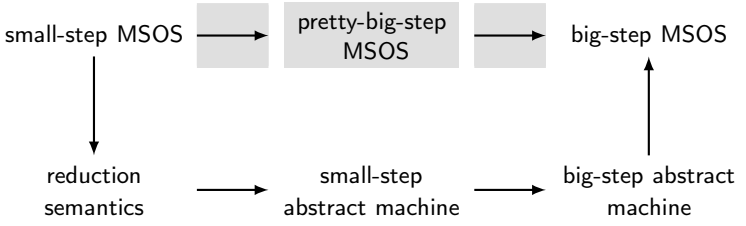
As the table illustrates, pretty-big-step rules eliminate the duplicate premise evaluating t_1 to n_1 in the big-step rules. However, both the big-step and pretty-big-step rules are less concise than their small-step counterparts – even more so if we generate small-step *congruence rules*, i.e., rules which perform a single contraction in the context of a term, such as [Small1] and [Small2], from *strictness annotations*, as used in the K-framework (Roşu and Şerbănuţă 2010).

We could ask ourselves: does (pretty-)big-step reasoning always come at the cost of less concise specifications? In this paper we answer this question in the negative. We show how we can have our cake and eat it by writing concise specifications in small-step style and automatically deriving their pretty-big-step counterparts. This allows us to do both small-step and big-step reasoning based on the same semantics. Our derivation differs in two ways from Charguéraud's manual transformation:

1. rather than transforming big-step rules into pretty-big-step rules, we transform small-step rules into pretty-big-step rules; and
2. our transformation is fully mechanical and has been automated.

Our pretty-big-step rules are derived by *refocusing* (Danvy and Nielsen 2004), which allows us to go from reduction-based (small-step) to reduction-free (big-step) evaluation (Danvy 2008b). We have previously adapted the techniques of Danvy and Nielsen to Modular SOS (MSOS) to generate efficient prototype interpreters (Bach Poulsen and Mosses 2014). Here, we extend and combine that with research in pretty-big-step semantics and modular semantics specification to make the following contributions to semantics engineering and its applications:

– We compare the effort required to extend a language with exceptions for big-step, pretty-big-step, and (modular) small-step semantics (Sect. 2). Our conclusion is that small-step MSOS specifications are more concise than corresponding pretty-big-step (SOS) and big-step (SOS and MSOS) specifications.
– We demonstrate that pretty-big-step semantics is within the range of refocusing by extending the diagram from (Danvy 2008a, p. 131) by the highlighted box and arrows[1]:



[1] Danvy (2008a) gave arrows for SOS rather than MSOS. The extension to MSOS follows from the correspondence between SOS and MSOS (Mosses 2004, Proposition 3 and 4).

By unfolding a refocused small-step semantics as described in Sect. 3 we derive pretty-big-step rules with fewer intermediate terms than Charguéraud's original formulation and which do not require auxiliary predicates.
– We adapt strictness annotations to MSOS (Sect. 4). By comparing the number of rules and premises required to specify an example language involving a considerable number of language features, we conclude that small-step MSOS with strictness annotations can be significantly more concise than the pretty-big-step style.

We claim that refocusing small-step MSOS specifications with strictness annotations gives the best of both the small-step and the big-step worlds: a concise specification format from which somewhat less concise pretty-big-step rules, amenable to the big-step proof techniques pioneered by Charguéraud (2013) and Leroy and Grall (2009), can be mechanically derived.

2 The Language and Its Semantics

We recall and contrast big-step semantics, pretty-big-step semantics, and small-step SOS and MSOS, and illustrate that small-step semantics is more concise than big-step semantics. Following Charguéraud (2013) and Leroy and Grall (2009), the language here considered is the call-by-value λ-calculus extended with constants. We consider the problem of extending this language with exceptions.

2.1 Big-Step Semantics

We give an environment-based semantics for the big-step semantics of the call-by-value λ-calculus based on closures[2]. Judgments take the form $\rho \vdash t \Rightarrow v$, asserting that, under environment ρ, t evaluates to v. Environments $\rho : \mathit{Var} \to \mathit{Val}$ map variables to values, and $\mathbb{N}$ are the natural numbers.

$$\mathit{Val} \ni v ::= n \mid \mathsf{clo}(x, t, \rho) \qquad n \in \mathbb{N} \qquad x \in \mathit{Var}$$

$$\frac{}{\rho \vdash v \Rightarrow v}\ [\text{B1}] \qquad \frac{\rho(x) = v}{\rho \vdash \mathsf{var}(x) \Rightarrow v}\ [\text{B2}] \qquad \frac{}{\rho \vdash \mathsf{abs}(x, t) \Rightarrow \mathsf{clo}(x, t, \rho)}\ [\text{B3}]$$

$$\frac{\rho \vdash t_1 \Rightarrow \mathsf{clo}(x, t, \rho') \qquad \rho \vdash t_2 \Rightarrow v \qquad \rho'[x \mapsto v] \vdash t \Rightarrow v'}{\rho \vdash \mathsf{app}(t_1, t_2) \Rightarrow v'}\ [\text{B4}]$$

Following Charguéraud (2013), we introduce an exception term for abruptly terminating evaluation. Under this extension, our judgment becomes $\rho \vdash t \Rightarrow b$ and now asserts that, under environment ρ, t results in the *behaviour* b. The grammar and rules immediately above are extended by:

$$\mathit{Behaviour} \ni b ::= v \mid \mathsf{exc}(v)$$

[2] By using closures we avoid the need to specify substitution.

$$\frac{\rho \vdash t_1 \Rightarrow \mathsf{exc}(v')}{\rho \vdash \mathsf{app}(t_1, t_2) \Rightarrow \mathsf{exc}(v')}\ [\text{B5}] \qquad \frac{\rho \vdash t_1 \Rightarrow \mathsf{clo}(x, t, \rho') \qquad \rho \vdash t_2 \Rightarrow \mathsf{exc}(v')}{\rho \vdash \mathsf{app}(t_1, t_2) \Rightarrow \mathsf{exc}(v')}\ [\text{B6}]$$

$$\frac{\rho \vdash t_1 \Rightarrow \mathsf{clo}(x, t, \rho') \qquad \rho \vdash t_2 \Rightarrow v \qquad \rho'[x \mapsto v] \vdash t \Rightarrow \mathsf{exc}(v')}{\rho \vdash \mathsf{app}(t_1, t_2) \Rightarrow \mathsf{exc}(v')}\ [\text{B7}]$$

In order to propagate the exception, the premise evaluating t_1 to a closure $\mathsf{clo}(x, t, \rho')$ becomes duplicated between the rules [B4], [B6], and [B7], the premise evaluating t_2 to a value v is duplicated between [B4] and [B7], and the number of rules defining the application construct grows from one ([B3]) to four ([B3]–[B7]). This is the *duplication problem* with abrupt termination in big-step semantics.

As illustrated by Charguéraud (2013) and Leroy and Grall (2009), a similar duplication problem arises if we express divergence following Cousot and Cousot (1992), e.g., by introducing a 'divergence relation' coinductively defined by rules similar to [B4]–[B7] above. Coinductive big-step semantics (Leroy and Grall 2009) avoids the duplication problem with divergence in big-step semantics by giving a dual (inductive and coinductive) interpretation of the same set of rules. Coinductive big-step semantics does not, however, offer any obvious solutions to the duplication problem with abrupt termination. Pretty-big-step semantics does.

2.2 Pretty-Big-Step Semantics

Charguéraud defines pretty-big-step rules as "rules that consider the evaluation of at most one subterm at a time" and are syntax-directed (Charguéraud 2013, Sect. 2.1), i.e., the initial term (conclusion source) for each rule is syntactically distinct. Using pretty-big-step rules, duplicate premises are eliminated:

$$\mathit{Outcome} \ni o ::= b \mid \mathsf{div} \qquad \mathit{Intermediate} \ni e ::= t \mid \mathsf{app1}(o, t) \mid \mathsf{app2}(v, o)$$

$$\frac{}{\rho \vdash v \Downarrow v}\ [\text{P1}] \qquad \frac{\rho(x) = v}{\rho \vdash \mathsf{var}(x) \Downarrow v}\ [\text{P2}] \qquad \frac{}{\rho \vdash \mathsf{abs}(x, t) \Downarrow \mathsf{clo}(x, t, \rho)}\ [\text{P3}]$$

$$\frac{\rho \vdash t_1 \Downarrow o_1 \qquad \rho \vdash \mathsf{app1}(o_1, t_2) \Downarrow o}{\rho \vdash \mathsf{app}(t_1, t_2) \Downarrow o}\ [\text{P4}] \qquad \frac{\rho \vdash t_2 \Downarrow o_2 \qquad \rho \vdash \mathsf{app2}(v_1, o_2) \Downarrow o}{\rho \vdash \mathsf{app1}(v_1, t_2) \Downarrow o}\ [\text{P5}]$$

$$\frac{\rho'[x \mapsto v] \vdash t \Downarrow o}{\rho \vdash \mathsf{app2}(\mathsf{clo}(x, t, \rho'), v) \Downarrow o}\ [\text{P6}] \qquad \frac{\mathsf{abort}(o)}{\rho \vdash \mathsf{app1}(o, t_2) \Downarrow o}\ [\text{P7}]$$

$$\frac{\mathsf{abort}(o)}{\rho \vdash \mathsf{app2}(v, o) \Downarrow o}\ [\text{P8}] \qquad \frac{}{\mathsf{abort}(\mathsf{exc}(v))}\ [\textsc{Abort-Exc}]$$

$$\frac{}{\mathsf{abort}(\mathsf{div})}\ [\textsc{Abort-Div}]$$

Here, e denotes *intermediate terms*. Like coinductive big-step semantics, each pretty-big-step rule has a dual interpretation: inductive and coinductive. Thus, the judgment $\rho \vdash e \Downarrow o$ asserts that, under environment ρ, t either terminates with, or *coevaluates* to, an *outcome* o. An outcome is either a behaviour b (e.g., an exception or a value), or div, the term representing divergence which is only

derivable under a coinductive interpretation. The $\mathsf{abort}(o)$ auxiliary predicate allows abrupt termination or divergence to be propagated in a generic way.

While pretty-big-step rules eliminate duplicate premises, they also introduce additional terms in the grammar of the language ($\mathsf{app1}(o,t)$ and $\mathsf{app2}(v,o)$), an auxiliary predicate ($\mathsf{abort}(o)$), and the number of rules compared to big-step semantics has increased from seven to eight.

2.3 Small-Step SOS

We now compare big-step and pretty-big-step semantics to small-step SOS. Consider the following small-step SOS rules for the call-by-value λ-calculus without exceptions:

$$\frac{\rho(x) = v}{\rho \vdash \mathsf{var}(x) \to v}\,[\text{S1}] \qquad \frac{}{\rho \vdash \mathsf{abs}(x,t) \to \mathsf{clo}(x,t,\rho)}\,[\text{S2}]$$

$$\frac{\rho \vdash t_1 \to t_1'}{\rho \vdash \mathsf{app}(t_1,t_2) \to \mathsf{app}(t_1',t_2)}\,[\text{S3}] \qquad \frac{\rho \vdash t_2 \to t_2'}{\rho \vdash \mathsf{app}(v_1,t_2) \to \mathsf{app}(v_1,t_2')}\,[\text{S4}]$$

$$\frac{\rho'[x \mapsto v] \vdash t \to t'}{\rho \vdash \mathsf{app}(\mathsf{clo}(x,t,\rho'),v) \to \mathsf{app}(\mathsf{clo}(x,t',\rho'),v)}\,[\text{S5}] \qquad \frac{}{\rho \vdash \mathsf{app}(\mathsf{clo}(x,v',\rho'),v) \to v'}\,[\text{S6}]$$

The small-step judgment $\rho \vdash t \to t'$ asserts that, under environment ρ, t makes a transition to t', which need not be a value. This formulation uses two more rules than the big-step style specification before adding exceptions (Sect. 2.1). This is in part due to the two congruence rules [S3] and [S4] which propagate the result of doing a contraction inside a subterm. Section 4.3 shows how these can be replaced by a strictness annotation.

Following Plotkin (2004), evaluation in a small-step SOS is given by (possibly infinite) sequences of transition steps in an underlying *labelled terminal transition system* (LTTS). The LTTS for the SOS with the rules [S1]–[S6] above is given by $\langle \mathit{Term}, \mathbf{1}, \to, \mathit{Val} \rangle$, where $\to \subseteq \mathit{Term} \times \mathbf{1} \times \mathit{Term}$ is the transition relation that our rules inductively define, and $\mathbf{1}$ denotes the singleton set containing a unit label that is implicitly present on all transitions. Divergence in small-step SOS corresponds to an infinite sequence of transition steps in the underlying LTTS.

To extend our small-step semantics with exceptions, we could follow Charguéraud (2013) in introducing an exception term. This would require us to introduce rules propagating exceptions similarly to the pretty-big-step rules. An alternative, following Mosses (2004), is to model exceptions as signals in the label of the transition relation. In this approach, a top-level term abruptly terminates evaluation if an exception signal is propagated to the top-level. Here, we take a different approach and model exceptions as *states* in the configurations of the underlying LTTS. Updating our relation, the judgment $\rho \vdash \langle t, a \rangle \to \langle t', a' \rangle$ asserts that, under environment ρ, the *configuration* $\langle t, a \rangle$ makes a transition to

$\langle t', a' \rangle$, where $a ::= \tau \mid \mathsf{exc}(v)$. Our small-step rules are updated to propagate the exception state:

$$\frac{\rho(x) = v}{\rho \vdash \langle \mathsf{var}(x), \tau \rangle \to \langle v, \tau \rangle}\,[\mathrm{S1}'] \qquad \frac{}{\rho \vdash \langle \mathsf{abs}(x,t), \tau \rangle \to \langle \mathsf{clo}(x,t,\rho), \tau \rangle}\,[\mathrm{S2}']$$

$$\frac{\rho \vdash \langle t_1, \tau \rangle \to \langle t_1', a' \rangle}{\rho \vdash \langle \mathsf{app}(t_1,t_2), \tau \rangle \to \langle \mathsf{app}(t_1',t_2), a' \rangle}\,[\mathrm{S3}'] \qquad \frac{\rho \vdash \langle t_2, \tau \rangle \to \langle t_2', a' \rangle}{\rho \vdash \langle \mathsf{app}(v_1,t_2), \tau \rangle \to \langle \mathsf{app}(v_1,t_2'), a' \rangle}\,[\mathrm{S4}']$$

$$\frac{\rho'[x \mapsto v] \vdash \langle t, \tau \rangle \to \langle t', a' \rangle}{\rho \vdash \langle \mathsf{app}(\mathsf{clo}(x,t,\rho'), v), \tau \rangle \to \langle \mathsf{app}(\mathsf{clo}(x,t',\rho'), v), a' \rangle}\,[\mathrm{S5}']$$

$$\frac{}{\rho \vdash \langle \mathsf{app}(\mathsf{clo}(x,v',\rho'), v), \tau \rangle \to \langle v', \tau \rangle}\,[\mathrm{S6}']$$

Here, τ indicates that no exception is being thrown. By matching on the exception state for each transition at the top-level of our LTTS, we can decide whether evaluation should continue (in case of a τ state), or whether to terminate (in case of an exception).

2.4 Small-Step Modular SOS

Unlike pretty-big-step semantics, introducing abrupt termination in the small-step SOS in previous subsection did not increase the number of rules. Unlike big-step semantics, nor did introducing abrupt termination result in duplication of premises. To introduce the exception state, we did, however, reformulate all of our rules. If we use MSOS instead of ordinary SOS, we do not need to update our rules at all. The MSOS rules corresponding to the small-step SOS rules [S1]–[S6] in the previous subsection are:

$$\frac{\rho(x) = v}{\mathsf{var}(x) \xrightarrow{\{\mathbf{env}=\rho,\text{—}\}} v}\,[\mathrm{M1}] \qquad \frac{}{\mathsf{abs}(x,t) \xrightarrow{\{\mathbf{env}=\rho,\text{—}\}} \mathsf{clo}(x,t,\rho)}\,[\mathrm{M2}]$$

$$\frac{t_1 \xrightarrow{\ell} t_1'}{\mathsf{app}(t_1,t_2) \xrightarrow{\ell} \mathsf{app}(t_1',t_2)}\,[\mathrm{M3}] \qquad \frac{t_2 \xrightarrow{\ell} t_2'}{\mathsf{app}(v_1,t_2) \xrightarrow{\ell} \mathsf{app}(v_1,t_2')}\,[\mathrm{M4}]$$

$$\frac{t \xrightarrow{\{\mathbf{env}=\rho'[x\mapsto v],\ldots\}} t'}{\mathsf{app}(\mathsf{clo}(x,t,\rho'),v) \xrightarrow{\{\mathbf{env}=\rho,\ldots\}} \mathsf{app}(\mathsf{clo}(x,t',\rho'),v)}\,[\mathrm{M5}] \qquad \frac{}{\mathsf{app}(\mathsf{clo}(x,v',\rho'),v) \xrightarrow{\{\text{—}\}} v'}\,[\mathrm{M6}]$$

The judgment $t \xrightarrow{\ell} t'$ asserts that, under label ℓ, t reduces to t'. Labels in MSOS are comprised of label components, such as $\mathbf{env} = \rho$ in the rules above, and are denoted using Standard ML syntax for record patterns (Milner et al. 1997). Whereas SOS requires auxiliary entities to be explicitly propagated, even for rules that don't explicitly use them, MSOS uses label variables to refer to label components that are not explicitly needed. In the rules above, '. . .' is an example of such a variable. The '—' in the rules above is a variable with a special meaning in MSOS: it says that no *side-effects* occur in the step. For example, if '—' refers

to, say, a pair of read-write store label components $\langle \mathbf{sto} = \sigma, \mathbf{sto}' = \sigma' \rangle$, where $\mathbf{sto} = \sigma$ is the store before the transition, and $\mathbf{sto}' = \sigma'$ is the store resulting from making the transition, the '—' variable requires that the state is not updated, i.e., that $\sigma = \sigma'$.

While side-effects (or lack hereof) on auxiliary entities in SOS are explicitly propagated, side-effects in MSOS are propagated by *label composition*. Formally, a label is a morphism in a product category, the members of the product being the label components. Propagating side-effects between transitions corresponds to composition in the product category. 'No side-effects' (or *unobservability*) is represented by identity morphisms. Recalling that composition in a product category corresponds to taking the product of compositions for each of the corresponding individual members of the product (Pierce 1991), composing two labels corresponds to propagating the side-effects for each of the underlying label components. We briefly recall the basic label component categories for MSOS and their composition principles:

Read-only: modelled by a *discrete category* where objects only have identity morphisms. This corresponds to environments which may be inspected but not changed. Composition principle for read-only entities ro: $ro \circ ro = ro$.

Read-write: modelled by a *preorder category* where morphisms between objects constitute a preorder. Corresponds to stores which may be inspected and changed by a transition. Each morphism is a pair; e.g., $\langle rw, rw' \rangle$. Composition principle for read-write entities $\langle rw, rw' \rangle$: $\langle rw', rw'' \rangle \circ \langle rw, rw' \rangle = \langle rw, rw'' \rangle$.

Write-only: modelled by a *free monoid* considered as a category with a single object. The morphisms are (possibly empty) sequences of observable actions and signals. One of the identity arrows corresponds to the unobservable action τ (the empty sequence); all others represent observable sequences of actions. Composition principle for write-only entities wo': $wo'_2 \circ wo'_1 = wo'_1 \bullet wo'_2$ where $\bullet$ is the composition operator in the monoid

By convention, readable label components are labelled by unprimed indices, such as **env**, and writable label components are labelled by primed indices, such as $\mathbf{sto}'$. For example, for the two labels $\ell_1 = \{\mathbf{env} = \rho, \mathbf{sto} = \sigma, \mathbf{sto}' = \sigma\}$ and $\ell_2 = \{\mathbf{env} = \rho, \mathbf{sto} = \sigma, \mathbf{sto}' = \sigma'\}$, their composition $\ell_2 \circ \ell_1$ is given by the label $\{\mathbf{env} = \rho, \mathbf{sto} = \sigma, \mathbf{sto}' = \sigma'\}$.

Following Mosses (2004), evaluation in MSOS corresponds to (possibly infinite) sequences of transition steps in an underlying *generalized transition system*. The generalized transition system for the MSOS given by rules [M1]–[M6] above is a tuple $\langle \mathit{Term}, \mathbb{L}, \rightarrow, \mathit{Val} \rangle$, where $\mathbb{L}$ is a product category consisting of a single discrete category, corresponding to the read-only label component $\mathbf{env} = \rho$.

In a similar style to Leroy and Grall (2009), the iteration of this GTS can be expressed by a relation $\rightarrow^*$ for which judgments take the form $t \xrightarrow{\ell}{}^* v$, asserting that term t evaluates to value v under label ℓ. $\rightarrow^*$ is defined by the rules:

$$\frac{}{v \xrightarrow{\{-\}}{}^* v}\ [\textsc{MRefl}] \qquad \frac{t \xrightarrow{\ell_1} t' \qquad t' \xrightarrow{\ell_2}{}^* v}{t \xrightarrow{\ell_2 \circ \ell_1}{}^* v}\ [\textsc{MTrans}]$$

To extend our semantics with exceptions, we extend the product category $\mathbb{L}$ by a new read-write label component $\langle \mathbf{exc} = a, \mathbf{exc}' = a' \rangle$, where $a ::= \tau \mid \mathsf{exc}(v)$. There are several ways of inhibiting further evaluation after an exception state is entered in MSOS. One is to follow the approach taken in our SOS rules and explicitly modify our rules such that each transition only matches when $\mathbf{exc} = \tau$. Another way, which does not require the modification of our transition rules, is to update the evaluation rules [MRefl] and [MTrans] as we illustrate in Sect. 3.1.

Table 2. Required number of rules, premises, and rule modifications in order to express abrupt termination. Rule modifications are counted by comparing with the corresponding semantics without abrupt termination, where we count each reformulated existing rule and each introduction of a new rule for previously defined constructs.

Variant	(exceptions)	Rules	Premises	Modifications
Big-step	terms	7 ([B1]–[B7])	10	3
	states	6	7	3
Pretty-big-step	terms	8 ([P1]–[P8])	8	2
	states	8	8	2
Small-step SOS	terms	8	4	2
	states	6 ([S1′]–[S6′])	4	6
Small-step MSOS	terms	8	4	2
	states	6 ([M1]–[M6])	4	0

Table 2 summarizes the effort required to specify and update our semantics. From this, we can see that SOS with exception labels requires fewer rules and premises to handle abrupt termination than big-step and pretty-big-step rules. This is in part due to the fact that we followed Charguéraud (2013) in using explicit exception terms rather than exception states. By refocusing our small-step MSOS rules in Sect. 3.2, we demonstrate how to derive more concise pretty-big-step rules based on small-step MSOS. Deriving pretty-big-step MSOS rules in this fashion also reduces the need for intermediate terms and auxiliary predicates.

3 From Small-Step to Pretty-Big-Step Modular SOS

After introducing some preliminary requirements, we show how to derive pretty-big-step rules from small-step rules by refocusing.

3.1 Preliminaries

To ensure the correctness of our derivation, we require that:

1. the small-step MSOS is syntax-directed; and
2. exception states are explicitly recognizable at the top-level of the semantics.

The first requirement ensures that derived pretty-big-step rules are syntax-directed. The second ensures that abrupt termination is propagated correctly in derived pretty-big-step rules.

Syntax-Directed Evaluation. Following Charguéraud (2013), rules are syntax-directed if the initial configuration (i.e., the conclusion source term) of each rule is distinct from all other rules. In MSOS rules, an initial configuration consist of the initial (conclusion source) term together with the readable label components in the conclusion. The small-step MSOS rules from the previous section are not syntax-directed: e.g., for some value v and term t, $\mathsf{app}(v, t)$ matches the conclusions of both [M3] and [M4]. Rather than introducing intermediate terms, like in Sect. 2.2, we modify the abstract syntax to distinguish terms and values:

$$\mathit{Val} \ni v ::= n \mid \mathsf{clo}(x, e, \rho) \qquad n \in \mathbb{N} \qquad x \in \mathit{Var}$$

$$\mathit{Term} \ni t ::= \mathsf{var}(x) \mid \mathsf{app}(e, e) \mid \mathsf{abs}(x, e)$$

$$\mathit{Expr} \ni e ::= \mathsf{term}(t) \mid \mathsf{val}(v)$$

Using this abstract syntax, values are no longer instances of terms. However, terms and values are both instances of *expressions* in *Expr*. To avoid the tedium of writing out the constructor names term and val each time we need them, we will leave them implicit, like Charguéraud (2013), and simply write t instead of $\mathsf{term}(t)$, and v instead of $\mathsf{val}(v)$. We revise our relations and rules from Sect. 2.4 to reflect the updated abstract syntax:

$$\frac{}{v \xrightarrow{\{-\}}{}^{*}\ v}\ [\textsc{ERefl}] \qquad \frac{t \xrightarrow{\ell_1} e \qquad e \xrightarrow{\ell_2}{}^{*}\ v}{t \xrightarrow{\ell_2 \circ \ell_1}{}^{*}\ v}\ [\textsc{ETrans}] \qquad \boxed{e \xrightarrow{\ell}{}^{*}\ e} \quad \boxed{t \xrightarrow{\ell} e}$$

$$\frac{\rho(x) = v}{\mathsf{var}(x) \xrightarrow{\{\mathbf{env}=\rho,-\}} v}\ [\text{E1}] \qquad \frac{}{\mathsf{abs}(x, e) \xrightarrow{\{\mathbf{env}=\rho,-\}} \mathsf{clo}(x, e, \rho)}\ [\text{E2}]$$

$$\frac{t_1 \xrightarrow{\ell} e_1}{\mathsf{app}(t_1, e_2) \xrightarrow{\ell} \mathsf{app}(e_1, e_2)}\ [\text{E3}] \qquad \frac{t_2 \xrightarrow{\ell} e_2}{\mathsf{app}(v_1, t_2) \xrightarrow{\ell} \mathsf{app}(v_1, e_2)}\ [\text{E4}]$$

$$\frac{t \xrightarrow{\{\mathbf{env}=\rho'[x \mapsto v],\ldots\}} e}{\mathsf{app}(\mathsf{clo}(x, t, \rho'), v) \xrightarrow{\{\mathbf{env}=\rho,\ldots\}} \mathsf{app}(\mathsf{clo}(x, e, \rho'), v)}\ [\text{E5}] \qquad \frac{}{\mathsf{app}(\mathsf{clo}(x, v', \rho'), v) \xrightarrow{\{-\}} v'}\ [\text{E6}]$$

These rules are syntax-directed: $\mathsf{app}(v, t)$ only matches the conclusion of [E4].

Exception State Recognition. Consider the extension of our language by a $\mathsf{throw}(v)$ construct for throwing exceptions:

$$t ::= \ldots \mid \mathsf{throw}(v) \qquad\qquad v ::= \ldots \mid \mathsf{unit}$$

$$\frac{}{\mathsf{throw}(v) \xrightarrow{\{\mathbf{exc}=\tau, \mathbf{exc}'=\mathsf{exc}(v),-\}} \mathsf{unit}}\ [\text{E7}]$$

We expect evaluation of the term $\mathsf{app}(\mathsf{abs}(x, \mathsf{var}(x)), \mathsf{throw}(42))$ to abruptly terminate after reducing $\mathsf{throw}(42)$. However, using [ETrans] as defined above,

this is not what happens. First, $\mathsf{abs}(x, \mathsf{var}(x))$ is evaluated to the closure $\mathsf{clo}(x, \mathsf{var}(x), \emptyset)$, where $\emptyset$ is the empty environment. The next step throws the exception, giving the subject term $\mathsf{app}(\mathsf{clo}(x, \mathsf{var}(x), \emptyset), \mathsf{unit})$ and label $\{\mathbf{exc} = \tau, \mathbf{exc'} = \mathsf{exc}(42), \ldots\}$. Rather than abruptly terminate at this point, the exception is forward propagated by label composition, whereafter evaluation of the subject term $\mathsf{app}(\mathsf{clo}(x, \mathsf{unit}, \emptyset), \mathsf{unit})$ and label $\{\mathbf{exc} = \mathsf{exc}(42), \mathbf{exc'} = a, \ldots\}$ continues. We update our evaluation rules to terminate when $\mathbf{exc} = \mathsf{exc}(v)$:

$$a ::= \tau \mid \mathsf{exc}(v)$$

$$\frac{t \xrightarrow{\{\mathbf{exc}=\tau, \mathbf{exc'}=a, X_1\}} e \qquad e \xrightarrow{\ell_2}{}^{*} e'}{t \xrightarrow{\ell_2 \circ \{\mathbf{exc}=\tau, \mathbf{exc'}=a, X_1\}}{}^{*} e'} \text{[Trans]} \qquad \frac{}{e \xrightarrow{\{\mathbf{exc}=\mathsf{exc}(v), \mathbf{exc'}=\mathsf{exc}(v), -\}}{}^{*} e} \text{[Exc]}$$

By the definition of label composition, the conclusion of the first rule only matches transitions with $\mathbf{exc} = \tau$, since the result of composing an arbitrary label with $\{\mathbf{exc} = \tau, \mathbf{exc'} = a, X_1\}$ is a label with $\mathbf{exc} = \tau$. The initial configurations for [Trans] and [Exc] are distinct and hence syntax-directed.

Evaluating the subject term $\mathsf{app}(\mathsf{clo}(x, \mathsf{var}(x), \emptyset), \mathsf{throw}(42))$ under [Trans] and [Exc] changes the exception state from τ to $\mathsf{exc}(42)$, after which only [Exc] matches the rule. Evaluation therefore abruptly terminates with label $\{\mathbf{exc} = \tau, \mathbf{exc'} = \mathsf{exc}(42), \ldots\}$ and term $\mathsf{app}(\mathsf{clo}(x, \mathsf{var}(x), \emptyset), \mathsf{unit})$.

It is equally straightforward to extend our language with a catch construct for catching and handling exceptions. We give a syntax-directed definition by introducing an $\mathsf{eq}(e, e)$ construct for checking syntactic equality for values and an $\mathsf{if}(e, e, e)$ construct for checking the outcome of the $\mathbf{exc'}$ label component:

$$t ::= \ldots \mid \mathsf{if}(e, e, e) \mid \mathsf{eq}(e, e) \mid \mathsf{catch}(e, e) \qquad v ::= \ldots \mid \mathsf{true} \mid \mathsf{false} \mid a$$

$$\frac{t \xrightarrow{\ell} e}{\mathsf{if}(t, e_1, e_2) \xrightarrow{\ell} \mathsf{if}(e, e_1, e_2)} \text{[E8]} \qquad \frac{}{\mathsf{if}(\mathsf{true}, e_1, e_2) \xrightarrow{\{-\}} e_1} \text{[E9]}$$

$$\frac{}{\mathsf{if}(\mathsf{false}, e_1, e_2) \xrightarrow{\{-\}} e_2} \text{[E10]} \qquad \frac{t_1 \xrightarrow{\ell} e_1}{\mathsf{eq}(t_1, e_2) \xrightarrow{\ell} \mathsf{eq}(e_1, e_2)} \text{[E11]}$$

$$\frac{t_2 \xrightarrow{\ell} e_2}{\mathsf{eq}(v_1, t_2) \xrightarrow{\ell} \mathsf{eq}(v_1, e_2)} \text{[E12]} \qquad \frac{v_1 = v_2}{\mathsf{eq}(v_1, v_2) \xrightarrow{\{-\}} \mathsf{true}} \text{[E13]}$$

$$\frac{v_1 \neq v_2}{\mathsf{eq}(v_1, v_2) \xrightarrow{\{-\}} \mathsf{false}} \text{[E14]} \qquad \frac{}{\mathsf{catch}(v_1, e_2) \xrightarrow{\{-\}} v_1} \text{[E15]}$$

$$\frac{t_1 \xrightarrow{\{\mathbf{exc}=\tau, \mathbf{exc'}=a, X\}} e_1}{\mathsf{catch}(t_1, e_2) \xrightarrow{\{\mathbf{exc}=\tau, \mathbf{exc'}=\tau, X\}} \mathsf{if}(\mathsf{eq}(a, \tau), \mathsf{catch}(e_1, e_2), \mathsf{app}(e_2, a))} \text{[E16]}$$

The resulting semantics is syntax-directed and explicitly recognizes abrupt termination at the top-level.

3.2 Deriving Pretty-Big-Step Rules by Refocusing

Following our previous work (Bach Poulsen and Mosses 2014), refocusing a small-step MSOS involves extending the MSOS evaluation rules from previous section by a refocusing rule. Renaming [EREFL] to [REFL], the evaluation rules extended by the [REFOCUS] rule are:

$$\frac{}{v \xrightarrow{\{-\}}{}^{*}\ v}\ [\textsc{Refl}] \qquad \frac{t \xrightarrow{\{\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\}} e \qquad e \xrightarrow{\ell_2}{}^{*}\ e'}{t \xrightarrow{\ell_2\circ\{\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\}}{}^{*}\ e'}\ [\textsc{Trans}]$$

$$\frac{}{e \xrightarrow{\{\mathbf{exc}=\mathsf{exc}(v),\mathbf{exc}'=\mathsf{exc}(v),-\}}{}^{*}\ e}\ [\textsc{Exc}] \qquad \frac{t \xrightarrow{\ell}{}^{*}\ e}{t \xrightarrow{\ell}\ e}\ [\textsc{Refocus}]$$

Introducing the [REFOCUS] rule allows evaluation to occur inside derivation trees, as opposed to always at the top-level. However, it also breaks syntax-direction: the initial configuration of [REFOCUS] matches that of every other transition rule. To get the effect of refocusing while preserving syntax-direction, we unfold the refocusing rule and replace our transition rules by the derived rules corresponding to the partial derivation for each transition rule $[r]$:

$$\cfrac{\cfrac{\cfrac{\cfrac{P_1 \quad \cdots \quad P_n}{t \xrightarrow{\{\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\}} e}\ [r] \qquad e \xrightarrow{\ell_2}{}^{*}\ e'}{t \xrightarrow{\ell_2\circ\{\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\}}{}^{*}\ e'}\ [\textsc{Trans}]}{t \xrightarrow{\ell_2\circ\{\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\}}\ e'}\ [\textsc{Refocus}]}$$

$$\sim \quad \frac{P_1 \quad \cdots \quad P_n \qquad e \xrightarrow{\ell_2}{}^{*}\ e'}{t \xrightarrow{\ell_2\circ\{\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\}}\ e'}$$

Noticing that we have to propagate the label components $\mathbf{exc}=\tau, \mathbf{exc}'=a$ many times, we introduce the notation $\|X\|$ for abbreviating $\{\mathbf{exc}=\tau, \mathbf{exc}'=a, X\}$.

Returning to our running example, *refocusing* the [E3] rule from Sect. 3.1 Gives the following partial derivation and derived (*refocused*) rule [ER3]:

$$\cfrac{\cfrac{\cfrac{\cfrac{t_1 \xrightarrow{\|X_1\|} e_1}{\mathsf{app}(t_1,e_2) \xrightarrow{\|X_1\|} \mathsf{app}(e_1,e_2)}\ [\text{E3}] \qquad \mathsf{app}(e_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(t_1,e_2) \xrightarrow{\ell_2\circ\|X_1\|}{}^{*}\ e'}\ [\textsc{Trans}]}{\mathsf{app}(t_1,e_2) \xrightarrow{\ell_2\circ\|X_1\|}\ e'}\ [\textsc{Refocus}]}$$

$$\sim \quad \frac{t_1 \xrightarrow{\|X_1\|} e_1 \qquad \mathsf{app}(e_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(t_1,e_2) \xrightarrow{\ell_2\circ\|X_1\|}\ e'}\ [\text{ER3}]$$

The refocused rules corresponding to [E1]–[E7] are:

$$\frac{\rho(x) = v}{\mathsf{var}(x) \xrightarrow{\|\mathbf{env}=\rho,-\|} v}\ [\text{ER1}] \qquad \frac{}{\mathsf{abs}(x,e) \xrightarrow{\|\mathbf{env}=\rho,-\|} \mathsf{clo}(x,e,\rho)}\ [\text{ER2}]$$

$$\frac{t_1 \xrightarrow{\|X_1\|} e_1 \qquad \mathsf{app}(e_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(t_1,e_2) \xrightarrow{\ell_2 \circ \|X_1\|} e'}\ [\text{ER3}] \qquad \frac{t_2 \xrightarrow{\|X_1\|} e_2 \qquad \mathsf{app}(v_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(v_1,t_2) \xrightarrow{\ell_2 \circ \|X_1\|} e'}\ [\text{ER4}]$$

$$\frac{t \xrightarrow{\|\mathbf{env}=\rho'[x \mapsto v],X_1\|} e \qquad \mathsf{app}(\mathsf{clo}(x,e,\rho'),v) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(\mathsf{clo}(x,t,\rho'),v) \xrightarrow{\ell_2 \circ \|\mathbf{env}=\rho,X_1\|} e'}\ [\text{ER5}]$$

$$\frac{}{\mathsf{app}(\mathsf{clo}(x,v',\rho'),v) \xrightarrow{\|-\|} v'}\ [\text{ER6}] \qquad \frac{}{\mathsf{throw}(v) \xrightarrow{\{\mathbf{exc}=\tau,\mathbf{exc}'=\mathbf{exc}(v),-\}} \mathsf{unit}}\ [\text{ER7}]$$

Our refocused rules are very closely related to pretty-big-step rules. Like pretty-big-step rules, each refocused rule:

- relates a term to a value or an exception state;
- reduces a single subterm at a time; and
- is syntax-directed.

A significant difference is that our refocused rules mutually define both $\to^*$ and $\to$. However, we can observe that each ordinary transition step ($\to$) either maps a term to an exception state, or maps a term to a value. From this, it follows that the top-level application of [Trans] has the form:

$$\frac{t \xrightarrow{\|X_1\|} e \qquad \dfrac{}{e \xrightarrow{\ell_2}{}^{*}\ e'}\ [R]}{t \xrightarrow{\ell_2 \circ \|X_1\|}{}^{*}\ e'}\ [\text{Trans}]$$

Either e is going to be a value v, in which case $[R]$=[Refl]. Otherwise, for the label $\ell_2 = \{\mathbf{exc} = a, \mathbf{exc}' = a', \ldots\}$ it is the case that $a \neq \tau$, whereby $[R]$=[Exc]. Therefore, in a semantics with refocused rules, all applications of [Trans] match the derived rule:

$$\frac{t \xrightarrow{\|X\|} e}{t \xrightarrow{\|X\|}{}^{*}\ e}\ [\text{TTrans}]$$

By applications of [Refocus] and [TTrans], each occurrence of an ordinary step ($\to$) can be replaced by a transitive step ($\to^*$). Replacing ordinary steps gives the MSOS pretty-big-step rules in Table 3. These rules describe the same language as the pretty-big-step rules given in (Charguéraud 2013, Fig. 2). In contrast to Charguéraud's pretty-big-step semantics, we have not introduced any intermediate terms or auxiliary predicates.

The correctness of the derivations presented in this section have been tested by using the MSOS Derivation Tool (Bach Poulsen and Mosses 2014) to generate and compare executable interpreters for the small-step semantics, its refocused, and its pretty-big-step counterpart. The generated interpreters and test suite are available online[3]. Sections 5 and 6 suggest future directions for a more formal treatment of correctness.

[3] `www.plancomps.org/bachpoulsen2014a`

Table 3. Derived pretty-big-step rules for [E1]–[E16]

$$\frac{}{v \xrightarrow{\{-\}}{}^{*}\ v}\ [\textsc{EPRefl}] \qquad \frac{}{e \xrightarrow{\{\mathbf{exc}=\mathbf{exc}(v),\mathbf{exc}'=\mathbf{exc}(v),-\}}{}^{*}\ e}\ [\textsc{EPExc}]$$

$$\frac{\rho(x) = v}{\mathsf{var}(x) \xrightarrow{\|\mathbf{env}=\rho,-\|}{}^{*}\ v}\ [\textsc{EP1}] \qquad \frac{}{\mathsf{abs}(x,e) \xrightarrow{\|\mathbf{env}=\rho,-\|}{}^{*}\ \mathsf{clo}(x,e,\rho)}\ [\textsc{EP2}]$$

$$\frac{t_1 \xrightarrow{\|X_1\|}{}^{*}\ e_1 \qquad \mathsf{app}(e_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(t_1,e_2) \xrightarrow{\ell_2\circ\|X_1\|}{}^{*}\ e'}\ [\textsc{EP3}] \qquad \frac{t_2 \xrightarrow{\|X_1\|}{}^{*}\ e_2 \qquad \mathsf{app}(v_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'x}{\mathsf{app}(v_1,t_2) \xrightarrow{\ell_2\circ\|X_1\|}{}^{*}\ e'}\ [\textsc{EP4}]$$

$$\frac{t \xrightarrow{\|\mathbf{env}=\rho'[x\mapsto v],X_1\|}{}^{*}\ e \qquad \mathsf{app}(\mathsf{clo}(x,e,\rho'),v) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{app}(\mathsf{clo}(x,t,\rho'),v) \xrightarrow{\ell_2\circ\|\mathbf{env}=\rho,X_1\|}{}^{*}\ e'}\ [\textsc{EP5}]$$

$$\frac{}{\mathsf{app}(\mathsf{clo}(x,v',\rho'),v) \xrightarrow{\|-\|}{}^{*}\ v'}\ [\textsc{EP6}] \qquad \frac{}{\mathsf{throw}(v) \xrightarrow{\{\mathbf{exc}=\tau,\mathbf{exc}'=\mathbf{exc}(v),-\}}{}^{*}\ \mathsf{unit}}\ [\textsc{EP7}]$$

$$\frac{t \xrightarrow{\|X_1\|}{}^{*}\ e \qquad \mathsf{if}(e,e_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{if}(t,e_1,e_2) \xrightarrow{\ell_2\circ\|X_1\|}{}^{*}\ e'}\ [\textsc{EP8}] \qquad \frac{e_1 \xrightarrow{\ell}{}^{*}\ e'}{\mathsf{if}(\mathsf{true},e_1,e_2) \xrightarrow{\ell}{}^{*}\ e'}\ [\textsc{EP9}]$$

$$\frac{e_2 \xrightarrow{\ell}{}^{*}\ e'}{\mathsf{if}(\mathsf{false},e_1,e_2) \xrightarrow{\ell}{}^{*}\ e'}\ [\textsc{EP10}] \qquad \frac{t_1 \xrightarrow{\|X_1\|}{}^{*}\ e_1 \qquad \mathsf{eq}(e_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{eq}(t_1,e_2) \xrightarrow{\ell_2\circ\|X_1\|}{}^{*}\ e'}\ [\textsc{EP11}]$$

$$\frac{t_2 \xrightarrow{\|X_1\|}{}^{*}\ e_2 \qquad \mathsf{eq}(v_1,e_2) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{eq}(v_1,t_2) \xrightarrow{\ell_2\circ\|X_1\|}{}^{*}\ e'}\ [\textsc{EP12}] \qquad \frac{v_1 = v_2}{\mathsf{eq}(v_1,v_2) \xrightarrow{\{-\}}{}^{*}\ \mathsf{true}}\ [\textsc{EP13}]$$

$$\frac{v_1 \neq v_2}{\mathsf{eq}(v_1,v_2) \xrightarrow{\{-\}}{}^{*}\ \mathsf{false}}\ [\textsc{EP14}] \qquad \frac{}{\mathsf{catch}(v_1,e_2) \xrightarrow{\{-\}}{}^{*}\ v_1}\ [\textsc{EP15}]$$

$$\frac{t_1 \xrightarrow{\|\mathbf{exc}=\tau,\mathbf{exc}'=a,X_1\|}{}^{*}\ e_1 \qquad \mathsf{if}(\mathsf{eq}(a,\tau),\mathsf{catch}(e_1,e_2),\mathsf{app}(e_2,a)) \xrightarrow{\ell_2}{}^{*}\ e'}{\mathsf{catch}(t_1,e_2) \xrightarrow{\ell_2\circ\|\mathbf{exc}=\tau,\mathbf{exc}'=\tau,X_1\|} e'}\ [\textsc{EP16}]$$

4 Scaling Up to Real Languages

Our running example in this paper has been the λ-calculus with exceptions. This section illustrates how the derivation in Sect. 3.2 scales up to other language features.

4.1 Side-Effects

We have already shown how to derive pretty-big-step rules for semantics with exceptions. Other kinds of abrupt termination can be handled in a similar way. Small-step MSOS rules with output channels (such as printing) and mutable

storage impose no additional constraints when deriving pretty-big-step rules by refocusing. To demonstrate, we extend our language with printing and ML-style references. To handle these features, we introduce two new label components: a read-write label component $\langle \mathbf{sto} = \sigma, \mathbf{sto'} = \sigma \rangle$, where $\sigma : \mathit{Loc} \to \mathit{Val}$ are stores mapping *locations* (such as memory addresses) to values; and a write-only label component $\mathbf{out'} = [v]$ containing a (possibly empty) list of printed values. The extended language is:

$$t ::= ... \mid \mathsf{print}(e) \mid \mathsf{ref}(e) \mid \mathsf{deref}(e) \mid \mathsf{assign}(e, e) \qquad v ::= ... \mid l \qquad l \in \mathit{Loc}$$

$$\frac{t \xrightarrow{\ell} e}{\mathsf{print}(t) \xrightarrow{\ell} \mathsf{print}(e)}\ [\text{E17}] \qquad \frac{}{\mathsf{print}(v) \xrightarrow{\{\mathbf{out'}=[v],-\}} \mathsf{unit}}\ [\text{E18}]$$

$$\frac{t \xrightarrow{\ell} e}{\mathsf{ref}(t) \xrightarrow{\ell} \mathsf{ref}(e)}\ [\text{E19}] \qquad \frac{l \notin \mathrm{dom}(\sigma)}{\mathsf{ref}(v) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto'}=\sigma[l \mapsto v],-\}} l}\ [\text{E20}]$$

$$\frac{t \xrightarrow{\ell} e}{\mathsf{deref}(t) \xrightarrow{\ell} \mathsf{deref}(e)}\ [\text{E21}] \qquad \frac{\sigma(l) = v}{\mathsf{deref}(l) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto'}=\sigma,-\}} v}\ [\text{E22}]$$

$$\frac{t_1 \xrightarrow{\ell} e_1}{\mathsf{assign}(t_1, e_2) \xrightarrow{\ell} \mathsf{assign}(e_1, e_2)}\ [\text{E23}] \qquad \frac{t_2 \xrightarrow{\ell} e_2}{\mathsf{assign}(l, t_2) \xrightarrow{\ell} \mathsf{assign}(l, e_2)}\ [\text{E24}]$$

$$\frac{}{\mathsf{assign}(l, v) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto'}=\sigma[l \mapsto v],-\}} v}\ [\text{E25}]$$

No modification of our evaluation rules is necessary. These syntax-directed rules are straightforwardly refocused and unfolded into pretty-big-step rules as described in Sect. 3.2.

4.2 C-Style for-Loops

Following Charguéraud (2013), we illustrate how to express a C-style for-loop construct. We recall Charguéraud's pretty-big-step rules, and compare with a corresponding small-step formulation and its derived pretty-big-step MSOS counterpart.

A C-style for-loop, $\mathsf{for}(e_1, e_2, e_3)$, continually evaluates body e_3 of a loop, until the condition e_1 no longer holds. Between each iteration of the for-loop, *incrementer* e_2 is evaluated. Charguéraud gives pretty-big-step rules that reflect this as follows:

$$t ::= \mathsf{for}(e, e, e) \mid v \qquad \mathit{Intermediate} \ni e ::= t \mid \mathsf{for}(i, o, t, t, t) \quad i \in \{1, 2, 3\}$$

$$b ::= v \mid \mathsf{exc}(v) \qquad o ::= \langle b, \sigma \rangle \mid \mathsf{div}$$

$$\frac{\langle t_1,\sigma\rangle \Downarrow o_1 \qquad \langle \mathsf{for}(1,o_1,t_1,t_2,t_3),\sigma\rangle \Downarrow o}{\langle \mathsf{for}(t_1,t_2,t_3),\sigma\rangle \Downarrow o} \qquad \frac{}{\langle \mathsf{for}(1,\langle \mathsf{false},\sigma\rangle,t_1,t_2,t_3),\sigma'\rangle \Downarrow \langle \mathsf{unit},\sigma\rangle}$$

$$\frac{\langle t_3,\sigma\rangle \Downarrow o_3 \qquad \langle \mathsf{for}(2,o_3,t_1,t_2,t_3),\sigma\rangle \Downarrow o}{\langle \mathsf{for}(1,\langle \mathsf{true},\sigma\rangle,t_1,t_2,t_3),\sigma'\rangle \Downarrow o} \qquad \frac{\langle t_2,\sigma\rangle \Downarrow o_2 \qquad \langle \mathsf{for}(3,o_2,t_1,t_2,t_3),\sigma\rangle \Downarrow o}{\langle \mathsf{for}(2,\langle \mathsf{unit},\sigma\rangle,t_1,t_2,t_3),\sigma'\rangle \Downarrow o}$$

$$\frac{\langle \mathsf{for}(t_1,t_2,t_3),\sigma\rangle \Downarrow o}{\langle \mathsf{for}(3,\langle \mathsf{unit},\sigma\rangle,t_1,t_2,t_3),\sigma'\rangle \Downarrow o} \qquad \frac{\mathsf{abort}(o)}{\langle \mathsf{for}(i,o,t_1,t_2,t_3),\sigma\rangle \Downarrow o}$$

$$\frac{}{\mathsf{abort}(\mathsf{exc}(v))}$$

In small-step MSOS, a corresponding specification of for-loops is in terms of the conditional $\mathsf{if}(e,e,e)$ defined in Sect. 3.1 rules [E8]–[E10], and sequential composition $\mathsf{seq}(e,e)$:

$$t ::= \ldots \mid \mathsf{seq}(e,e) \mid \mathsf{for}(e,e,e)$$

$$\frac{\dfrac{t_1 \xrightarrow{\ell} e_1}{\mathsf{seq}(t_1,e_2) \xrightarrow{\ell} \mathsf{seq}(e_1,e_2)}\,[\text{E26}] \qquad \dfrac{}{\mathsf{seq}(v_1,e_2) \xrightarrow{\{-\}} e_2}\,[\text{E27}]}{\mathsf{for}(e_1,e_2,e_3) \xrightarrow{\{-\}} \mathsf{if}(e_1,\mathsf{seq}(e_3,\mathsf{seq}(e_2,\mathsf{for}(e_1,e_2,e_3))),\mathsf{unit})}\,[\text{E28}]$$

Deriving the pretty-big-step MSOS rules gives:

$$\frac{t_1 \xrightarrow{\|X_1\|}{}^{*} e_1 \qquad \mathsf{seq}(e_1,e_2) \xrightarrow{\ell_2}{}^{*} e'}{\mathsf{seq}(t_1,e_2) \xrightarrow{\ell}{}^{*} e'}\,[\text{EP26}] \qquad \frac{e_2 \xrightarrow{\|X_1\|}{}^{*} e'}{\mathsf{seq}(v_1,e_2) \xrightarrow{\|X_1\|}{}^{*} e'}\,[\text{EP27}]$$

$$\frac{\mathsf{if}(e_1,\mathsf{seq}(e_3,\mathsf{seq}(e_2,\mathsf{for}(e_1,e_2,e_3))),\mathsf{unit}) \xrightarrow{\|X\|}{}^{*} e}{\mathsf{for}(e_1,e_2,e_3) \xrightarrow{\|X\|}{}^{*} e}\,[\text{EP28}]$$

These pretty-big-step rules correspond to Charguéraud's rules. In fact, we can derive Charguéraud's pretty-big-step rules directly from these rules. Replacing a rule by the derived rule(s) corresponding to all possible partial derivations is trivially correct. We can compress transitions, similar to (Danvy 2008b), by unfolding the rightmost 'continuation' premise in pretty-big-step rules. This corresponds to *striding* as described in (Bach Poulsen and Mosses 2014). Transition compressing [EP28] once gives:

$$\frac{e_1 \xrightarrow{\|X_1\|}{}^{*} e_1' \qquad \mathsf{if}(e_1',\mathsf{seq}(e_3,\mathsf{seq}(e_2,\mathsf{for}(e_1,e_2,e_3))),\mathsf{unit}) \xrightarrow{\ell_2}{}^{*} e}{\mathsf{for}(e_1,e_2,e_3) \xrightarrow{\ell_2 \circ \|X_1\|}{}^{*} e}\,[\text{EP28}']$$

$$\frac{}{\mathsf{for}(\mathsf{false},e_2,e_3) \xrightarrow{\{-\}}{}^{*} \mathsf{unit}}\,[\text{EP29}] \qquad \frac{\mathsf{seq}(e_3,\mathsf{seq}(e_2,\mathsf{for}(e_1,e_2,e_3))) \xrightarrow{\ell}{}^{*} e}{\mathsf{for}(\mathsf{true},e_2,e_3) \xrightarrow{\ell}{}^{*} e}\,[\text{EP30}]$$

If we continue doing this, we get a set of classic big-step rules. Decomposing the derived big-step rules into pretty-big-step rules, as described by Charguéraud (2013), we obtain a set of rules that coincides with the pretty-big-step semantics for C-style for loops given in the beginning of this subsection.

4.3 Strictness Annotations

Inspired by the K-framework (Roşu and Şerbănuţă 2010), we can use strictness annotations to automatically generate congruence rules. For example, for application in the λ-calculus:

$$\begin{array}{l} t ::= \ldots \\ \quad \mid \ \mathsf{app}(e, e) \qquad [\textbf{seq-strict}] \\ \quad \mid \ \ldots \end{array}$$

The **seq-strict** annotation automatically generates congruence rules for evaluating each sub-term fully in left-to-right order. Recall the original rules defining $\mathsf{app}(e, e)$:

$$\frac{t_1 \xrightarrow{\ell} e_1}{\mathsf{app}(t_1, e_2) \xrightarrow{\ell} \mathsf{app}(e_1, e_2)}\ [\text{E3}] \qquad \frac{t_2 \xrightarrow{\ell} e_2}{\mathsf{app}(v_1, t_2) \xrightarrow{\ell} \mathsf{app}(v_1, e_2)}\ [\text{E4}]$$

$$\frac{t \xrightarrow{\{\mathbf{env}=\rho'[x \mapsto v],\ldots\}} e}{\mathsf{app}(\mathsf{clo}(x, t, \rho'), v) \xrightarrow{\{\mathbf{env}=\rho,\ldots\}} \mathsf{app}(\mathsf{clo}(x, e, \rho'), v)}\ [\text{E5}] \qquad \frac{}{\mathsf{app}(\mathsf{clo}(x, v', \rho'), v) \xrightarrow{\{-\}} v'}\ [\text{E6}]$$

We can omit [E3] and [E4], since these congruence rules correspond exactly to the rules generated by **seq-strict**.

To generate congruence rules for subterm positions $n_1, n_2, \ldots$ numbered in the order they should be evaluated, we use the annotation **strict(**$n_1\ n_2\ \ldots$**)**. E.g., $\mathsf{if}(e, e, e)$ can be specified as:

$$\begin{array}{l} t ::= \ldots \\ \quad \mid \ \mathsf{if}(e, e, e) \qquad [\textbf{strict(1)}] \\ \quad \mid \ \ldots \end{array}$$

$$\frac{}{\mathsf{if}(\mathsf{true}, e_1, e_2) \xrightarrow{\{-\}} e_1}\ [\text{E9}] \qquad \frac{}{\mathsf{if}(\mathsf{false}, e_1, e_2) \xrightarrow{\{-\}} e_2}\ [\text{E10}]$$

This annotation automatically generates rule [E8] from Sect. 3.1.

Table 4. Comparison of number of rules and premises for strictness annotated small-step, ordinary small-step, and derived pretty-big-step MSOS

Variant	Explicit rules	Premises	Generated rules	Generated premises
Strictness-annotated small-step MSOS	20	9	31	20
Small-step MSOS	31	20	31	20
Pretty-big-step MSOS	30	35	30	35

Table 4 summarizes how the use of strictness annotations reduces the number of explicitly specified rules by a third. As expected, small-step specifications are

more concise than their pretty-big-step counterparts. By deriving the pretty-big-step rules automatically as described in this paper, we get the best of both worlds: a concise small-step specification format, and derived pretty-big-step rules that can be used for (pretty-)big-step reasoning.

5 Related Work

As illustrated throughout this paper, specifications in small-step MSOS require less effort to specify than corresponding big-step and pretty-big-step specifications. By automatic derivation, it is possible to apply both small-step and (pretty-)big-step reasoning to the same semantics. Many other authors have considered the relationship between small-step and big-step semantics.

Danvy et al. (2004; 2008a; 2008b) have explored this relationship by inter-deriving functional programs implementing many different semantic styles by provably correct transformations. Refocusing (Danvy and Nielsen 2004) was originally formulated for reduction semantics (Felleisen and Hieb 1992), but is also applicable to the K-framework, whose *heating* and *cooling* rules closely resemble reduction contexts (Roşu and Şerbănuţă 2010).

Recently, Ciobâcă (2013) described a means of deriving big-step semantics automatically from small-step semantics. His transformation essentially corresponds to the derivation we describe here. Unlike this work, his transformation does not describe the intermediate steps involved in the derivation, and is defined for substitution-based small-step semantics, which are transformed into substitution-based classic big-step rules. The correctness of Ciobâcă's transformation is based on notions of *star-soundness* and *star-completeness*. Comparing with Leroy and Grall's proof method for relating small-step and big-step semantics, these notions coincide with their proof method (Leroy and Grall 2009, Theorem 9)[4]. Star-soundness corresponds to the helper lemmas required for the "easy induction" used by Leroy and Grall, which holds for semicompositional semantics in the sense of Jones (2004). Similarly, star-completeness says that a big-step can be decomposed into a small-step followed by a big-step on the resulting term, corresponding to the second step of the "only if" part of Leroy and Grall's proof. The decomposition of a big-step into a small-step followed by a big-step is correct when the semantics is either confluent or deterministic, which corresponds to the unique decomposition requirement of refocusing in reduction semantics, and to Ciobâcă's requirement that the semantics is confluent.

We have taken a syntactic approach to deriving pretty-big-step semantics by describing each of the intermediate steps involved in the derivation. To ensure correct derivations, we required (Sect. 3.1) that:

1. the small-step semantics is syntax-directed; and
2. exception states are explicitly recognizable at the top-level of the semantics.

By insisting that our semantics is syntax-directed we avoid the issue of having to prove unique decomposition, as is required for refocusing in reduction semantics (Xiao et al. 2001). Syntax-direction implies determinism, which in turn

[4] See also their Coq proofs: `http://gallium.inria.fr/~xleroy/coindsem/`

implies unique decomposition. Whereas Ciobâcă and Danvy and Nielsen prove their transformations correct, we have so far relied on testing by generating executable interpreters using the MSOS Derivation Tool (Bach Poulsen and Mosses 2014) and comparing their outputs for example programs.

6 Conclusion and Future Directions

Small-step MSOS requires less effort than big-step and pretty-big-step rules to specify. We have shown that pretty-big-step semantics is within the range of refocusing, and that it is therefore possible to automatically derive pretty-big-step rules. In our examples, the derived pretty-big-step rules do not require auxiliary predicates, and are more concise than the pretty-big-step rules one would specify manually.

Future work includes exploring whether all pretty-big-step semantics are derivable by refocusing, and whether refocusing always yields a pretty-big-step semantics. A first step towards answering these questions is to mechanically verify, using, e.g., Coq, the correctness criteria for the derivation presented in Sect. 3.2[5]. Existing work by Leroy and Grall (2009), Ciobâcă (2013), and Sieczkowski et al. (2011) are notable sources of reference for aiding such mechanization.

Acknowledgements. We would like to thank Martin Churchill, Paolo Torrini, and the anonymous referees for their useful comments. This work was supported by an EPSRC grant (EP/I032495/1) to Swansea University in connection with the *PLanCompS* project (`www.plancomps.org`).

References

Bach Poulsen, C., Mosses, P.D.: Generating specialized interpreters for modular structural operational semantics. In: LOPSTR 2013. LNCS, Springer, Heidelberg (to appear, 2014)

Charguéraud, A.: Pretty-big-step semantics. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 41–60. Springer, Heidelberg (2013)

Ciobâcă, Ş.: From small-step semantics to big-step semantics, automatically. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 347–361. Springer, Heidelberg (2013)

Cousot, P., Cousot, R.: Inductive definitions, semantics and abstract interpretations. In: POPL 1992, pp. 83–94. ACM (1992)

Danvy, O.: Defunctionalized interpreters for programming languages. In: Hook, J., Thiemann, P. (eds.) ICFP 2008, pp. 131–142. ACM (2008a)

Danvy, O.: From reduction-based to reduction-free normalization. In: Koopman, P., Plasmeijer, R., Swierstra, D. (eds.) AFP 2008. LNCS, vol. 5832, pp. 66–164. Springer, Heidelberg (2009)

[5] Preliminary work on a Coq mechanization of the correctness proofs for the derivations in Sect. 3 and 4 is available online: `www.plancomps.org/bachpoulsen2014a`

Danvy, O., Nielsen, L.R.: Refocusing in reduction semantics. BRICS Research Series RS-04-26, Dept. of Comp. Sci., Aarhus University (2004)
Felleisen, M., Hieb, R.: The revised report on the syntactic theories of sequential control and state. Theor. Comput. Sci. 103(2), 235–271 (1992)
Jones, N.D.: Transformation by interpreter specialisation. Sci. Comput. Program. 52(1-3), 307–339 (2004)
Kahn, G.: Natural semantics. In: Brandenburg, F.J., Wirsing, M., Vidal-Naquet, G. (eds.) STACS 1987. LNCS, vol. 247, pp. 22–39. Springer, Heidelberg (1987)
Leroy, X., Grall, H.: Coinductive big-step operational semantics. Inf. Comput. 207(2), 284–304 (2009)
Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
Milner, R., Tofte, M., Macqueen, D.: The Definition of Standard ML. MIT Press, Cambridge (1997)
Mosses, P.D.: Modular structural operational semantics. J. Log. Algebr. Program. 60-61, 195–228 (2004)
Pierce, B.C.: Basic Category Theory for Computer Scientists. MIT Press (1991)
Pierce, B.C.: Types and programming languages. MIT Press (2002)
Plotkin, G.D.: A structural approach to operational semantics. J. Log. Algebr. Program. 60-61, 17–139 (2004)
Roşu, G., Şerbănuţă, T.F.: An overview of the K semantic framework. J. Log. Algebr. Program. 79(6), 397–434 (2010)
Sieczkowski, F., Biernacka, M., Biernacki, D.: Automating derivations of abstract machines from reduction semantics. In: Hage, J., Morazán, M.T. (eds.) IFL 2011. LNCS, vol. 6647, pp. 72–88. Springer, Heidelberg (2011)
Wright, A., Felleisen, M.: A syntactic approach to type soundness. Inf. Comput. 115(1), 38–94 (1994)
Xiao, Y., Sabry, A., Ariola, Z.M.: From syntactic theories to interpreters: Automating the proof of unique decomposition. Higher-Order and Symbolic Computation 14(4), 387–409 (2001)