Coinductive Big-Step Semantics for Concurrency

Tarmo Uustalu
Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia
tarmo@cs.ioc.ee

In a paper presented at SOS 2010 [13], we developed a framework for big-step semantics for interactive input-output in combination with divergence, based on coinductive and mixed inductive-coinductive notions of resumptions, evaluation and termination-sensitive weak bisimilarity. In contrast to standard inductively defined big-step semantics, this framework handles divergence properly; in particular, runs that produce some observable effects and then diverge, are not "lost". Here we scale this approach for shared-variable concurrency on a simple example language. We develop the metatheory of our semantics in a constructive logic.

1 Introduction

The purpose of this paper is to advocate two ideas. First, big-step operational semantics can handle divergence as well as small-step semantics, so that both terminating and diverging behaviors can be reasoned about uniformly. Big-step semantics that account for divergence properly are achieved by working with coinductive semantic entities (transcripts of possible infinite computation paths or nonwellfounded computation trees) and coinductive evaluation. Second, contrary to what is so often stated, concurrency is not inherently small-step, or at least not more inherently than any kind of effect produced incrementally during a program's run (e.g., interactive output). Big-step semantics for concurrency can be built by borrowing the suitable denotational machinery, except that we do not want to use domains and fixpoints to deal with partiality, but coinductively defined sets and corecursion. In this paper, we use resumptions, more specifically coinductive resumptions.

We build on our previous work [13] and develop two resumption-based big-step semantics for a simple imperative language with shared-variable concurrency. The metatheory of these semantics—e.g., the equivalence of evaluation in the big-step semantics to maximal multi-step reduction in a reference small-step semantics—is entirely constructive—meaning that we can compute evaluations from maximal multi-step reductions and vice versa. Moreover, evaluation is deterministic and can be turned into a computable function.

The idea that divergence can be properly accounted for by switching to coinductively defined semantic entities such as possibly infinitely delayed states or possibly infinite traces is due to Capretta [3]. The deeper underlying theory is based on completely iterative monads and has been treated in detail by Goncharov and Schröder [7].

Leroy and Grall [9] attempted to use coinductive big-step semantics to reason about both terminating and diverging program runs in the Compcert project on a formally certified compiler, but ran into certain semantic anomalies (proving the big-step and small-step semantics equivalent required the use of excluded middle, which should not be needed; infinite loops were not specifically arranged to be productive, with the effect that infinite loops with no observable effects led to finite traces, to which other traces could be appended). (Cf. also the simultaneous work by Cousot and Cousot [5].) Nakata and Uustalu [11] fixed the anomalies and arrived at a systematic account of trace-based big-step semantics for divergence in a purely sequential, side-effect-free setting (in relational and also functional styles).

Further [12, 13], they also developed a matching Hoare logic and a resumption-based big-step semantics for a combination of interactive input/output with divergence. Danielsson [6] has promoted especially functional-style coinductive big-step semantics. Ancona [2] used a coinductive big-step semantics of Java to show it type-sound in a sense that covers also divergence: if a program is type-sound, it produces a trace.

The tool of resumptions was originated by Plotkin [14] and has since been developed and used by several authors [4, 8]. An inductive trace-based big-step semantics for a concurrent language (not handling divergence) has appeared in the work of Mitchell [10].

The paper is organized as follows. In Section 2, we introduce our example language with pre-emptive scheduling, give it two resumption-based big-step semantics and relate them to a small-step semantics. In Section 3, we discuss notions of equivalence of resumptions. In Section 4, we show that our semantics can also be formulated functionally rather than relationally. In Section 5, we discuss the major alternative to resumptions—traces. We conclude in Section 6.

In Appendix A, we consider cooperative scheduling.

Haskell implementations of the functional-style semantics of Section 4 are available online at `http://cs.ioc.ee/~tarmo/papers/`, to be completed with an Agda formalization of the whole paper.

2 An example language and resumption-based semantics

2.1 Syntax

We look at a minimal language with shared-variable concurrency (cf. Amadio [1]) whose statements are given inductively by the grammar

$$s ::= x := e \mid \mathsf{skip} \mid s_0; s_1 \mid \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f \mid \mathsf{while}\ e\ \mathsf{do}\ s_t \mid s_0 \parallel s_1 \mid \mathsf{atomic}\ s \mid \mathsf{await}\ e\ \mathsf{do}\ s$$

The intention is that $s_0 \parallel s_1$ is parallel composition of s_0 and s_1 (in particular, it terminates when both branches have terminated). The statement $\mathsf{atomic}\ s$ is executed by running s atomically; the statement $\mathsf{await}\ e\ \mathsf{do}\ s$ is executed by waiting until e is true (other computations can have their chance in the meantime) and then running s atomically. Throughout the paper proper, scheduling is preemptive, with only assignments and boolean guards atomic implicitly.

In Appendix A, we look at a cooperative scheduling interpretation of the same syntax.

2.2 Big-step semantics

We will first introduce a semantics that captures all runs of a statement from a state until the closest control release points in one step. (In the next section, we will introduce a semantics that deals with return of control.)

The central semantic entities of this semantics are *resumptions* (computation trees). Resumptions are defined coinductively by the following rules (in this text, inductive definitions are shown by single rule lines, coinductive definitions are indicated by double rule-lines).

$$\frac{\sigma : state}{ret\ \sigma : res} \quad \frac{r : res}{\delta\ r : res} \quad \frac{r_0 : res \quad r_1 : res}{r_0 + r_1 : res} \quad \frac{s : stmt \quad \sigma : state}{yield\ s\ \sigma : res}$$

The resumption $ret\ \sigma$ denotes a computation that terminated in a state σ. The resumption $\delta\ r$ is a computation that first produces an unit delay (makes an internal small step) and continues then as r.

The resumption $r_0 + r_1$ is a choice between two resumptions r_0 and r_1. The resumption *yield* $s\ \sigma$ is a computation that has released control in a state σ and will further execute a statement s when (and if) it regains control (notice the presence of a syntactic entity here!). The definition being coinductive has the effect that resumptions can be non-wellfounded, i.e., computations can go on forever.

E.g., the following is a resumption that involves some internal small steps, two choices; one path terminates, one diverges, one suspends:

$$\delta^3\,(\delta^2\,(\mathit{ret}\ [x \mapsto 5]) + \delta^4\,(\delta^\infty + \delta\,(\mathit{yield}\ x := x + 7\ [x \mapsto 3])))$$

(by δ^∞ we mean the diverging resumption defined corecursively by $\delta^\infty = \delta\ \delta^\infty$).

Evaluation of a statement s relates a (pre-)state to a (post-)resumption and is defined coinductively by the rules

$$\dfrac{}{x := e, \sigma \Rightarrow \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma])} \qquad \dfrac{}{\mathsf{skip}, \sigma \Rightarrow \mathit{ret}\ \sigma} \qquad \dfrac{s_0, \sigma \Rightarrow r \quad s_1, r \Rightarrow^{\mathrm{seq}} r'}{s_0; s_1, \sigma \Rightarrow r'}$$

$$\dfrac{\sigma \models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow \delta\ (\mathit{yield}\ s_t\ \sigma)} \qquad \dfrac{\sigma \not\models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow \delta\ (\mathit{yield}\ s_f\ \sigma)}$$

$$\dfrac{\sigma \models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow \delta\ (\mathit{yield}\ (s_t; \mathsf{while}\ e\ \mathsf{do}\ s_t)\ \sigma)} \qquad \dfrac{\sigma \not\models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow \delta\ (\mathit{ret}\ \sigma)}$$

$$\dfrac{s_0, \sigma \Rightarrow r_0 \quad s_1, r_0 \Rightarrow^{\mathrm{parR}} r_0' \quad s_1, \sigma \Rightarrow r_1 \quad s_0, r_1 \Rightarrow^{\mathrm{parL}} r_1'}{s_0 \parallel s_1, \sigma \Rightarrow r_0' + r_1'} \qquad \dfrac{s, \sigma \Rightarrow r \quad r \rightsquigarrow r'}{\mathsf{atomic}\ s, \sigma \Rightarrow r'}$$

$$\dfrac{\sigma \models e \quad s, \sigma \Rightarrow r \quad r \rightsquigarrow r'}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow \delta\ r'} \qquad \dfrac{\sigma \not\models e}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow \delta\ (\mathit{yield}\ (\mathsf{await}\ e\ \mathsf{do}\ s)\ \sigma)}$$

We have made sure that internal small steps take their time by inserting unit delays at all places where assignments or boolean guards are evaluated. This makes evaluation deterministic, allowing us to turn it into a function, as we will see later in Sec. 4. The *yield*s in the rules for await, if and while signify control release points. Control release also occurs at the "midpoint" of evaluation of any sequential or parallel composition (i.e., at the termination of the first resp. faster statement). This is handled by the *ret* rules for sequential and parallel extensions of evaluation.

Sequential extension of evaluation relates a (pre-)resumption (the resumption present before some statement is evaluated) to a (post-)resumption (the total resumption after). It is defined coinductively by the rules

$$\dfrac{}{s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{seq}} \mathit{yield}\ s\ \sigma} \qquad \dfrac{s, r \Rightarrow^{\mathrm{seq}} r'}{s, \delta\ r \Rightarrow^{\mathrm{seq}} \delta\ r'} \qquad \dfrac{s, r_0 \Rightarrow^{\mathrm{seq}} r_0' \quad s, r_1 \Rightarrow^{\mathrm{seq}} r_1'}{s, r_0 + r_1 \Rightarrow^{\mathrm{seq}} r_0' + r_1'} \qquad \dfrac{}{s, \mathit{yield}\ s_0\ \sigma \Rightarrow^{\mathrm{seq}} \mathit{yield}\ (s_0; s)\ \sigma}$$

Essentially, sequential extension of evaluation is a form of coinductive prefix closure of evaluation. But, in addition, the *ret* rule inserts a control release between the termination of the first statement and the start of the second statement of a sequential composition. In the case of a *yield* pre-resumption, we simply grow the residual statement.

Parallel extension of evaluation, which also relates a resumption to a resumption, is for evaluating a given statement in parallel with a given resumption. The idea is to create an opportunity for the given statement to start when (and if) the resumption terminates or releases control. Also this relation is defined coinductively. Also here, in the base case (where the given resumption has terminated), we have a control release point.

$$\frac{}{s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{parR}} \mathit{yield}\ s\ \sigma} \quad \frac{s, r \Rightarrow^{\mathrm{parR}} r'}{s, \delta\ r \Rightarrow^{\mathrm{parR}} \delta\ r'} \quad \frac{s, r_0 \Rightarrow^{\mathrm{parR}} r'_0 \quad s, r_1 \Rightarrow^{\mathrm{parR}} r'_1}{s, r_0 + r_1 \Rightarrow^{\mathrm{parR}} r'_0 + r'_1} \quad \frac{}{s, \mathit{yield}\ s_0\ \sigma \Rightarrow^{\mathrm{parR}} \mathit{yield}\ (s_0 \parallel s)\ \sigma}$$

$$\frac{}{s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{parL}} \mathit{yield}\ s\ \sigma} \quad \frac{s, r \Rightarrow^{\mathrm{parL}} r'}{s, \delta\ r \Rightarrow^{\mathrm{parL}} \delta\ r'} \quad \frac{s, r_0 \Rightarrow^{\mathrm{parL}} r'_0 \quad s, r_1 \Rightarrow^{\mathrm{parL}} r'_1}{s, r_0 + r_1 \Rightarrow^{\mathrm{parL}} r'_0 + r'_1} \quad \frac{}{s, \mathit{yield}\ s_1\ \sigma \Rightarrow^{\mathrm{parL}} \mathit{yield}\ (s \parallel s_1)\ \sigma}$$

Finally, *closing a resumption* makes sure it does not release control. This is done by (repeatedly) "stitching" a resumption at every control release point by evaluating the residual statement from the state at this point. The corresponding relation between two resumptions is defined coinductively by

$$\frac{}{\mathit{ret}\ \sigma \rightsquigarrow \mathit{ret}\ \sigma} \quad \frac{r \rightsquigarrow r'}{\delta\ r \rightsquigarrow \delta\ r'} \quad \frac{r_0 \rightsquigarrow r'_0 \quad r_1 \rightsquigarrow r'_1}{r_0 + r_1 \rightsquigarrow r'_0 + r'_1} \quad \frac{s, \sigma \Rightarrow r \quad r \rightsquigarrow r'}{\mathit{yield}\ s\ \sigma \rightsquigarrow \delta\ r'}$$

In the last rule, the constructor *yield* does not disappear without leaving a trace, it is replaced with *delay*, corresponding to an internal small step.

To give only two smallest examples, for $s = x := 1 \parallel (x := x+2; x := x+2)$, $\sigma = [x \mapsto 0]$, we have

$$s, \sigma \Rightarrow \delta(\mathit{yield}\ (x := x+2; x := x+2)\ [x \mapsto 1]) + \delta(\mathit{yield}\ (x := 1 \parallel x := x+2)\ [x \mapsto 2])$$

while

$$\mathsf{atomic}\ s, \sigma \Rightarrow \delta^5(\mathit{ret}\ [x \mapsto 5]) + \delta^2(\delta^3(\mathit{ret}\ [x \mapsto 3]) + \delta^3(\mathit{ret}\ [x \mapsto 1]))$$

For $s = (\mathsf{await}\ x = 0\ \mathsf{do}\ x := 1) \parallel x := 2$, $\sigma = [x \mapsto 0]$, we have

$$s, \sigma \Rightarrow \delta^2(\mathit{yield}\ x := 2\ [x \mapsto 1]) + \delta^1(\mathit{yield}\ (\mathsf{await}\ x = 0\ \mathsf{do}\ x := 1)\ [x \mapsto 2])$$

whereas

$$\mathsf{atomic}\ s, \sigma \Rightarrow \delta^4(\mathit{ret}\ [x \mapsto 2]) + \delta^\infty$$

In this semantics there is no fairness, all schedules are considered. The resumption for statement $\mathsf{atomic}\ (x := 1 \parallel \mathsf{while}\ x = 0\ \mathsf{do}\ \mathsf{skip})$ and state $[x \mapsto 0]$ contains a path that never terminates. Note that fairness is a property of a path in a resumption, not of a resumption. Being an inductive property, fairness cannot be refuted based on an initial segment of a path, so unfair paths cannot be cut out of a resumption.

2.3 Giant-step semantics

An alternative to what we have considered in the previous section is to run statements beyond control release points for any states that control may potentially be returned in, i.e., for all states.

This leads to what we call a giant-step semantics here in order to have a different name for it.[1]

In this semantics, resumptions are purely semantic, they do not contain any statement syntax. They are defined as before, except that the *yield* constructor is typed differently.

$$\frac{\sigma : \mathit{state}}{\mathit{ret}\ \sigma : \mathit{res}_{\mathrm{g}}} \quad \frac{r : \mathit{res}}{\delta\ r : \mathit{res}_{\mathrm{g}}} \quad \frac{r_0 : \mathit{res}_{\mathrm{g}} \quad r_1 : \mathit{res}_{\mathrm{g}}}{r_0 + r_1 : \mathit{res}_{\mathrm{g}}} \quad \frac{k : \mathit{state} \to \mathit{res}_{\mathrm{g}} \quad \sigma : \mathit{state}}{\mathit{yield}\ k\ \sigma : \mathit{res}_{\mathrm{g}}}$$

[1]One might, of course, argue, that what I have called the "big-step" semantics here should be called "medium-step", and the "giant-step" semantics should be called "big-step". I would not disagree at all. My choice of terminology here was motivated by the intuition that "big-step" evaluation should run a statement to its completion. When a statement's run has reached a control release point, it is complete in the sense that it cannot run further on its own; what will happen further depends on the scheduler (it might even be unfair and not return control to it at all). Note, however, that big-step and giant-step evaluation agree fully for statements of the form $\mathsf{atomic}\ s$.

yield $k\ \sigma$ is a resumption that has released control in a given state σ and, when returned control in some state σ', will continue as $k\ \sigma'$. We call functions from states to resumptions *continuations*.

Evaluation is defined essentially as before, but with appropriate adjustments, as what were residual statements must now be evaluated.

$$\begin{array}{c} \\ \hline\hline x := e, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma]) \end{array}$$

$$\begin{array}{c} \\ \hline\hline \mathsf{skip}, \sigma \Rightarrow_{\mathrm{g}} \mathit{ret}\ \sigma \end{array} \qquad \begin{array}{c} s_0, \sigma \Rightarrow_{\mathrm{g}} r \quad s_1, r \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} r' \\ \hline\hline s_0; s_1, \sigma \Rightarrow_{\mathrm{g}} r' \end{array}$$

$$\begin{array}{c} \sigma \models e \quad \forall \sigma'. s_t, \sigma' \Rightarrow_{\mathrm{g}} k\ \sigma' \\ \hline\hline \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ k\ \sigma) \end{array} \quad \begin{array}{c} \sigma \not\models e \quad \forall \sigma'. s_f, \sigma' \Rightarrow_{\mathrm{g}} k\ \sigma' \\ \hline\hline \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ k\ \sigma) \end{array}$$

$$\begin{array}{c} \sigma \models e \quad \forall \sigma'.\ s_t, \sigma' \Rightarrow_{\mathrm{g}} k\ \sigma' \quad \forall \sigma'.\ \mathsf{while}\ e\ \mathsf{do}\ s_t, k\ \sigma' \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} k'\ \sigma' \\ \hline\hline \mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ k'\ \sigma) \end{array} \quad \begin{array}{c} \sigma \not\models e \\ \hline\hline \mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{ret}\ \sigma) \end{array}$$

$$\begin{array}{c} s_0, \sigma \Rightarrow_{\mathrm{g}} r_0 \quad \forall \sigma'. s_1, \sigma' \Rightarrow_{\mathrm{g}} k_1\ \sigma' \quad k_1, r_0 \gg_{\mathrm{g}}^{\mathrm{R}} r_0' \quad s_1, \sigma \Rightarrow_{\mathrm{g}} r_1 \quad \forall \sigma'. s_0, \sigma' \Rightarrow_{\mathrm{g}} k_0\ \sigma' \quad k_0, r_1 \gg_{\mathrm{g}}^{\mathrm{L}} r_1' \\ \hline\hline s_0 \parallel s_1, \sigma \Rightarrow_{\mathrm{g}} r_0' + r_1' \end{array}$$

$$\begin{array}{c} s, \sigma \Rightarrow_{\mathrm{g}} r \quad r \leadsto_{\mathrm{g}} r' \\ \hline\hline \mathsf{atomic}\ s, \sigma \Rightarrow_{\mathrm{g}} r' \end{array}$$

$$\begin{array}{c} \sigma \models e \quad s, \sigma \Rightarrow_{\mathrm{g}} r \quad r \leadsto_{\mathrm{g}} r' \\ \hline\hline \mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow_{\mathrm{g}} \delta\ r' \end{array} \quad \begin{array}{c} \sigma \not\models e \quad \forall \sigma'.\ \mathsf{await}\ e\ \mathsf{do}\ s, \sigma' \Rightarrow_{\mathrm{g}} k\ \sigma' \\ \hline\hline \mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ k\ \sigma) \end{array}$$

In sequential extension of evaluation, the rule for *yield* is now similar to those for δ and $+$, so we are dealing with a proper coinductive prefix closure of the evaluation relation modulo the extra *yield* constructor in the *ret* rule to cater for control release at the midpoint of evaluation of a sequential composition.

$$\begin{array}{c} \forall \sigma'.\ s, \sigma' \Rightarrow_{\mathrm{g}} k \sigma' \\ \hline\hline s, \mathit{ret}\ \sigma \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} \mathit{yield}\ k\ \sigma \end{array} \quad \begin{array}{c} s, r \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} r' \\ \hline\hline s, \delta\ r \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} \delta\ r' \end{array} \quad \begin{array}{c} s, r_0 \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} r_0' \quad s, r_1 \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} r_1' \\ \hline\hline s, r_0 + r_1 \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} r_0' + r_1' \end{array} \quad \begin{array}{c} \forall \sigma'.\ s, k\ \sigma' \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} k'\ \sigma' \\ \hline\hline s, \mathit{yield}\ k\ \sigma \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} \mathit{yield}\ k'\ \sigma \end{array}$$

Instead of parallel extension of evaluation, we define *merging a continuation into a resumption*.

$$\begin{array}{c} \\ \hline\hline k, \mathit{ret}\ \sigma \gg_{\mathrm{g}}^{\mathrm{R}} \mathit{yield}\ k\ \sigma \end{array} \quad \begin{array}{c} k, r \gg_{\mathrm{g}}^{\mathrm{R}} r' \\ \hline\hline k, \delta\ r \gg_{\mathrm{g}}^{\mathrm{R}} \delta\ r' \end{array} \quad \begin{array}{c} k, r_0 \gg_{\mathrm{g}}^{\mathrm{R}} r_0' \quad k, r_1 \gg_{\mathrm{g}}^{\mathrm{R}} r_1' \\ \hline\hline k, r_0 + r_1 \gg_{\mathrm{g}}^{\mathrm{R}} r_0' + r_1' \end{array}$$

$$\begin{array}{c} \forall \sigma'.\ k, k_0\ \sigma' \gg_{\mathrm{g}}^{\mathrm{R}} k_0'\ \sigma' \quad \forall \sigma'.\ k_0, k\ \sigma' \gg_{\mathrm{g}}^{\mathrm{L}} k_1'\ \sigma' \\ \hline\hline k, \mathit{yield}\ k_0\ \sigma \gg_{\mathrm{g}}^{\mathrm{R}} \mathit{yield}\ (\lambda \sigma'. k_0'\ \sigma' + k_1'\ \sigma')\ \sigma \end{array}$$

$$\begin{array}{c} \\ \hline\hline k, \mathit{ret}\ \sigma \gg_{\mathrm{g}}^{\mathrm{L}} \mathit{yield}\ k\ \sigma \end{array} \quad \begin{array}{c} k, r \gg_{\mathrm{g}}^{\mathrm{L}} r' \\ \hline\hline k, \delta\ r \gg_{\mathrm{g}}^{\mathrm{L}} \delta\ r' \end{array} \quad \begin{array}{c} k, r_0 \gg_{\mathrm{g}}^{\mathrm{L}} r_0' \quad k, r_1 \gg_{\mathrm{g}}^{\mathrm{L}} r_1' \\ \hline\hline k, r_0 + r_1 \gg_{\mathrm{g}}^{\mathrm{L}} r_0' + r_1' \end{array}$$

$$\begin{array}{c} \forall \sigma'.\ k_1, k\ \sigma' \gg_{\mathrm{g}}^{\mathrm{R}} k_0'\ \sigma' \quad \forall \sigma'.\ k, k_1\ \sigma' \gg_{\mathrm{g}}^{\mathrm{L}} k_1'\ \sigma' \\ \hline\hline k, \mathit{yield}\ k_1\ \sigma \gg_{\mathrm{g}}^{\mathrm{L}} \mathit{yield}\ (\lambda \sigma'. k_0'\ \sigma' + k_1'\ \sigma')\ \sigma \end{array}$$

Here, in the rules for *yield*, we construct continuations corresponding to evaluating suitable $\parallel$ statements from any given states.

Closing a resumption is straightforward. To close a *yield* resumption, we apply the given continuation to the given state, close the resulting resumption and add a unit delay.

$$\frac{}{ret\ \sigma \leadsto_g ret\ \sigma} \quad \frac{r \leadsto_g r'}{\delta\ r \leadsto_g \delta\ r'} \quad \frac{r_0 \leadsto_g r_0' \quad r_1 \leadsto_g r_1'}{r_0 + r_1 \leadsto_g r_0' + r_1'} \quad \frac{k\,\sigma \leadsto_g r}{yield\ k\ \sigma \leadsto_g \delta\ r}$$

For example, $s = x := 1 \parallel (x := x+2; x := x+2)$, $\sigma = [x \mapsto 0]$, we have

$$\begin{aligned} s, \sigma \Rightarrow_g\ & \delta(yield\ (\lambda\sigma'.\,\delta(yield\ (\lambda\sigma''.\,\delta(ret\ \sigma''[x \mapsto \sigma''\ x+2]))\ [x \mapsto \sigma'\ x+2]))\ [x \mapsto 1]) \\ & +\delta(yield\ (\lambda\sigma'.\,\delta(yield\ (\lambda\sigma''.\,\delta(ret\ \sigma''[x \mapsto \sigma''\ x+2]))\ [x \mapsto 1]) \\ & \qquad +\delta(yield\ (\lambda\sigma''.\,\delta(ret\ [x \mapsto 1]))\ [x \mapsto \sigma'\ x+2]))\ [x \mapsto 2]) \end{aligned}$$

2.4 Small-step semantics

To validate the big-step semantics of Sec. 2.2, we can compare it to a small-step semantics.

To get a close match with the big-step semantics, where we capture all runs of a program in a single resumption, we give a (perhaps somewhat nonstandard) small-step semantics that makes it possible to keep track of all runs of a statement at once.

This semantics works with *extended configurations*. They are defined as follows. (We use the notation of an inductive definition, but in fact this datatype is a simple disjoint union.)

$$\frac{\sigma : state}{ret\ \sigma : xcfg} \quad \frac{s : stmt \quad \sigma : state}{\delta\ (s,\sigma) : xcfg} \quad \frac{s_0 : stmt \quad \sigma_0 : state \quad s_1 : stmt \quad \sigma_1 : state}{(s_0,\sigma_0) + (s_1,\sigma_1) : xcfg} \quad \frac{s : stmt \quad \sigma : state}{yield\ s\ \sigma : xcfg}$$

ret σ is a terminated computation. $\delta\ (s,\sigma)$ is a computation that after an internal step is in a state σ and has s to execute yet. $(s_0,\sigma_0)+(s_1,\sigma_1)$ is a computation that makes a choice and is then in a state σ_0 with s_0 to execute or in a state σ_1 with s_1 to execute. *yield* $s\ \sigma$ is a computation that has released control in a state σ and has s to execute when (and if) it regains control.

Reduction relates a state to an extended configuration and is defined inductively(!). So small steps are justified by finite derivations.

$$\frac{}{x := e, \sigma \to \delta\ (\mathsf{skip}, \sigma[x \mapsto [\![e]\!]\ \sigma])}$$

$$\frac{}{\mathsf{skip}, \sigma \to ret\ \sigma}$$

$$\frac{s_0, \sigma \to ret\ \sigma'}{s_0; s_1, \sigma \to yield\ s_1 \sigma'} \quad \frac{s_0, \sigma \to \delta\ (s_0', \sigma')}{s_0; s_1, \sigma \to \delta\ (s_0'; s_1, \sigma')}$$

$$\frac{s_0, \sigma \to (s_{00}, \sigma_0) + (s_{01}, \sigma_1)}{s_0; s_1, \sigma \to (s_{00}; s_1, \sigma_0) + (s_{01}; s_1, \sigma_1)} \quad \frac{s_0, \sigma \to yield\ s_0'\ \sigma'}{s_0; s_1, \sigma \to yield\ (s_0'; s_1)\ \sigma'}$$

$$\frac{\sigma \models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \to \delta\ (\mathsf{skip}; s_t, \sigma)} \quad \frac{\sigma \not\models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \to \delta\ (\mathsf{skip}; s_f, \sigma)}$$

$$\frac{\sigma \models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \to \delta\ (\mathsf{skip}; (s_t; \mathsf{while}\ e\ \mathsf{do}\ s_t), \sigma)} \quad \frac{\sigma \not\models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \to \delta\ (\mathsf{skip}, \sigma)}$$

$$\frac{}{s_0 \parallel s_1, \sigma \to (s_0 \lfloor\!\lfloor s_1, \sigma) + (s_0 \rfloor\!\rfloor s_1, \sigma)}$$

$$\frac{s_0, \sigma \to \mathit{ret}\ \sigma'}{s_0 \lfloor\!\lfloor s_1, \sigma \to \mathit{yield}\ s_1 \sigma'} \qquad \frac{s_0, \sigma \to \delta\ (s_0', \sigma')}{s_0 \lfloor\!\lfloor s_1, \sigma \to \delta\ (s_0' \lfloor\!\lfloor s_1, \sigma')}$$

$$\frac{s_0, \sigma \to (s_{00}, \sigma_0) + (s_{01}, \sigma_1)}{s_0 \lfloor\!\lfloor s_1, \sigma \to (s_{00} \lfloor\!\lfloor s_1, \sigma_0) + (s_{01} \lfloor\!\lfloor s_1, \sigma_1)} \qquad \frac{s_0, \sigma \to \mathit{yield}\ s_0'\ \sigma'}{s_0 \lfloor\!\lfloor s_1, \sigma \to \mathit{yield}\ (s_0' \parallel s_1)\ \sigma'}$$

$$\frac{s_1, \sigma \to \mathit{ret}\ \sigma'}{s_0 \rfloor\!\rfloor s_1, \sigma \to \mathit{yield}\ s_0 \sigma'} \qquad \frac{s_1, \sigma \to \delta\ (s_1', \sigma')}{s_0 \rfloor\!\rfloor s_1, \sigma \to \delta\ (s_0 \rfloor\!\rfloor s_1', \sigma')}$$

$$\frac{s_1, \sigma \to (s_{10}, \sigma_0) + (s_{11}, \sigma_1)}{s_0 \rfloor\!\rfloor s_1, \sigma \to (s_0 \rfloor\!\rfloor s_1, \sigma_{10}) + (s_0 \rfloor\!\rfloor s_1, \sigma_{11})} \qquad \frac{s_1, \sigma \to \mathit{yield}\ s_1'\ \sigma'}{s_0 \rfloor\!\rfloor s_1, \sigma \to \mathit{yield}\ (s_0 \parallel s_1')\ \sigma'}$$

$$\frac{s, \sigma \to \mathit{ret}\ \sigma'}{\mathsf{atomic}\ s, \sigma \to \mathit{ret}\ \sigma'} \qquad \frac{s, \sigma \to \delta\ (s', \sigma')}{\mathsf{atomic}\ s, \sigma \to \delta\ (\mathsf{atomic}\ s', \sigma')}$$

$$\frac{s, \sigma \to (s_0, \sigma_0) + (s_1, \sigma_1)}{\mathsf{atomic}\ s, \sigma \to (\mathsf{atomic}\ s_0, \sigma_0) + (\mathsf{atomic}\ s_1, \sigma_1)} \qquad \frac{s, \sigma \to \mathit{yield}\ s'\ \sigma'}{\mathsf{atomic}\ s, \sigma \to \delta\ (\mathsf{atomic}\ s', \sigma')}$$

$$\frac{\sigma \models e}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \to \delta\ (\mathsf{atomic}\ s, \sigma)} \qquad \frac{\sigma \not\models e}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \to \delta\ (\mathsf{skip}; \mathsf{await}\ e\ \mathsf{do}\ s, \sigma)}$$

Notice that $\mathsf{skip}; s$ differs from s by allowing a control release before s is started. To use some device like this is unavoidable, if we want the reduction relation to capture exactly one small step leading to a configuration. We have also used auxiliary statement forms $s_0 \lfloor\!\lfloor s_1$ and $s_0 \rfloor\!\rfloor s_1$ which are like parallel composition except that s_0 resp. s_1 makes the first small step.

Maximal multi-step reduction relates a state to a resumption and is defined coinductively:

$$\frac{s, \sigma \to \mathit{ret}\ \sigma'}{s, \sigma \to^{\mathrm{m}} \mathit{ret}\ \sigma'} \qquad \frac{s, \sigma \to \delta\ (s', \sigma') \quad s', \sigma' \to^{\mathrm{m}} r}{s, \sigma \to^{\mathrm{m}} \delta\ r}$$

$$\frac{s, \sigma \to (s_0, \sigma_0) + (s_1, \sigma_1) \quad s_0, \sigma_0 \to^{\mathrm{m}} r_0 \quad s_1, \sigma_1 \to^{\mathrm{m}} r_1}{s, \sigma \to^{\mathrm{m}} r_0 + r_1} \qquad \frac{s, \sigma \to \mathit{yield}\ s'\ \sigma'}{s, \sigma \to^{\mathrm{m}} \mathit{yield}\ s'\ \sigma'}$$

It applies single-step reduction repeatedly as many times as possible viewing *ret* and *yield* configurations as terminal and develops a resumption.

Evaluation of the big-step semantics agrees with maximal multi-step reduction: $s, \sigma \Rightarrow r$ iff $s, \sigma \to^{\mathrm{m}} r$.

A variation of maximal multi-step reduction that also reduces under *yield*s develops a resumption of the giant-step semantics of Sec. 2.3.

$$\frac{s, \sigma \to \mathit{ret}\ \sigma'}{s, \sigma \to^{\mathrm{m}}_{\mathrm{g}} \mathit{ret}\ \sigma'} \qquad \frac{s, \sigma \to \delta\ (s', \sigma') \quad s', \sigma' \to^{\mathrm{m}}_{\mathrm{g}} r}{s, \sigma \to^{\mathrm{m}}_{\mathrm{g}} \delta\ r}$$

$$\frac{s, \sigma \to (s_0, \sigma_0) + (s_1, \sigma_1) \quad s_0, \sigma_0 \to^{\mathrm{m}}_{\mathrm{g}} r_0 \quad s_1, \sigma_1 \to^{\mathrm{m}}_{\mathrm{g}} r_1}{s, \sigma \to^{\mathrm{m}}_{\mathrm{g}} r_0 + r_1} \qquad \frac{s, \sigma \to \mathit{yield}\ s'\ \sigma' \quad \forall \sigma''. s', \sigma'' \to^{\mathrm{m}}_{\mathrm{g}} k\ \sigma''}{s, \sigma \to^{\mathrm{m}}_{\mathrm{g}} \mathit{yield}\ k\ \sigma'}$$

Evaluation of the giant-step semantics agrees with this variation of maximal multi-step reduction: $s, \sigma \Rightarrow_{\mathrm{g}} r$ iff $s, \sigma \to^{\mathrm{m}}_{\mathrm{g}} r$.

3 Equivalences of resumptions

When are two resumptions to be considered equivalent? This depends on the purpose at hand. The finest sensible notion is *strong bisimilarity* defined for big-step resumptions coinductively by the rules

$$\frac{}{\mathit{ret}\ \sigma \sim \mathit{ret}\ \sigma} \qquad \frac{r \sim r_*}{\delta\ r \sim \delta\ r_*} \qquad \frac{r_0 \sim r_{0*} \quad r_1 \sim r_{1*}}{r_0 + r_1 \sim r_{0*} + r_{1*}} \qquad \frac{}{\mathit{yield}\ s\ \sigma \sim \mathit{yield}\ s\ \sigma}$$

Classically, this predicate is just equality of big-step resumptions. But in intensional type theory, propositional equality is stronger; strong bisimilarity as just defined does not imply propositional equality[2].

Strong bisimilarity in the sense just defined may feel entirely uninteresting. Yet it is meaningful and important constructively. E.g., big-step evaluation is deterministic up to strong bisimilarity, but not up to propositional equality.

For giant-step resumptions, strong bisimilarity is defined by the rules

$$\frac{}{\mathit{ret}\ \sigma \sim_{\mathrm{g}} \mathit{ret}\ \sigma} \qquad \frac{r \sim_{\mathrm{g}} r_*}{\delta\ r \sim_{\mathrm{g}} \delta\ r_*} \qquad \frac{r_0 \sim_{\mathrm{g}} r_{0*} \quad r_1 \sim_{\mathrm{g}} r_{1*}}{r_0 + r_1 \sim_{\mathrm{g}} r_{0*} + r_{1*}} \qquad \frac{\forall \sigma'.\, k\ \sigma' \sim_{\mathrm{g}} k_*\ \sigma'}{\mathit{yield}\ k\ \sigma \sim_{\mathrm{g}} \mathit{yield}\ k_*\ \sigma}$$

The useful coarser notions ignore order and multiplicity of choices (strong bisimilarity as in process algebras), exact durations of finite delays (termination-sensitive weak bisimilarity) or both. The definition of termination-sensitive weak bisimilarity requires combining or mixing induction and coinduction, with several caveats to avoid. First, it is easy to misdefine weak bisimilarity so that it equates any resumption with the divergent resumption and therefore all resumptions. Second, a fairly attractive definition fails to give reflexivity without the use of excluded middle, which is a warning that the definition is not the "right one" from the constructive point of view.

Let us look at the definition of weak bisimilarity for big-step resumptions. First we define *convergence* of a resumption inductively by the rules

$$\frac{}{\mathit{ret}\ \sigma \downarrow \mathit{ret}\ \sigma} \qquad \frac{r \downarrow r'}{\delta\ r \downarrow r'} \qquad \frac{r_0 \downarrow r'_0 \quad r_1 \downarrow r'_1}{r_0 + r_1 \downarrow r'_0 + r'_1} \qquad \frac{}{\mathit{yield}\ s\ \sigma \downarrow \mathit{yield}\ s\ \sigma}$$

Intuitively, a resumption r converges to another resumption r' if r' is a *ret*, $+$ or *yield* resumption and can be obtained from r by removing a finite number of initial unit delays.

We also define *divergence* coinductively by the rule

$$\frac{r \uparrow}{\delta\ r \uparrow}$$

Now *termination-sensitive weak bisimilarity* is defined coinductively by the rules

$$\frac{r \downarrow r' \quad r' \cong r'_* \quad r_* \downarrow r'_*}{r \approx r_*}$$

$$\frac{r \approx r_*}{\delta\ r \approx \delta\ r_*}$$

using an auxiliary predicate defined as a disjunction by the rules

$$\frac{}{\mathit{ret}\ \sigma \cong \mathit{ret}\ \sigma} \qquad \frac{r_0 \approx r_{0*} \quad r_1 \approx r_{1*}}{r_0 + r_1 \cong r_{0*} + r_{1*}} \qquad \frac{}{\mathit{yield}\ s\ \sigma \cong \mathit{yield}\ s\ \sigma}$$

[2]This phenomenon is similar to extensional function equality, i.e., propositional equality of two functions on all arguments: it does not imply propositional equality of the functions.

While it might seem reasonable to replace the second rule in the definition of weak bisimilarity by

$$\frac{r \uparrow \quad r_* \uparrow}{r \approx r_*}$$

it is actually not a good idea in a constructive setting (classically one gets an equivalent definition). Constructively, it is not the case that any resumption would either converge or diverge (it takes the lesser principle of omniscience, a weak instance of excluded middle to prove this). Therefore, we would not be able to prove weak bisimilarity reflexive.

4 Functional-style semantics

Since we collect the possible executions of a statement into a single computation tree and divergence is represented by infinite delays, evaluation of the big-step semantics is deterministic (up to strong bisimilarity of big-step resumptions) and total: on one hand, for any r, r_*, if $s, \sigma \Rightarrow r$ and $s, \sigma \Rightarrow r_*$, then $r \sim r_*$, and on the other, there exists r such that $s, \sigma \Rightarrow r$. This means that the evaluation relation can be turned into a function. As a result, from the constructive point of view, evaluations can not only be checked, but also computed—which is only good of course.

Here is an equational specification of this function that can be massaged into an honest definition by structural corecursion.
Evaluation:

$$\begin{array}{llcl}
\mathit{eval}\ (x := e) & \sigma & = & \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma]) \\
\mathit{eval}\ \mathsf{skip} & \sigma & = & \mathit{ret}\ \sigma \\
\mathit{eval}\ (s_0; s_1) & \sigma & = & \mathit{evalseq}\ s_1\ (\mathit{eval}\ s_0\ \sigma) \\
\mathit{eval}\ (\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f) & \sigma & = & \mathit{if}\ [\![e]\!]\ \sigma\ \mathit{then}\ \delta\ (\mathit{yield}\ s_t\ \sigma)\ \mathit{else}\ \delta\ (\mathit{yield}\ s_t\ \sigma) \\
\mathit{eval}\ (\mathsf{while}\ e\ \mathsf{do}\ s_t) & \sigma & = & \mathit{if}\ [\![e]\!]\ \sigma\ \mathit{then}\ \delta\ (\mathit{yield}\ (s_t; \mathsf{while}\ e\ \mathsf{do}\ s_t)\ \sigma)\ \mathit{else}\ \delta\ (\mathit{ret}\ \sigma) \\
\mathit{eval}\ (s_0 \parallel s_1) & \sigma & = & \mathit{evalparR}\ s_1\ (\mathit{eval}\ s_0\ \sigma) + \mathit{evalparL}\ s_0\ (\mathit{eval}\ s_1\ \sigma) \\
\mathit{eval}\ (\mathsf{atomic}\ s) & \sigma & = & \mathit{close}\ (\mathit{eval}\ s\ \sigma) \\
\mathit{eval}\ (\mathsf{await}\ e\ \mathsf{do}\ s) & \sigma & = & \mathit{if}\ [\![e]\!]\ \sigma\ \mathit{then}\ \delta\ (\mathit{close}\ (\mathit{eval}\ s\ \sigma))\ \mathit{else}\ \delta\ (\mathit{yield}\ (\mathsf{await}\ e\ \mathsf{do}\ s)\ \sigma)
\end{array}$$

Sequential extension of evaluation:

$$\begin{array}{lllcl}
\mathit{evalseq} & s & (\mathit{ret}\ \sigma) & = & \mathit{yield}\ s\ \sigma \\
\mathit{evalseq} & s & (\delta\ r) & = & \delta\ (\mathit{evalseq}\ s\ r) \\
\mathit{evalseq} & s & (r_0 + r_1) & = & \mathit{evalseq}\ s\ r_0 + \mathit{evalseq}\ s\ r_1 \\
\mathit{evalseq} & s & (\mathit{yield}\ s_0\ \sigma) & = & \mathit{yield}\ (s_0; s)\ \sigma
\end{array}$$

Parallel extension of evaluation:

$$\begin{array}{lllcl}
\mathit{evalparR} & s & (\mathit{ret}\ \sigma) & = & \mathit{yield}\ s\ \sigma \\
\mathit{evalparR} & s & (\delta\ r) & = & \delta\ (\mathit{evalparR}\ s\ r) \\
\mathit{evalparR} & s & (r_0 + r_1) & = & \mathit{evalparR}\ s\ r_0 + \mathit{evalparR}\ s\ r_1 \\
\mathit{evalparR} & s & (\mathit{yield}\ s_0\ \sigma) & = & \mathit{yield}\ (s_0 \parallel s)\ \sigma \\
\\
\mathit{evalparL} & s & (\mathit{ret}\ \sigma) & = & \mathit{yield}\ s\ \sigma \\
\mathit{evalparL} & s & (\delta\ r) & = & \delta\ (\mathit{evalparL}\ s\ r) \\
\mathit{evalparL} & s & (r_0 + r_1) & = & \mathit{evalparL}\ s\ r_0 + \mathit{evalparL}\ s\ r_1 \\
\mathit{evalparL} & s & (\mathit{yield}\ s_1\ \sigma) & = & \mathit{yield}\ (s \parallel s_1)\ \sigma
\end{array}$$

Closing a resumption:

$$\begin{array}{llcl}
close & (ret\ \sigma) & = & ret\ \sigma \\
close & (\delta\ r) & = & \delta\ (close\ r) \\
close & (r_0 + r_1) & = & close\ r_0 + close\ r_1 \\
close & (yield\ s\ \sigma) & = & \delta\ (close\ (eval\ s\ \sigma))
\end{array}$$

Functional and relational evaluation of the big-step semantics agree: $eval\ s\ \sigma \sim r$ iff $s, \sigma \Rightarrow r$.

Similarly, evaluation of the giant-step semantics is deterministic and total and can be turned into a function.

Evaluation:

$$\begin{array}{lllcl}
eval_g & (x := e) & \sigma & = & \delta\ (ret\ \sigma[x \mapsto [\![e]\!]\ \sigma]) \\
eval_g & \mathsf{skip} & \sigma & = & ret\ \sigma \\
eval_g & (s_0; s_1) & \sigma & = & evalseq_g\ s_1\ (eval_g\ s_0\ \sigma) \\
eval_g & (\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f) & \sigma & = & if\ [\![e]\!]\ \sigma\ then\ \delta\ (yield\ (eval_g\ s_t)\ \sigma)\ else\ \delta\ (yield\ (eval_g\ s_t)\ \sigma) \\
eval_g & (\mathsf{while}\ e\ \mathsf{do}\ s_t) & \sigma & = & if\ [\![e]\!]\ \sigma\ then\ \delta\ (yield\ (evalseq_g\ (\mathsf{while}\ e\ \mathsf{do}\ s_t) \circ eval_g\ s_t)\ \sigma) \\
 & & & & \qquad else\ \delta\ (ret\ \sigma) \\
eval_g & (s_0 \parallel s_1) & \sigma & = & mergeR_g\ (eval_g\ s_1)\ (eval_g\ s_0\ \sigma) + mergeL_g\ (eval_g\ s_0)\ (eval_g\ s_1\ \sigma) \\
eval_g & (\mathsf{atomic}\ s) & \sigma & = & close_g\ (eval_g\ s\ \sigma) \\
eval_g & (\mathsf{await}\ e\ \mathsf{do}\ s) & \sigma & = & if\ [\![e]\!]\ \sigma\ then\ \delta\ (close_g\ (eval_g\ s\ \sigma)) \\
 & & & & \qquad else\ \delta\ (yield\ (eval_g\ (\mathsf{await}\ e\ \mathsf{do}\ s))\ \sigma)
\end{array}$$

Sequential extension of evaluation:

$$\begin{array}{lllcl}
evalseq_g & s & (ret\ \sigma) & = & yield\ (eval_g\ s)\ \sigma \\
evalseq_g & s & (\delta\ r) & = & \delta\ (evalseq_g\ s\ r) \\
evalseq_g & s & (r_0 + r_1) & = & evalseq_g\ s\ r_0 + evalseq_g\ s\ r_1 \\
evalseq_g & s & (yield\ k\ \sigma) & = & yield\ (evalseq_g\ s \circ k)\ \sigma
\end{array}$$

Merge of a continuation into a resumption:

$$\begin{array}{lllcl}
mergeR_g & k & (ret\ \sigma) & = & yield\ k\ \sigma \\
mergeR_g & k & (\delta\ r) & = & \delta\ (mergeR_g\ k\ r) \\
mergeR_g & k & (r_0 + r_1) & = & mergeR_g\ k\ r_0 + mergeR_g\ k\ r_1 \\
mergeR_g & k & (yield\ k_0\ \sigma) & = & yield\ (\lambda \sigma'.\ mergeR_g\ k\ (k_0\ \sigma') + mergeL_g\ k_0\ (k\ \sigma'))\ \sigma \\
mergeL_g & k & (ret\ \sigma) & = & yield\ k\ \sigma \\
mergeL_g & k & (\delta\ r) & = & \delta\ (mergeL_g\ k\ r) \\
mergeL_g & k & (r_0 + r_1) & = & mergeL_g\ k\ r_0 + mergeL_g\ k\ r_1 \\
mergeL_g & k & (yield\ k_1\ \sigma) & = & yield\ (\lambda \sigma'.\ mergeR_g\ k_1\ (k\ \sigma') + mergeL_g\ k\ (k_1\ \sigma'))\ \sigma
\end{array}$$

Closing a resumption:

$$\begin{array}{llcl}
close_g & (ret\ \sigma) & = & ret\ \sigma \\
close_g & (\delta\ r) & = & \delta\ (close_g\ r) \\
close_g & (r_0 + r_1) & = & close_g\ r_0 + close_g\ r_1 \\
close_g & (yield\ k\ \sigma) & = & \delta\ (close_g(k\ \sigma))
\end{array}$$

Functional and relational evaluation of the giant-step semantics agree: $eval_g\ s\ \sigma \sim_g r$ iff $s, \sigma \Rightarrow_g r$.

The reduction relation of the small-step semantics is also deterministic and total and can thus be turned into a function. We refrain from spelling out the details here.

5 Trace-based semantics

A trace-based big-step semantics is obtained from the resumption-based big-step semantics straightforwardly by removing the + constructor of resumptions, splitting the evaluation rule for $\parallel$ into two rules (thereby turning evaluation nondeterministic) and removing the rules for + in the definitions of extended evaluations and closing. Because of the nondeterminism, trace-based evaluation cannot be turned into a function. But it is still total, as any scheduling leads to a valid trace. Differently from standard inductive big-step semantics, divergence from endless work or waiting does not lead to a "lost trace".

In detail, the different ingredients of the semantics are defined as follows.
Traces:

$$\frac{\sigma : \mathit{state}}{\mathit{ret}\ \sigma : \mathit{trace}} \qquad \frac{t : \mathit{trace}}{\delta\ t : \mathit{trace}} \qquad \frac{s : \mathit{stmt} \quad \sigma : \mathit{state}}{\mathit{yield}\ s\ \sigma : \mathit{trace}}$$

Evaluation:

$$\frac{}{x := e, \sigma \Rightarrow \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma])} \qquad \frac{}{\mathsf{skip}, \sigma \Rightarrow \mathit{ret}\ \sigma} \qquad \frac{s_0, \sigma \Rightarrow t \quad s_1, t \Rightarrow^{\mathrm{seq}} t'}{s_0; s_1, \sigma \Rightarrow t'}$$

$$\frac{\sigma \models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow \delta\ (\mathit{yield}\ s_t\ \sigma)} \qquad \frac{\sigma \not\models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow \delta\ (\mathit{yield}\ s_f\ \sigma)}$$

$$\frac{\sigma \models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow \delta\ (\mathit{yield}\ (s_t; \mathsf{while}\ e\ \mathsf{do}\ s_t)\ \sigma)} \qquad \frac{\sigma \not\models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow \delta\ (\mathit{ret}\ \sigma)}$$

$$\frac{s_0, \sigma \Rightarrow t \quad s_1, t \Rightarrow^{\mathrm{parR}} t'}{s_0 \parallel s_1, \sigma \Rightarrow t'} \qquad \frac{s_1, \sigma \Rightarrow t \quad s_0, t \Rightarrow^{\mathrm{parL}} t'}{s_0 \parallel s_1, \sigma \Rightarrow t'} \qquad \frac{s, \sigma \Rightarrow t \quad t \leadsto t'}{\mathsf{atomic}\ s, \sigma \Rightarrow t'}$$

$$\frac{\sigma \models e \quad s, \sigma \Rightarrow t \quad t \leadsto t'}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow \delta\ t'} \qquad \frac{\sigma \not\models e}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow \delta\ (\mathit{yield}\ (\mathsf{await}\ e\ \mathsf{do}\ s)\ \sigma)}$$

Sequential extension of evaluation:

$$\frac{}{s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{seq}} \mathit{yield}\ s\ \sigma} \qquad \frac{s, t \Rightarrow^{\mathrm{seq}} t'}{s, \delta\ t \Rightarrow^{\mathrm{seq}} \delta\ t'} \qquad \frac{}{s, \mathit{yield}\ s_0\ \sigma \Rightarrow^{\mathrm{seq}} \mathit{yield}\ (s_0; s)\ \sigma}$$

Parallel extension of evaluation:

$$\frac{}{s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{parR}} \mathit{yield}\ s\ \sigma} \qquad \frac{s, t \Rightarrow^{\mathrm{parR}} t'}{s, \delta\ t \Rightarrow^{\mathrm{parR}} \delta\ t'} \qquad \frac{}{s, \mathit{yield}\ s_0\ \sigma \Rightarrow^{\mathrm{parR}} \mathit{yield}\ (s_0 \parallel s)\ \sigma}$$

$$\frac{}{s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{parL}} \mathit{yield}\ s\ \sigma} \qquad \frac{s, t \Rightarrow^{\mathrm{parL}} t'}{s, \delta\ t \Rightarrow^{\mathrm{parL}} \delta\ t'} \qquad \frac{}{s, \mathit{yield}\ s_1\ \sigma \Rightarrow^{\mathrm{parL}} \mathit{yield}\ (s \parallel s_1)\ \sigma}$$

Closing a trace:

$$\frac{}{\mathit{ret}\ \sigma \leadsto \mathit{ret}\ \sigma} \qquad \frac{t \leadsto t'}{\delta\ t \leadsto \delta\ t'} \qquad \frac{s, \sigma \Rightarrow t \quad t \leadsto t'}{\mathit{yield}\ s\ \sigma \leadsto \delta\ t'}$$

The giant-step case is more interesting. In giant-step resumptions, we had two kinds of branching: in addition to the binary branching of $+$, the branching over all states of *yield*. For a trace-based giant-step semantics, we would like to have a fully linear concept of traces with neither kind of branching. Evaluation must then not only "guess" which part of a parallel composition gets to make the first small step, but also which state control is regained in after suspension.

Accordingly, we define traces without a $+$ constructor. Moreover, we modify the typing of *yield*.

$$\frac{\sigma : \mathit{state}}{\mathit{ret}\ \sigma : \mathit{trace}_{\mathrm{g}}} \quad \frac{t : \mathit{trace}_{\mathrm{g}}}{\delta\ t : \mathit{trace}_{\mathrm{g}}} \quad \frac{\sigma' : \mathit{state} \quad t : \mathit{trace}_{\mathrm{g}} \quad \sigma : \mathit{state}}{\mathit{yield}\ (\sigma', t)\ \sigma : \mathit{trace}_{\mathrm{g}}}$$

The idea is to have the trace $\mathit{yield}\ (\sigma',t)\ \sigma$ to stand for a computation that is suspended in state σ. Control is returned to it in state σ' and then it continues as recorded in trace t.

Evaluation is defined as for the resumption-based semantics, but there are two rules for parallel composition and in the rules where a *yield* trace is produced σ' is quantified existentially in premises rather than universally.

$$\frac{}{x := e, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma])}$$

$$\frac{}{\mathsf{skip}, \sigma \Rightarrow_{\mathrm{g}} \mathit{ret}\ \sigma} \quad \frac{s_0, \sigma \Rightarrow_{\mathrm{g}} t \quad s_1, t \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} t'}{s_0; s_1, \sigma \Rightarrow_{\mathrm{g}} t'}$$

$$\frac{\sigma \models e \quad s_t, \sigma' \Rightarrow_{\mathrm{g}} t}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ (\sigma', t)\ \sigma)} \quad \frac{\sigma \not\models e \quad s_f, \sigma' \Rightarrow_{\mathrm{g}} t}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ (\sigma', t)\ \sigma)}$$

$$\frac{\sigma \models e \quad s_t, \sigma' \Rightarrow_{\mathrm{g}} t \quad \mathsf{while}\ e\ \mathsf{do}\ s_t, t \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} t'}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ (\sigma', t)\ \sigma)} \quad \frac{\sigma \not\models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{ret}\ \sigma)}$$

$$\frac{s_0, \sigma \Rightarrow_{\mathrm{g}} t_0 \quad s_1, \sigma' \Rightarrow_{\mathrm{g}} t_1 \quad (\sigma', t_1), t_0 \gg_{\mathrm{g}}^{\mathrm{R}} t'}{s_0 \parallel s_1, \sigma \Rightarrow_{\mathrm{g}} t'} \quad \frac{s_1, \sigma \Rightarrow_{\mathrm{g}} t_1 \quad s_0, \sigma' \Rightarrow_{\mathrm{g}} t_0 \quad (\sigma', t_0), t_1 \gg_{\mathrm{g}}^{\mathrm{L}} t'}{s_0 \parallel s_1, \sigma \Rightarrow_{\mathrm{g}} t'}$$

$$\frac{s, \sigma \Rightarrow_{\mathrm{g}} t \quad t \rightsquigarrow_{\mathrm{g}} t'}{\mathsf{atomic}\ s, \sigma \Rightarrow_{\mathrm{g}} t'}$$

$$\frac{\sigma \models e \quad s, \sigma \Rightarrow_{\mathrm{g}} t \quad t \rightsquigarrow_{\mathrm{g}} t'}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow_{\mathrm{g}} \delta\ t'} \quad \frac{\sigma \not\models e \quad \mathsf{await}\ e\ \mathsf{do}\ s, \sigma' \Rightarrow_{\mathrm{g}} t}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ (\sigma', t)\ \sigma)}$$

Similar considerations apply to sequential extension of evaluation—

$$\frac{s, \sigma' \Rightarrow_{\mathrm{g}} t}{s, \mathit{ret}\ \sigma \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} \mathit{yield}\ (\sigma', t)\ \sigma} \quad \frac{s, t \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} t'}{s, \delta\ t \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} \delta\ t'} \quad \frac{s, t \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} t'}{s, \mathit{yield}\ (\sigma', t)\ \sigma \Rightarrow_{\mathrm{g}}^{\mathrm{seq}} \mathit{yield}\ (\sigma', t')\ \sigma}$$

—and to merging a continuation into a trace—

$$\frac{}{k, \mathit{ret}\ \sigma \gg_{\mathrm{g}}^{\mathrm{R}} \mathit{yield}\ k\ \sigma} \quad \frac{k, t \gg_{\mathrm{g}}^{\mathrm{R}} t'}{k, \delta\ t \gg_{\mathrm{g}}^{\mathrm{R}} \delta\ t'} \quad \frac{k, t \gg_{\mathrm{g}}^{\mathrm{R}} t'}{k, \mathit{yield}\ (\sigma', t)\ \sigma \gg_{\mathrm{g}}^{\mathrm{R}} \mathit{yield}\ (\sigma', t')\ \sigma} \quad \frac{k, t \gg_{\mathrm{g}}^{\mathrm{L}} t'}{(\sigma', t), \mathit{yield}\ k\ \sigma \gg_{\mathrm{g}}^{\mathrm{R}} \mathit{yield}\ (\sigma', t')\ \sigma}$$

$$\frac{}{k, \mathit{ret}\ \sigma \gg_{\mathrm{g}}^{\mathrm{L}} \mathit{yield}\ k\ \sigma} \quad \frac{k, t \gg_{\mathrm{g}}^{\mathrm{L}} t'}{k, \delta\ t \gg_{\mathrm{g}}^{\mathrm{L}} \delta\ t'} \quad \frac{k, t \gg_{\mathrm{g}}^{\mathrm{R}} t'}{(\sigma', t), \mathit{yield}\ k\ \sigma \gg_{\mathrm{g}}^{\mathrm{L}} \mathit{yield}\ (\sigma', t')\ \sigma} \quad \frac{k, t \gg_{\mathrm{g}}^{\mathrm{L}} t'}{k, \mathit{yield}\ (\sigma', t)\ \sigma \gg_{\mathrm{g}}^{\mathrm{L}} \mathit{yield}\ (\sigma', t')\ \sigma}$$

Closing a trace is defined as follows. Closing a *yield* trace can only succeed if the control release and grab states coincide, i.e., the grab state has been guessed correctly for the closed-system situation.

$$\frac{}{\mathit{ret}\ \sigma \rightsquigarrow_{\mathrm{g}} \mathit{ret}\ \sigma} \quad \frac{t \rightsquigarrow_{\mathrm{g}} t'}{\delta\ t \rightsquigarrow_{\mathrm{g}} \delta\ t'} \quad \frac{t \rightsquigarrow_{\mathrm{g}} t'}{\mathit{yield}\ (\sigma, t)\ \sigma \rightsquigarrow_{\mathrm{g}} \delta\ t'}$$

6 Conclusion

We have shown that, with coinductive denotations and coinductive evaluation, it is possible to give simple and meaningful big-step descriptions of semantics of languages with concurrency. The key ideas remain the same as in the purely sequential case. Most importantly, due care must be taken of the possibilities of divergence. In particular, even diverging loops or await statements must be productive (by growing resumptions or traces by unit delays). Finite delays can then be equated by a suitable notion of weak bisimilarity.

Although we could not delve into this topic here, our definitions and proofs benefit heavily from the fact that the datatype of resumptions is a monad, in fact a completely iterative monad, and moreover a free one (as long as we equate only strongly bisimilar resumptions).

With Wolfgang Ahrendt and Keiko Nakata, we have devised a coinductive big-step semantics for ABS, an exploratory object-oriented language with an intricate concurrency model, developed in the FP7 ICT project HATS. ABS has cooperative scheduling of tasks (method invocations) communicating via shared memory (fields) within every object and preemptive scheduling of objects communicating via asynchronous method calls and futures. This work will be reported elsewhere.

Acknowledgements This work was first presented at the SEFM 2011 Summer School in Montevideo in November 2011. I thank the organizers for the invitation.

This research was supported by the EU FP7 ICT project HATS, the ERDF financed CoE project EXCS, the Estonian Science Foundation grant no. 9475 and the Estonian Ministry of Education and Research target-financed research theme no. 0140007s12.

References

[1] R. Amadio (2012): *Operational methods for concurrency*. Draft lecture notes. URL `http://www.pps.univ-paris-diderot.fr/~amadio/Ens/concurrency.pdf`

[2] D. Ancona (2012): *Soundness of object-oriented languages with coinductive big-step semantics*. In: J. Noble (ed.) *Proc. of 26th Europ. Conf. on Object-Oriented Programming, ECOOP 2012 (Beijing, June 2012). Lect. Notes in Comput. Sci.* 7313. Springer, Berlin, pp. 459–483. doi: 10.1007/978-3-642-31057-7_21

[3] V. Capretta (2005): *General recursion via coinductive types*. *Log. Methods in Comput. Sci.* 1(2), article 1. doi: 10.2168/lmcs-1(2:1)2005

[4] P. Cenciarelli & E. Moggi (1993): *A syntactic approach to modularity in denotational semantics*. In: *Proc. of 5th Biennial Meeting on Category Theory and Computer Science, CTCS '93 (Amsterdam, Sept. 1993).* Tech. report, CWI, Amsterdam.

[5] P. Cousot & R. Cousot (2009): *Bi-inductive operational semantics*. *Inf. and Comput.* 207(2), pp. 258–283. doi: 10.1016/j.ic.2008.03.025

[6] N. A. Danielsson (2012): *Operational semantics using the partiality monad*. In: *Proc. of 17th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP '12 (Copenhagen, Sept. 2012).* ACM Press, New York, pp. 127–138. doi: 10.1145/2364527.2364546

[7] S. Goncharov & L. Schröder (2011): *A coinductive calculus for asynchronous side-effecting processes*. In: O. Owe, M. Steffen & J. A. Telle (eds.) *Proc. of 18th Int. Symp. on Fundamentals of Computation Theory, FCT 2011 (Oslo, Aug. 2011). Lect. Notes in Comput. Sci.* 6914. Springer, Berlin, pp. 276–287. doi: 10.1007/978-3-642-22953-4_24

[8] W. L. Harrison (2006): *The essence of multitasking*. In: M. Johnson & V. Vene (eds.) *Proc. of 11th Int. Conf. on Algebraic Methdology and Software Technology, AMAST 2006 (Kuressaare, July 2006). Lect. Notes in Comput. Sci.* 4019. Springer, Berlin, pp. 158–172. doi: 10.1007/11784180_14

[9] X. Leroy & H. Grall (2009): *Coinductive big-step operational semantics*. *Inf. and Comput.* 207(2), pp. 285–305. doi: 10.1016/j.ic.2007.12.004

[10] K. Mitchell (1994): *Concurrency in a natural semantics*. Report ECS-LFCS-94-311. Univ. of Edinburgh.

[11] K. Nakata & T. Uustalu (2009): *Trace-based coinductive operational semantics for While: big-step and small-step, relational and functional styles*. In: S. Berghofer, T. Nipkow, C. Urban & M. Wenzel (eds.) *Proc. of 22nd Int. Conf. on Theorem Proving in Higher-Order Logics, TPHOLs 2009 (Munich, Aug. 2009). Lect. Notes in Comput. Sci.* 5674. Springer, Berlin, pp. 375–390. doi: 10.1007/978-3-642-03359-9_26

[12] K. Nakata & T. Uustalu (2010): *A Hoare logic for the coinductive trace-based big-step semantics of While*. In: A. D. Gordon (ed.) *Proc. of 19th Europ. Symp. on Programming, ESOP 2010 (Paphos, March 2010). Lect. Notes in Comput. Sci.* 6012. Springer, Berlin, pp. 488–506. doi: 10.1007/978-3-642-11957-6_26

[13] K. Nakata & T. Uustalu (2010). *Resumptions, weak bisimilarity and big-step semantics for While with interactive I/O: an exercise in mixed induction-coinduction*. In: L. Aceto & P. Sobocinski (eds.) *Proc. of 7th Wksh. on Structural Operational Semantics, SOS 2010 (Paris, Aug. 2010). Electron. Proc. in Theor. Comput. Sci.* 32. Open Publishing Assoc., Sydney, pp. 57–75. doi: 10.4204/eptcs.32.5

[14] G. D. Plotkin (1976): *A powerdomain construction*. *SIAM J. of Comput.* 5(3), pp. 452–487. doi: 10.1137/0205035

A Resumption-based semantics for cooperative scheduling

Here we give the syntax of Section 2.1 a cooperative scheduling interpretation.

It might be argued that this interpretation is more foundational than the pre-emptive scheduling interpretation—all control release is explicit and is only due to await statements. Hence all *yield*s stem from evaluation of await statements.

A.1 Big-step semantics

Evaluation:

$$\begin{array}{c} \\ \hline\hline x := e, \sigma \Rightarrow \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma]) \end{array} \qquad \begin{array}{c} \\ \hline\hline \mathsf{skip}, \sigma \Rightarrow \mathit{ret}\ \sigma \end{array} \qquad \begin{array}{c} s_0, \sigma \Rightarrow r \quad s_1, r \Rightarrow^{\mathrm{seq}} r' \\ \hline\hline s_0; s_1, \sigma \Rightarrow r' \end{array}$$

$$\begin{array}{c} \sigma \models e \quad s_t, \sigma \Rightarrow r \\ \hline\hline \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow \delta\ r \end{array} \qquad \begin{array}{c} \sigma \not\models e \quad s_f, \sigma \Rightarrow r \\ \hline\hline \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \Rightarrow \delta\ r \end{array}$$

$$\begin{array}{c} \sigma \models e \quad s_t, \sigma \Rightarrow r \quad \mathsf{while}\ e\ \mathsf{do}\ s_t, r \Rightarrow^{\mathrm{seq}} r' \\ \hline\hline \mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow \delta\ r' \end{array} \qquad \begin{array}{c} \sigma \not\models e \\ \hline\hline \mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \Rightarrow \delta\ (\mathit{ret}\ \sigma) \end{array}$$

$$\begin{array}{c} s_0, \sigma \Rightarrow r_0 \quad s_1, r_0 \Rightarrow^{\mathrm{parR}} r_0' \quad s_1, \sigma \Rightarrow r_1 \quad s_0, r_1 \Rightarrow^{\mathrm{parL}} r_1' \\ \hline\hline s_0 \parallel s_1, \sigma \Rightarrow r_0' + r_1' \end{array} \qquad \begin{array}{c} s, \sigma \Rightarrow r \quad r \rightsquigarrow r' \\ \hline\hline \mathsf{atomic}\ s, \sigma \Rightarrow r' \end{array}$$

$$\begin{array}{c} \sigma \models e \quad s, \sigma \Rightarrow r \quad r \rightsquigarrow r' \\ \hline\hline \mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow \delta\ r' \end{array} \qquad \begin{array}{c} \sigma \not\models e \\ \hline\hline \mathsf{await}\ e\ \mathsf{do}\ s, \sigma \Rightarrow \delta\ (\mathit{yield}\ (\mathsf{await}\ e\ \mathsf{do}\ s)\ \sigma) \end{array}$$

Sequential extension of evaluation:

$$\begin{array}{c} s, \sigma \Rightarrow r \\ \hline\hline s, \mathit{ret}\ \sigma \Rightarrow^{\mathrm{seq}} r \end{array} \qquad \begin{array}{c} s, r \Rightarrow^{\mathrm{seq}} r' \\ \hline\hline s, \delta\ r \Rightarrow^{\mathrm{seq}} \delta\ r' \end{array} \qquad \begin{array}{c} s, r_0 \Rightarrow^{\mathrm{seq}} r_0' \quad s, r_1 \Rightarrow^{\mathrm{seq}} r_1' \\ \hline\hline s, r_0 + r_1 \Rightarrow^{\mathrm{seq}} r_0' + r_1' \end{array} \qquad \begin{array}{c} \\ \hline\hline s, \mathit{yield}\ s_0\ \sigma \Rightarrow^{\mathrm{seq}} \mathit{yield}\ (s_0; s)\ \sigma \end{array}$$

Parallel extension of evaluation:

$$\frac{s,\sigma \Rightarrow r}{s,\mathit{ret}\ \sigma \Rightarrow^{\mathrm{parR}} r} \quad \frac{s,r \Rightarrow^{\mathrm{parR}} r'}{s,\delta\ r \Rightarrow^{\mathrm{parR}} \delta\ r'} \quad \frac{s,r_0 \Rightarrow^{\mathrm{parR}} r'_0 \quad s,r_1 \Rightarrow^{\mathrm{parR}} r'_1}{s,r_0+r_1 \Rightarrow^{\mathrm{parR}} r'_0+r'_1} \quad \frac{}{s,\mathit{yield}\ s_0\ \sigma \Rightarrow^{\mathrm{parR}} \mathit{yield}\ (s_0 \parallel s)\ \sigma}$$

$$\frac{s,\sigma \Rightarrow r}{s,\mathit{ret}\ \sigma \Rightarrow^{\mathrm{parL}} r} \quad \frac{s,r \Rightarrow^{\mathrm{parL}} r'}{s,\delta\ r \Rightarrow^{\mathrm{parL}} \delta\ r'} \quad \frac{s,r_0 \Rightarrow^{\mathrm{parL}} r'_0 \quad s,r_1 \Rightarrow^{\mathrm{parL}} r'_1}{s,r_0+r_1 \Rightarrow^{\mathrm{parL}} r'_0+r'_1} \quad \frac{}{s,\mathit{yield}\ s_1\ \sigma \Rightarrow^{\mathrm{parL}} \mathit{yield}\ (s \parallel s_1)\ \sigma}$$

Closing a resumption:

$$\frac{}{\mathit{ret}\ \sigma \rightsquigarrow \mathit{ret}\ \sigma} \quad \frac{r \rightsquigarrow r'}{\delta\ r \rightsquigarrow \delta\ r'} \quad \frac{r_0 \rightsquigarrow r'_0 \quad r_1 \rightsquigarrow r'_1}{r_0+r_1 \rightsquigarrow r'_0+r'_1} \quad \frac{s,\sigma \Rightarrow r \quad r \rightsquigarrow r'}{\mathit{yield}\ s\ \sigma \rightsquigarrow \delta\ r'}$$

A.2 Giant-step semantics

Evaluation:

$$\frac{}{x := e,\sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{ret}\ \sigma[x \mapsto [\![e]\!]\ \sigma])} \quad \frac{}{\mathsf{skip},\sigma \Rightarrow_{\mathrm{g}} \mathit{ret}\ \sigma} \quad \frac{s_0,\sigma \Rightarrow_{\mathrm{g}} r \quad s_1,r \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r'}{s_0;s_1,\sigma \Rightarrow_{\mathrm{g}} r'}$$

$$\frac{\sigma \models e \quad s_t,\sigma \Rightarrow_{\mathrm{g}} r}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f,\sigma \Rightarrow_{\mathrm{g}} \delta\ r} \quad \frac{\sigma \not\models e \quad s_f,\sigma \Rightarrow_{\mathrm{g}} r}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f,\sigma \Rightarrow_{\mathrm{g}} \delta\ r}$$

$$\frac{\sigma \models e \quad s_t,\sigma \Rightarrow_{\mathrm{g}} r \quad \mathsf{while}\ e\ \mathsf{do}\ s_t,r \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r'}{\mathsf{while}\ e\ \mathsf{do}\ s_t,\sigma \Rightarrow_{\mathrm{g}} \delta\ r'} \quad \frac{\sigma \not\models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t,\sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{ret}\ \sigma)}$$

$$\frac{s_0,\sigma \Rightarrow_{\mathrm{g}} r_0 \quad \forall \sigma'.\, s_1,\sigma' \Rightarrow_{\mathrm{g}} k_1\ \sigma' \quad k_1,r_0 \gg^{\mathrm{R}}_{\mathrm{g}} r'_0 \quad s_1,\sigma \Rightarrow_{\mathrm{g}} r_1 \quad \forall \sigma'.\, s_0,\sigma' \Rightarrow_{\mathrm{g}} k_0\ \sigma' \quad k_0,r_1 \gg^{\mathrm{L}}_{\mathrm{g}} r'_1}{s_0 \parallel s_1,\sigma \Rightarrow_{\mathrm{g}} r'_0+r'_1}$$

$$\frac{s,\sigma \Rightarrow_{\mathrm{g}} r \quad r \rightsquigarrow_{\mathrm{g}} r'}{\mathsf{atomic}\ s,\sigma \Rightarrow_{\mathrm{g}} r'}$$

$$\frac{\sigma \models e \quad s,\sigma \Rightarrow_{\mathrm{g}} r \quad r \rightsquigarrow_{\mathrm{g}} r'}{\mathsf{await}\ e\ \mathsf{do}\ s,\sigma \Rightarrow_{\mathrm{g}} \delta\ r'} \quad \frac{\sigma \not\models e \quad \forall \sigma'.\, \mathsf{await}\ e\ \mathsf{do}\ s,\sigma' \Rightarrow_{\mathrm{g}} k\ \sigma'}{\mathsf{await}\ e\ \mathsf{do}\ s,\sigma \Rightarrow_{\mathrm{g}} \delta\ (\mathit{yield}\ k\ \sigma)}$$

Sequential extension of evaluation:

$$\frac{s,\sigma \Rightarrow_{\mathrm{g}} r}{s,\mathit{ret}\ \sigma \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r} \quad \frac{s,r \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r'}{s,\delta\ r \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} \delta\ r'} \quad \frac{s,r_0 \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r'_0 \quad s,r_1 \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r'_1}{s,r_0+r_1 \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} r'_0+r'_1} \quad \frac{\forall \sigma'.\, s,k\ \sigma' \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} k'\ \sigma'}{s,\mathit{yield}\ k\ \sigma \Rightarrow^{\mathrm{seq}}_{\mathrm{g}} \mathit{yield}\ k'\ \sigma}$$

Merging a continuation into a resumption:

$$\frac{}{k,\mathit{ret}\ \sigma \gg^{\mathrm{R}}_{\mathrm{g}} k\ \sigma} \quad \frac{k,r \gg^{\mathrm{R}}_{\mathrm{g}} r'}{k,\delta\ r \gg^{\mathrm{R}}_{\mathrm{g}} \delta\ r'} \quad \frac{k,r_0 \gg^{\mathrm{R}}_{\mathrm{g}} r'_0 \quad k,r_1 \gg^{\mathrm{R}}_{\mathrm{g}} r'_1}{k,r_0+r_1 \gg^{\mathrm{R}}_{\mathrm{g}} r'_0+r'_1} \quad \frac{\forall \sigma'.\, k,k_0\ \sigma' \gg^{\mathrm{R}}_{\mathrm{g}} k'_0\ \sigma' \quad \forall \sigma'.\, k_0,k\ \sigma' \gg^{\mathrm{L}}_{\mathrm{g}} k'_1\ \sigma'}{k,\mathit{yield}\ k_0\ \sigma \gg^{\mathrm{R}}_{\mathrm{g}} \mathit{yield}\ (\lambda \sigma'.k'_0\ \sigma' + k'_1\ \sigma')\ \sigma}$$

$$\frac{}{k,\mathit{ret}\ \sigma \gg^{\mathrm{L}}_{\mathrm{g}} k\ \sigma} \quad \frac{k,r \gg^{\mathrm{L}}_{\mathrm{g}} r'}{k,\delta\ r \gg^{\mathrm{L}}_{\mathrm{g}} \delta\ r'} \quad \frac{k,r_0 \gg^{\mathrm{L}}_{\mathrm{g}} r'_0 \quad k,r_1 \gg^{\mathrm{L}}_{\mathrm{g}} r'_1}{k,r_0+r_1 \gg^{\mathrm{L}}_{\mathrm{g}} r'_0+r'_1} \quad \frac{\forall \sigma'.\, k_1,k\ \sigma' \gg^{\mathrm{R}}_{\mathrm{g}} k'_0\ \sigma' \quad \forall \sigma'.\, k,k_1\ \sigma' \gg^{\mathrm{L}}_{\mathrm{g}} k'_1\ \sigma'}{k,\mathit{yield}\ k_1\ \sigma \gg^{\mathrm{L}}_{\mathrm{g}} \mathit{yield}\ (\lambda \sigma'.k'_0\ \sigma' + k'_1\ \sigma')\ \sigma}$$

Closing a resumption:

$$\frac{}{\mathit{ret}\ \sigma \rightsquigarrow_{\mathrm{g}} \mathit{ret}\ \sigma} \quad \frac{r \rightsquigarrow_{\mathrm{g}} r'}{\delta\ r \rightsquigarrow_{\mathrm{g}} \delta\ r'} \quad \frac{r_0 \rightsquigarrow_{\mathrm{g}} r'_0 \quad r_1 \rightsquigarrow_{\mathrm{g}} r'_1}{r_0+r_1 \rightsquigarrow_{\mathrm{g}} r'_0+r'_1} \quad \frac{k\ \sigma \rightsquigarrow_{\mathrm{g}} r}{\mathit{yield}\ k\ \sigma \rightsquigarrow_{\mathrm{g}} \delta\ r}$$

A.3 Small-step semantics

Reduction:

$$\frac{}{x := e, \sigma \to \delta\ (\mathsf{skip}, \sigma[x \mapsto [\![e]\!]\ \sigma])}$$

$$\frac{}{\mathsf{skip}, \sigma \to \mathit{ret}\ \sigma}$$

$$\frac{s_0, \sigma \to \mathit{ret}\ \sigma' \quad s_1, \sigma' \to c}{s_0; s_1, \sigma \to c} \qquad \frac{s_0, \sigma \to \delta\ (s_0', \sigma')}{s_0; s_1, \sigma \to \delta\ (s_0'; s_1, \sigma')}$$

$$\frac{s_0, \sigma \to (s_{00}, \sigma_0) + (s_{01}, \sigma_1)}{s_0; s_1, \sigma \to (s_{00}; s_1, \sigma_0) + (s_{01}; s_1, \sigma_1)} \qquad \frac{s_0, \sigma \to \mathit{yield}\ s_0'\ \sigma'}{s_0; s_1, \sigma \to \mathit{yield}\ (s_0'; s_1)\ \sigma'}$$

$$\frac{\sigma \models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \to \delta\ (s_t, \sigma)} \qquad \frac{\sigma \not\models e}{\mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f, \sigma \to \delta\ (s_f, \sigma)}$$

$$\frac{\sigma \models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \to \delta\ (s_t; \mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma)} \qquad \frac{\sigma \not\models e}{\mathsf{while}\ e\ \mathsf{do}\ s_t, \sigma \to \delta\ (\mathsf{skip}, \sigma)}$$

$$\frac{}{s_0 \parallel s_1, \sigma \to (s_0 \mathbin{\lfloor\!\lfloor} s_1, \sigma) + (s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma)}$$

$$\frac{s_0, \sigma \to \mathit{ret}\ \sigma' \quad s_1, \sigma' \to c}{s_0 \mathbin{\lfloor\!\lfloor} s_1, \sigma \to c} \qquad \frac{s_0, \sigma \to \delta\ (s_0', \sigma')}{s_0 \mathbin{\lfloor\!\lfloor} s_1, \sigma \to \delta\ (s_0' \mathbin{\lfloor\!\lfloor} s_1, \sigma')}$$

$$\frac{s_0, \sigma \to (s_{00}, \sigma_0) + (s_{01}, \sigma_1)}{s_0 \mathbin{\lfloor\!\lfloor} s_1, \sigma \to (s_{00} \mathbin{\lfloor\!\lfloor} s_1, \sigma_0) + (s_{01} \mathbin{\lfloor\!\lfloor} s_1, \sigma_1)} \qquad \frac{s_0, \sigma \to \mathit{yield}\ s_0'\ \sigma'}{s_0 \mathbin{\lfloor\!\lfloor} s_1, \sigma \to \mathit{yield}\ (s_0' \parallel s_1)\ \sigma'}$$

$$\frac{s_1, \sigma \to \mathit{ret}\ \sigma' \quad s_0, \sigma' \to c}{s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma \to c} \qquad \frac{s_1, \sigma \to \delta\ (s_1', \sigma')}{s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma \to \delta\ (s_0 \mathbin{\rfloor\!\rfloor} s_1', \sigma')}$$

$$\frac{s_1, \sigma \to (s_{10}, \sigma_0) + (s_{11}, \sigma_1)}{s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma \to (s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma_{10}) + (s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma_{11})} \qquad \frac{s_1, \sigma \to \mathit{yield}\ s_1'\ \sigma'}{s_0 \mathbin{\rfloor\!\rfloor} s_1, \sigma \to \mathit{yield}\ (s_0 \parallel s_1')\ \sigma'}$$

$$\frac{s, \sigma \to \mathit{ret}\ \sigma'}{\mathsf{atomic}\ s, \sigma \to \mathit{ret}\ \sigma'} \qquad \frac{s, \sigma \to \delta\ (s', \sigma')}{\mathsf{atomic}\ s, \sigma \to \delta\ (\mathsf{atomic}\ s', \sigma')}$$

$$\frac{s, \sigma \to (s_0, \sigma_0) + (s_1, \sigma_1)}{\mathsf{atomic}\ s, \sigma \to (\mathsf{atomic}\ s_0, \sigma_0) + (\mathsf{atomic}\ s_1, \sigma_1)} \qquad \frac{s, \sigma \to \mathit{yield}\ s'\ \sigma'}{\mathsf{atomic}\ s, \sigma \to \delta\ (\mathsf{atomic}\ s', \sigma')}$$

$$\frac{\sigma \models e}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \to \delta\ (\mathsf{atomic}\ s, \sigma)} \qquad \frac{\sigma \not\models e}{\mathsf{await}\ e\ \mathsf{do}\ s, \sigma \to \delta\ (\mathsf{suspend}; \mathsf{await}\ e\ \mathsf{do}\ s, \sigma)}$$

$$\frac{}{\mathsf{suspend}, \sigma \to \mathit{yield}\ \mathsf{skip}\ \sigma}$$

Here suspend is an auxiliary statement form that we need for giving the reduction rule for await. It releases control immediately (differently from await true do skip which makes a small internal step first).