

The Yield Operator (Talk Handout)

Roshan James

October 4, 2006

1 Operational Semantics

Definitions:

$$\begin{aligned} \text{Expressions, } e &= x \mid \lambda x.e \mid e_1 e_2 \mid \text{case } e_1 e_2 e_3 \mid \text{pair } v_1 v_2 \mid \text{just } v \mid \text{yield } e \mid \text{run } e \mid \text{cont } E \\ \text{Values, } v &= \lambda x.e \mid \text{pair } v_1 v_2 \mid \text{just } v \mid \text{cont } E \end{aligned}$$

$$\text{Contexts, } E = \square \mid E e \mid v E \mid \text{case } E e_1 e_2 \mid \text{run } E \mid \text{yield } E$$

The definitions for syntactic categories of expressions ‘ e ’, values ‘ v ’ and evaluation contexts ‘ E ’ are given above. The expressions in this language consist of the call-by-value lambda calculus extended with ‘pair’ and ‘just’ values and a corresponding pattern matching ‘case’ expression. The language also allows for first class continuations. ‘run’ and ‘yield’ statements to used to model first class iterators.

The term \square in denotes the empty evaluation context ‘ E ’. The first two terms $E e$ and $v E$ indicates call-by-value reduction, where the operator is evaluated before the operand. The ‘case’ expression allows for reduction for its first parameter expression ‘ e_1 ’. The ‘run’ and ‘yield’ expressions allows for reduction of their parameter expressions before their reductions are applicable.

Reductions:

$$\begin{aligned} E[(\lambda x.e) v] &\mapsto E[e[v/x]] && \text{BETA} \\ E[\text{case } (\text{pair } v_1 v_2) e_1 e_2] &\mapsto E[(e_1 v_1) v_2] && \text{CASE PAIR} \\ E[\text{case } (\text{just } v) e_1 e_2] &\mapsto E[e_2 v] && \text{CASE JUST} \\ E[\text{run } E'[\text{yield } v]] &\mapsto E[\text{pair } v (\text{cont } E')] && \text{YIELD} \\ E[\text{run } v] &\mapsto E[\text{just } v] && \text{RETURN} \\ E[(\text{cont } E') v] &\mapsto E[\text{run } E'[v]] && \text{RESUME} \end{aligned}$$

1.1 Reference: Semantics for Callcc/Abort

Language Definition:

$$\begin{aligned} \text{Expressions, } e &= x \mid \lambda x.e \mid e_1 e_2 \mid p \mid \text{callcc } e \mid \mathcal{A} e \\ \text{Values, } v &= \lambda x.e \mid \text{pair } v_1 v_2 \mid \text{just } v \mid \text{cont } E \end{aligned}$$

$$\text{Contexts, } E = \square \mid E e \mid v E$$

Reductions:

$$\begin{aligned} E[(\lambda x.e) v] &\mapsto E[e[v/x]] && \text{BETA} \\ E[\text{callcc } e] &\mapsto E[e (\lambda x.\mathcal{A} E[x])] && \text{CALLCC} \\ E[\mathcal{A} e] &\mapsto e && \text{ABORT} \end{aligned}$$

1.2 CPS and Yield

The same way CPS eliminates callcc, the partial CPS transformation ‘T’ eliminates yield.

$$\begin{array}{lll} T[[x]] & = & x \qquad \text{VAR} \\ T[[\lambda x.e]] & = & \lambda x.T[[e]] \qquad \text{LAM} \\ T[[e_1 e_2]] & = & T[[e_1]] T[[e_2]] \qquad \text{APP} \\ T[[run e]] & = & CPS[[e]] (\lambda x.x) \qquad \text{RUN} \\ \\ CPS[[x]] & = & \lambda k.(k x) \qquad \text{CPS VAR} \\ CPS[[\lambda x.e]] & = & \lambda k.(k \lambda x.CPS[[e]]) \qquad \text{CPS ABS} \\ CPS[[e_1 e_2]] & = & \lambda k.(CPS[[e_1]] (\lambda f.CPS[[e_2]] (\lambda v.f v k))) \qquad \text{CPS APP} \\ CPS[[run e]] & = & \lambda k.(k (CPS[[e]] (\lambda x.x))) \qquad \text{CPS RUN} \\ CPS[[yield e]] & = & \lambda k.(CPS[[e]] \lambda v.(v, k)) \qquad \text{CPS YIELD} \end{array}$$