Programming Language Concepts

CSCI-344 **Recitation 11**
Term 20161

Control Operators & Reduction Semantics

1 Introduction

You will undertake a few simple problems in order to develop basic skills with control operators and reduction semantics.

2 Description

- (Adapted from Exercises 4 and 5 of Chapter 3 from *Programming Languages: Build, Prove, and Compare* (p. 292).)

  Control operators combine in tricky ways, both with each other and with regular operators. See what you make of these combinations.

  - Is there a simpler expression that behaves the same as `(return (return 1))` in every context?
  - Is there a simpler expression that behaves the same as `(return (continue))` in every context?
  - Is there a simpler expression that behaves the same as `(return (break))` in every context?
  - Is there a simpler expression that behaves the same as `(throw (return 3))` in every context?
  - Is there a simpler expression that behaves the same as `(return (throw 4))` in every context?
  - Is it true that expression `(begin` e_1 e_2 $\cdots$ `(throw exn)` $\cdots$ e_n`)` can always be replaced by `(begin` e_1 e_2 $\cdots$ `(throw exn))` without changing the behavior of the program?
  - Can the function application `(`e e_1 e_2 $\cdots$ `(throw exn)` $\cdots$ e_n`)` be simplified without changing the behavior of the program? If so, what is the simpler version? If not, why not?

  Combinations of try-catch with control operators are a bit less tricky. What outcome do you expect from these expressions?

  - ```
    (try-catch
      (try-catch (throw 'inner) (lambda (exn) (list2 'caught exn)))
      (throw 'outer))
    ```
  - ```
    (try-catch
      (throw (try-catch 'exception (lambda (exn) (list2 'inner-caught exn))))
      (lambda (exn) (list2 'outer-caught exn)))
    ```

- A number of languages (for example, the POSIX shell language) extend `break` and `continue` to exit or resume from within a nest of several loops.

  Suppose that we extend (simplified) μScheme+ with two new expression forms:

Concrete Syntax				Abstract Syntax			
exp	e	=	$\cdots$	Exp	e	=	$\cdots$
		|	`(breakk` e_k`)`			|	BREAKK(e_k)
		|	`(continuek` e_k`)`			|	CONTINUEK(e_k)

  Informally, the `(breakk` e_k`)` expression evaluates e_k to a number k greater than or equal to 1 and then exits the k^{th} enclosing loop. Similarly, the `(continuek` e_k`)` expression evaluates e_k to a number k greater than or equal to 1 and then resumes the k^{th} enclosing loop. Thus, `(breakk 1)` behaves like `(break)` and `(continuek 1)` behaves like `(continue)`.

Extend the reduction semantics for μScheme+ to support these new expression forms. You will need to add one or more frame forms as well as one or more inference rules; you should not need to modify or remove any of the existing inference rules.

- Based on the experience of the previous problem, it would be a simple exercise to extend μScheme+ with a new expression form `(returnk` e e_k`)` that evaluates e to value and then evaluates e_k to a number k greater than or equal to 1 and then returns v to the k^{th} enclosing function call. Thus, `(returnk` e `1)` behaves like `(return` e`)`.

  However, I know of no language that provides a construct to return a value from anything other than the immediately enclosing function call (which necessarily corresponds to the function in which the `return` expression appears), although games with global mutable state and `try-catch/throw`, `call/cc`, or `prompt/control` may be played to provide similar functionality.

  Do you think that this would be a useful feature? Why or why not? Do your answer and/or reasons change if considering a dynamically-typed language or a statically-typed language?

3 Requirements and Submission

You may use the reference interpreter (see Appendix A), but there may only be one active laptop in each group.

At the end of class, submit the group's solutions as hard-copy; be sure to include the names of all group members in the submission.

A Interpreter

A reference μScheme+ interpreter is available on the CS Department Linux systems (e.g., `glados.cs.rit.edu` and `queeg.cs.rit.edu` and ICLs 1 and 2) at:

`/usr/local/pub/mtf/plc/bin/uschemeplus`

Use the reference interpreter to check your code.

B Abstract Syntax of (simplified) μScheme+

$$
\begin{array}{lcl}
\text{Exp} & e & = \text{NUME}(n) \\
& & \mid \text{BOOLE}(b) \\
& & \mid \text{SYME}(s) \\
& & \mid \text{VAR}(x) \\
& & \mid \text{SET}(x, e_x) \\
& & \mid \text{IF}(e_c, e_t, e_f) \\
& & \mid \text{WHILE}(e_c, e_b) \\
& & \mid \text{BREAK} \\
& & \mid \text{CONTINUE} \\
& & \mid \text{BEGIN}(e_1, e_2) \\
& & \mid \text{LAMBDA}(x, e) \\
& & \mid \text{APPLY}(e_f, e_a) \\
& & \mid \text{RETURN}(e) \\
& & \mid \text{LET}(x, e_x, e_b) \\
& & \mid \text{LETREC}(x, e_x, e_b) \\
& & \mid \text{TRY}-\text{CATCH}(e_b, e_h) \\
& & \mid \text{THROW}(e) \\
\\
\text{Val} & v & = \text{NUMV}(n) \\
& & \mid \text{BOOLV}(b) \\
& & \mid \text{SYMV}(s) \\
& & \mid \text{CLOSURE}(x, e, \rho)
\end{array}
$$

$$
\begin{array}{lcl}
\text{Frame} & F & = \text{SET}(x, \textcircled{e_x}) \\
& & \mid \text{IF}(\textcircled{e_c}, e_t, e_f) \\
& & \mid \text{WHILE}(\textcircled{e_c}, e_b) \\
& & \mid \text{WHILE}(e_c, \textcircled{e_b}) \\
& & \mid \text{BREAK} \\
& & \mid \text{CONTINUE} \\
& & \mid \text{BEGIN}(\textcircled{e_1}, e_2) \\
& & \mid \text{BEGIN}(v_1, \textcircled{e_2}) \\
& & \mid \text{APPLY}(\textcircled{e_f}, e_a) \\
& & \mid \text{APPLY}(v_f, \textcircled{e_a}) \\
& & \mid \text{CALLENV}(\rho) \\
& & \mid \text{RETURN}(\textcircled{e}) \\
& & \mid \text{LET}(x, \textcircled{e_x}, e_b) \\
& & \mid \text{LETREC}(x, \textcircled{e_x}, e_b) \\
& & \mid \text{LETENV}(\rho) \\
& & \mid \text{TRY}-\text{CATCH}(e_b, \textcircled{e_h}) \\
& & \mid \text{TRY}-\text{CATCH}(\textcircled{e_b}, v_h) \\
& & \mid \text{THROW}(\textcircled{e})
\end{array}
$$

C Reduction Semantics of (simplified) μScheme+

$\boxed{\langle e/v, \rho, \sigma, S\rangle \to \langle e'/v', \rho', \sigma', S'\rangle}$

$$\frac{}{\langle \text{NUME}(n), \rho, \sigma, S\rangle \to \langle \text{NUMV}(n), \rho, \sigma, S\rangle} \qquad \frac{}{\langle \text{BOOLE}(b), \rho, \sigma, S\rangle \to \langle \text{BOOLV}(b), \rho, \sigma, S\rangle} \qquad \frac{}{\langle \text{SYME}(s), \rho, \sigma, S\rangle \to \langle \text{SYMV}(s), \rho, \sigma, S\rangle}$$

$$\frac{x \in \text{dom } \rho \qquad \rho(x) \in \text{dom } \sigma}{\langle \text{VAR}(x), \rho, \sigma, S\rangle \to \langle \sigma(\rho(x)), \rho, \sigma, S\rangle}$$

$$\frac{x \in \text{dom } \rho}{\langle \text{SET}(x, e_x), \rho, \sigma, S\rangle \to \langle e_x, \rho, \sigma, \text{SET}(x, \textcircled{e_x}) :: S\rangle} \qquad \frac{x \in \text{dom } \rho \qquad \rho(x) \in \text{dom } \sigma}{\langle v_x, \rho, \sigma, \text{SET}(x, \textcircled{e_x}) :: S\rangle \to \langle v_x, \rho, \sigma\{\rho(x) \mapsto v_x\}, S\rangle}$$

$$\frac{}{\langle \text{IF}(e_c, e_t, e_f), \rho, \sigma, S\rangle \to \langle e_c, \rho, \sigma, \text{IF}(\textcircled{e_c}, e_t, e_f) :: S\rangle}$$

$$\frac{v_c \neq \text{BOOLV}(\texttt{\#f})}{\langle v_c, \rho, \sigma, \text{IF}(\textcircled{e_c}, e_t, e_f) :: S\rangle \to \langle e_t, \rho, \sigma, S\rangle} \qquad \frac{v_c = \text{BOOLV}(\texttt{\#f})}{\langle v_c, \rho, \sigma, \text{IF}(\textcircled{e_c}, e_t, e_f) :: S\rangle \to \langle e_f, \rho, \sigma, S\rangle}$$

$$\frac{}{\langle \text{WHILE}(e_c, e_b), \rho, \sigma, S\rangle \to \langle e_c, \rho, \sigma, \text{WHILE}(\textcircled{e_c}, e_b) :: S\rangle}$$

$$\frac{v_c \neq \text{BOOLV}(\texttt{\#f})}{\langle v_c, \rho, \sigma, \text{WHILE}(\textcircled{e_c}, e_b) :: S\rangle \to \langle e_b, \rho, \sigma, \text{WHILE}(e_c, \textcircled{e_b}) :: S\rangle} \qquad \frac{v_c = \text{BOOLV}(\texttt{\#f})}{\langle v_c, \rho, \sigma, \text{WHILE}(\textcircled{e_c}, e_b) :: S\rangle \to \langle \text{BOOLV}(\texttt{\#f}), \rho, \sigma, S\rangle}$$

$$\frac{}{\langle v_b, \rho, \sigma, \text{WHILE}(e_c, \textcircled{e_b}) :: S\rangle \to \langle e_c, \rho, \sigma, \text{WHILE}(\textcircled{e_c}, e_b) :: S\rangle}$$

$$\frac{}{\langle \text{BREAK}, \rho, \sigma, S\rangle \to \langle \text{BOOLE}(\texttt{\#f}), \rho, \sigma, \text{BREAK} :: S\rangle}$$

$$\frac{}{\langle \text{CONTINUE}, \rho, \sigma, S\rangle \to \langle \text{BOOLE}(\texttt{\#f}), \rho, \sigma, \text{CONTINUE} :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{BREAK} :: \text{WHILE}(e_c, \textcircled{e_b}) :: S\rangle \to \langle \text{BOOLV}(\texttt{\#f}), \rho, \sigma, S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{CONTINUE} :: \text{WHILE}(e_c, \textcircled{e_b}) :: S\rangle \to \langle e_c, \rho, \sigma, \text{WHILE}(\textcircled{e_c}, e_b) :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{BREAK} :: \text{LETENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, \text{BREAK} :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{CONTINUE} :: \text{LETENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, \text{CONTINUE} :: S\rangle}$$

$$\frac{F \neq \text{WHILE}(\_, \textcircled{\_}) \qquad F \neq \text{LETENV}(\_) \qquad F \neq \text{CALLENV}(\_)}{\langle v, \rho, \sigma, \text{BREAK} :: F :: S\rangle \to \langle v, \rho, \sigma, \text{BREAK} :: S\rangle}$$

$$\frac{F \neq \text{WHILE}(\_, \textcircled{\_}) \qquad F \neq \text{LETENV}(\_) \qquad F \neq \text{CALLENV}(\_)}{\langle v, \rho, \sigma, \text{CONTINUE} :: F :: S\rangle \to \langle v, \rho, \sigma, \text{CONTINUE} :: S\rangle}$$

$$\frac{}{\langle \text{BEGIN}(e_1, e_2), \rho, \sigma, S\rangle \to \langle e_1, \rho, \sigma, \text{BEGIN}(\textcircled{e_1}, e_2) :: S\rangle}$$

$$\frac{}{\langle v_1, \rho, \sigma, \text{BEGIN}(\textcircled{e_1}, e_2) :: S\rangle \to \langle e_2, \rho, \sigma, \text{BEGIN}(v_1, \textcircled{e_2}) :: S\rangle}$$

$$\frac{}{\langle v_2, \rho, \sigma, \text{BEGIN}(v_1, \textcircled{e_2}) :: S\rangle \to \langle v_2, \rho, \sigma, S\rangle}$$

$\boxed{\langle e/v, \rho, \sigma, S\rangle \to \langle e'/v', \rho', \sigma', S'\rangle}$ *(continued)*

$$\frac{}{\langle \text{LAMBDA}(x, e), \rho, \sigma, S\rangle \to \langle \text{CLOSURE}(x, e, \rho), \rho, \sigma, S\rangle}$$

$$\frac{}{\langle \text{APPLY}(e_f, e_a), \rho, \sigma, S\rangle \to \langle e_f, \rho, \sigma, \text{APPLY}((\overline{e_f}), e_a) :: S\rangle} \qquad \frac{}{\langle v_f, \rho, \sigma, \text{APPLY}((\overline{e_f}), e_a) :: S\rangle \to \langle e_a, \rho, \sigma, \text{APPLY}(v_f, (\overline{e_a})) :: S\rangle}$$

$$\frac{v_f = \text{CLOSURE}(x, e, \rho_c) \qquad \ell \notin \text{dom } \sigma}{\langle v_a, \rho, \sigma, \text{APPLY}(v_f, (\overline{e_a})) :: S\rangle \to \langle e, \rho_c\{x \mapsto \ell\}, \sigma\{\ell \mapsto v_a\}, \text{CALLENV}(\rho) :: S\rangle} \qquad \frac{}{\langle v, \rho, \sigma, \text{CALLENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, S\rangle}$$

$$\frac{}{\langle \text{RETURN}(e), \rho, \sigma, S\rangle \to \langle e, \rho, \sigma, \text{RETURN}((\overline{e})) :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{RETURN}((\overline{e})) :: \text{CALLENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{RETURN}((\overline{e})) :: \text{LETENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, \text{RETURN}((\overline{e})) :: S\rangle}$$

$$\frac{F \neq \text{LETENV}(\_) \qquad F \neq \text{CALLENV}(\_)}{\langle v, \rho, \sigma, \text{RETURN}((\overline{e})) :: F :: S\rangle \to \langle v, \rho, \sigma, \text{RETURN}((\overline{e})) :: S\rangle}$$

$$\frac{}{\langle \text{LET}(x, e_x, e_b), \rho, \sigma, S\rangle \to \langle e_x, \rho, \sigma, \text{LET}(x, (\overline{e_x}), e_b) :: S\rangle} \qquad \frac{\ell \notin \text{dom } \sigma}{\langle v_x, \rho, \sigma, \text{LET}(x, (\overline{e_x}), e_b) :: S\rangle \to \langle e_b, \rho\{x \mapsto \ell\}, \sigma\{\ell \mapsto v_x\}, \text{LETENV}(\rho) :: S\rangle}$$

$$\frac{\ell \notin \text{dom } \sigma}{\langle \text{LETREC}(x, e_x, e_b), \rho, \sigma, S\rangle \to \langle e_x, \rho\{x \mapsto \ell\}, \sigma\{\ell \mapsto \text{unspec}\}, \text{LETREC}(x, (\overline{e_x}), e_b) :: \text{LETENV}(\rho) :: S\rangle}$$

$$\frac{x \in \text{dom } \rho \qquad \rho(x) \in \text{dom } \sigma}{\langle v_x, \rho, \sigma, \text{LETREC}(x, (\overline{e_x}), e_b) :: S\rangle \to \langle e_b, \rho, \sigma\{\rho(x) \mapsto v_x\}, S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{LETENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, S\rangle}$$

$$\frac{}{\langle \text{TRY}{-}\text{CATCH}(e_b, e_h), \rho, \sigma, S\rangle \to \langle e_h, \rho, \sigma, \text{TRY}{-}\text{CATCH}(e_b, (\overline{e_h})) :: S\rangle}$$

$$\frac{}{\langle v_h, \rho, \sigma, \text{TRY}{-}\text{CATCH}(e_b, (\overline{e_h})) :: S\rangle \to \langle e_b, \rho, \sigma, \text{TRY}{-}\text{CATCH}((\overline{e_b}), v_h) :: S\rangle}$$

$$\frac{}{\langle v_b, \rho, \sigma, \text{TRY}{-}\text{CATCH}((\overline{e_b}), v_h) :: S\rangle \to \langle v_b, \rho, \sigma, S\rangle}$$

$$\frac{}{\langle \text{THROW}(e), \rho, \sigma, S\rangle \to \langle e, \rho, \sigma, \text{THROW}((\overline{e})) :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{THROW}((\overline{e})) :: \text{TRY}{-}\text{CATCH}((\overline{e_b}), v_h) :: S\rangle \to \langle v, \rho, \sigma, \text{APPLY}(v_h, (\overline{e})) :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{THROW}((\overline{e})) :: \text{LETENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, \text{THROW}((\overline{e})) :: S\rangle}$$

$$\frac{}{\langle v, \rho, \sigma, \text{THROW}((\overline{e})) :: \text{CALLENV}(\rho') :: S\rangle \to \langle v, \rho', \sigma, \text{THROW}((\overline{e})) :: S\rangle}$$

$$\frac{F \neq \text{TRY}{-}\text{CATCH}((\overline{\_}), \_) \qquad F \neq \text{LETENV}(\_) \qquad F \neq \text{CALLENV}(\_)}{\langle v, \rho, \sigma, \text{THROW}((\overline{e})) :: F :: S\rangle \to \langle v, \rho, \sigma, \text{THROW}((\overline{e})) :: S\rangle}$$