

# Operational Semantics of ImpCore

Bill Harrison

Computer Science Department, University of Missouri

CS4450: Principles of Programming Languages

- Today we begin looking at operational semantics for ImpCore
- An *operational semantics* for a language gives a precise account of what it means to execute programs in that language
- It is one common means of defining precisely what a language means
- Reminder: homework 3 is due on Friday, October 16, by 3pm

# Abstract Syntax for ImpCore

TopLevel

```
= EXP (Exp)
| DEFINE (Name, Namelist, Exp)
| VAL (Name, Exp)
| USE (Name)
```

Exp = LITERAL (Value)

```
| VAR (Name)
| SET (Name, Exp)
| IF (Exp, Exp, Exp)
| WHILE (Exp, Exp)
| BEGIN (Explist)
| APPLY (Name, Explist)
```

- From Ramsey/Kamin, page 12
- What does this mean?
- How do we represent it in Haskell?
- How do we represent concrete ImpCore terms?

# ImpCore Abstract Syntax: What does it mean?

```
TopLevel
= EXP (Exp)
| DEFINE (Name, Namelist, Exp)
| VAL (Name, Exp)
| USE (Name)

Exp = LITERAL (Value)
    | VAR (Name)
    | SET (Name, Exp)
    | IF (Exp, Exp, Exp)
    | WHILE (Exp, Exp)
    | BEGIN (Explist)
    | APPLY (Name, Explist)
```

- Distinguishes between top level *declarations* (Toplevel) and *expressions* (Exp)
- A Toplevel declaration defines names of procedures, values, etc.
- An Exp is a program chunk
- What's an example of a Toplevel and Exp for Haskell?

# ImpCore Abstract Syntax: Haskell Representation?

```
TopLevel
= EXP (Exp)
| DEFINE (Name, Namelist, Exp)
| VAL (Name, Exp)
| USE (Name)

Exp = LITERAL (Value)
    | VAR (Name)
    | SET (Name, Exp)
    | IF (Exp, Exp, Exp)
    | WHILE (Exp, Exp)
    | BEGIN (Explist)
    | APPLY (Name, Explist)
```

Clearly, these will correspond to **data** declarations—which ones?

# ImpCore Abstract Syntax: Haskell Representation?

```
TopLevel
= EXP (Exp)
| DEFINE (Name, Namelist, Exp)
| VAL (Name, Exp)
| USE (Name)

Exp = LITERAL (Value)
    | VAR (Name)
    | SET (Name, Exp)
    | IF (Exp, Exp, Exp)
    | WHILE (Exp, Exp)
    | BEGIN (Explist)
    | APPLY (Name, Explist)
```

Clearly, these will correspond to **data** declarations—which ones?

Here's a start (what's left?):

```
data TopLevel
= EXP Exp
| DEFINE Name Namelist Exp
| VAL Name Exp
| USE Name

data Exp
= LITERAL Value
| VAR Name
| SET Name Exp
| IF Exp Exp Exp
| WHILE Exp Exp
| BEGIN Explist
| APPLY Name Explist
```

# ImpCore Abstract Syntax: Haskell Representation, what's left?

```
data TopLevel
  = EXP Exp
  | DEFINE Name Namelist Exp
  | VAL Name Exp
  | USE Name
```

```
data Exp
  = LITERAL Value
  | VAR Name
  | SET Name Exp
  | IF Exp Exp Exp
  | WHILE Exp Exp
  | BEGIN Explist
  | APPLY Name Explist
```

Need definitions for Name, Value and the lists...

```
type Name      = String
type Value     = Int
type Explist   = [Exp]
type Namelist  = [Name]
```

# ImpCore Abstract Syntax: Representing Concrete Terms

```
data TopLevel
  = EXP Exp
  | DEFINE Name Namelist Exp
  | VAL Name Exp
  | USE Name
type Name      = String
type Value     = Int
type Explist   = [Exp]
type Namelist  = [Name]

data Exp
  = LITERAL Value
  | VAR Name
  | SET Name Exp
  | IF Exp Exp Exp
  | WHILE Exp Exp
  | BEGIN Explist
  | APPLY Name Explist
```

How would you represent the following concrete ImpCore expression?

```
(set i (- (+ (* 2 j) i) (/ k 3)))
```

- An expression like “ $(* x 3)$ ” has no value per se. Why?

# Environments

- An expression like “ $(* x 3)$ ” has no value per se. Why?
- ...because  $x$  is *free* in it (i.e., it is just a placeholder with no meaning).

# Environments

- An expression like “ $(* x 3)$ ” has no value per se. Why?
- ...because  $x$  is *free* in it (i.e., it is just a placeholder with no meaning).
- An *environment* is a **structure** used to bind names to meanings
- in some environments,  $x$  has an appropriate meaning and in others, it doesn't

# Environments

- An expression like “ $(* x 3)$ ” has no value per se. Why?
- ...because  $x$  is *free* in it (i.e., it is just a placeholder with no meaning).
- An *environment* is a **structure** used to bind names to meanings
- in some environments,  $x$  has an appropriate meaning and in others, it doesn't

Say  $\rho$  is an environment

# Environments

- An expression like “ $(* x 3)$ ” has no value per se. Why?
- ...because  $x$  is *free* in it (i.e., it is just a placeholder with no meaning).
- An *environment* is a **structure** used to bind names to meanings
- in some environments,  $x$  has an appropriate meaning and in others, it doesn't

Say  $\rho$  is an environment

- $\rho(x)$  is the value of  $x$  in environment  $\rho$

# Environments

- An expression like “ $(* x 3)$ ” has no value per se. Why?
- ...because  $x$  is *free* in it (i.e., it is just a placeholder with no meaning).
- An *environment* is a **structure** used to bind names to meanings
- in some environments,  $x$  has an appropriate meaning and in others, it doesn't

Say  $\rho$  is an environment

- $\rho(x)$  is the value of  $x$  in environment  $\rho$
- $\rho\{x \mapsto v\}$  is the extension of  $\rho$  with a new binding

# Environments

- An expression like “ $(* x 3)$ ” has no value per se. Why?
- ...because  $x$  is *free* in it (i.e., it is just a placeholder with no meaning).
- An *environment* is a **structure** used to bind names to meanings
- in some environments,  $x$  has an appropriate meaning and in others, it doesn't

Say  $\rho$  is an environment

- $\rho(x)$  is the value of  $x$  in environment  $\rho$
- $\rho\{x \mapsto v\}$  is the extension of  $\rho$  with a new binding
- ...defined by:

$$\rho\{x \mapsto v\}(y) = \begin{cases} v & \text{when } x = y \\ \rho(y) & \text{when } x \neq y \end{cases}$$

# Environments as they occur in ImpCore

Consider the following transcript:

```
ImpCore> (val x 2)
2
ImpCore> (define x (y) (+ x y))
ImpCore> (define z (x) (x x))
ImpCore> (z 4)
6
```

# Environments as they occur in ImpCore

Consider the following transcript:

```
ImpCore> (val x 2)
2
ImpCore> (define x (y) (+ x y))
ImpCore> (define z (x) (x x))
ImpCore> (z 4)
6
```

- How many different ways is `x` used in this code?

# Environments as they occur in ImpCore

Consider the following transcript:

```
ImpCore> (val x 2)
2
ImpCore> (define x (y) (+ x y))
ImpCore> (define z (x) (x x))
ImpCore> (z 4)
6
```

- How many different ways is `x` used in this code?
- Why does `(z 4)` return 6?

# Environments as they occur in ImpCore

There are three different versions of `x`:

```
ImpCore> (val x 2)                -- global binding
2
ImpCore> (define x (y) (+ x y))  -- function binding
ImpCore> (define z (x) (x x))    -- formal argument
ImpCore> (z 4)
6
```

# Environments as they occur in ImpCore

There are three different versions of x:

```
ImpCore> (val x 2)                -- global binding
2
ImpCore> (define x (y) (+ x y))  -- function binding
ImpCore> (define z (x) (x x))    -- formal argument
ImpCore> (z 4)
6
```

Upshot: Each version requires its own environment

# Operational Semantics

- An operational semantics specifies precisely what the meaning of a program is. Or, rather, precisely how a program is executed

# Operational Semantics

- An operational semantics specifies precisely what the meaning of a program is. Or, rather, precisely how a program is executed
- Operational semantics are typically state machines
  - There is an initial state for the machine
  - There are rules for state transitions
  - Program execution starts in the initial state and makes transitions (until, possibly, an answer is reached)

- An operational semantics specifies precisely what the meaning of a program is. Or, rather, precisely how a program is executed
- Operational semantics are typically state machines
  - There is an initial state for the machine
  - There are rules for state transitions
  - Program execution starts in the initial state and makes transitions (until, possibly, an answer is reached)
- Expressions: state of an ImpCore machine:  $\langle e, \xi, \phi, \rho \rangle$ 
  - $e$  — expression being evaluated
  - $\xi$  — values of globals (“zee”)
  - $\phi$  — function definitions (“fee”)
  - $\rho$  — values of formal arguments (“row”)
- Top level: state of an ImpCore machine:  $\langle t, \xi, \phi \rangle$ 
  - $t$  — top level declaration

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$
- Function Definitions Environment ( $\phi$ 's): binds Name's to *function definitions*; these have two forms:

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$
- Function Definitions Environment ( $\phi$ 's): binds Name's to *function definitions*; these have two forms:
  - User-defined: value created by the top-level declaration

```
(define f (x1...xn) e) -- Top-level decl.  
USER(x1...xn, e)      -- ...& its value
```

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$
- Function Definitions Environment ( $\phi$ 's): binds Name's to *function definitions*; these have two forms:
  - User-defined: value created by the top-level declaration

```
(define f (x1...xn) e) -- Top-level decl.  
USER(x1...xn, e)      -- ...& its value
```

- ...and the corresponding binding is  $f \mapsto \text{USER}(x_1 \cdots x_n, e)$

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$
- Function Definitions Environment ( $\phi$ 's): binds Name's to *function definitions*; these have two forms:
  - User-defined: value created by the top-level declaration

```
(define f (x1...xn) e) -- Top-level decl.  
USER(x1...xn, e)      -- ...& its value
```

- ...and the corresponding binding is  $f \mapsto \text{USER}(x_1 \cdots x_n, e)$
- Primitives: if  $\oplus$  is a primitive, then  $\text{PRIMITIVE}(\oplus)$  is its value

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$
- Function Definitions Environment ( $\phi$ 's): binds Name's to *function definitions*; these have two forms:
  - User-defined: value created by the top-level declaration

```
(define f (x1...xn) e) -- Top-level decl.  
USER(x1...xn, e)      -- ...& its value
```

- ...and the corresponding binding is  $f \mapsto \text{USER}(x_1 \dots x_n, e)$
- Primitives: if  $\oplus$  is a primitive, then  $\text{PRIMITIVE}(\oplus)$  is its value
- ...and the corresponding binding is  $\oplus \mapsto \text{PRIMITIVE}(\oplus)$

# Environments **bind** names to values

- Typical binding in an environment written:  $x \mapsto v$
- Function Definitions Environment ( $\phi$ 's): binds Name's to *function definitions*; these have two forms:
  - User-defined: value created by the top-level declaration

```
(define f (x1...xn) e) -- Top-level decl.  
USER(x1...xn, e)      -- ...& its value
```

- ...and the corresponding binding is  $f \mapsto \text{USER}(x_1 \dots x_n, e)$
  - Primitives: if  $\oplus$  is a primitive, then  $\text{PRIMITIVE}(\oplus)$  is its value
  - ...and the corresponding binding is  $\oplus \mapsto \text{PRIMITIVE}(\oplus)$
- Global ( $\xi$ ) and Formal Parameter ( $\rho$ ) environments bind Name's to integers
  - ...they consist of bindings like:  $x \mapsto 99$

# Operational Semantics (Judgments)

- A **judgment** is a transition in an operational semantics

# Operational Semantics (Judgments)

- A **judgment** is a transition in an operational semantics
- $\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle$ 
  - ...means “evaluating  $e$  produces value  $v$ ”
  - ...may represent more than one transition

# Operational Semantics (Judgments)

- A **judgment** is a transition in an operational semantics
- $\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle$ 
  - ...means “evaluating  $e$  produces value  $v$ ”
  - ...may represent more than one transition
- Evaluating Expressions
  - ...always produces a value (unless non-termination/error occurs)
  - ...may change global variables (in  $\xi$ ) or formal params (in  $\rho$ )
  - ...never results in a new function definition ( $\phi$  is always unchanged)

# Operational Semantics (Judgments)

- A **judgment** is a transition in an operational semantics
- $\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle$ 
  - ...means “evaluating  $e$  produces value  $v$ ”
  - ...may represent more than one transition
- Evaluating Expressions
  - ...always produces a value (unless non-termination/error occurs)
  - ...may change global variables (in  $\xi$ ) or formal params (in  $\rho$ )
  - ...never results in a new function definition ( $\phi$  is always unchanged)
- Judgments are either true or false
  - Some judgments will hold:  $\langle (+\ 1\ 1), \xi, \phi, \rho \rangle \Downarrow \langle 2, \xi, \phi, \rho \rangle$
  - ...some judgments won't:  $\langle (+\ 1\ 1), \xi, \phi, \rho \rangle \not\Downarrow \langle 4, \xi, \phi, \rho \rangle$

# Judgments for top-level declarations

- There is another form of judgment for top-level declarations:
  - $\langle t, \xi, \phi \rangle \rightarrow \langle \xi', \phi' \rangle$
  - N.b., different judgment, different arrow.
- This means: "evaluating a top-level item in environments  $\xi$  and  $\phi$  results in new environments  $\xi'$  and  $\phi'$ ."

# Rules of Inference

- Q: How do you decide which judgments hold and which do not?

# Rules of Inference

- Q: How do you decide which judgments hold and which do not?
- A: Rules of inference: they say *if I know these judgments hold, then this other judgment holds*

# Rules of Inference

- Q: How do you decide which judgments hold and which do not?
- A: Rules of inference: they say *if I know these judgments hold, then this other judgment holds*
- Rules of inference are written as:

$$\frac{\text{premises}}{\text{conclusion}} \text{ (Name-of-Rule)}$$

- Here is a simple inference rule for ImpCore: formal variable lookup:

$$\frac{x \in \text{dom}(\rho)}{\langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle} \text{ (FormalVar)}$$

What does it mean?

# Rules of Inference

- Q: How do you decide which judgments hold and which do not?
- A: Rules of inference: they say *if I know these judgments hold, then this other judgment holds*
- Rules of inference are written as:

$$\frac{\text{premises}}{\text{conclusion}} \text{ (Name-of-Rule)}$$

- Here is a simple inference rule for ImpCore: formal variable lookup:

$$\frac{x \in \text{dom}(\rho)}{\langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle} \text{ (FormalVar)}$$

What does it mean?

If  $x \mapsto v$  is a binding in  $\rho$ , then it is a formal parameter to a function and  $\text{VAR}(x)$  evaluates to  $v$

$$\frac{x \in \text{dom}(\rho)}{\langle \text{LITERAL}(v), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi, \phi, \rho \rangle} \text{ (Literal)}$$

$$\frac{x \notin \text{dom}(\rho) \quad x \in \text{dom}(\xi)}{\langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \xi x, \xi, \phi, \rho \rangle} \text{ (GlobalVar)}$$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \quad (\text{FormalAssign})$$

$$\frac{\begin{array}{l} x \notin \text{dom}(\rho) \\ x \in \text{dom}(\xi) \\ \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle \end{array}}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi' \{x \mapsto v\}, \phi, \rho \rangle} \quad (\text{GlobalAssign})$$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \text{ (FormalAssign)}$$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \text{ (FormalAssign)}$$

- 1 Want to evaluate  $\text{SET}(x, e)$  in  $\xi, \phi$  and  $\rho$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \text{ (FormalAssign)}$$

- 1 Want to evaluate  $\text{SET}(x, e)$  in  $\xi, \phi$  and  $\rho$
- 2  $x$  is a formal parameter:  $x \in \rho$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \text{ (FormalAssign)}$$

- 1 Want to evaluate  $\text{SET}(x, e)$  in  $\xi, \phi$  and  $\rho$
- 2  $x$  is a formal parameter:  $x \in \rho$
- 3 Evaluating  $e$  produces value  $v$  and new global and formal env's  $\xi', \rho'$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \text{ (FormalAssign)}$$

- 1 Want to evaluate  $\text{SET}(x, e)$  in  $\xi, \phi$  and  $\rho$
- 2  $x$  is a formal parameter:  $x \in \rho$
- 3 Evaluating  $e$  produces value  $v$  and new global and formal env's  $\xi', \rho'$
- 4 ...so return  $v$  value,  $\xi'$  and updated formal arg. env.,  $\rho' \{x \mapsto v\}$

$$\frac{x \in \text{dom}(\rho) \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle} \text{ (FormalAssign)}$$

$\text{eval} :: \text{ExpStore} \rightarrow \text{ExpStore}$

$\text{eval} \langle \text{SET}(x, e), \xi, \phi, \rho \rangle \mid (x \in \text{dom}(\rho)) = \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle$

**where**

$\langle v, \xi', \phi, \rho' \rangle = \text{eval} \langle e, \xi, \phi, \rho \rangle$

N.b., this gives you a template for translating the operational semantics of ImpCore into Haskell. Etch this in your minds as it will be very useful in HW4.

$$\begin{array}{l} \phi(f) = \text{PRIMITIVE}(+) \\ \langle e_0, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_0, \xi_1, \phi, \rho_1 \rangle \\ \langle e_1, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_1, \xi_2, \phi, \rho_2 \rangle \\ \hline \langle \text{APPLY}(f, e_0, e_1), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2, \phi, \rho_2 \rangle \end{array} \quad (\text{Primitive})$$

$$\begin{array}{c} \phi(f) = \text{USER}(\langle x_1, \dots, x_n \rangle, \text{body}) \\ \langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_0, \xi_1, \phi, \rho_1 \rangle \\ \vdots \\ \langle e_n, \xi_{n-1}, \phi, \rho_{n-1} \rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle \\ \langle \text{body}, \xi_n, \phi, \rho_n \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle \\ \hline \langle \text{APPLY}(f, e_1, \dots, e_n), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle \end{array} \quad (\text{ApplyUser})$$

$$\frac{\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \quad v_1 \neq 0 \quad \langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle}{\langle IF(e_1, e_2, e_3), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle} \textit{IfTrue}$$

$$\frac{\begin{array}{c} \langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \\ v_1 \neq 0 \\ \langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle \end{array}}{\langle IF(e_1, e_2, e_3), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle} \textit{IfTrue}$$

$$\frac{\begin{array}{c} \langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \\ v_1 = 0 \\ \langle e_3, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_3, \xi_3, \phi, \rho_3 \rangle \end{array}}{\langle IF(e_1, e_2, e_3), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_3, \xi_3, \phi, \rho_3 \rangle} \textit{IfFalse}$$

# Control Flow: Iteration

$$\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle$$

$$v_1 \neq 0$$

$$\langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle$$

?????

$$\frac{\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \quad \langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle}{\langle \text{WHILE}(e_1, e_2), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_3, \xi_3, \phi, \rho_3 \rangle} \textit{WhileIterate}$$

# Control Flow: Iteration

$$\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle$$

$$v_1 \neq 0$$

$$\langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle$$

?????

$$\frac{}{\langle \text{WHILE}(e_1, e_2), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_3, \xi_3, \phi, \rho_3 \rangle} \textit{WhileIterate}$$

$$\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle$$

$$v_1 \neq 0$$

$$\langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle$$

$$\langle \text{WHILE}(e_1, e_2), \xi_2, \phi, \rho_2 \rangle \Downarrow \langle v_3, \xi_3, \phi, \rho_3 \rangle$$

$$\frac{}{\langle \text{WHILE}(e_1, e_2), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_3, \xi_3, \phi, \rho_3 \rangle} \textit{WhileIterate}$$

$$\frac{\langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \quad v_1 = 0}{\langle \text{WHILE}(e_1, e_2), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle 0, \xi_1, \phi, \rho_1 \rangle} \textit{WhileEnd}$$