15-411: Dynamic Semantics

Jan Hoffmann

Dynamic Semantics

- **Static semantics:** definition of valid programs
- **Dynamic semantics:** definition of how programs are executed
- So far: Dynamic semantics is given in English on lab handouts
  - This only works since you know how C programs should behave
  - Sometimes needed to consult the reference compiler
- A description in English will always be ambiguous

➡ **Need precise ways of defining the meaning of programs**

Types of (Formal) Dynamic Semantics

- **Denotational Semantics:** Abstract and elegant.
  - Each part of a program is associated with a denotation (math. object)
  - For example: a procedure is associated with a mathematical function
- **Axiomatic Semantics:** Strongly related to program logic.
  - Gives meaning to phrases using logical axioms
  - The meaning is identical to the set of properties that can be proved
- **Operational Semantics:** Describes how programs are executed
  - Related to interpreters and abstract machines
  - Most popular and flexible form of semantics

Types of (Formal) Dynamic Semantics

Dana Scott

- **Denotational Semantics:** Abstract and elegant.
  - Each part of a program is associated with a denotation (math. object)
  - For example: a procedure is associated with a mathematical function
- **Axiomatic Semantics:** Strongly related to program logic.
  - Gives meaning to phrases using logical axioms
  - The meaning is identical to the set of properties that can be proved
- **Operational Semantics:** Describes how programs are executed
  - Related to interpreters and abstract machines
  - Most popular and flexible form of semantics

Types of (Formal) Dynamic Semantics

- **Denotational Semantics:** Abstract and elegant.

  Dana Scott

  - Each part of a program is associated with a denotation (math. object)
  - For example: a procedure is associated with a mathematical function
- **Axiomatic Semantics:** Strongly related to program logic.

  Tony Hoare

  - Gives meaning to phrases using logical axioms
  - The meaning is identical to the set of properties that can be proved
- **Operational Semantics:** Describes how programs are executed
  - Related to interpreters and abstract machines
  - Most popular and flexible form of semantics

Types of (Formal) Dynamic Semantics

- **Denotational Semantics:** Abstract and elegant.
  - Each part of a program is associated with a denotation (math. object)
  - For example: a procedure is associated with a mathematical function

Dana Scott

- **Axiomatic Semantics:** Strongly related to program logic.
  - Gives meaning to phrases using logical axioms
  - The meaning is identical to the set of properties that can be proved

Tony Hoare

- **Operational Semantics:** Describes how programs are executed
  - Related to interpreters and abstract machines
  - Most popular and flexible form of semantics

Bob Harper

Operational Semantics

- **Many different styles**
  - Natural semantics (or big-step semantics or evaluation dynamics)
  - Structural operational semantics
  - Substructural operational semantics
  - Abstract machine (or small-step with continuation)
- **We will use an abstract machine**
  - Very general: can describe non-termination, concurrency, …
  - Low-level and elaborate

Operational Semantics

- **Many different styles**
  - Natural semantics (or big-step semantics or evaluation dynamics)
  - Structural operational semantics
  - Substructural operational semantics
  - Abstract machine (or small-step with continuation)
- **We will use an abstract machine**
  - Very general: can describe non-termination, concurrency, …
  - Low-level and elaborate

Frank Pfenning

Operational Semantics

- **Many different styles**
  - Natural semantics (or big-step semantics or evaluation dynamics)
  - Structural operational semantics
  - Substructural operational semantics
  - Abstract machine (or small-step with continuation)

Frank Pfenning

- **We will use an abstract machine**
  - Very general: can describe non-termination, concurrency, …
  - Low-level and elaborate

How to pick the right dynamic semantics?

Evaluating Expressions

Continuations

Want to model a single evaluation step

$$e \to e'$$

Continuations

Want to model a single evaluation step

$$e \to e'$$

For example: $((4+5)*10+2) \to (9*10+2)$

Continuations

Want to model a single evaluation step

$$e \to e'$$

For example: $((4+5)*10+2) \to (9*10+2)$

How can we find the right place at which to make the step?

Continuations

Want to model a single evaluation step

$$e \to e'$$

For example: $((4 + 5) * 10 + 2) \to (9 * 10 + 2)$

How can we find the right place at which to make the step?

Use a continuation K:

$$e \triangleright K$$

“Evaluate expression e and pass the result to K”

The continuation has a ‘hole’ for the result value of e.

Continuations

Want to model a single evaluation step

$$e \to e'$$

For example: $((4+5)*10+2) \to (9*10+2)$

How can we find the right place at which to make the step?

Use a continuation K:

A stack of partial computations.

$$e \rhd K$$

"Evaluate expression e and pass the result to K"

The continuation has a 'hole' for the result value of e.

Evaluation Rules: Addition

$$e_1 + e_2 \rhd K \qquad \longrightarrow \qquad e_1 \rhd (\_ + e_2 \; , K)$$

Evaluation Rules: Addition

First evaluate e1.

$$e_1 + e_2 \triangleright K \qquad \longrightarrow \qquad e_1 \triangleright (\_ + e_2\ ,\ K)$$

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \triangleright K \quad \longrightarrow \quad e_1 \triangleright (\_ + e_2 \; , \; K)$$

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \triangleright K \quad \longrightarrow \quad e_1 \triangleright (\_ + e_2 \ , \ K)$$

$$c_1 \triangleright (\_ + e_2 \ , \ K) \quad \longrightarrow \quad e_2 \triangleright (c_1 + \_ \ , \ K)$$

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \rhd K \quad \longrightarrow \quad e_1 \rhd (\_ + e_2\ ,\ K)$$

e is a constant.

$$c_1 \rhd (\_ + e_2\ ,\ K) \quad \longrightarrow \quad e_2 \rhd (c_1 + \_\ ,\ K)$$

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \triangleright K \quad \longrightarrow \quad e_1 \triangleright (\_ + e_2\ ,\ K)$$

e is a constant.

$$c_1 \triangleright (\_ + e_2\ ,\ K) \quad \longrightarrow \quad e_2 \triangleright (c_1 + \_\ ,\ K)$$

Continuation is an addition.

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \rhd K \quad \longrightarrow \quad e_1 \rhd (\_ + e_2\ ,\ K)$$

e is a constant.

Continue with evaluating e2.

$$c_1 \rhd (\_ + e_2\ ,\ K) \quad \longrightarrow \quad e_2 \rhd (c_1 + \_\ ,\ K)$$

Continuation is an addition.

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \rhd K \quad \longrightarrow \quad e_1 \rhd (\_ + e_2\ ,\ K)$$

e is a constant.

Continue with evaluating e2.

Plug the result here.

$$c_1 \rhd (\_ + e_2\ ,\ K) \quad \longrightarrow \quad e_2 \rhd (c_1 + \_\ ,\ K)$$

Continuation is an addition.

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \rhd K \longrightarrow e_1 \rhd (\_ + e_2 \ , K)$$

e is a constant.

Continue with evaluating e2.

Plug the result here.

$$c_1 \rhd (\_ + e_2 \ , K) \longrightarrow e_2 \rhd (c_1 + \_ \ , K)$$

Continuation is an addition.

$$c_2 \rhd (c_1 + \_ \ , K) \longrightarrow c \rhd K \qquad (c = c_1 + c_2 \bmod 2^{32})$$

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \rhd K \quad \longrightarrow \quad e_1 \rhd (\_ + e_2 \;, K)$$

e is a constant.

Continue with evaluating e2.

Plug the result here.

$$c_1 \rhd (\_ + e_2 \;, K) \quad \longrightarrow \quad e_2 \rhd (c_1 + \_ \;, K)$$

Continuation is an addition.

$$c_2 \rhd (c_1 + \_ \;, K) \quad \longrightarrow \quad c \rhd K \qquad (c = c_1 + c_2 \bmod 2^{32})$$

Two constants

Evaluation Rules: Addition

First evaluate e1.

Plug the result here.

$$e_1 + e_2 \rhd K \longrightarrow e_1 \rhd (\_ + e_2 \ , K)$$

e is a constant.

Continue with evaluating e2.

Plug the result here.

$$c_1 \rhd (\_ + e_2 \ , K) \longrightarrow e_2 \rhd (c_1 + \_ \ , K)$$

Continuation is an addition.

$$c_2 \rhd (c_1 + \_ \ , K) \longrightarrow c \rhd K \qquad (c = c_1 + c_2 \bmod 2^{32})$$

Two constants

Actual addition.

Evaluation Rules: Binary Operations

Arithmetic operations are treated like addition

$$e_1 \oplus e_2 \triangleright K \quad \longrightarrow \quad e_1 \triangleright (\_ \oplus e_2\ , K)$$

$$c_1 \triangleright (\_ \oplus e_2\ , K) \quad \longrightarrow \quad e_2 \triangleright (c_1 \oplus \_\ , K)$$

$$c_2 \triangleright (c_1 \oplus \_\ , K) \quad \longrightarrow \quad c \triangleright K \qquad (c = c_1 \oplus c_2 \bmod 2^{32})$$

Arithmetic is modulo 2^{32} to match our x86 architecture

Evaluation Rules: Binary Operations

Arithmetic operations are treated like addition

$$e_1 \oplus e_2 \rhd K \quad \longrightarrow \quad e_1 \rhd (\_ \oplus e_2 \ , K)$$

$$c_1 \rhd (\_ \oplus e_2 \ , K) \quad \longrightarrow \quad e_2 \rhd (c_1 \oplus \_ \ , K)$$

$$c_2 \rhd (c_1 \oplus \_ \ , K) \quad \longrightarrow \quad c \rhd K \qquad (c = c_1 \oplus c_2 \bmod 2^{32})$$

Arithmetic is modulo 2^{32} to match our x86 architecture

What about effects?

Evaluation Rules: Binops with Effects

In case of an arithmetic exception: Abort the computation and report and error

$$
\begin{array}{llll}
e_1 \oslash e_2 \triangleright K & \longrightarrow & e_1 \triangleright (\_ \oslash e_2 \ , K) & \\
c_1 \triangleright (\_ \oslash e_2 \ , K) & \longrightarrow & e_2 \triangleright (c_1 \oslash \_ \ , K) & \\
c_2 \triangleright (c_1 \oslash \_ \ , K) & \longrightarrow & c \triangleright K & (c = c_1 \oslash c_2) \\
c_2 \triangleright (c_1 \oslash \_ \ , K) & \longrightarrow & \mathsf{exception}(\mathsf{arith}) & (c_1 \oslash c_2 \text{ undefined})
\end{array}
$$

There is no rule for further evaluating an exception.

Example Evaluation

$$((4+5)*10)+2 \quad \triangleright \quad \cdot$$

Example Evaluation

$$
\begin{array}{llll}
 & ((4+5)*10)+2 & \triangleright & \cdot \\
\longrightarrow & (4+5)*10 & \triangleright & \_ + 2
\end{array}
$$

Example Evaluation

$$
\begin{array}{llll}
 & ((4+5)*10)+2 & \triangleright & \cdot \\
\\
\longrightarrow & (4+5)*10 & \triangleright & \_+2 \\
\longrightarrow & 4+5 & \triangleright & \_*10\ ,\ \_+2
\end{array}
$$

Example Evaluation

$$
\begin{array}{llll}
 & ((4+5)*10)+2 & \rhd & \cdot \\
\\
\longrightarrow & (4+5)*10 & \rhd & \_ + 2 \\
\longrightarrow & 4+5 & \rhd & \_ * 10\ ,\ \_ + 2 \\
\longrightarrow & 4 & \rhd & \_ + 5\ ,\ \_ * 10\ ,\ \_ + 2
\end{array}
$$

Example Evaluation

$$
\begin{array}{llll}
 & ((4+5)*10)+2 & \rhd & \cdot \\
\\
\longrightarrow & (4+5)*10 & \rhd & \_+2 \\
\longrightarrow & 4+5 & \rhd & \_*10\ ,\ \_+2 \\
\longrightarrow & 4 & \rhd & \_+5\ ,\ \_*10\ ,\ \_+2 \\
\longrightarrow & 5 & \rhd & 4+\_\ ,\ \_*10\ ,\ \_+2
\end{array}
$$

Example Evaluation

$$
\begin{array}{rlcl}
 & ((4+5)*10)+2 & \rhd & \cdot \\
\\
\longrightarrow & (4+5)*10 & \rhd & \_+2 \\
\longrightarrow & 4+5 & \rhd & \_*10\ ,\ \_+2 \\
\longrightarrow & 4 & \rhd & \_+5\ ,\ \_*10\ ,\ \_+2 \\
\longrightarrow & 5 & \rhd & 4+\_\ ,\ \_*10\ ,\ \_+2 \\
\longrightarrow & 9 & \rhd & \_*10\ ,\ \_+2
\end{array}
$$

Example Evaluation

$$
\begin{array}{llll}
& ((4+5)*10)+2 & \rhd & \cdot \\
\\
\longrightarrow & (4+5)*10 & \rhd & \_ + 2 \\
\longrightarrow & 4+5 & \rhd & \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 4 & \rhd & \_ + 5 \;,\; \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 5 & \rhd & 4 + \_ \;,\; \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 9 & \rhd & \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 10 & \rhd & 9 * \_ \;,\; \_ + 2
\end{array}
$$

Example Evaluation

$$
\begin{array}{llcl}
 & ((4+5)*10)+2 & \rhd & \cdot \\
\\
\longrightarrow & (4+5)*10 & \rhd & \_+2 \\
\longrightarrow & 4+5 & \rhd & \_*10\ ,\ \_+2 \\
\longrightarrow & 4 & \rhd & \_+5\ ,\ \_*10\ ,\ \_+2 \\
\longrightarrow & 5 & \rhd & 4+\_\ ,\ \_*10\ ,\ \_+2 \\
\longrightarrow & 9 & \rhd & \_*10\ ,\ \_+2 \\
\longrightarrow & 10 & \rhd & 9*\_\ ,\ \_+2 \\
\longrightarrow & 90 & \rhd & \_+2
\end{array}
$$

Example Evaluation

$$
\begin{array}{llcl}
 & ((4+5)*10)+2 & \rhd & \cdot \\
\\
\longrightarrow & (4+5)*10 & \rhd & \_ + 2 \\
\longrightarrow & 4+5 & \rhd & \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 4 & \rhd & \_ + 5 \;,\; \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 5 & \rhd & 4 + \_ \;,\; \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 9 & \rhd & \_ * 10 \;,\; \_ + 2 \\
\longrightarrow & 10 & \rhd & 9 * \_ \;,\; \_ + 2 \\
\longrightarrow & 90 & \rhd & \_ + 2 \\
\longrightarrow & 2 & \rhd & 90 + \_
\end{array}
$$

Example Evaluation

$$
\begin{array}{llll}
 & ((4+5)*10)+2 & \rhd & \cdot \\
 & & & \\
\longrightarrow & (4+5)*10 & \rhd & \_+2 \\
\longrightarrow & 4+5 & \rhd & \_*10 \;,\; \_+2 \\
\longrightarrow & 4 & \rhd & \_+5 \;,\; \_*10 \;,\; \_+2 \\
\longrightarrow & 5 & \rhd & 4+\_ \;,\; \_*10 \;,\; \_+2 \\
\longrightarrow & 9 & \rhd & \_*10 \;,\; \_+2 \\
\longrightarrow & 10 & \rhd & 9*\_ \;,\; \_+2 \\
\longrightarrow & 90 & \rhd & \_+2 \\
\longrightarrow & 2 & \rhd & 90+\_ \\
\longrightarrow & 92 & \rhd & \cdot
\end{array}
$$

Evaluation Rules: End of and Evaluation

If we reach a constant and the empty continuation then we stop

$$c \rhd \cdot \quad \longrightarrow \quad \mathsf{value}(c)$$

Evaluation Rules: Boolean Expressions

$$e_1 \texttt{ \&\& } e_2 \rhd K \qquad \longrightarrow \qquad e_1 \rhd (\texttt{\_ \&\& } e_2 \ , K)$$

$$\mathsf{false} \rhd (\texttt{\_ \&\& } e_2 \ , K) \qquad \longrightarrow \qquad \mathsf{false} \rhd K$$

$$\mathsf{true} \rhd (\texttt{\_ \&\& } e_2 \ , K) \qquad \longrightarrow \qquad e_2 \rhd K$$

true and *false* are also values

(We could also use 1 and 0 but distinguishing helps detect errors.)

Evaluation Rules: Boolean Expressions

$$e_1 \texttt{ \&\& } e_2 \triangleright K \longrightarrow e_1 \triangleright (\_ \texttt{ \&\& } e_2 \ , \ K)$$

$$\mathsf{false} \triangleright (\_ \texttt{ \&\& } e_2 \ , \ K) \longrightarrow \mathsf{false} \triangleright K$$

Notice the short-cutting.

$$\mathsf{true} \triangleright (\_ \texttt{ \&\& } e_2 \ , \ K) \longrightarrow e_2 \triangleright K$$

true and *false* are also values

(We could also use 1 and 0 but distinguishing helps detect errors.)

Variables and Environments

How do we evaluate variable?

Variables and Environments

How do we evaluate variable?

$$x \rhd K \quad \longrightarrow \quad ?$$

Variables and Environments

How do we evaluate variable?

$$x \triangleright K \quad \longrightarrow \quad ?$$

Need to have a environment that maps variables to values

Variables and Environments

How do we evaluate variable?

$$x \triangleright K \quad \longrightarrow \quad ?$$

Need to have a environment that maps variables to values

$$\eta ::= \cdot \mid \eta, x \mapsto v$$

Variables and Environments

How do we evaluate variable?

$$x \triangleright K \quad \longrightarrow \quad ?$$

Need to have a environment that maps variables to values

$$\eta ::= \cdot \mid \eta, x \mapsto v$$

The machine state consists now of an expression, a continuation, and an environment

Variables and Environments

How do we evaluate variable?

$$x \triangleright K \quad \longrightarrow \quad ?$$

Need to have a environment that maps variables to values

$$\eta ::= \cdot \mid \eta, x \mapsto v$$

The machine state consists now of an expression, a continuation, and an environment

$$\eta \vdash e \triangleright K$$

Variables and Environments II

The rules we have seen so far just carry over

$$\eta \vdash e_1 \oplus e_2 \rhd K \quad \longrightarrow \quad \eta \vdash e_1 \rhd (\_ \oplus e_2 \ , K)$$

$$\eta \vdash c_1 \rhd (\_ \oplus e_2 \ , K) \quad \longrightarrow \quad \eta \vdash e_2 \rhd (c_1 \oplus \_ \ , K)$$

$$\eta \vdash c_2 \rhd (c_1 \oplus \_ \ , K) \quad \longrightarrow \quad \eta \vdash c \rhd K \qquad (c = c_1 \oplus c_2 \bmod 2^{32})$$

Variables and Environments II

The rules we have seen so far just carry over

$$\eta \vdash e_1 \oplus e_2 \rhd K \quad \longrightarrow \quad \eta \vdash e_1 \rhd (\_ \oplus e_2\ , K)$$

$$\eta \vdash c_1 \rhd (\_ \oplus e_2\ , K) \quad \longrightarrow \quad \eta \vdash e_2 \rhd (c_1 \oplus \_\ , K)$$

$$\eta \vdash c_2 \rhd (c_1 \oplus \_\ , K) \quad \longrightarrow \quad \eta \vdash c \rhd K \qquad (c = c_1 \oplus c_2 \bmod 2^{32})$$

Variables are simply looked up

$$\eta \vdash x \rhd K \quad \longrightarrow \quad \eta \vdash \eta(x) \rhd K$$

Variables and Environments II

The rules we have seen so far just carry over

$$\eta \vdash e_1 \oplus e_2 \rhd K \quad \longrightarrow \quad \eta \vdash e_1 \rhd (\_ \oplus e_2 \ , K)$$

$$\eta \vdash c_1 \rhd (\_ \oplus e_2 \ , K) \quad \longrightarrow \quad \eta \vdash e_2 \rhd (c_1 \oplus \_ \ , K)$$

$$\eta \vdash c_2 \rhd (c_1 \oplus \_ \ , K) \quad \longrightarrow \quad \eta \vdash c \rhd K \qquad (c = c_1 \oplus c_2 \bmod 2^{32})$$

Variables are simply looked up

$$\eta \vdash x \rhd K \quad \longrightarrow \quad \eta \vdash \eta(x) \rhd K$$

The environment never changes when evaluating expressions

Executing Statements

Executing Statements I

Executions of statements don’t pass values to the continuation

Statements have usually an effect on the environment

Machine configurations:

$$\eta \vdash s \blacktriangleright K$$

Executing Statements I

Executions of statements don't pass values to the continuation

Statements have usually an effect on the environment

Machine configurations:

$$\eta \vdash s \blacktriangleright K$$

Continuations contain statements.

Executing Statements I

Executions of statements don't pass values to the continuation

Statements have usually an effect on the environment

Machine configurations:

$$\eta \vdash s \blacktriangleright K$$

Continuations contain statements.

Sequences:

$$\eta \vdash \mathsf{seq}(s_1, s_2) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash s_1 \blacktriangleright (s_2\ ,\ K)$$

Executing Statements I

Executions of statements don't pass values to the continuation

Statements have usually an effect on the environment

Machine configurations:

$$\eta \vdash s \blacktriangleright K$$

Continuations contain statements.

Sequences:

$$\eta \vdash \mathsf{seq}(s_1, s_2) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash s_1 \blacktriangleright (s_2 \;,\; K)$$

No ops:

$$\eta \vdash \mathsf{nop} \blacktriangleright (s \;,\; K) \quad \longrightarrow \quad \eta \vdash s \blacktriangleright K$$

Executing Statements I

Executions of statements don't pass values to the continuation

Statements have usually an effect on the environment

Machine configurations:

$$\eta \vdash s \blacktriangleright K$$

Continuations contain statements.

Sequences:

$$\eta \vdash \mathsf{seq}(s_1, s_2) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash s_1 \blacktriangleright (s_2\ ,\ K)$$

No ops:

$$\eta \vdash \mathsf{nop} \blacktriangleright (s\ ,\ K) \quad \longrightarrow \quad \eta \vdash s \blacktriangleright K$$

A terminating execution ends with a nop.

Executing Statements II

Interaction with expressions is straightforward

Assignments:

$$\eta \vdash \mathsf{assign}(x, e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \rhd (\mathsf{assign}(x, \_)\ ,\ K)$$

$$\eta \vdash v \rhd (\mathsf{assign}(x, \_)\ ,\ K) \quad \longrightarrow$$

Executing Statements II

Interaction with expressions is straightforward

Assignments:

$$\eta \vdash \mathsf{assign}(x, e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \rhd (\mathsf{assign}(x, \_)\ ,\ K)$$

$$\eta \vdash v \rhd (\mathsf{assign}(x, \_)\ ,\ K) \quad \longrightarrow \quad \eta[x \mapsto v] \vdash \mathsf{nop} \blacktriangleright K$$

Executing Statements II

Interaction with expressions is straightforward

Assignments:

$$\eta \vdash \mathsf{assign}(x, e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \rhd (\mathsf{assign}(x, \_)\ ,\ K)$$

$$\eta \vdash v \rhd (\mathsf{assign}(x, \_)\ ,\ K) \quad \longrightarrow \quad \eta[x \mapsto v] \vdash \mathsf{nop} \blacktriangleright K$$

Update the environment with new mapping.

Executing Statements III

Conditionals:

$$\eta \vdash \mathsf{if}(e, s_1, s_2) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \rhd (\mathsf{if}(\_, s_1, s_2)\ ,\ K)$$

$$\eta \vdash \mathsf{true} \rhd (\mathsf{if}(\_, s_1, s_2), K) \quad \longrightarrow \quad \eta \vdash s_1 \blacktriangleright K$$

$$\eta \vdash \mathsf{false} \rhd (\mathsf{if}(\_, s_1, s_2), K) \quad \longrightarrow \quad \eta \vdash s_2 \blacktriangleright K$$

Executing Statements IV

Loops:

$$\eta \vdash \mathsf{while}(e, s) \blacktriangleright K \qquad \longrightarrow \quad ?$$

Executing Statements IV

Loops:

$$\eta \vdash \mathsf{while}(e, s) \blacktriangleright K \qquad \longrightarrow \quad ?$$

Not that the following statements are equivalent:

$$\mathsf{while}(e, s) \equiv \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop})$$

Executing Statements IV

Loops:

$$\eta \vdash \mathsf{while}(e, s) \blacktriangleright K \qquad \longrightarrow \quad ?$$

Not that the following statements are equivalent:

$$\mathsf{while}(e, s) \equiv \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop})$$

$$\eta \vdash \mathsf{while}(e, s) \blacktriangleright K \qquad \longrightarrow \qquad \eta \vdash \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop}) \blacktriangleright K$$

Executing Statements IV

Loops:

$$\eta \vdash \mathsf{while}(e, s) \blacktriangleright K \qquad \longrightarrow \quad ?$$

Not that the following statements are equivalent:

$$\mathsf{while}(e, s) \equiv \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop})$$

$$\eta \vdash \mathsf{while}(e, s) \blacktriangleright K \qquad \longrightarrow \qquad \eta \vdash \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop}) \blacktriangleright K$$

Non-termination:

$$s_0 \quad \longrightarrow \quad s_1 \quad \longrightarrow \quad s_2 \quad \longrightarrow \quad \cdots$$

We can make an infinite number of steps without reaching a final state

Executing Statements V

Assertions:

$$\eta \vdash \mathsf{assert}(e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \triangleright (\mathsf{assert}(\_), K)$$

$$\eta \vdash \mathsf{true} \triangleright (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \eta \vdash \mathsf{nop} \blacktriangleright K$$

$$\eta \vdash \mathsf{false} \triangleright (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \mathsf{exception}(\mathsf{abort})$$

Executing Statements V

Assertions:

$$\eta \vdash \mathsf{assert}(e) \blacktriangleright K \qquad \longrightarrow \qquad \eta \vdash e \rhd (\mathsf{assert}(\_), K)$$

$$\eta \vdash \mathsf{true} \rhd (\mathsf{assert}(\_), K) \qquad \longrightarrow \qquad \eta \vdash \mathsf{nop} \blacktriangleright K$$

$$\eta \vdash \mathsf{false} \rhd (\mathsf{assert}(\_), K) \qquad \longrightarrow \qquad \mathsf{exception}(\mathsf{abort})$$

Declarations:

$$\eta \vdash \mathsf{decl}(x, \tau, s) \blacktriangleright K \qquad \longrightarrow \qquad \eta[x \mapsto \mathsf{nothing}] \vdash s \blacktriangleright K$$

Executing Statements V

Assertions:

$$\eta \vdash \mathsf{assert}(e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \triangleright (\mathsf{assert}(\_), K)$$

$$\eta \vdash \mathsf{true} \triangleright (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \eta \vdash \mathsf{nop} \blacktriangleright K$$

$$\eta \vdash \mathsf{false} \triangleright (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \mathsf{exception}(\mathsf{abort})$$

Declarations:

$$\eta \vdash \mathsf{decl}(x, \tau, s) \blacktriangleright K \quad \longrightarrow \quad \eta[x \mapsto \mathsf{nothing}] \vdash s \blacktriangleright K$$

If C0 had shadowing then we would have to save and restore the previous value of x.

Executing Statements V

Assertions:

$$\eta \vdash \mathsf{assert}(e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \triangleright (\mathsf{assert}(\_), K)$$

$$\eta \vdash \mathsf{true} \triangleright (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \eta \vdash \mathsf{nop} \blacktriangleright K$$

$$\eta \vdash \mathsf{false} \triangleright (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \mathsf{exception}(\mathsf{abort})$$

Declarations:

$$\eta \vdash \mathsf{decl}(x, \tau, s) \blacktriangleright K \quad \longrightarrow \quad \eta[x \mapsto \mathsf{nothing}] \vdash s \blacktriangleright K$$

If C0 had shadowing then we would have to save and restore the previous value of x.

Final states:

$$\mathsf{exception}(E), \qquad \mathsf{nop} \blacktriangleright \cdot$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$
$\longrightarrow$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{ll}
 & [x \mapsto 1] \vdash \mathsf{while}(x > 0, s) \quad \blacktriangleright \quad \cdot \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x{\mapsto}1]$

$$
\begin{array}{ll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) \quad \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) \quad \blacktriangleright \quad \cdot \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow & \\
\longrightarrow &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x\ >\ 0, \mathsf{assign}(x, x+1)) \qquad \eta\ =\ [x{\mapsto}1]$

$$
\begin{array}{rlll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x{\mapsto}1]$

$$
\begin{array}{lll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{lll}
 & [x \mapsto 1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x \mapsto 1] \vdash \mathsf{if}(x>0, \mathsf{seq}(s, \mathsf{while}(x>0, s)), \mathsf{nop}) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x \mapsto 1] \vdash x > 0 & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x \mapsto 1] \vdash x & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x \mapsto 1] \vdash 1 & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{lll}
 & [x\mapsto 1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{if}(x>0, \mathsf{seq}(s, \mathsf{while}(x>0, s)), \mathsf{nop}) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x\mapsto 1] \vdash x > 0 & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 0 & \triangleright \quad 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{lllll}
 & [x\mapsto 1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{llll}
 & [x\mapsto 1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright & \cdot \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x{\mapsto}1]$

$$
\begin{array}{rlcl}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright & \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x\ >\ 0, \mathsf{assign}(x, x+1)) \qquad \eta\ =\ [x{\mapsto}1]$

$$
\begin{array}{rlcl}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright & \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & [x{\mapsto}1] \vdash x+1 & \triangleright & \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x\ >\ 0, \mathsf{assign}(x, x + 1)) \qquad \eta\ =\ [x{\mapsto}1]$

$$
\begin{array}{llll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x + 1)) & \blacktriangleright & \mathsf{while}(x > 0, \mathsf{assign}(x, x + 1)) \\
\longrightarrow & [x{\mapsto}1] \vdash x + 1 & \triangleright & \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright & \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & & \\
\longrightarrow & & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{lll}
 & [x\mapsto 1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x\mapsto 1] \vdash x > 0 & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 0 & \triangleright \quad 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{true} & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright \quad \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & [x\mapsto 1] \vdash x + 1 & \triangleright \quad \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright \quad \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright \quad \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x \;>\; 0, \mathsf{assign}(x, x+1)) \qquad \eta \;=\; [x{\mapsto}1]$

$$
\begin{array}{lll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \;\; \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright \;\; \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright \;\; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \;\; \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \;\; \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright \;\; 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright \;\; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright \;\; \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x + 1)) & \blacktriangleright \;\; \mathsf{while}(x > 0, \mathsf{assign}(x, x + 1)) \\
\longrightarrow & [x{\mapsto}1] \vdash x + 1 & \triangleright \;\; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \;\; \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \;\; \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \;\; 1 + \_; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & & \\
\longrightarrow & & \\
\longrightarrow & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x \; > \; 0, \mathsf{assign}(x, x+1)) \qquad \eta \; = \; [x{\mapsto}1]$

$$
\begin{array}{lll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright \quad 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright \quad \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & [x{\mapsto}1] \vdash x + 1 & \triangleright \quad \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \quad \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \quad \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \quad 1 + \_; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 2 & \triangleright \quad \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & & \\
\longrightarrow & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x{\mapsto}1]$

$$
\begin{array}{lll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \quad \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright \quad 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright \quad \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright \quad \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright \quad \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & [x{\mapsto}1] \vdash x + 1 & \triangleright \quad \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright \quad \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \quad \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright \quad 1 + \_; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 2 & \triangleright \quad \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}2] \vdash \mathsf{nop} & \blacktriangleright \quad \mathsf{while}(x > 0, s) \\
\longrightarrow & &
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x \mapsto 1]$

$$
\begin{array}{llcl}
 & [x\mapsto 1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright & \cdot \\
\longrightarrow & [x\mapsto 1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright & \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & [x\mapsto 1] \vdash x + 1 & \triangleright & \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 1] \vdash x & \triangleright & \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright & \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 1] \vdash 1 & \triangleright & 1 + \_; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 1] \vdash 2 & \triangleright & \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 2] \vdash \mathsf{nop} & \blacktriangleright & \mathsf{while}(x > 0, s) \\
\longrightarrow & [x\mapsto 2] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot
\end{array}
$$

Example: Infinite Loop

$\mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \qquad \eta = [x{\mapsto}1]$

$$
\begin{array}{llll}
 & [x{\mapsto}1] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{if}(x{>}0, \mathsf{seq}(s, \mathsf{while}(x{>}0, s)), \mathsf{nop}) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash x > 0 & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright & \_ > 0; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash 0 & \triangleright & 1 > \_; \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{true} & \triangleright & \mathsf{if}(\_, \mathsf{seq}(s, \mathsf{while}(x > 0, s)), \mathsf{nop}) \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{seq}(s, \mathsf{while}(x > 0, s)) & \blacktriangleright & \cdot \\
\longrightarrow & [x{\mapsto}1] \vdash \mathsf{assign}(x, x+1)) & \blacktriangleright & \mathsf{while}(x > 0, \mathsf{assign}(x, x+1)) \\
\longrightarrow & [x{\mapsto}1] \vdash x + 1 & \triangleright & \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash x & \triangleright & \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright & \_ + 1; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 1 & \triangleright & 1 + \_; \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}1] \vdash 2 & \triangleright & \mathsf{assign}(x, \_)); \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}2] \vdash \mathsf{nop} & \blacktriangleright & \mathsf{while}(x > 0, s) \\
\longrightarrow & [x{\mapsto}2] \vdash \mathsf{while}(x > 0, s) & \blacktriangleright & \cdot \\
\ldots & & &
\end{array}
$$

Functions

Function Calls

What needs to happen at a function call?

Function Calls

What needs to happen at a function call?

- Evaluate the arguments in left-to-right order

Function Calls

What needs to happen at a function call?

- Evaluate the arguments in left-to-right order
- Save the environment of the caller to continue the execution after the function call

Function Calls

What needs to happen at a function call?

- Evaluate the arguments in left-to-right order
- Save the environment of the caller to continue the execution after the function call
- Save the continuation of the caller

Function Calls

What needs to happen at a function call?

- Evaluate the arguments in left-to-right order
- Save the environment of the caller to continue the execution after the function call
- Save the continuation of the caller
- Execute the body of the callee in a new environment that maps the formal parameters to the argument values

Function Calls

What needs to happen at a function call?

- Evaluate the arguments in left-to-right order
- Save the environment of the caller to continue the execution after the function call
- Save the continuation of the caller
- Execute the body of the callee in a new environment that maps the formal parameters to the argument values
- Pass the return value the the environment of the caller

Call Stack

We need to keep track of continuations and environment in stack frames

Call stack:

$$S \quad ::= \quad \cdot \mid S \ , \langle \eta, K \rangle$$

Call Stack

We need to keep track of continuations and environment in stack frames

Call stack:

$$S \quad ::= \quad \cdot \mid S \ , \ \langle \eta, K \rangle$$

Environment

Call Stack

We need to keep track of continuations and environment in stack frames

Call stack:

Continuation

$$S \quad ::= \quad \cdot \mid S\ ,\ \langle \eta, K \rangle$$

Environment

Call Stack

We need to keep track of continuations and environment in stack frames

Call stack:

$$S \quad ::= \quad \cdot \mid S \,,\, \langle \underbrace{\eta}_{\text{Environment}}, \overbrace{K}^{\text{Continuation}} \rangle$$

Configurations:

Evaluation $\quad S \,;\, \eta \vdash e \rhd K$

Execution $\quad S \,;\, \eta \vdash s \blacktriangleright K$

Call Stack

We need to keep track of continuations and environment in stack frames

Call stack:

Continuation

$$S \quad ::= \quad \cdot \mid S \;,\; \langle \eta, K \rangle$$

Environment

Configurations:

Evaluation $\quad S \;;\; \eta \vdash e \rhd K$

Execution $\quad S \;;\; \eta \vdash s \blacktriangleright K$

Existing rules can be lifted to the new configurations by passing through the call stack

Rules for Function Calls

We only show the special case of 0 and 2 arguments

Rules for Function Calls

n args is similar.

We only show the special case of 0 and 2 arguments

Rules for Function Calls

n args is similar.

We only show the special case of 0 and 2 arguments

No arguments:

$$S \; ; \eta \vdash f(\,) \rhd K \qquad \longrightarrow \qquad (S\;,\langle \eta, K\rangle)\;; \cdot \vdash s \blacktriangleright \cdot$$

(given that f *is defined as* $f(\,)\{s\}$*)*

Rules for Function Calls

n args is similar.

We only show the special case of 0 and 2 arguments

No arguments:

$$S\ ;\ \eta \vdash f(\,) \rhd K \qquad \longrightarrow \qquad (S\ ,\ \langle \eta, K \rangle)\ ;\ \cdot \vdash s \blacktriangleright \cdot$$

(given that f *is defined as* $f(\,)\{s\}$*)*

Store callee's stack frame

Rules for Function Calls

n args is similar.

We only show the special case of 0 and 2 arguments

Evaluate s in empty environment.

No arguments:

$$S\ ;\eta \vdash f(\,) \rhd K \qquad \longrightarrow \qquad (S\ ,\ \langle \eta, K\rangle)\ ;\cdot \vdash s \blacktriangleright \cdot$$

(given that f *is defined as* $f(\,)\{s\}$*)*

Store callee's stack frame

Rules for Function Calls

n args is similar.

We only show the special case of 0 and 2 arguments

Evaluate s in empty environment.

No arguments:

$$S \; ; \eta \vdash f(\,) \rhd K \qquad \longrightarrow \qquad (S \; , \langle \eta, K \rangle) \; ; \cdot \vdash s \blacktriangleright \cdot$$

(given that f *is defined as* $f(\,)\{s\}$*)*

Store callee's stack frame

Two arguments:

$$S \; ; \eta \vdash f(e_1, e_2) \rhd K \qquad \longrightarrow \qquad S \; ; \eta \vdash e_1 \rhd (f(\_, e_2) \; , K)$$

$$S \; ; \eta \vdash c_1 \rhd (f(\_, e_2) \; , K) \qquad \longrightarrow \qquad S \; ; \eta \vdash e_2 \rhd (f(c_1, \_) \; , K)$$

Rules for Function Calls

n args is similar.

We only show the special case of 0 and 2 arguments

Evaluate s in empty environment.

No arguments:

$$S\,;\eta \vdash f(\,) \rhd K \qquad \longrightarrow \qquad (S\ ,\ \langle \eta, K\rangle)\ ;\cdot \vdash s \blacktriangleright \cdot$$

(given that f is defined as $f(\,)\{s\}$*)*

Store callee's stack frame

Two arguments:

$$S\,;\eta \vdash f(e_1, e_2) \rhd K \qquad \longrightarrow \qquad S\,;\eta \vdash e_1 \rhd (f(\_, e_2)\ ,\ K)$$

$$S\,;\eta \vdash c_1 \rhd (f(\_, e_2)\ ,\ K) \qquad \longrightarrow \qquad S\,;\eta \vdash e_2 \rhd (f(c_1, \_)\ ,\ K)$$

$$S\,;\eta \vdash c_2 \rhd (f(c_1, \_)\ ,\ K) \qquad \longrightarrow \qquad (S\ ,\ \langle \eta, K\rangle)\ ;[x_1 \mapsto c_1, x_2 \mapsto c_2] \vdash s \blacktriangleright \cdot$$

(given that f is defined as $f(x_1, x_2)\{s\}$*)*

Rules for Returns

Need to restore continuation and environment and pass return value

$$S \; ; \eta \vdash \mathsf{return}(e) \blacktriangleright K$$

$$S \; , \langle \eta', K' \rangle \; ; \eta \vdash v \triangleright (\mathsf{return}(\_) \; , K)$$

Rules for Returns

Need to restore continuation and environment and pass return value

$$S\ ;\ \eta \vdash \mathsf{return}(e) \blacktriangleright K \quad \longrightarrow \quad S\ ;\ \eta \vdash e \rhd (\mathsf{return}(\_)\ ,\ K)$$

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash v \rhd (\mathsf{return}(\_)\ ,\ K)$$

Rules for Returns

Need to restore continuation and environment and pass return value

$$S\ ;\ \eta \vdash \mathsf{return}(e) \blacktriangleright K \qquad \longrightarrow \qquad S\ ;\ \eta \vdash e \rhd (\mathsf{return}(\_)\ ,\ K)$$

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash v \rhd (\mathsf{return}(\_)\ ,\ K) \qquad \longrightarrow \qquad S\ ;\ \eta' \vdash v \rhd K'$$

Rules for Returns

Need to restore continuation and environment and pass return value

$$S\ ;\ \eta \vdash \mathsf{return}(e) \blacktriangleright K \quad\longrightarrow\quad S\ ;\ \eta \vdash e \rhd (\mathsf{return}(\_)\ ,\ K)$$

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash v \rhd (\mathsf{return}(\_)\ ,\ K) \quad\longrightarrow\quad S\ ;\ \eta' \vdash v \rhd K'$$

Special case: returning void

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright \cdot \quad\longrightarrow\quad S\ ;\ \eta' \vdash \mathsf{nothing} \rhd K'$$

Rules for Returns

Need to restore continuation and environment and pass return value

$$S\ ;\ \eta \vdash \mathsf{return}(e) \blacktriangleright K \quad\longrightarrow\quad S\ ;\ \eta \vdash e \rhd (\mathsf{return}(\_)\ ,\ K)$$

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash v \rhd (\mathsf{return}(\_)\ ,\ K) \quad\longrightarrow\quad S\ ;\ \eta' \vdash v \rhd K'$$

Special case: returning void

Will only be reached by functions without return.

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright \cdot \quad\longrightarrow\quad S\ ;\ \eta' \vdash \mathsf{nothing} \rhd K'$$

Rules for Returns

Need to restore continuation and environment and pass return value

$$S \; ; \eta \vdash \mathsf{return}(e) \blacktriangleright K \quad \longrightarrow \quad S \; ; \eta \vdash e \rhd (\mathsf{return}(\_) \; , K)$$

$$S \; , \langle \eta', K' \rangle \; ; \eta \vdash v \rhd (\mathsf{return}(\_) \; , K) \quad \longrightarrow \quad S \; ; \eta' \vdash v \rhd K'$$

Special case: returning void

Will only be reached by functions without return.

$$S \; , \langle \eta', K' \rangle \; ; \eta \vdash \mathsf{nop} \blacktriangleright \cdot \quad \longrightarrow \quad S \; ; \eta' \vdash \mathsf{nothing} \rhd K'$$

Dummy value

Rules for Returns

Need to restore continuation and environment and pass return value

$$S\ ;\ \eta \vdash \mathsf{return}(e) \blacktriangleright K \quad\longrightarrow\quad S\ ;\ \eta \vdash e \triangleright (\mathsf{return}(\_)\ ,\ K)$$

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash v \triangleright (\mathsf{return}(\_)\ ,\ K) \quad\longrightarrow\quad S\ ;\ \eta' \vdash v \triangleright K'$$

Special case: returning void

Will only be reached by functions without return.

$$S\ ,\ \langle \eta', K' \rangle\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright \cdot \quad\longrightarrow\quad S\ ;\ \eta' \vdash \mathsf{nothing} \triangleright K'$$

Dummy value

Alternative: elaborate each function that returns void with return(nothing) statements.

Execution of the Main Function

How can we execute a program?

Execution of the Main Function

How can we execute a program?

$$\cdot\, ; \cdot \vdash \mathsf{main}(\,) \rhd \cdot \qquad \text{(initial state)}$$

Execution of the Main Function

How can we execute a program?

$$\cdot\ ;\ \cdot \vdash \mathsf{main}(\,) \rhd \cdot \qquad \text{(initial state)}$$

$$\cdot\ ;\ \eta \vdash c \rhd \cdot \quad \longrightarrow \quad \mathsf{value}(c) \qquad \textbf{(final state)}$$

Statics, Dynamics, and Safety

Overview of Machine States (Configurations)

$S\ ;\eta \vdash e \rhd K$ – Evaluating the expression e with the continuation K

$S\ ;\eta \vdash s \blacktriangleright K$ – Evaluating the statement s with the continuation K

$\mathsf{value}(c)$ – Final state, return a value

$\mathsf{exception}(E)$ – Final state, report an error

Overview of Machine States (Configurations)

$S\;;\eta \vdash e \triangleright K$ – Evaluating the expression e with the continuation K

$S\;;\eta \vdash s \blacktriangleright K$ – Evaluating the statement s with the continuation K

$\mathsf{value}(c)$ – Final state, return a value

$\mathsf{exception}(E)$ – Final state, report an error

What do we expect from the transitions?

Overview of Machine States (Configurations)

$S\ ;\eta \vdash e \rhd K$ – Evaluating the expression e with the continuation K

$S\ ;\eta \vdash s \blacktriangleright K$ – Evaluating the statement s with the continuation K

$\mathsf{value}(c)$ – Final state, return a value

$\mathsf{exception}(E)$ – Final state, report an error

What do we expect from the transitions?

There shouldn't be more steps after reaching a final state

Overview of Machine States (Configurations)

$S\,;\eta \vdash e \rhd K$ – Evaluating the expression e with the continuation K

$S\,;\eta \vdash s \blacktriangleright K$ – Evaluating the statement s with the continuation K

$\mathsf{value}(c)$ – Final state, return a value

$\mathsf{exception}(E)$ – Final state, report an error

What do we expect from the transitions?

There shouldn’t be more steps after reaching a final state

The language should be deterministic: there at most one transition per state

Progress

There are many non-final states that don’t have transitions, e.g.

Progress

There are many non-final states that don't have transitions, e.g.

$$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K)$$

Progress

There are many non-final states that don't have transitions, e.g.

$$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K) \qquad\qquad \cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$$

Progress

There are many non-final states that don't have transitions, e.g.

$$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K) \qquad\qquad \cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$$

Stuck states.

Progress

There are many non-final states that don't have transitions, e.g.

$$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K) \qquad \cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$$

Stuck states.

The behavior of these states is undefined.

Progress

There are many non-final states that don't have transitions, e.g.

$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K)$

$\cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$

Stuck states.

The behavior of these states is undefined.

Central relationship between static and dynamic semantics:

Programs that are well-defined according to the static semantics should be free of undefined behavior.

Progress

There are many non-final states that don't have transitions, e.g.

$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K)$ $\qquad$ $\cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$

Stuck states.

The behavior of these states is undefined.

Central relationship between static and dynamic semantics:

Programs that are well-defined according to the static semantics should be free of undefined behavior.

Theorem 1 (No undefined behavior) *If a program passes all the static semantics, and*

$$\cdot; \cdot \vdash \mathsf{main}() \longrightarrow \mathcal{ST}_1 \longrightarrow \ldots \longrightarrow \mathcal{ST}_n$$

then either $\mathcal{ST}_n$ is a final state or else $\mathcal{ST}_n$ is not-stuck because there exists a state $\mathcal{ST}'$ such that $\mathcal{ST}_n \longrightarrow \mathcal{ST}'$.

Progress

Well-typed programs don't go wrong!

There are many non-final states that don't have transitions, e.g.

$$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K) \qquad \cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$$

Stuck states.

The behavior of these states is undefined.

Central relationship between static and dynamic semantics:

Programs that are well-defined according to the static semantics should be free of undefined behavior.

Theorem 1 (No undefined behavior) *If a program passes all the static semantics, and*

$$\cdot; \cdot \vdash \mathsf{main}() \quad \longrightarrow \quad \mathcal{ST}_1 \quad \longrightarrow \quad \ldots \quad \longrightarrow \quad \mathcal{ST}_n$$

then either $\mathcal{ST}_n$ is a final state or else $\mathcal{ST}_n$ is not-stuck because there exists a state $\mathcal{ST}'$ such that $\mathcal{ST}_n \quad \longrightarrow \quad \mathcal{ST}'$.

Progress

Well-typed programs don't go wrong!

There are many non-final states that don't have transitions, e.g.

$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K)$ $\quad\quad$ $\cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$

Stuck states.

The behavior of these states is undefined.

Central relationship between static and dynamic semantics:

Programs that are well-defined according to the static semantics should be free of undefined behavior.

Theorem 1 (No undefined behavior) *If a program passes all the static semantics, and*

$$\cdot; \cdot \vdash \mathsf{main}() \quad\longrightarrow\quad \mathcal{ST}_1 \quad\longrightarrow\quad \ldots \quad\longrightarrow\quad \mathcal{ST}_n$$

then either $\mathcal{ST}_n$ is a final state or else $\mathcal{ST}_n$ is not-stuck because there exists a state $\mathcal{ST}'$ such that $\mathcal{ST}_n \quad\longrightarrow\quad \mathcal{ST}'$.

How to prove this?

Progress

Well-typed programs don't go wrong!

There are many non-final states that don't have transitions, e.g.

$S; \eta \vdash 42 \rhd (\mathsf{if}(\_, s_1, s_2); K)$ $\quad\quad$ $\cdot; \cdot \vdash \mathsf{nop} \blacktriangleright \cdot$

Stuck states.

The behavior of these states is undefined.

Central relationship between static and dynamic semantics:

Programs that are well-defined according to the static semantics should be free of undefined behavior.

Theorem 1 (No undefined behavior) *If a program passes all the static semantics, and*

$$\cdot; \cdot \vdash \mathsf{main}() \longrightarrow \mathcal{ST}_1 \longrightarrow \ldots \longrightarrow \mathcal{ST}_n$$

then either $\mathcal{ST}_n$ is a final state or else $\mathcal{ST}_n$ is not-stuck because there exists a state $\mathcal{ST}'$ such that $\mathcal{ST}_n \longrightarrow \mathcal{ST}'$.

How to prove this?

15-312 next term.

Expressions	e	$::=$	$c \mid e_1 \odot e_2 \mid \mathsf{true} \mid \mathsf{false} \mid e_1 \;\&\&\; e_2 \mid x \mid f(e_1, e_2) \mid f(\,)$
Statements	s	$::=$	$\mathsf{nop} \mid \mathsf{seq}(s_1, s_2) \mid \mathsf{assign}(x, e) \mid \mathsf{decl}(x, \tau, s)$
		$\mid$	$\mathsf{if}(e, s_1, s_2) \mid \mathsf{while}(e, s) \mid \mathsf{return}(e) \mid \mathsf{assert}(e)$
Values	v	$::=$	$c \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{nothing}$
Environments	η	$::=$	$\cdot \mid \eta, x \mapsto c$
Stacks	S	$::=$	$\cdot \mid S \;,\; \langle \eta, K \rangle$
Cont. frames	ϕ	$::=$	$\_ \odot e \mid c \odot \_ \mid \_ \;\&\&\; e \mid f(\_, e) \mid f(c, \_)$
		$\mid$	$s \mid \mathsf{assign}(x, \_) \mid \mathsf{if}(\_, s_1, s_2) \mid \mathsf{return}(\_) \mid \mathsf{assert}(\_)$
Continuations	K	$::=$	$\cdot \mid \phi \;,\; K$
Exceptions	E	$::=$	$\mathsf{arith} \mid \mathsf{abort} \mid \mathsf{mem}$

Summary I

All ops.

Expressions	e	$::=$	$c \mid e_1 \odot e_2 \mid \mathsf{true} \mid \mathsf{false} \mid e_1 \;\&\&\; e_2 \mid x \mid f(e_1, e_2) \mid f(\,)$
Statements	s	$::=$	$\mathsf{nop} \mid \mathsf{seq}(s_1, s_2) \mid \mathsf{assign}(x, e) \mid \mathsf{decl}(x, \tau, s)$
		$\mid$	$\mathsf{if}(e, s_1, s_2) \mid \mathsf{while}(e, s) \mid \mathsf{return}(e) \mid \mathsf{assert}(e)$
Values	v	$::=$	$c \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{nothing}$
Environments	η	$::=$	$\cdot \mid \eta, x \mapsto c$
Stacks	S	$::=$	$\cdot \mid S\ ,\ \langle \eta, K \rangle$
Cont. frames	ϕ	$::=$	$\_ \odot e \mid c \odot \_ \mid \_ \;\&\&\; e \mid f(\_, e) \mid f(c, \_)$
		$\mid$	$s \mid \mathsf{assign}(x, \_) \mid \mathsf{if}(\_, s_1, s_2) \mid \mathsf{return}(\_) \mid \mathsf{assert}(\_)$
Continuations	K	$::=$	$\cdot \mid \phi\ ,\ K$
Exceptions	E	$::=$	$\mathsf{arith} \mid \mathsf{abort} \mid \mathsf{mem}$

Summary I

$$
\begin{array}{lcll}
S\ ;\eta \vdash e_1 \odot e_2 \rhd K & \longrightarrow & S\ ;\eta \vdash e_1 \rhd (\_ \odot e_2\ , K) & \\
S\ ;\eta \vdash c_1 \rhd (\_ \odot e_2\ , K) & \longrightarrow & S\ ;\eta \vdash e_2 \rhd (c_1 \odot \_\ , K) & \\
S\ ;\eta \vdash c_2 \rhd (c_1 \odot \_\ , K) & \longrightarrow & S\ ;\eta \vdash c \rhd K & (c = c_1 \odot c_2) \\
S\ ;\eta \vdash c_2 \rhd (c_1 \odot \_\ , K) & \longrightarrow & \mathsf{exception}(\mathsf{arith}) & (c_1 \odot c_2 \text{ undefined}) \\
\\
S\ ;\eta \vdash e_1\ \texttt{\&\&}\ e_2 \rhd K & \longrightarrow & S\ ;\eta \vdash e_1 \rhd (\_\ \texttt{\&\&}\ e_2\ , K) & \\
S\ ;\eta \vdash \mathsf{false} \rhd (\_\ \texttt{\&\&}\ e_2\ , K) & \longrightarrow & S\ ;\eta \vdash \mathsf{false} \rhd K & \\
S\ ;\eta \vdash \mathsf{true} \rhd (\_\ \texttt{\&\&}\ e_2\ , K) & \longrightarrow & S\ ;\eta \vdash e_2 \rhd K & \\
\\
S\ ;\eta \vdash x \rhd K & \longrightarrow & S\ ;\eta \vdash \eta(x) \rhd K &
\end{array}
$$

Summary: Expressions

$$
\begin{array}{lcl}
S\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright (s\ ,\ K) & \longrightarrow & S\ ;\ \eta \vdash s \blacktriangleright K \\
S\ ;\ \eta \vdash \mathsf{assign}(x, e) \blacktriangleright K & \longrightarrow & S\ ;\ \eta \vdash e \rhd (\mathsf{assign}(x, \_)\ ,\ K) \\
S\ ;\ \eta \vdash c \rhd (\mathsf{assign}(x, \_)\ ,\ K) & \longrightarrow & S\ ;\ \eta[x \mapsto c] \vdash \mathsf{nop} \blacktriangleright K \\
\\
S\ ;\ \eta \vdash \mathsf{decl}(x, \tau, s) \blacktriangleright K & \longrightarrow & S\ ;\ \eta[x \mapsto \mathsf{nothing}] \vdash s \blacktriangleright K \\
\\
S\ ;\ \eta \vdash \mathsf{assert}(e) \blacktriangleright K & \longrightarrow & S\ ;\ \eta \vdash e \rhd (\mathsf{assert}(\_)\ ,\ K) \\
S\ ;\ \eta \vdash \mathsf{true} \rhd (\mathsf{assert}(\_)\ ,\ K) & \longrightarrow & S\ ;\ \eta \vdash \mathsf{nop} \blacktriangleright K \\
S\ ;\ \eta \vdash \mathsf{false} \rhd (\mathsf{assert}(\_)\ ,\ K) & \longrightarrow & \mathsf{exception(abort)} \\
\\
S\ ;\ \eta \vdash \mathsf{if}(e, s_1, s_2) \blacktriangleright K & \longrightarrow & S\ ;\ \eta \vdash e \rhd (\mathsf{if}(\_, s_1, s_2)\ ,\ K) \\
S\ ;\ \eta \vdash \mathsf{true} \rhd (\mathsf{if}(\_, s_1, s_2), K) & \longrightarrow & S\ ;\ \eta \vdash s_1 \blacktriangleright K \\
S\ ;\ \eta \vdash \mathsf{false} \rhd (\mathsf{if}(\_, s_1, s_2), K) & \longrightarrow & S\ ;\ \eta \vdash s_2 \blacktriangleright K \\
\\
S\ ;\ \eta \vdash \mathsf{while}(e, s) \blacktriangleright K & \longrightarrow & S\ ;\ \eta \vdash \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop}) \blacktriangleright K
\end{array}
$$

Summary: Statements

$$
\begin{array}{lcl}
S \;;\, \eta \vdash f(e_1, e_2) \rhd K & \longrightarrow & S \;;\, \eta \vdash e_1 \rhd (f(\_, e_2)\ ,\ K) \\
S \;;\, \eta \vdash c_1 \rhd (f(\_, e_2)\ ,\ K) & \longrightarrow & S \;;\, \eta \vdash e_2 \rhd (f(c_1, \_)\ ,\ K) \\
S \;;\, \eta \vdash c_2 \rhd (f(c_1, \_)\ ,\ K) & \longrightarrow & (S\ ,\ \langle \eta, K \rangle) \;;\, [x_1 \mapsto c_1, x_2 \mapsto c_2] \vdash s \blacktriangleright \cdot \\
 & & \textit{(given that } f \textit{ is defined as } \ f(x_1, x_2)\{s\}\textit{)} \\
S \;;\, \eta \vdash f(\,) \rhd K & \longrightarrow & (S\ ,\ \langle \eta, K \rangle) \;;\, \cdot \vdash s \blacktriangleright \cdot \\
 & & \textit{(given that } f \textit{ is defined as } \ f(\,)\{s\}\textit{)} \\
S \;;\, \eta \vdash \mathsf{return}(e) \blacktriangleright K & \longrightarrow & S \;;\, \eta \vdash e \rhd (\mathsf{return}(\_)\ ,\ K) \\
(S\ ,\ \langle \eta', K' \rangle) \;;\, \eta \vdash v \rhd (\mathsf{return}(\_)\ ,\ K) & \longrightarrow & S \;;\, \eta' \vdash v \rhd K' \\
\cdot \;;\, \eta \vdash c \rhd (\mathsf{return}(\_)\ ,\ K) & \longrightarrow & \mathsf{value}(c)
\end{array}
$$

Summary: Functions