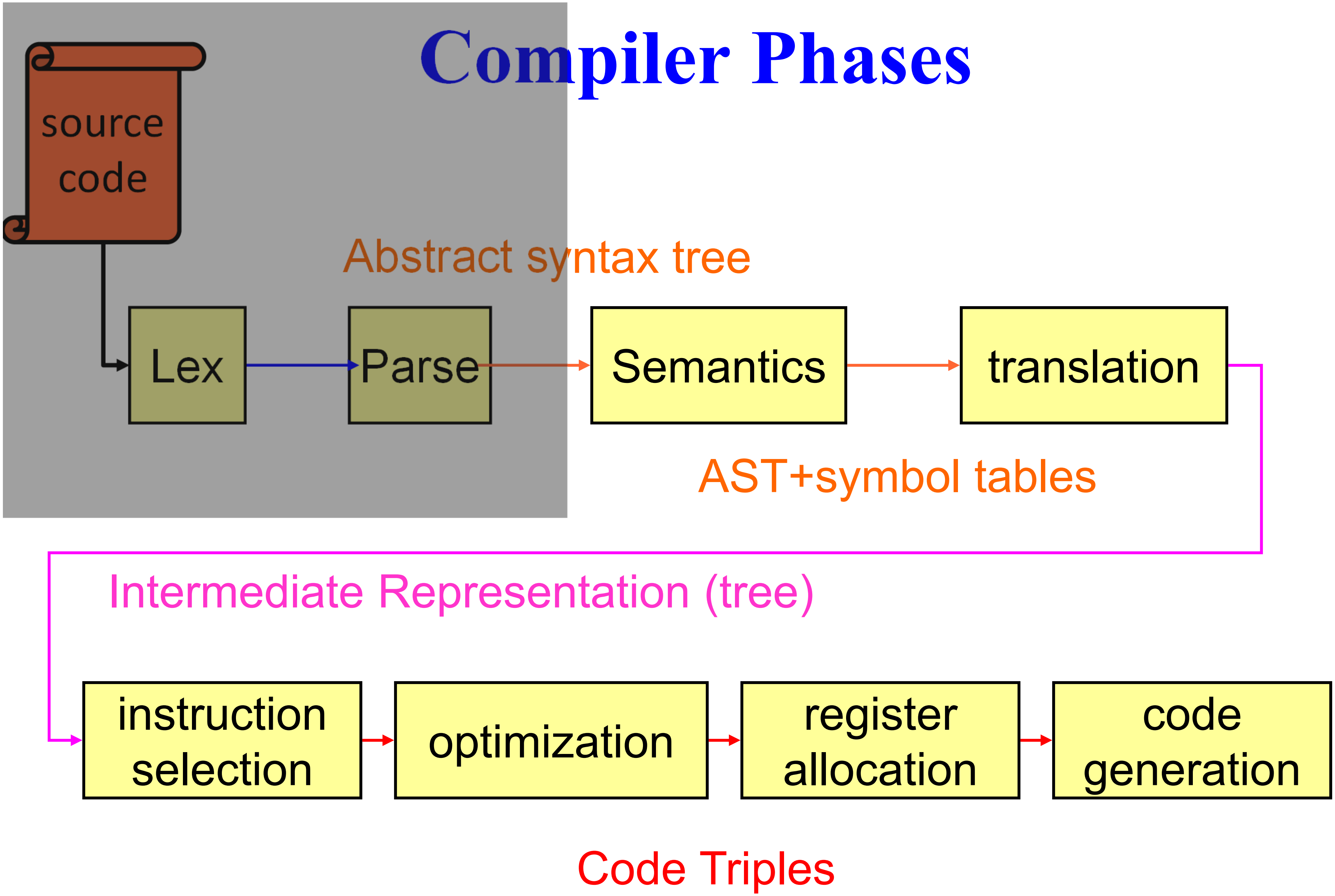
Dynamic Semantics

15-411/15-611 Compiler Design

Seth Copen Goldstein

October 15, 2020

Compiler Phases



Today

- Overview
- Our destination
- Assumptions
- Evaluation
- Variables and the environment
- Execution
- Functions, returns, and the stack
- L3 summary

Dynamic Semantics

- Formally describe how programs execute
- Concise and precise definition
- Our Purpose: Informs compiler writing.
- Could: prove properties about
  - source programs
  - compiler transformations
  - resulting executable

Static → Dynamic

- Static semantics describes which programs are well-formed
- Dynamic semantics describes how well-formed programs execute
- A language is safe when all well-formed programs are well-behaved.

Approaches to Dynamic Semantics

- Denotational:
  What does the program mean?
- Axiomatic:
  What can we prove about the program?
- Operational:
  How does the program execute?

Operational Semantics

- Structural (small-step semantics)
  What are the basic steps of the execution
- Natural (large-step semantics)
  Relationship of operations to effects

- operational semantics on abstract machines
  - syntax directed
  - inductive
  - transition rules which formally describe how a piece of syntax will change the abstract machines

Our destination

aka: End of the next lecture

Our destination

Evaluation of expression e in the context of

- a Heap,
- Stack, and
- binding environment.

$$H; S; \eta \vdash e \rhd K$$

Our destination

Evaluation of expression e in the context of

- a Heap,
- Stack, and
- binding environment.

$$H; S; \eta \vdash e \rhd K$$

Small-step semantics: where is the program counter?

Our destination

Evaluation of expression e in the context of

- a Heap,
- Stack, and
- binding environment.

$$H; S; \eta \vdash e \rhd K$$

K is a continuation, i.e.,

evaluate e and pass result to *K*

Our destination

Execution of a statement s in the context of

- a Heap,
- Stack, and
- binding environment.

$$H ; S ; \eta \vdash s \blacktriangleright K$$

Execute *s* and then the next statement in *K*

Assumptions

- Working on our standard AST:
  - expressions (n,x,$\oplus$,…) and
  - statements (decl, assign, return, …)
- Working on well-formed ASTs, i.e., they pass static semantics
- It bears repeating: well-formed programs are well-behaved
  - Or, as Milner quipped: “well typed programs do not go wrong.”
  - “well typed programs do not get stuck.”

$e \rhd K$

- Evaluate *e* pass result into *K*
- For example,

$$e_1 + e_2 \rhd K$$

$e \rhd K$

- Evaluate e pass result into K
- For example, we have the judgement:

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

- Evaluate $e_1 + e_2$ by evaluating e_1 and then pass value into $\blacksquare$ and continue.
- $\blacksquare$ is "hole" into which we put the value of e_1 after it is evaluated.

$e \rhd K$

- Evaluate *e* pass result into *K*
- For example, we have the judgements:

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

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

$$c_2 \rhd (c_1 + \blacksquare, K) \longrightarrow c \rhd K$$

Where, $c = c_1 + c_2 \bmod 2^{32}$

Pure arithmetic ops, $\oplus$

$$e_1 \oplus e_2 \rhd K \longrightarrow e_1 \rhd (\blacksquare \oplus e_2, K)$$

$$c_1 \rhd (\blacksquare \oplus e_2, K) \longrightarrow e_2 \rhd (c_1 \oplus \blacksquare, K)$$

$$c_2 \rhd (c_1 \oplus \blacksquare, K) \longrightarrow c \rhd K$$

Where, $c = c_1 \oplus c_2 \bmod 2^{32}$

ops that can cause exceptions: $\oslash$

$$e_1 \oslash e_2 \rhd K \longrightarrow e_1 \rhd (\blacksquare \oslash e_2, K)$$

$$c_1 \rhd (\blacksquare \oslash e_2, K) \longrightarrow e_2 \rhd (c_1 \oslash \blacksquare, K)$$

$$c_2 \rhd (c_1 \oslash \blacksquare, K) \longrightarrow c \rhd K \qquad \text{if } c = c_1 \oslash c_2$$

$$c_2 \rhd (c_1 \oslash \blacksquare, K) \longrightarrow \mathrm{excpt(arith} \quad) \qquad \text{if } c_1 \oslash c_2 \text{ undef}$$

The empty continuation

- $$c \rhd \cdot$$
  indicates there is nothing more to do
- We stop and return
$$\mathrm{value}(c)$$
- Giving the judgement:
$$c \rhd \cdot \longrightarrow \mathrm{value}(c)$$

short-circuiting

$$
\begin{array}{lcl}
e_1 \;\&\&\; e_2 \rhd K & \longrightarrow & e_1 \rhd (\_ \;\&\&\; e_2 \;,\; K) \\
\mathsf{false} \rhd (\_ \;\&\&\; e_2 \;,\; K) & \longrightarrow & \mathsf{false} \rhd K \\
\mathsf{true} \rhd (\_ \;\&\&\; e_2 \;,\; K) & \longrightarrow & e_2 \rhd K
\end{array}
$$

- of note:
  - Booleans are not 0 & 1, but false & true

Example

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

Example

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

Example

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

Example

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

$$\longrightarrow \quad \boxed{(4+5)} * \boxed{10} \quad \rhd \quad \_ + 2$$

Example

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

Example

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

Example

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

Example

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

Example

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

Example

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

Example

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

variables and η

- We need to keep track of variables and their values
- $\boldsymbol{\eta}$ defines the environment
  - if x has the value v in the environment, then
$$\eta(x) = v$$
  - We add a value v for x to the environment
$$\eta[x \mapsto v]$$
    yielding
$$\eta, x \mapsto v$$

Our new abstract machine

$$\eta \vdash e \rhd K$$

- We add a rule for variables,

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

- Why is this rule ok? I.e., what if x is undefined?

Our new abstract machine

$$\eta \vdash e \rhd K$$

- We add a rule for variables,

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

- Why is this rule ok? x is never undefined since we already passed static semantics

Our new abstract machine

$$\eta \vdash e \rhd K$$

- We add a rule for variables,

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

- And, augment old rules with η, e. g.,

$$\eta \vdash e_1 \oplus e_2 \rhd K \;\longrightarrow \eta \vdash e_1 \rhd (\blacksquare \oplus e_2, K)$$

Execution

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

- Statements alter the environment and then become a **nop**, and then goto the statement in K

$$\begin{array}{lcl} \eta \vdash \mathsf{seq}(s_1, s_2) \blacktriangleright K & \longrightarrow & \eta \vdash s_1 \blacktriangleright (s_2\ , K) \\ & \longrightarrow & \eta \vdash \text{ nop } \blacktriangleright (s_2, K) \\ & \longrightarrow & \eta \vdash s_2 \blacktriangleright K \end{array}$$

Execution

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

- Statements alter the environment and then become a **`nop`**, and then goto the statement in K

$$
\begin{array}{lcl}
\eta \vdash \mathsf{seq}(s_1, s_2) \blacktriangleright K & \longrightarrow & \eta \vdash s_1 \blacktriangleright (s_2\ ,\ K) \\
\eta \vdash \mathsf{nop} \blacktriangleright (s\ ,\ K) & \longrightarrow & \eta \vdash s \blacktriangleright K
\end{array}
$$

Modifying η

- Declaration adds a mapping to η

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

- Assignment, changes the value in η

Modifying η

- Declaration adds a mapping to η

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

- Assignment, changes the value in η (after evaluating the right hand side.)

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

Modifying η

- Declaration adds a mapping to η

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

- Assignment, changes the value in η (after evaluating the right hand side.)

$$\begin{array}{lcl} \eta \vdash \mathsf{assign}(x, e) \blacktriangleright K & \longrightarrow & \eta \vdash e \triangleright (\mathsf{assign}(x, \_)\ ,\ K) \\ \eta \vdash v \triangleright (\mathsf{assign}(x, \_)\ ,\ K) & \longrightarrow & \eta[x \mapsto v] \vdash \mathsf{nop} \blacktriangleright K \end{array}$$

Scoping

$$[x \mapsto v_1] \vdash \mathrm{assign}(x, e) \blacktriangleright K$$

$$\longrightarrow \; [x \mapsto v_1] \vdash e \rhd (\mathrm{assign}(x, \blacksquare), K)$$

$$\longrightarrow \; [x \mapsto v_1] \vdash v_2 \rhd (\mathrm{assign}(x, \blacksquare), K)$$

$$\longrightarrow \; [x \mapsto v_2] \vdash \mathrm{nop} \quad \rhd K \qquad \blacktriangleright$$

Now, what does $[x \mapsto v_1, x \mapsto v_2] \vdash x \rhd K$ evaluate to?

Statements

- if

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

Statements

- if

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

- while

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

Statements

- if

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

- while

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

- assert

$$\eta \vdash \mathsf{assert}(e) \blacktriangleright K \quad \longrightarrow \quad \eta \vdash e \rhd (\mathsf{assert}(\_), K)$$
$$\eta \vdash \mathsf{true} \rhd (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \eta \vdash \mathsf{nop} \blacktriangleright K$$
$$\eta \vdash \mathsf{false} \rhd (\mathsf{assert}(\_), K) \quad \longrightarrow \quad \mathsf{exception}(\mathsf{abort})$$

Statements

- if

$$\begin{array}{lcl}
\eta \vdash \mathsf{if}(e, s_1, s_2) \blacktriangleright K & \longrightarrow & \eta \vdash e \rhd (\mathsf{if}(_, s_1, s_2)\ ,\ K) \\
\eta \vdash \mathsf{true} \rhd (\mathsf{if}(_, s_1, s_2), K) & \longrightarrow & \eta \vdash s_1 \blacktriangleright K \\
\eta \vdash \mathsf{false} \rhd (\mathsf{if}(_, s_1, s_2), K) & \longrightarrow & \eta \vdash s_2 \blacktriangleright K
\end{array}$$

- while

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

- assert

$$\begin{array}{lcl}
\eta \vdash \mathsf{assert}(e) \blacktriangleright K & \longrightarrow & \eta \vdash e \rhd (\mathsf{assert}(_), K) \\
\eta \vdash \mathsf{true} \rhd (\mathsf{assert}(_), K) & \longrightarrow & \eta \vdash \mathsf{nop} \blacktriangleright K \\
\eta \vdash \mathsf{false} \rhd (\mathsf{assert}(_), K) & \longrightarrow & \mathsf{exception}(\mathsf{abort})
\end{array}$$

- return?

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

$[x \mapsto 1] \vdash \mathsf{while}(x > 0, s) \quad \blacktriangleright \quad \cdot$

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s ≡ x=x+1

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

while(x>0,assign(x,x+1))

- Assuming $\eta = [x \mapsto 1]$
- and s $\equiv$ x=x+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 & [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}
$$

The return Statement

$$
\begin{aligned}
&\eta \vdash \mathrm{return}(e) \blacktriangleright K \\
&\qquad\qquad\qquad\qquad \longrightarrow \eta \vdash e \rhd (\mathrm{return}(\blacksquare), K) \\
&\qquad\qquad\qquad\qquad \longrightarrow \eta \vdash v \rhd (\mathrm{return}(\blacksquare), K)
\end{aligned}
$$

- But now what?

The return Statement

$$\eta \vdash \mathrm{return}(e) \blacktriangleright K$$

$$\longrightarrow \eta \vdash e \rhd (\mathrm{return}(\blacksquare), K)$$

$$\longrightarrow \eta \vdash v \rhd (\mathrm{return}(\blacksquare), K)$$

- We need to represent the stack, S, which will have
  - an environment
  - a continuation

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

- Our new abstract machine augments all old rules with S

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

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

The return Statement

$$
\begin{aligned}
S, \langle \eta', K' \rangle; \eta \vdash \mathrm{return}(e) \blacktriangleright K \\
&\longrightarrow S, \langle \eta', K' \rangle; \eta \vdash e \rhd (\mathrm{return}(\blacksquare), K) \\
&\longrightarrow S, \langle \eta', K' \rangle; \eta \vdash v \rhd (\mathrm{return}(\blacksquare), K) \\
&\longrightarrow S; \eta' \vdash v \rhd K'
\end{aligned}
$$

And, for void functions we need:

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

Function calls

- Special case with no arguments

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

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

Function calls

- Special case with 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 $f(\,)\{s\}$*)*

- And, two arguments

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

Function calls

- Special case with 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 $f(\,)\{s\}$*)*

- And, two arguments

$$\begin{array}{lcl} S\;;\eta\vdash f(e_1,e_2)\rhd K & \longrightarrow & S\;;\eta\vdash e_1\rhd(f(\_,e_2)\;,K) \\ S\;;\eta\vdash c_1\rhd(f(\_,e_2)\;,K) & \longrightarrow & S\;;\eta\vdash e_2\rhd(f(c_1,\_)\;,K) \end{array}$$

Function calls

- Special case with no arguments

$$S\ ;\eta \vdash f(\,) \rhd K \quad \longrightarrow \quad (S\ ,\langle \eta, K\rangle)\ ;\cdot \vdash s \blacktriangleright \cdot$$
$$\textit{(given that } f \textit{ is defined as } \quad f(\,)\{s\}\textit{)}$$

- And, two arguments

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

Putting it all together

- We start with

$$\cdot ; \cdot \vdash \mathrm{main}(\,) \rhd \cdot$$

- We stop with (assuming main returns *c*)

$$\cdot ; \eta \vdash c \rhd \cdot \quad \longrightarrow \quad \mathrm{value}(c)$$

Putting it all together

- We start with

$$\cdot;\cdot \vdash \text{main}(\,) \rhd \cdot$$

- We stop with (assuming main returns *c*)

$$\cdot;\eta \vdash c \rhd \cdot \quad \longrightarrow \quad \text{value}(c)$$

- Unless, we get an error

$$\text{exception}(E)$$

Putting it all together

- We start with

$$\cdot ; \cdot \vdash \mathrm{main}(\,) \rhd \cdot$$

- We stop with (assuming main returns *c*)

$$\cdot ; \eta \vdash c \rhd \cdot \quad \longrightarrow \quad \mathrm{value}(c)$$

- Unless, we get an error

$$\mathrm{exception}(E)$$

- And, along the way,

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

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

L3

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	$::=$	$\mathsf{arith} \mid \mathsf{abort}$

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

$$
\begin{array}{lll}
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}(\mathsf{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 \\
\\
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 \\
 & & \quad \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 \\
 & & \qquad \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}
$$

Pretty Amazing

- Clear, Concise
- What about rule set?
  - deterministic?
  - ?
- But, the amazing thing is:

Theorem 1 (No undefined behavior) *If a program is valid as defined by 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 $\mathcal{ST}_n \quad \longrightarrow \quad \mathcal{ST}'$.

Next Time

- memory!