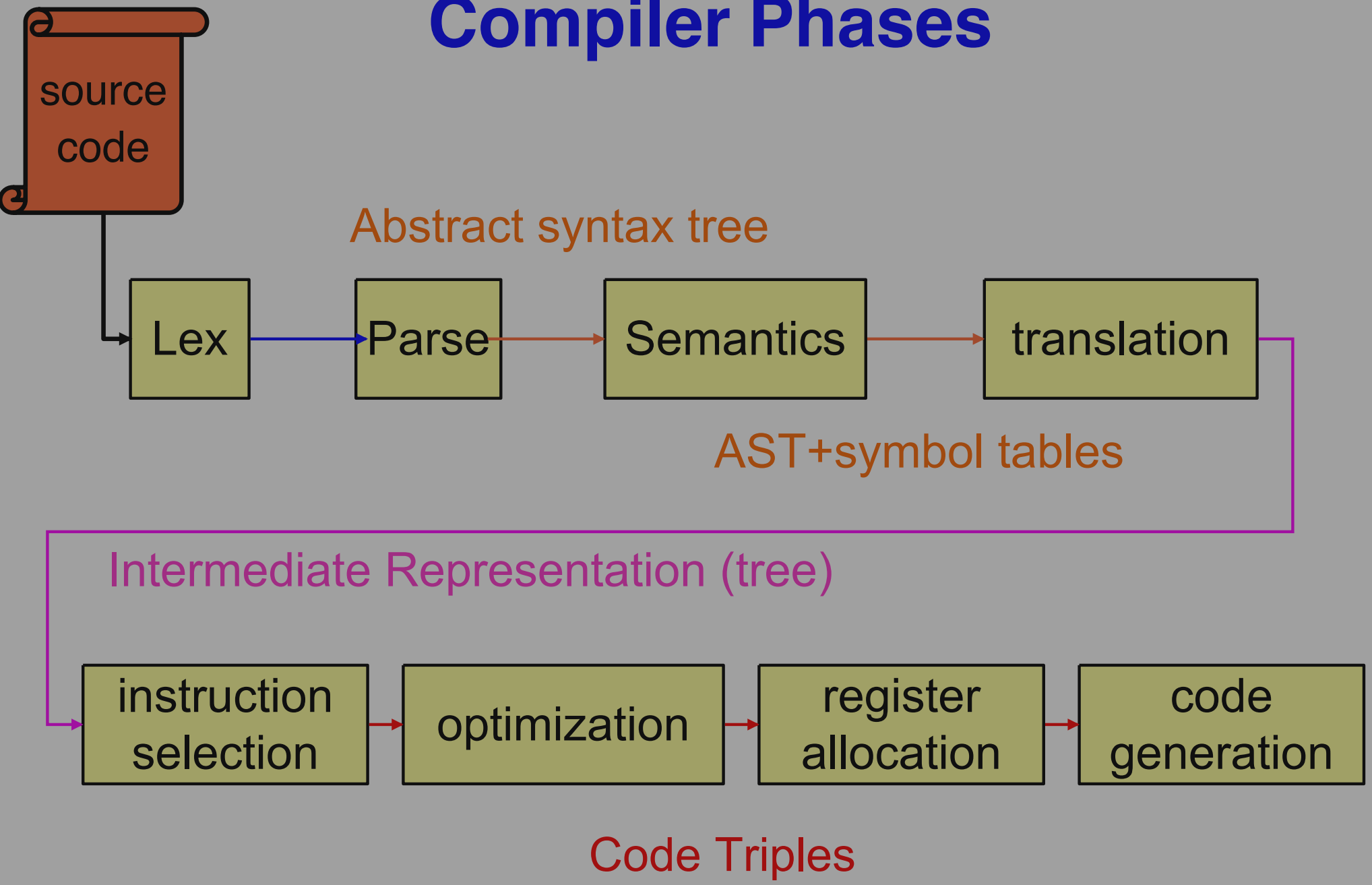
Dynamic Semantics

15-411/15-611 Compiler Design

Seth Copen Goldstein

February 27, 2025

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

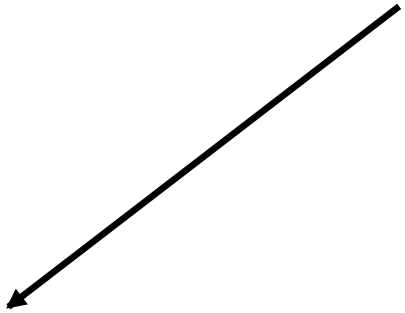
- Denotational:
  What does the program mean?
- Axiomatic:
  What can we prove about the program?
- Operational:
  How does the program execute?

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

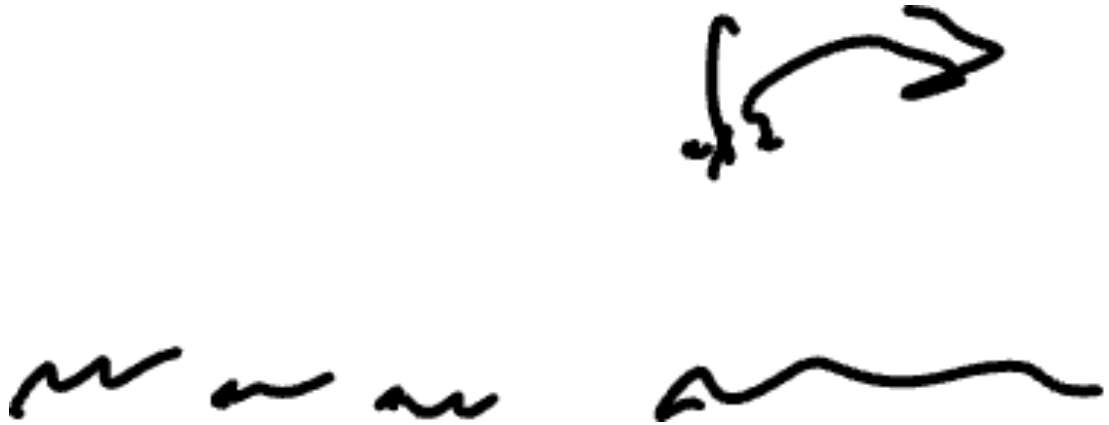


aka: End of the next lecture

Our destination

Evaluation of expression e in the context of

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



Our destination

Evaluation of expression e in the context of

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

Small-step semantics: where is the program counter?

Our destination

Evaluation of expression e in the context of

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

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.

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

Assumptions

- Working on our standard AST:
  - expressions (*n*,*x*,□ … 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."

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

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

- Evaluate by evaluating and then pass value into and continue.
- is “hole” into which we put the value of after it is evaluated.

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

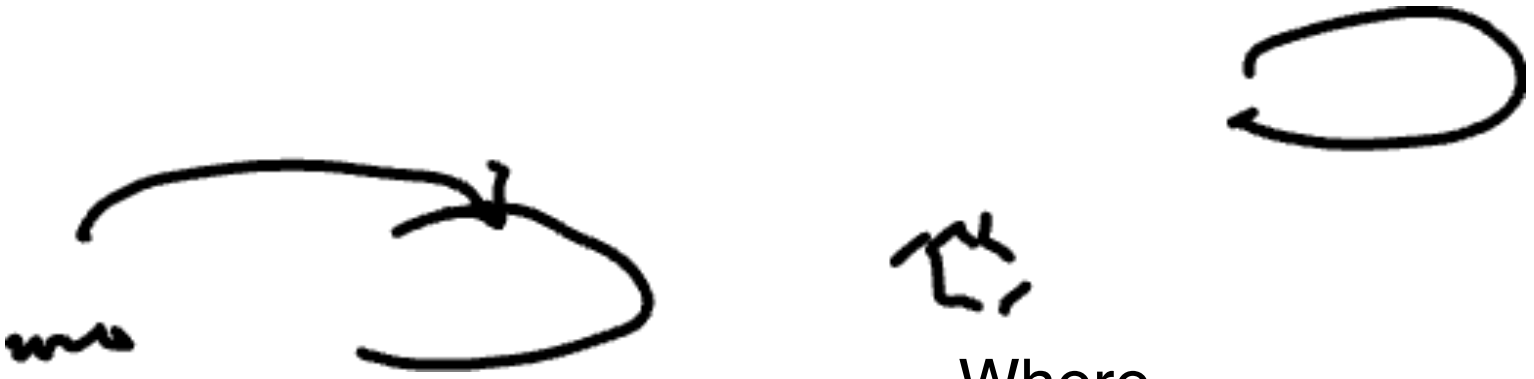
- Evaluate by evaluating and then pass value into and continue.
- is “hole” into which we put the value of after it is evaluated.

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

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

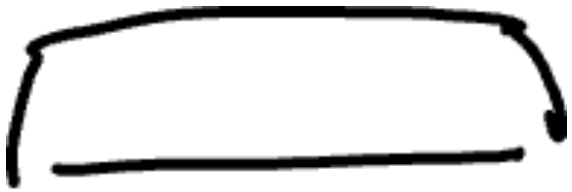
- Evaluate *e* pass result into *K*
- For example, we have the judgements:

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

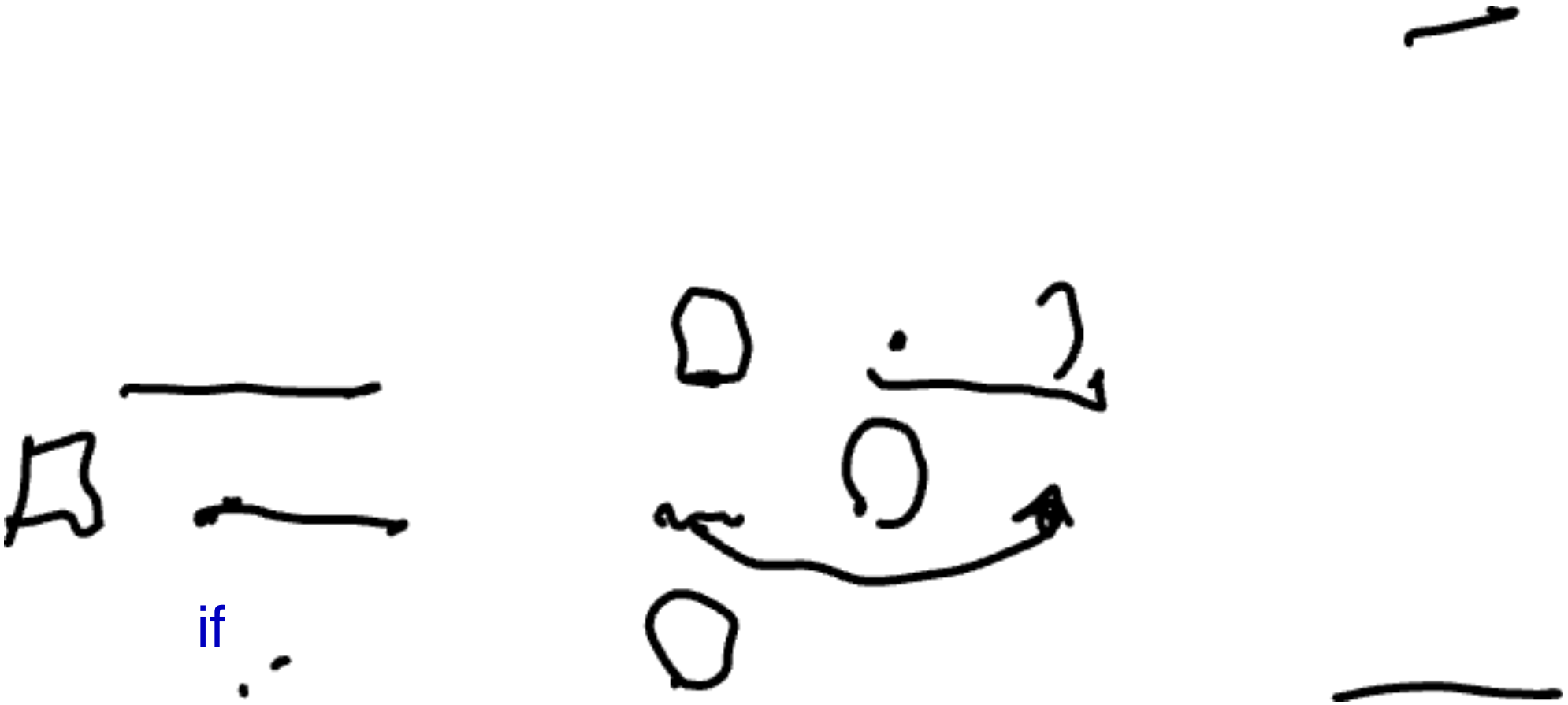


Where,

Pure arithmetic ops,

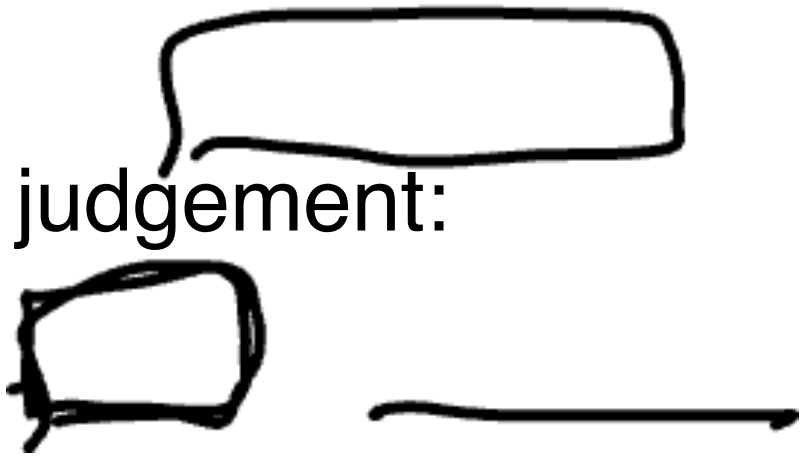
Where, 

ops that can cause exceptions:



The empty continuation

- 
  indicates there is nothing more to do
- We stop and return
- Giving the judgement:



short-circuiting

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

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

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

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

short-circuiting

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

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

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

$e_1 \,||\, e_2 \rhd K \Rightarrow e_1 \rhd (\_ \,||\, e_2, K)$

$\mathsf{false} \rhd (\_ \,||\, e_2, K) \Rightarrow e_2 \rhd K$

$\mathsf{true} \rhd (\_ \,||\, e_2, K) \Rightarrow \mathsf{true} \rhd K$

Example

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

((4+5)*10) ▷ (B+2 ; ·)

Example

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

Example

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

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

e_1 e_2 K

Example

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

Example

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

Example

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

Example

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

Example

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

Example

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

Example

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

Example

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

variables and

- We need to keep track of variables and their values
- defines the environment
  - if x has the value v in the environment, then
  - We add a value v for x to the environment

    yielding

    ,

Our new abstract machine

- We add a rule for variables,

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

Our new abstract machine

- We add a rule for variables,

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

Our new abstract machine

- We add a rule for variables,

- And, augment old rules with

Execution

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

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

Execution

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

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

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

Modifying ?

- Declaration adds a mapping to ?

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

- Assignment, changes the value in ?

Modifying ?

- Declaration adds a mapping to ?

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

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

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

Modifying ?

- Declaration adds a mapping to ?

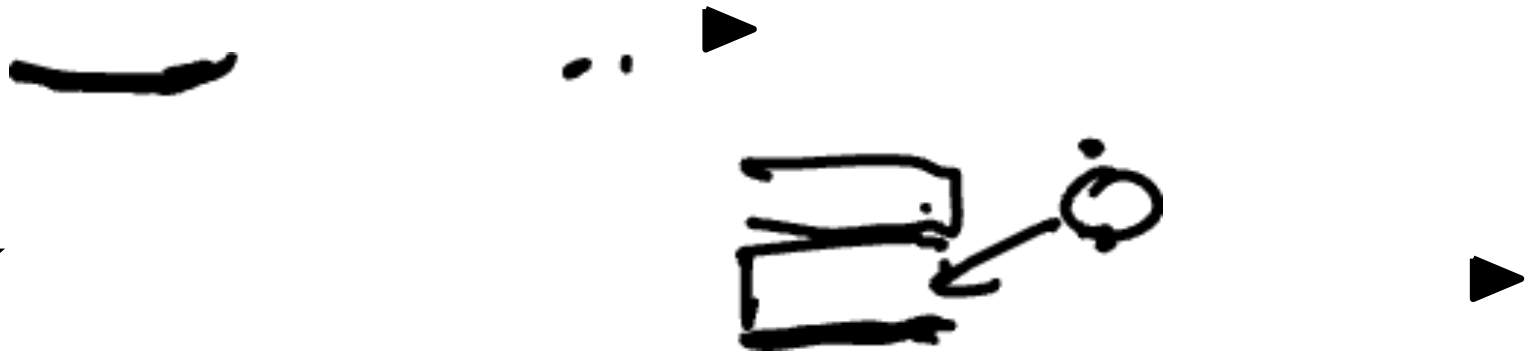
$$\eta \vdash \mathsf{decl}(x, \tau, s) \blacktriangleright K \longrightarrow \eta[x \mapsto \mathsf{nothing}] \vdash s \blacktriangleright K$$

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

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

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

Scoping



K

Now, what does *K* evaluate to?

Statements

- if

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

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

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

Statements

- if

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

- while

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

Statements

- if

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

- while

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

- assert

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

Statements

- if

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

- 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

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

- return?

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

- Assuming
- and s ? x=x+1

$[x \mapsto 1] \vdash \mathsf{while}(x > 0, s)$ ▶ ·

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

- Assuming
- and s ⍰ x=x+1

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

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

- Assuming
- and s ? x=x+1

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

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

- Assuming
- 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 \\
\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 & [x\mapsto 2] \vdash \mathsf{while}(x>0, s) & \blacktriangleright \quad \cdot
\end{array}
$$

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

- Assuming
- and s ? x=x+1

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

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

- Assuming
- 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
- and s ? x=x+1

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

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

- Assuming
- 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
- 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 \\
\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 & [x\mapsto 2] \vdash \mathsf{while}(x>0, s) & \blacktriangleright \quad \cdot
\end{array}
$$

The return Statement

- But now what?

The return Statement

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

The return Statement

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

- Our new abstract machine augments all old rules with S

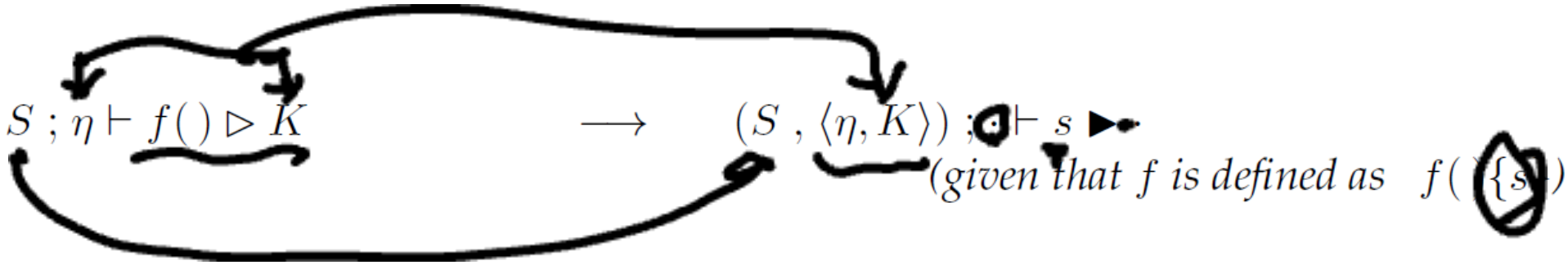
The return Statement

The return Statement

And, for void functions we need:

Function calls

- Special case with no arguments



Function calls

- Special case with no arguments

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

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

- And, two arguments

$$S\ ;\ \eta \vdash f(e_1, e_2) \triangleright K \quad \longrightarrow \quad S\ ;\ \eta \vdash e_1 \triangleright (f(\_, e_2)\ ,\ K)$$

Function calls

- Special case with no arguments

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

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

- And, two arguments

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

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

Function calls

- Special case with no arguments

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

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

- And, two arguments

$$\begin{aligned}
S\ ;\ \eta \vdash f(e_1, e_2) \rhd K &\longrightarrow S\ ;\ \eta \vdash e_1 \rhd (f(_, e_2)\ ,\ K) \\
S\ ;\ \eta \vdash c_1 \rhd (f(_, e_2)\ ,\ K) &\longrightarrow S\ ;\ \eta \vdash e_2 \rhd (f(c_1, _)\ ,\ K) \\
S\ ;\ \eta \vdash c_2 \rhd (f(c_1, _)\ ,\ K) &\longrightarrow (S\ ,\ \langle \eta, K \rangle)\ ;\ [x_1 \mapsto c_1, x_2 \mapsto c_2] \vdash s \blacktriangleright \cdot
\end{aligned}$$

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

Putting it all together

- We start with

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

Putting it all together

- We start with

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

- Unless, we get an error

Putting it all together

- We start with
- We stop with (assuming main returns *c*)
- Unless, we get an error
- And, along the way,

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

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

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

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

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* $\mathcal{ST}_n \longrightarrow \mathcal{ST}'$.

Next Time

- memory!

Next Time

- memory!

Sign up for code review

Enjoy spring break!