CS558
Programming Languages

Fall 2023
Lecture 4a

Andrew Tolmach
Portland State University

Formal Operational Semantics

- So far, we've presented operational semantics using interpreters: precise and executable, but verbose and too concrete
- One alternative: describe semantics using state transition judgments

Example language

```
exp := var | int
     |  '(' '+' exp exp ')'
     |  '(' 'let' var exp exp ')'
     |  '(' ':=' var exp ')'
     |  '(' 'if' exp exp exp ')'
     |  '(' 'while' exp exp ')'
     |  etc.
```

State machine transition judgments

current expression

result value

$$\langle e, E, S \rangle \Downarrow \langle v, S' \rangle$$

current environment
maps vars x to locations l

final store
maps locations l to values v

initial store
maps locations l to values v

Evaluating expression e in environment E and store S yields value v and (possibly) changed store S'

Evaluation by inference

- To describe the machine's operation, we give rules of inference that say when a judgment can be derived from judgments about sub-expressions

$$\frac{\textit{premises}}{\textit{conclusion}}\ \text{(Name of rule)}$$

- We can view evaluation of the program as the process of building an inference tree

- Notation has similarities to axiomatic semantics: idea of derivation is that same, but contents of judgments is different

Environments and Stores, Formally

• We write $E(x)$ means the result of looking up x in environment E. (This notation is because an environment is like a **function** taking a name as argument and returning a meaning as result.)

• We write $E + \{x \mapsto v\}$ for the environment obtained from existing environment E by **extending** it with a new binding from x to v. If E already has a binding for x, this new binding replaces it.

The **domain** of an environment, *dom*(E), is the set of names bound in E.

Analogously with environments, we'll write

• $S(l)$ to mean the value at location l of store S

• $S + \{l \mapsto v\}$ to mean the store obtained from store S by extending (or updating) it so that location l maps to value v.

• *dom*(S) for the set of locations bound in store S.

Also, we'll write

• $S - \{l\}$ to mean the store obtained from store S by removing the binding for location l.

EVALUATION RULES (1)

$$\frac{l = E(x) \quad v = S(l)}{\langle x, E, S\rangle \Downarrow \langle v, S\rangle} \text{ (Var)}$$

$$\frac{}{\langle i, E, S\rangle \Downarrow \langle i, S\rangle} \text{ (Int)}$$

$$\frac{\langle e_1, E, S\rangle \Downarrow \langle v_1, S'\rangle \quad \langle e_2, E, S'\rangle \Downarrow \langle v_2, S''\rangle}{\langle \texttt{(+}\ e_1\ \ e_2\texttt{)}, E, S\rangle \Downarrow \langle v_1 + v_2, S''\rangle} \text{ (Add)}$$

$$\frac{\begin{array}{c}\langle e_1, E, S\rangle \Downarrow \langle v_1, S'\rangle \qquad l \notin \textit{dom}(S') \\ \langle e_2, E + \{x \mapsto l\}, S' + \{l \mapsto v_1\}\rangle \Downarrow \langle v_2, S''\rangle\end{array}}{\langle \texttt{(let}\ x\ \ e_1\ \ e_2\texttt{)}, E, S\rangle \Downarrow \langle v_2, S'' - \{l\}\rangle} \text{ (Let)}$$

$$\frac{\langle e, E, S\rangle \Downarrow \langle v, S'\rangle \quad l = E(x)}{\langle \texttt{(:=}\ x\ \ e\texttt{)}, E, S\rangle \Downarrow \langle v, S' + \{l \mapsto v\}\rangle} \text{ (Assgn)}$$

Evaluation Rules (2)

$$\frac{\langle e_1, E, S\rangle \Downarrow \langle v_1, S'\rangle \quad v_1 \neq 0 \quad \langle e_2, E, S'\rangle \Downarrow \langle v_2, S''\rangle}{\langle \texttt{(if}\ \ e_1\ \ e_2\ \ e_3\texttt{)}, E, S\rangle \Downarrow \langle v_2, S''\rangle} \text{ (If-nzero)}$$

$$\frac{\langle e_1, E, S\rangle \Downarrow \langle 0, S'\rangle \quad \langle e_3, E, S'\rangle \Downarrow \langle v_3, S''\rangle}{\langle \texttt{(if}\ \ e_1\ \ e_2\ \ e_3\texttt{)}, E, S\rangle \Downarrow \langle v_3, S''\rangle} \text{ (If-zero)}$$

$$\frac{\begin{array}{c}\langle e_1, E, S\rangle \Downarrow \langle v_1, S'\rangle \qquad v_1 \neq 0 \qquad \langle e_2, E, S'\rangle \Downarrow \langle v_2, S''\rangle \\ \langle \texttt{(while}\ \ e_1\ \ e_2\texttt{)}, E, S''\rangle \Downarrow \langle v_3, S'''\rangle\end{array}}{\langle \texttt{(while}\ \ e_1\ \ e_2\texttt{)}, E, S\rangle \Downarrow \langle v_3, S'''\rangle} \text{ (While-nzero)}$$

$$\frac{\langle e_1, E, S\rangle \Downarrow \langle 0, S'\rangle}{\langle \texttt{(while}\ \ e_1\ \ e_2\texttt{)}, E, S\rangle \Downarrow \langle 0, S'\rangle} \text{ (While-zero)}$$

Example Derivation Trees

$$\dfrac{\dfrac{}{\langle \texttt{10}, \emptyset, \emptyset\rangle \Downarrow \langle 10, \emptyset\rangle}\ \text{(Int)} \quad \dfrac{\dfrac{\dfrac{}{\langle \texttt{21}, E_1, S_1\rangle \Downarrow \langle 21, S_1\rangle}\ \text{(Int)}}{\langle \texttt{(:= x 21)}, E_1, S_1\rangle \Downarrow \langle 21, S_2\rangle}\ \text{(Assgn)} \quad \dfrac{}{\langle \texttt{x}, E_1, S_2\rangle \Downarrow \langle 21, S_2\rangle}\ \text{(Var)}}{\langle \texttt{(+ (:= x 21) x)}, E_1, S_1\rangle \Downarrow \langle 42, S_2\rangle}\ \text{(Add)}}{\langle \texttt{(let x 10 (+ (:= x 21) x))}, \emptyset, \emptyset\rangle \Downarrow \langle 42, \emptyset\rangle}\ \text{(Let)}$$

where $E_1 = \{x \mapsto L_1\}$, $S_1 = \{L_1 \mapsto 10\}$, $S_2 = \{L_1 \mapsto 21\}$.

$$\dfrac{\dfrac{}{\langle \texttt{x}, E_1, S_1\rangle \Downarrow \langle -1, S_1\rangle}\ \text{(Var)} \quad \dfrac{\dfrac{\dfrac{}{\langle \texttt{x}, E_1, S_1\rangle \Downarrow \langle -1, S_1\rangle}\ \text{(Var)} \quad \dfrac{}{\langle \texttt{1}, E_1, S_1\rangle \Downarrow \langle 1, S_1\rangle}\ \text{(Int)}}{\langle \texttt{(+ x 1)}, E_1, S_1\rangle \Downarrow \langle 0, S_1\rangle}\ \text{(Add)}}{\langle \texttt{(:= x (+ x 1))}, E_1, S_1\rangle \Downarrow \langle 0, S_2\rangle}\ \text{(Assgn)} \quad \dfrac{\dfrac{}{\langle \texttt{x}, E_1, S_2\rangle \Downarrow \langle 0, S_2\rangle}\ \text{(Var)}}{\langle \texttt{(while x (:= x (+ x 1)))}, E_1, S_2\rangle \Downarrow \langle 0, S_2\rangle}\ \text{(While-zero)}}{\langle \texttt{(while x (:= x (+ x 1)))}, E_1, S_1\rangle \Downarrow \langle 0, S_2\rangle}\ \text{(While-nzero)}$$

where $E_1 = \{x \mapsto L_1\}$, $S_1 = \{L_1 \mapsto -1\}$, $S_2 = \{L_1 \mapsto 0\}$.

About the rules

- As usual in inference systems, a rule only applies if all the premises above the line can be shown
- Structure of rules guarantees that at most one rule is applicable at any point
- Store relationships constrain the order of evaluation of premises
  - (For simplicity here, we use just a single global store)
- If no rules apply, the evaluation gets stuck; this corresponds to an (unchecked) runtime error

Rules vs. Interpreter

- We can view a recursive interpreter as implementing a bottom-up exploration of the inference tree

- A function like

```
Value eval(Exp e, Env env) { .... }
```

returns a value `v` and has side effects on a global store `store` such that

$$\langle \texttt{e}, \texttt{env}, \texttt{store}_{\textit{before}} \rangle \Downarrow \langle \texttt{v}, \texttt{store}_{\textit{after}} \rangle$$

- The implementation of `eval` dispatches on the syntactic form of `e`, choosing the appropriate rule,

- and makes recursive calls on `eval` corresponding to the premises of that rule.

- Question: How deep can the derivation tree get?