Big-Step Semantics

(+ introduction to Semantics)

"Semantics" = meaning/significance (ancient Greek)
of programs

vs. "Syntax" (rules for writing valid programs) i.e., given a program — a collection of valid syntactic objects . . .

semantic analysis describes mathematically the program's behavior

"Mathematical"?

- rigorous / precise
- abstract / symbolic representation
- amenable to proof methods (e.g., proof by induction)

Approaches to modeling semantics

- **Operational semantics** — describe the execution of a program and its effects directly, as on an abstract machine
- **Denotational semantics** — model language constructs as "denotations", i.e., values in a mathematical domain, and reason about those
- **Axiomatic semantics** — describe a program in terms of logical predicates that are satisfied before and after its execution

Types of Operational Semantics

- **Big-step** - evaluate an entire expression in one big step

  $e \Downarrow v$: expression e evaluates to value v

- **Small-step** - evaluation is modeled as a series of small steps

  $e \rightarrow e' \rightarrow e'' \rightarrow e''' \rightarrow \cdots \rightarrow v$

  $e \rightarrow^* v$: e evaluate to v after a series of steps

How can we assert/prove that evaluations and computations proceed a certain way?

need to be able to:

1. make assertions about individual syntactic objects
2. connect these assertions
3. use these connections to create larger narratives (proofs!)

Inference Rules

form of a rule: $$[\text{NAME}]\ \frac{\text{premise}_1 \ \ldots\ \text{premise}_k}{\text{conclusion}}\ [\text{side conditions}]$$

where each premise & the conclusion is an assertion about a syntactic object

aka "judgement"

e.g., assertions: n is odd

$e \rightarrow e'$

$e \Downarrow v$

$\vdash e : t$ ← type of e is t

Inference Rules

form of a rule:
$$[\text{NAME}]\ \frac{\text{premise}_1 \ \ldots \ \text{premise}_k}{\text{conclusion}}\ [\text{side conditions}]$$

meaning of a rule: if all premises (and side condition(s), if present) hold, then the conclusion holds

a rule w/ <u>no</u> premises is an <u>axiom</u>,
otherwise it is a <u>proper rule</u>

e.g., inference rules for even/odd integers over multiplication

$$\text{MOD0}\ \frac{x \bmod 2 = 0}{x \text{ is even}} \qquad \text{MOD1}\ \frac{x \bmod 2 = 1}{x \text{ is odd}}$$

$$\text{EVENXEVEN}\ \frac{x \text{ is even} \quad y \text{ is even}}{x * y \text{ is even}} \qquad \text{ODDXEVEN}\ \frac{x \text{ is odd} \quad y \text{ is even}}{x * y \text{ is even}}$$

$$\text{EVENXODD}\ \frac{x \text{ is even} \quad y \text{ is odd}}{x * y \text{ is even}} \qquad \text{ODDXODD}\ \frac{x \text{ is odd} \quad y \text{ is odd}}{x * y \text{ is odd}}$$

Proof Trees

- we can derive a proof tree for a given assertion, where:
  - the root of the proof tree (its final conclusion) is the assertion we wish to prove
  - each subtree is a proof tree of the conclusion's premises

e.g.

leaves = axioms

$$\cfrac{\cfrac{P_1}{C_1} \qquad P_2 \qquad \cfrac{P_3 \qquad \cfrac{P_4 \quad P_5}{C_4}}{C_3}}{C}$$

"root"

Derivation strategies

- forward chaining / bottom-up construction
  - start w/ axioms and work towards conclusion
  - end goal may not be reachable — when to stop?
- backward chaining / top-down construction
  - start w/ conclusion and work "backwards" towards axioms
  - goal is known — can we derive it?

e.g., prove "21347 × 12345 is odd"

$$
\dfrac{\dfrac{21347 \bmod 2 = 1}{21347 \text{ is odd}}\ \text{MOD1} \qquad \dfrac{12345 \bmod 2 = 1}{12345 \text{ is odd}}\ \text{MOD1}}{21347 \times 12345 \text{ is odd}}\ \text{ODD} \times \text{ODD}
$$

Big-step Semantics

- a big-step semantic rule makes an assertion about what a syntactic object would evaluate to on an abstract machine
- state of the abstract machine (aka "configuration") includes:
    - the syntactic object to evaluate (e.g., expressions, statements, etc.)
    - the environment (e.g., variable → value mappings)

The Evaluation Relation: "$\Downarrow$"

$\Downarrow$ is a relation over:

- syntactic objects
- environments
- possible results

cartesian product

i.e., $\Downarrow \in S \times \Sigma \times R$

we write (in "mixfix" form): $\langle s, \sigma \rangle \Downarrow r$

"Syntactic Objects"

i.e., as found in a program written in some language.

- what syntactic objects are possible/valid?
- need a formal way of specifying a language
    - Grammar

a grammar $G = (V, T, S, P)$ consists of:

V, the vocabulary: a non-empty set of symbols.

$T \subset V$: the set of terminal symbols

cannot be rewritten as other syms.

$V - T = N$: the set of non-terminal symbols

$S \in V$: the start symbol

P: a set of productions

each production is a "rewrite rule" of form LHS → RHS; LHS & RHS consist of symbols, and LHS contains at least one non-terminal.

e.g., use the following grammar to generate some sentences:

$V = \{ (,), E \}$

$T = \{ (,) \}$

$S = E$

$P = \{ E \to EE, E \to (E), E \to \varepsilon \}$

epsilon: empty string.

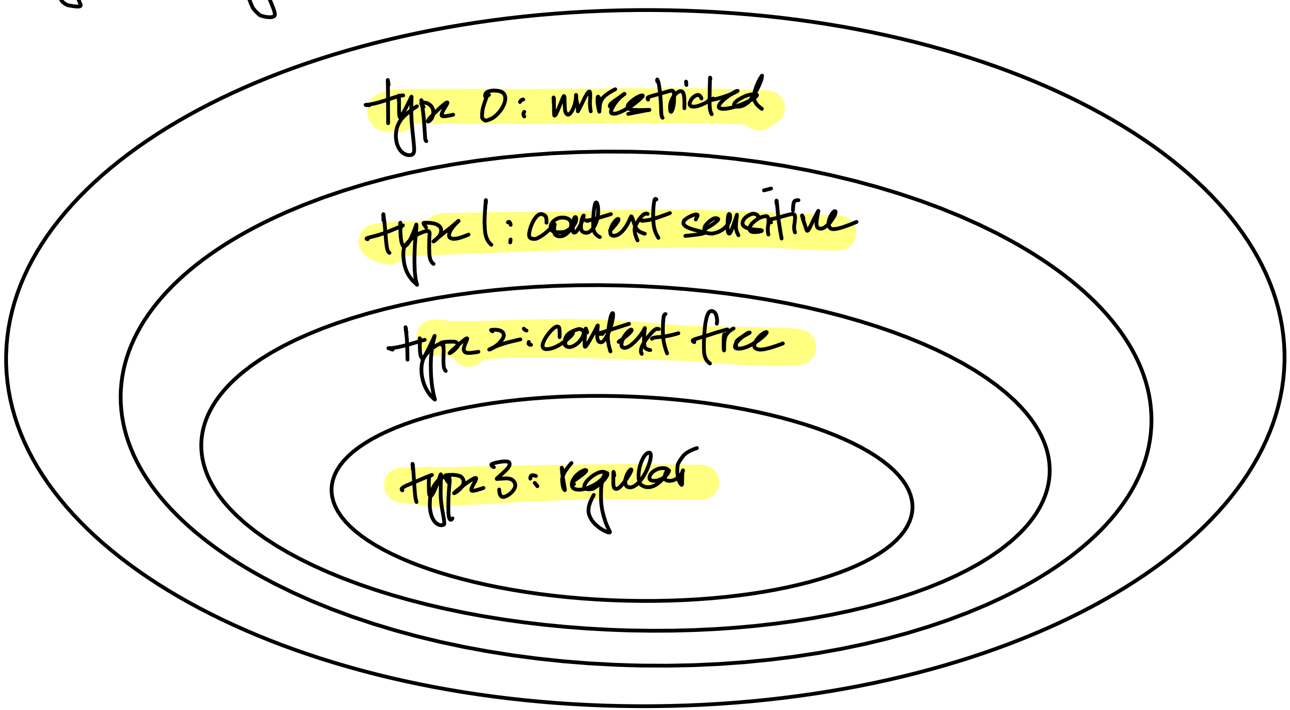
1. $E \Rightarrow \varepsilon = $ " "

2. $E \Rightarrow (E) \Rightarrow ((E)) \Rightarrow ((\varepsilon)) \Rightarrow (())$

3. $E \Rightarrow EE \Rightarrow (E)E \Rightarrow (\varepsilon)E \Rightarrow ()(E) \Rightarrow ()(\varepsilon) = ()()$

Noam Chomsky created a classification system for formal grammar in 1956.

Chomsky hierarchy:



type 0 (unrestricted): no restrictions on grammar

A in some context

type 1 (context-sensitive): productions have form $\alpha A \beta \to \alpha \gamma \beta$, where A is a non-terminal and α, β, γ are strings of terminals/non-terminals (γ non-empty).

LHS has no context

type 2 (context-free): productions have form $A \to \alpha$, where A is a non-terminal and α is a string of terminals/non-terminals.

type 3 (regular): [right regular]: productions have form $A \to aB$ or $A \to a$; [left regular]: productions have form $A \to Ba$ or $A \to a$, where A, B are non-terminals, and a is a terminal symbol (or ε)

Regular grammars, the simplest type, are often used to specify the syntax of tokens for parsers

- regular expressions (regexes) are a way to specify these grammars

Virtually all programming languages are specified using Context-free grammars (CFGs)

- Backus-Naur Form (BNF) is often used to specify a CFG

Regular Expressions

e.g., for a floating point number

$$\underbrace{[+-]}_{\text{char group}}?([0-9]*\backslash.)?[0-9]+$$

1 or more of preceding elem

optional preceding elem

0 or more of preceding elem

+42.15
−99.0
123
+8
⋮

- note lack of "memory"
  - cannot describe structures w/ matching/nested elements

(real world regex libraries are more complex)

Backus-Naur Form

- list all productions in form

<non-terminal> ::= <term/non-term>+ (| <term/non-term>)*

e.g., English phrases

SENTENCE ::= NP VP
NP ::= ARTICLE ADJ NOUN | ARTICLE NOUN
VP ::= VERB ADV | VERB
ARTICLE ::= the | a
ADJ ::= red | big
NOUN ::= car | computer
VERB ::= runs | jumps
ADV ::= hungrily | happily

"Simple Imperative Programming Language" (IMP)

$+, \times$

arithmetic exps: $E ::= \text{Integer} \mid \text{Var} \mid E \oplus E$

boolean exps: $B ::= \text{true} \mid \text{false} \mid E \sim E$

$<, >, =, \ldots$

statements: $S ::= \text{skip}$

$\mid \text{Var} := E$

$\mid S_1 ; S_2$

sequencing

$\mid \text{if } B \text{ then } S_1 \text{ else } S_2$

$\mid \text{while } B \text{ do } S$

Evaluation relations:

exp   environ (vars)

resulting value (integer)

arith exps : $\langle E, \sigma \rangle \Downarrow_e v$

boolean value

boolean exps : $\langle B, \sigma \rangle \Downarrow_b b$

new environment

statements : $\langle S, \sigma \rangle \Downarrow_s \sigma'$

* we will overload $\Downarrow$ when it is clear by context which relation to use

Arithmetic Expression Rules

LITERAL $\dfrac{}{\langle i, \sigma \rangle \Downarrow i}$ if $i \in \mathbb{Z}$

x is a var, and $x := v \in \sigma$

VARIABLE $\dfrac{\overbrace{\sigma(x) = v}}{\langle x, \sigma \rangle \Downarrow v}$

ARITH $\dfrac{\langle e_1, \sigma \rangle \Downarrow v_1 \quad \langle e_2, \sigma \rangle \Downarrow v_2}{\langle e_1 \oplus e_2, \sigma \rangle \Downarrow v_1 \oplus v_2}$

Boolean Expression Rules

$$\text{LITERAL}\ \frac{}{\langle b, \sigma \rangle \Downarrow b}\ \text{if } b \in \{\text{true}, \text{false}\}$$

$$\text{REL}\ \frac{\langle e_1, \sigma \rangle \Downarrow_e v_1 \quad \langle e_2, \sigma \rangle \Downarrow_e v_2}{\langle e_1 \sim e_2, \sigma \rangle \Downarrow_b v_1 \sim v_2}$$

skip Rule

$$\text{skip}\quad \frac{}{\langle \text{skip}, \sigma \rangle \Downarrow \sigma}$$

Assignment Rule

$$\text{ASSIGN}\ \frac{\langle e, \sigma \rangle \Downarrow_e v}{\langle x := e, \sigma \rangle \Downarrow \sigma \underbrace{(x := v)}_{\text{create new/update var mapping in env}}}$$

Sequencing ($S_1 ; S_2$) Rule

$$\text{SEQ}\ \frac{\langle S_1, \sigma \rangle \Downarrow \sigma' \quad \langle S_2, \sigma' \rangle \Downarrow \sigma''}{\langle S_1 ; S_2, \sigma \rangle \Downarrow \sigma''}$$

if-statement Rules

$$\text{IF-T}\ \frac{\langle b, \sigma\rangle \Downarrow_b \text{true} \qquad \langle S_1, \sigma\rangle \Downarrow \sigma'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, \sigma\rangle \Downarrow \sigma'}$$

$$\text{IF-F}\ \frac{\langle b, \sigma\rangle \Downarrow_b \text{false} \qquad \langle S_2, \sigma\rangle \Downarrow \sigma'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, \sigma\rangle \Downarrow \sigma'}$$

while-statement Rules

$$\text{WHILE-F}\quad \frac{\langle b, \sigma\rangle \Downarrow \text{false}}{\langle \text{while } b \text{ do } S, \sigma\rangle \Downarrow \sigma}$$

$$\text{WHILE-T}\quad \frac{\langle b, \sigma\rangle \Downarrow \text{true} \quad \langle S, \sigma\rangle \Downarrow \sigma' \quad \langle \text{while } b \text{ do } S, \sigma'\rangle \Downarrow \sigma''}{\langle \text{while } b \text{ do } S, \sigma\rangle \Downarrow \sigma''}$$

e.g., proof trees for IMP

- prove that $3 \times a + 4 \times b = 39$ in $\sigma = \{a := 5, b := 6\}$

- prove that while b do s
  is equivalent to if b then (s ; while b do s) else skip

– prove that $3\times a + 4\times b = 39$ in $\sigma = \{a := 5, b := 6\}$

$$\text{ARITH}\ \dfrac{\dfrac{\text{LITERAL}\ \dfrac{}{\langle 3,\sigma\rangle \Downarrow 3} \quad \text{VAR}\ \dfrac{}{\langle a,\sigma\rangle \Downarrow 5}}{\langle 3\times a,\sigma\rangle \Downarrow 15} \qquad \dfrac{\text{LITERAL}\ \dfrac{}{\langle 4,\sigma\rangle \Downarrow 4} \quad \text{VAR}\ \dfrac{}{\langle b,\sigma\rangle \Downarrow 6}}{\langle 4\times b,\sigma\rangle \Downarrow 24}}{\langle 3\times a + 4\times b,\sigma\rangle \Downarrow 39}$$

- prove that while b do s

is equivalent to if b then (s ; while b do s) else skip

- two statements s_1 and s_2 are equivalent ($s_1 \sim s_2$) if, for any two environments σ, σ',

$$\langle s_1, \sigma \rangle \Downarrow \sigma' \longleftrightarrow \langle s_2, \sigma \rangle \Downarrow \sigma'$$

biconditional / iff

- let W be "while b do S"; want to show:

$$\langle W, \sigma \rangle \Downarrow \sigma' \longleftrightarrow \langle \text{if } b \text{ then } (s ; W) \text{ else skip}, \sigma \rangle \Downarrow \sigma'$$

- prove that $\underbrace{\text{while } b \text{ do } s}_{W}$

is equivalent to if b then $(s\,;\,\underbrace{\text{while } b \text{ do } s}_{W})$ else skip

- consider WHILE-F and WHILE-T rules for W

$$\text{WHILE-F}\;\frac{\dfrac{\vdots\;(1)}{\langle b,\sigma\rangle \Downarrow \text{false}}}{\langle W,\sigma\rangle \Downarrow \sigma}
\qquad
\text{IF-F}\;\frac{\dfrac{\vdots\;(1)}{\langle b,\sigma\rangle \Downarrow \text{false}} \quad \text{SKIP}\,\dfrac{}{\langle \text{skip},\sigma\rangle \Downarrow \sigma}}{\langle \text{if } b \text{ then } (s;W) \text{ else skip},\sigma\rangle \Downarrow \sigma}$$

- prove that while b do s W

is equivalent to if b then (s ; while b do s) else skip

- consider WHILE-F and WHILE-T rules for W

$$\text{WHILE-T}\ \dfrac{\dfrac{\overset{①}{\vdots}}{\langle b, \sigma \rangle \Downarrow \text{true}} \quad \dfrac{\overset{②}{\vdots}}{\langle s, \sigma \rangle \Downarrow \sigma''} \quad \dfrac{\overset{③}{\vdots}}{\langle W, \sigma'' \rangle \Downarrow \sigma'}}{\langle W, \sigma \rangle \Downarrow \sigma'}$$

- prove that while b do s (W) is equivalent to if b then (s ; while b do s (W)) else skip

- consider WHILE-F and WHILE-T rules for W

$$
\textsf{IF-T}\;\dfrac{\dfrac{\vdots\;①}{\langle b, \sigma \rangle \Downarrow \text{true}} \qquad \textsf{SEQ}\;\dfrac{\dfrac{\vdots\;②}{\langle s, \sigma \rangle \Downarrow \sigma''} \qquad \dfrac{\vdots\;③}{\langle w, \sigma'' \rangle \Downarrow \sigma'}}{\langle s ; w, \sigma \rangle \Downarrow \sigma'}}{\langle \text{if } b \text{ then } (s ; w) \text{ else skip}, \sigma \rangle \Downarrow \sigma'}
$$

$\square$

Back to Interpreters

$\Downarrow \equiv$ eval

$\sigma \equiv$ env

i.e., Big step semantics also gives us a rigorous, concise way to specify the behavior of an interpreter/language!

e.g., dozens of pages of formal semantics vs. hundreds of English for specification of Java lang.