Our sponsors make OPLSS possible.

Diamond Sponsor

https://www.janestreet.com/

Platinum Sponsors

National Science Foundation
WHERE DISCOVERIES BEGIN

https://www.nsf.gov/

SIGPLAN

https://www.sigplan.org/

Silver Sponsors

OREGON
PROGRAMMING
LANGUAGES
SUMMER
SCHOOL

Introduction to Programming
Language Semantics

Lecture 1: Structural Operational Semantics

Some motivation

- It's hopeless to try to reason about program behavior without knowing program semantics
- Programs can be very complicated beasts — modularity, compositionality, and abstraction are essential
- Reduce as much as possible to a few formal rules that can be applied mindlessly
- Ideally the rules should be consistent and complete, and the simpler the better

Styles of Semantics

Three Styles of Semantics

- Operational: Focus on states, local, forward-moving in time
- Denotational: Focus on states, global, forward-moving in time
- Axiomatic: Focus on observations, backward-moving in time

Roadmap

- Lecture 1: Structural operational semantics
  IMP language, small-step rules, big-step rules, evaluation contexts, big-step rules as binary relations, algebraic properties
- Lecture 2: The λ-calculus
  λ-terms, safe substitution, binding and scope, α- and β-reduction, Church-Rosser theorem, data encodings, recursion, simple types, progress and preservation
- Lecture 3: Denotational semantics
  CPOs, least fixpoints, Knaster-Tarski theorem, domain equations
- Lecture 4: Axiomatic semantics, probabilistic semantics
  Weakest preconditions, semantics of Hoare logic, Dynamic logic, probabilistic semantics

A simple imperative language — IMP

Syntax specified by a BNF grammar

- Arithmetic expressions (AExp)
  $a \ ::= \ n \in \mathbb{N} \mid x \in \mathsf{Var} \mid a_1 + a_2 \mid a_1 * a_2 \mid \ldots$
- Boolean expressions (BExp)
  $b \ ::= \ \texttt{true} \mid \texttt{false} \mid a_1 \leq a_2 \mid \ldots \mid b_1 \wedge b_2 \mid \neg b \mid \ldots$
- Commands (Com)
  $c \ ::= \ x := a \mid c_1; c_2 \mid \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{while}\ b\ \texttt{do}\ c \mid \texttt{skip}$

This specifies the abstract syntax trees (ASTs) of expressions

Environments and configurations

An environment is a function $s : \text{Var} \to \mathbb{N}$

- used to evaluate variables in an expression
- Env is the set of environments

A configuration is a pair $\langle c, s \rangle$, where $c \in \text{Com}$ and $s \in \text{Env}$

- represents the current state of the computation

Small-step semantics of IMP

Small-step semantics specifies the operation of a program one step at a time

- Formally: define a small-step relation $\rightarrow_c$ on configurations
  $\langle c, s\rangle \rightarrow_c \langle c', s'\rangle$ means $\langle c, s\rangle$ goes to $\langle c', s'\rangle$ in one step
- $\stackrel{*}{\rightarrow}_c$ is the reflexive transitive closure of $\rightarrow_c$
  $\langle c, s\rangle \stackrel{*}{\rightarrow}_c \langle c', s'\rangle$ means $\langle c, s\rangle$ goes to $\langle c', s'\rangle$ in 0 or more steps
- Continue until reaching a halt configuration $\langle \texttt{skip}, s\rangle$ (if ever!)

Small-step semantics of IMP

Need auxiliary evaluation relations $\rightarrow_a$ and $\rightarrow_b$ for arithmetic and Boolean expressions

$$\rightarrow_a : (\mathsf{AExp} \times \mathsf{Env}) \rightarrow (\mathsf{AExp} \times \mathsf{Env})$$
$$\rightarrow_b : (\mathsf{BExp} \times \mathsf{Env}) \rightarrow (\mathsf{BExp} \times \mathsf{Env})$$
$$\rightarrow_c : (\mathsf{Com} \times \mathsf{Env}) \rightarrow (\mathsf{Com} \times \mathsf{Env})$$

Arithmetic and Boolean expressions

$$\frac{\langle a_1, s\rangle \to_a \langle a_1', s\rangle}{\langle a_1 + a_2, s\rangle \to_a \langle a_1' + a_2, s\rangle} \qquad \frac{\langle a_2, s\rangle \to_a \langle a_2', s\rangle}{\langle n_1 + a_2, s\rangle \to_a \langle n_1 + a_2', s\rangle}$$

$$\frac{}{\langle x, s\rangle \to_a \langle s(x), s\rangle} \qquad \frac{}{\langle n_1 + n_2, s\rangle \to_a \langle n_3, s\rangle}$$

where in the last rule, n_3 is the sum of n_1 and n_2 in $\mathbb{N}$

These rules say: If the reduction above the line can be performed, then the reduction below the line can be performed

- There is no rule that applies to $\langle n, s\rangle$ — it is irreducible
- In all other cases, exactly one rule applies
- No rule affects s
- The rules for * and Boolean operators are similar

Commands

- Assignment:

$$\frac{}{\langle x := n, s\rangle \to_c \langle \texttt{skip}, s[n/x]\rangle} \qquad \frac{\langle a, s\rangle \to_a \langle a', s'\rangle}{\langle x := a, s\rangle \to_c \langle x := a', s'\rangle}$$

- Sequential composition:

$$\frac{\langle c_0, s\rangle \to_c \langle c_0', s'\rangle}{\langle c_0 ; c_1, s\rangle \to_c \langle c_0' ; c_1, s'\rangle} \qquad \frac{}{\langle \texttt{skip} ; c_1, s\rangle \to_c \langle c_1, s\rangle}$$

- Conditional test:

$$\frac{\langle b, s\rangle \to_b \langle b', s'\rangle}{\langle \texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1, s\rangle \to_c \langle \texttt{if } b' \texttt{ then } c_0 \texttt{ else } c_1, s'\rangle}$$

$$\frac{}{\langle \texttt{if true then } c_0 \texttt{ else } c_1, s\rangle \to_c \langle c_0, s\rangle}$$

$$\frac{}{\langle \texttt{if false then } c_0 \texttt{ else } c_1, s\rangle \to_c \langle c_1, s\rangle}$$

Commands

- While loop:

$$\frac{}{\langle \texttt{while}\ b\ \texttt{do}\ c, s\rangle \to_c \langle \texttt{if}\ b\ \texttt{then}\ (c; \texttt{while}\ b\ \texttt{do}\ c)\ \texttt{else}\ \texttt{skip}, s\rangle}$$

- There is no rule for `skip`; any configuration $\langle \texttt{skip}, s\rangle$ is irreducible
- In all other cases, there is exactly one rule that applies

Example

$$
\begin{aligned}
&\langle x := 0; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [\,]\rangle \\
&\rightarrow_c \langle \texttt{skip}; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 0]\rangle \\
&\rightarrow_c \langle \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 0]\rangle \\
&\rightarrow_c \langle \texttt{if}\ x \leq 2\ \texttt{then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 0]\rangle \\
&\rightarrow_c \langle \texttt{if}\ 0 \leq 2\ \texttt{then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 0]\rangle \\
&\rightarrow_c \langle \texttt{if true then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 0]\rangle \\
&\rightarrow_c \langle x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 0]\rangle \\
&\rightarrow_c \langle x := 0 + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 0]\rangle \\
&\rightarrow_c \langle x := 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 0]\rangle \\
&\rightarrow_c \langle \texttt{skip}; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 1]\rangle \\
&\rightarrow_c \langle \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1, [x = 1]\rangle \\
&\rightarrow_c \langle \texttt{if}\ x \leq 2\ \texttt{then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 1]\rangle \\
&\rightarrow_c \ldots \\
&\rightarrow_c \langle \texttt{if}\ x \leq 2\ \texttt{then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 3]\rangle \\
&\rightarrow_c \langle \texttt{if}\ 3 \leq 2\ \texttt{then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 3]\rangle \\
&\rightarrow_c \langle \texttt{if false then}\ (x := x + 1; \texttt{while}\ x \leq 2\ \texttt{do}\ x := x + 1)\ \texttt{else skip}, [x = 3]\rangle \\
&\rightarrow_c \langle \texttt{skip}, [x = 3]\rangle
\end{aligned}
$$

Big-step semantics of IMP

Big-step semantics specifies a relation between a configuration and its final output

- Formally define a big-step relation $\downarrow_c$ between configurations and environments
  $\langle c, s\rangle \downarrow_c s'$ means $\langle c, s\rangle$ eventually halts in final configuration $\langle \texttt{skip}, s'\rangle$
- Also need auxiliary evaluation big-step relations $\downarrow_a$ and $\downarrow_b$ for arithmetic and Boolean expressions

$$\downarrow_a : (\mathsf{AExp} \times \mathsf{Env}) \to \mathbb{N}$$
$$\downarrow_b : (\mathsf{BExp} \times \mathsf{Env}) \to \{\texttt{true}, \texttt{false}\}$$
$$\downarrow_c : (\mathsf{Com} \times \mathsf{Env}) \rightharpoonup \mathsf{Env} \quad \text{partial function!}$$

Arithmetic and Boolean expressions

$$\frac{}{\langle n, s\rangle \downarrow_a n} \qquad \frac{}{\langle x, s\rangle \downarrow_a s(x)}$$

$$\frac{\langle a_1, s\rangle \downarrow_a n_1 \quad \langle a_2, s\rangle \downarrow_a n_2}{\langle a_1 + a_2, s\rangle \downarrow_a n_3}$$

where in the last rule, n_3 is the sum of n_1 and n_2 in $\mathbb{N}$

The rules for $*$, comparisons, and Boolean expressions are similar

Commands

- Skip:

$$\frac{}{\langle \texttt{skip}, s\rangle \downarrow_c s}$$

- Assignment:

$$\frac{\langle a, s\rangle \downarrow_a n}{\langle x := a, s\rangle \downarrow_c s[n/x]}$$

- Sequential composition:

$$\frac{\langle c_0, s\rangle \downarrow_c s' \quad \langle c_1, s'\rangle \downarrow_c s''}{\langle c_0\,; c_1, s\rangle \downarrow_c s''}$$

- Conditional test:

$$\frac{\langle b, s\rangle \downarrow_b \texttt{true} \quad \langle c_0, s\rangle \downarrow_c s'}{\langle \texttt{if}\ b\ \texttt{then}\ c_0\ \texttt{else}\ c_1, s\rangle \downarrow_c s'} \qquad \frac{\langle b, s\rangle \downarrow_b \texttt{false} \quad \langle c_1, s\rangle \downarrow_c s'}{\langle \texttt{if}\ b\ \texttt{then}\ c_0\ \texttt{else}\ c_1, s\rangle \downarrow_c s'}$$

Commands

- While loop:

$$\frac{\langle b, s\rangle \downarrow_b \texttt{false}}{\langle \texttt{while}\ b\ \texttt{do}\ c, s\rangle \downarrow_c s}$$

$$\frac{\langle b, s\rangle \downarrow_b \texttt{true} \qquad \langle c, s\rangle \downarrow_c s' \qquad \langle \texttt{while}\ b\ \texttt{do}\ c, s'\rangle \downarrow_c s''}{\langle \texttt{while}\ b\ \texttt{do}\ c, s\rangle \downarrow_c s''}$$

Big-step vs small-step SOS

Big-step semantics essentially specifies a recursive evaluation procedure — easy to build an interpreter from the rules

Small-step semantics gives finer view of computation — can reason about processes that are not supposed to halt

For terminating IMP programs, big- and small-step agree:

Theorem

For all commands $c \in \text{Com}$ and environments $s, s' \in \text{Env}$,

$$\langle c, s\rangle \xrightarrow{*}_c \langle \texttt{skip}, s'\rangle \;\Leftrightarrow\; \langle c, s\rangle \downarrow_c s'$$

Evaluation Contexts

The rules for SOS can be classified into two types:

- reduction rules describing actual computation steps, e.g.

$$\frac{}{\langle \texttt{if true then } c_0 \texttt{ else } c_1, s\rangle \rightarrow_c \langle c_0, s\rangle}$$

$$\frac{}{\langle \texttt{if false then } c_0 \texttt{ else } c_1, s\rangle \rightarrow_c \langle c_1, s\rangle}$$

- evaluation order rules describing which reduction to perform next, e.g.

$$\frac{\langle b, s\rangle \rightarrow_b \langle b', s'\rangle}{\langle \texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1, s\rangle \rightarrow_c \langle \texttt{if } b' \texttt{ then } c_0 \texttt{ else } c_1, s'\rangle}$$

Evaluation Contexts

The evaluation order rules, although accurate, require a lot of work for complicated expressions, e.g.

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle x, s\rangle \to_b \langle 0, s'\rangle}{\langle x \leq y, s\rangle \to_b \langle 0 \leq y, s'\rangle}}{\langle x \leq y \wedge \neg(y \leq z), s\rangle \to_b \langle 0 \leq y \wedge \neg(y \leq z), s'\rangle}}{\langle \neg(x \leq y \wedge \neg(y \leq z)), s\rangle \to_b \langle \neg(0 \leq y \wedge \neg(y \leq z)), s'\rangle}}{\langle \texttt{if}\ \neg(x \leq y \wedge \neg(y \leq z))\ \texttt{then}\ c_0\ \texttt{else}\ c_1, s\rangle \to_c \langle \texttt{if}\ \neg(0 \leq y \wedge \neg(y \leq z))\ \texttt{then}\ c_0\ \texttt{else}\ c_1, s'\rangle}$$

One would like a more succinct way of expressing evaluation order

Evaluation Contexts

- Evaluation contexts are a family of expressions with one occurrence of a special variable (called the hole, usually denoted [ ]) that say where rules can be applied
- Typically specified by a BNF grammar
- The evaluation contexts for IMP are

$$
\begin{aligned}
E_a &::= [\,]_a \mid E_a + a \mid n + E_a \mid E_a * a \mid n * E_a \\
E_b &::= [\,]_b \mid E_b \wedge b \mid \neg E_b \mid E_a \leq a \mid n \leq E_a \\
E_c &::= [\,]_c \mid \texttt{if}\ E_b\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid x := E_a \mid E_c; c
\end{aligned}
$$

- There should be exactly one evaluation context matching any unevaluated expression

Evaluation Contexts

So if $s(x) = 0$, instead of

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle x, s\rangle \to_b \langle 0, s'\rangle}{\langle x \leq y, s\rangle \to_b \langle 0 \leq y, s'\rangle}}{\langle x \leq y \wedge \neg(y \leq z), s\rangle \to_b \langle 0 \leq y \wedge \neg(y \leq z), s'\rangle}}{\langle \neg(x \leq y \wedge \neg(y \leq z)), s\rangle \to_b \langle \neg(0 \leq y \wedge \neg(y \leq z)), s'\rangle}}{\langle \texttt{if } \neg(x \leq y \wedge \neg(y \leq z)) \texttt{ then } c_0 \texttt{ else } c_1, s\rangle \to_c \langle \texttt{if } \neg(0 \leq y \wedge \neg(y \leq z)) \texttt{ then } c_0 \texttt{ else } c_1, s'\rangle}$$

we just have

$$\texttt{if } \neg(\texttt{[]} \leq y \wedge \neg(y \leq z)) \texttt{ then } c_0 \texttt{ else } c_1$$

Evaluation Contexts

Every evaluation context E represents a context rule

$$\frac{\langle e, s\rangle \to \langle e', s'\rangle}{\langle E\,[e]\,, s\rangle \to \langle E\,[e']\,, s'\rangle}$$

which says that we may apply the reduction $\langle e, s\rangle \to \langle e', s'\rangle$ in the context $E\,[e]$
Here $E\,[e]$ simply means E with e substituted for the special variable $[\,]$

Evaluation contexts vastly simplify the specification of an evaluation order

Big-step semantics as binary relations

As presented earlier, the big-step relations had types

$$\downarrow_a : (\mathsf{AExp} \times \mathsf{Env}) \to \mathbb{N}$$
$$\downarrow_b : (\mathsf{BExp} \times \mathsf{Env}) \to \{\mathtt{true}, \mathtt{false}\}$$
$$\downarrow_c : (\mathsf{Com} \times \mathsf{Env}) \rightharpoonup \mathsf{Env}$$

For $c \in \mathsf{Com}$ and $b \in \mathsf{BExp}$, define

$$[\![c]\!] = \{(s, s') \mid \langle c, s\rangle \downarrow_c s'\} \subseteq \mathsf{Env} \times \mathsf{Env}$$
$$[\![b]\!] = \{(s, s) \mid \langle b, s\rangle \downarrow_b \mathtt{true}\} \subseteq \mathsf{Env} \times \mathsf{Env}$$

$[\![c]\!]$ is the input/output relation of c, a binary relation on Env
$[\![b]\!]$ is the guard relation of b, a subset of the identity relation on Env

Some operations on binary relations

For $R, S \subseteq X \times X$,

$$R \cup S = \{(u, v) \mid (u, v) \in R \text{ or } (u, v) \in S\}$$

$$R \cdot S = \{(u, v) \mid \exists w\, (u, w) \in R,\ (w.v) \in S\}$$

the relational composition of R and S

$$\mathsf{id}_X = \{(u, u) \mid u \in X\}$$ the identity relation on X

$$R^* = \bigcup_n R^n, \text{ where } R^0 = \mathsf{id}_X \text{ and } R^{n+1} = R^n \cdot R$$

the reflexive transitive closure of R

For $B \subseteq \mathsf{id}_X$,

$$\sim B = \mathsf{id}_x \setminus B = \{(u, u) \mid (u, u) \notin B\}$$

the complement of B in id_X

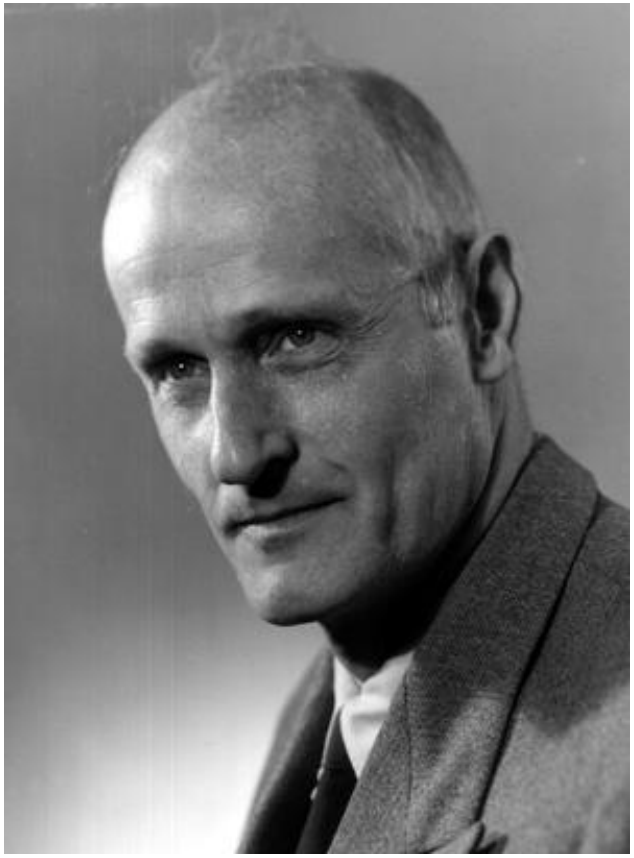
Programs as binary relations on Env

The input/output relations of programs can be defined in terms of operations on binary relations

$$
\begin{aligned}
[\![c_1; c_2]\!] &= [\![c_1]\!] \cdot [\![c_2]\!] \\
[\![\neg b]\!] &= \sim [\![b]\!] \\
[\![b_1 \vee b_2]\!] &= [\![b_1]\!] \cup [\![b_2]\!] \\
[\![\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2]\!] &= ([\![b]\!] \cdot [\![c_1]\!]) \cup (\sim [\![b]\!] \cdot [\![c_2]\!]) \\
[\![\texttt{while}\ b\ \texttt{do}\ c]\!] &= ([\![b]\!] \cdot [\![c]\!])^* \cdot \sim [\![b]\!] \\
[\![\texttt{skip}]\!] &= \mathsf{id}_{\mathsf{Env}}
\end{aligned}
$$

These are equivalent to the big-step definitions

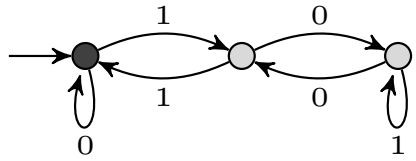
Kleene Algebra (KA)



Stephen Cole Kleene
(1909–1994)

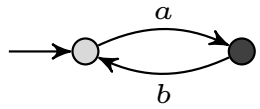
$(0 + 1(01^*0)^*1)^*$

{multiples of 3 in binary}



$(ab)^*a = a(ba)^*$

$\{a, aba, ababa, \ldots\}$



$(a + b)^* = a^*(ba^*)^*$

{all strings over $\{a, b\}$}

Axioms of Kleene algebra

Idempotent Semiring Axioms

$$p + (q + r) = (p + q) + r \qquad p(qr) = (pq)r$$
$$p + q = q + p \qquad 1p = p1 = p$$
$$p + 0 = p \qquad p0 = 0p = 0$$
$$p + p = p$$

$$p(q + r) = pq + pr \qquad a \leq b \overset{\triangle}{\iff} a + b = b$$
$$(p + q)r = pr + qr$$

Axioms for *

$$1 + pp^* \leq p^* \qquad q + px \leq x \;\Rightarrow\; p^*q \leq x$$
$$1 + p^*p \leq p^* \qquad q + xp \leq x \;\Rightarrow\; qp^* \leq x$$

Standard Model

Regular sets of strings over Σ

$$
\begin{aligned}
A + B &= A \cup B \\
AB &= \{xy \mid x \in A,\ y \in B\} \\
A^* &= \bigcup_{n \geq 0} A^n = A^0 \cup A^1 \cup A^2 \cup \cdots \\
1 &= \{\varepsilon\} \\
0 &= \varnothing
\end{aligned}
$$

This is the free KA on generators Σ

An equation is a deductive consequence of the axioms iff it holds in this model

Relational Models

Binary relations on a set X

$$
\begin{aligned}
R + S &= R \cup S \\
RS &= R \cdot S = \{(u,v) \mid \exists w\ (u,w) \in R,\ (w,v) \in S\} \\
R^* &= \textstyle\bigcup_n R^n = R^0 \cup R^1 \cup R^2 \cup \cdots \\
1 &= \mathsf{id}_X = \{(u,u) \mid u \in X\} \\
0 &= \varnothing
\end{aligned}
$$

KA is sound and complete for relational models

An equation is a deductive consequence of the axioms iff it holds in all relational models

Kleene Algebra with Tests (KAT)

$(K, B, +, \cdot, ^*, ^-, 0, 1), \quad B \subseteq K$

- $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra
- $(B, +, \cdot, ^-, 0, 1)$ is a Boolean algebra
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, 0, 1)$

- $p, q, r, \ldots$ range over K
- $a, b, c, \ldots$ range over B

Program verification with KAT

Modeling IMP

$p\,;q = pq \quad \texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q = bp + \bar{b}q \quad \texttt{while}\ b\ \texttt{do}\ p = (bp)^*\bar{b}$

KAT Subsumes Hoare Logic

$$\{b\}\,p\,\{c\} \;\Leftrightarrow\; bp \leq pc \;\Leftrightarrow\; bp = bpc \;\Leftrightarrow\; bp\bar{c} = 0$$

The Hoare while rules become universal Horn formulas — e.g.

$$\frac{\{bc\}\,p\,\{c\}}{\{c\}\ \texttt{while}\ b\ \texttt{do}\ p\ \{\bar{b}c\}}$$

becomes

$$bcp \leq pc \;\Rightarrow\; c(bp)^*\bar{b} \leq (bp)^*\bar{b}\bar{b}c$$

Example

The following two programs are equivalent:

while b do {
  p ;
  while c do q
}

if b then {
  p ;
  while $b + c$ do {
    if c then q else p
  }
} else skip

Expressed in KAT:

$$(bp(cq)^*\overline{c})^*\overline{b} = bp((b+c)(cq+\overline{c}p))^*\overline{b+c} + \overline{b}$$

First-order reasoning — SKAT

$$
\begin{array}{lcll}
x := s\,; y := t & = & y := t[s/x]\,; x := s & y \notin \mathsf{FV}(s) \\
x := s\,; y := t & = & x := s\,; y := t[s/x] & x \notin \mathsf{FV}(s) \\
x := s\,; x := t & = & x := t[s/x] & \\
\varphi[t/x]\,; x := t & = & x := t\,; \varphi &
\end{array}
$$

where in the first two, x and y are distinct variables and $\mathsf{FV}(s)$ denotes the set of free variables of s

Special cases of the first and last are

$$
\begin{array}{lcll}
x := s\,; y := t & = & y := t\,; x := s & x \notin \mathsf{FV}(t),\ y \notin \mathsf{FV}(s) \\
\varphi\,; x := t & = & x := t\,; \varphi & x \notin \mathsf{FV}(\varphi)
\end{array}
$$

First-order reasoning — SKAT

The assignment axiom of Hoare Logic is

$$\{\varphi[t/x]\}\, x := t\, \{\varphi\}$$

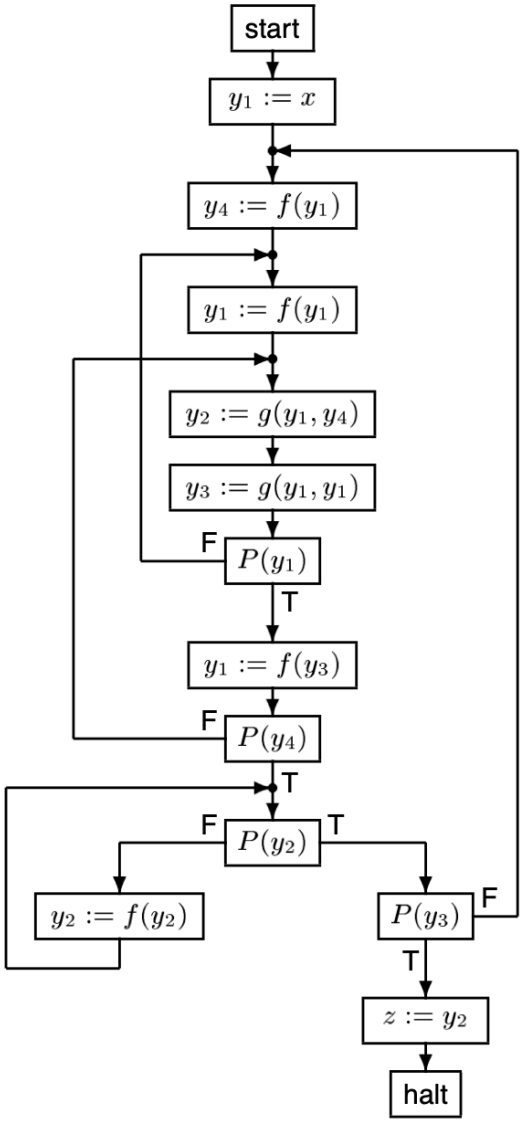
which is represented in SKAT by any of

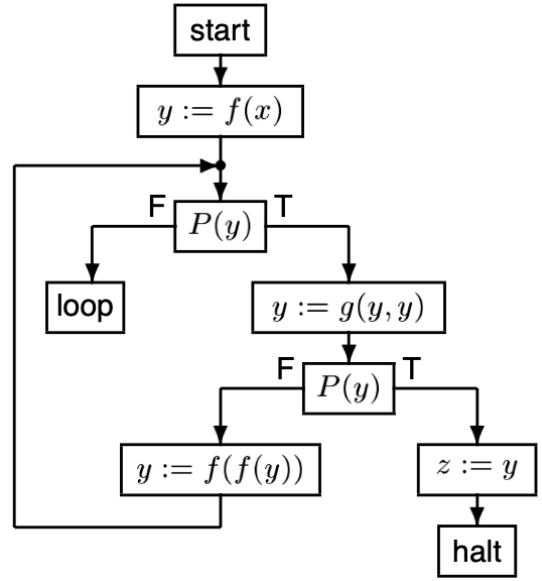
$$\begin{aligned} \varphi[t/x]\,;x := t\,;\varphi &= \varphi[t/x]\,;x := t \\ \varphi[t/x]\,;x := t\,;\neg\varphi &= 0 \\ \varphi[t/x]\,;x := t &\leq x := t\,;\varphi \end{aligned}$$

The equation $\varphi[t/x]\,;x := t \;=\; x := t\,;\varphi$ is equivalent to

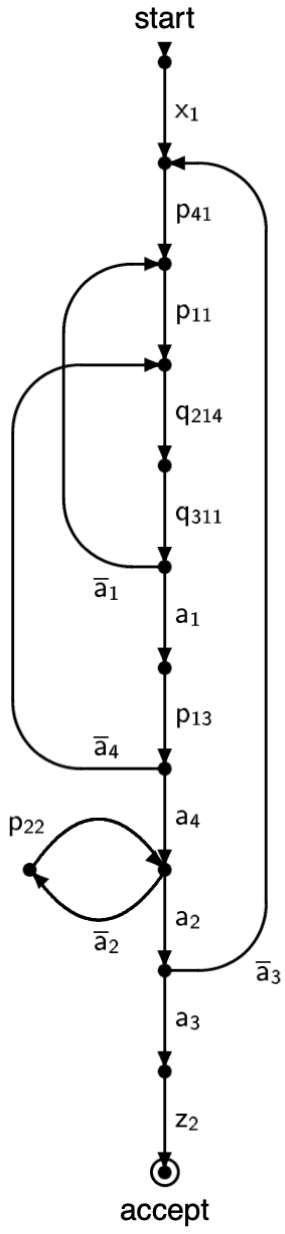
$$\{\varphi[t/x]\}\, x := t\, \{\varphi\} \text{ and } \{\neg\varphi[t/x]\}\, x := t\, \{\neg\varphi\}$$

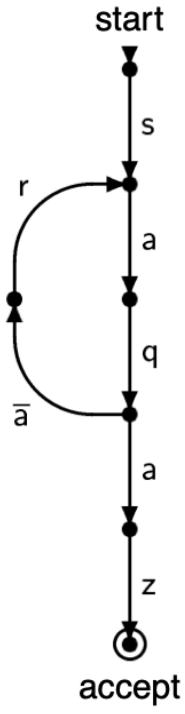
First-order reasoning — SKAT





First-order reasoning — SKAT





Next time — the λ-calculus!

- Lecture 1: Structural operational semantics
  IMP language, small-step rules, big-step rules, evaluation contexts, big-step rules as binary relations, algebraic properties
- Lecture 2: The λ-calculus
  λ-terms, safe substitution, binding and scope, α- and β-reduction, Church-Rosser theorem, data encodings, recursion, simple types, progress and preservation
- Lecture 3: Denotational semantics
  CPOs, least fixpoints, Knaster-Tarski theorem, domain equations
- Lecture 4: Axiomatic semantics, probabilistic semantics
  Weakest preconditions, semantics of Hoare logic, Dynamic logic, probabilistic semantics

Our sponsors make OPLSS possible.

Diamond Sponsor

https://www.janestreet.com/

Platinum Sponsors

National Science Foundation
WHERE DISCOVERIES BEGIN

https://www.nsf.gov/

SIGPLAN

https://www.sigplan.org/

Silver Sponsors