CS2104 Prog. Lang. Concepts

Operational Semantics

Abhik Roychoudhury

Department of Computer Science

National University of Singapore

Organization

- An imperative language IMP
- Formalizing the syntax of IMP
- Meaning of arithmetic expressions
- Meaning of Boolean Expressions
- Meaning of statements/commands
- Discussion on Axiomatic Semantics

IMP : a toy imperative language

- **IMP** is an imperative language in the style of PASCAL or C (even though some of the syntax may be different)
- The language contains arithmetic and boolean expressions as well as if-then-else, while statements.
- The syntax of the program will be described by BNF grammars.

IMP : a toy imperative language

- During execution of IMP program, the state of execution will be captured by the values of program variables.
- Operational semantics will be described by rules which specify how
  - Expressions in IMP pgm. are evaluated
  - Statements in IMP pgm. change the state

Syntax of IMP

- non-negative integers N
- truth values $T = \{\text{true, false}\}$
- variables V
- arithmetic expressions A
- boolean expressions B
- statements/commands C

Syntax of N

- Regular Grammar:

$$N \rightarrow DN \mid D$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

- Regular Expression

$$(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)^*$$

or

$$(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)^+$$

Syntax of T

- Regular Grammar

$$T \rightarrow true \mid false$$

- Regular Expression

$$(true|false)$$

Syntax of V

- Regular Grammar:

$$
\begin{aligned}
V &\rightarrow LV \mid DV \mid L \\
D &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
L &\rightarrow a \mid b \mid \ldots \mid z
\end{aligned}
$$

- Construct the equivalent regular expression.

Syntax of Arithmetic expressions A

$$
\begin{aligned}
A &\rightarrow N \\
A &\rightarrow V \\
A &\rightarrow A + A \\
A &\rightarrow A * A
\end{aligned}
$$

This grammar is ambiguous but an equivalent unambiguous grammar of arithmetic expressions can be easily constructed. Try it ?

Syntax of Boolean expressions B

$$
\begin{aligned}
B &\rightarrow T \\
B &\rightarrow A = A \\
B &\rightarrow A \leq A \\
B &\rightarrow \neg B \\
B &\rightarrow B \wedge B
\end{aligned}
$$

This grammar is ambiguous but an equivalent unambiguous grammar of boolean expressions can be easily constructed. Try it ?

Syntax of Commands C

$$C \rightarrow skip$$

$$C \rightarrow V := A$$

$$C \rightarrow C; C$$

$$C \rightarrow if\ B\ then\ C\ else\ C$$

$$C \rightarrow while\ B\ do\ C$$

Is there ambiguity in the above grammar ?

Execution model

- Operational semantics of IMP describes how programs in that language are excuted.
- To describe this, it needs to assume an underlying execution model.
- The execution model could be thought as a state machine although not necessarily a finite state machine.

States of Execution Model

- Each state of the execution model is a unique assignment of values to variables.
- Thus, if $\{a, b\}$ are the only variables in an IMP program, then each of the following are states in the execution model
  - a = 0, b = 0
  - a = 0, b = 1
  - a = 0, b = 2
  - ...
  - a = 1, b = 0
  - ...

Operational Semantics

Operational Semantics for the IMP language will give rules to describe the following:

Given a state s

- how to evaluate arithmetic expressions
- how to evaluate boolean expressions
- how the commands can alter s to a new state s'

Meaning of Arith. Expressions A - (1)

- Numbers: $\langle n, s\rangle \equiv n$
  Number n in any state s evaluates to n
  e.g. $\langle 0, s\rangle \equiv 0$, $\langle 1, s\rangle \equiv 1$
- Variables: $\langle X, s\rangle \equiv s(X)$
  Variable X in state s evaluates to value of X in s.
  e.g. $\langle a, (a = 5, b = 20)\rangle \equiv 5$
  $\langle b, (a = 5, b = 20)\rangle \equiv 27$

Meaning of Arith. Expressions A - (2)

- Sums:

$$\frac{\langle a_0, s\rangle \equiv n_0 \quad \langle a_1, s\rangle \equiv n_1}{\langle a_0 + a_1, s\rangle \equiv n}$$ where n is the sum of n_0 and n_1

e.g. $\langle a + b,\ (a = 5, b = 20)\rangle \equiv 25$

- Products:

$$\frac{\langle a_0, s\rangle \equiv n_0 \quad \langle a_1, s\rangle \equiv n_1}{\langle a_0 * a_1, s\rangle \equiv n}$$ where n is the product of n_0 and n_1

e.g. $\langle a * b,\ (a = 5, b = 20)\rangle \equiv 100$

Example arith. expr. evaluation

Evaluating meaning of a complicated arith. expr. will require

- several application of the above rules
- operator precedence

$$\dfrac{\dfrac{\langle a,\ (a=5,b=20)\rangle \equiv 5 \quad \langle b,\ (a=5,b=20)\rangle \equiv 20}{\langle a*b,\ (a=5,b=20)\rangle \equiv 100} \qquad \langle b,\ (a=5,b=20)\rangle \equiv 20}{\langle a*b+b,\ (a=5,b=20)\rangle \equiv 120}$$

Syntax of Boolean expressions B - Recap

$$
\begin{aligned}
B &\rightarrow T \\
B &\rightarrow A = A \\
B &\rightarrow A \leq A \\
B &\rightarrow \neg B \\
B &\rightarrow B \wedge B
\end{aligned}
$$

Meaning of Boolean Expressions B - (1)

- $\langle true, s\rangle \equiv true$
- $\langle false, s\rangle \equiv false$
- Equality Check

$$\frac{\langle a_0, s\rangle \equiv n_0 \quad \langle a_1, s\rangle \equiv n_1}{\langle a_0 = a_1, s\rangle \equiv true}$$ where n_0 and n_1 are equal

$$\frac{\langle a_0, s\rangle \equiv n_0 \quad \langle a_1, s\rangle \equiv n_1}{\langle a_0 = a_1, s\rangle \equiv false}$$ where n_0 and n_1 are unequal

Meaning of Boolean Expressions B - (2)

- LEQ check

$$\frac{\langle a_0, s\rangle \equiv n_0 \quad \langle a_1, s\rangle \equiv n_1}{\langle a_0 \leq a_1, s\rangle \equiv true}$$ where n_0 is l.e.q. to n_1

$$\frac{\langle a_0, s\rangle \equiv n_0 \quad \langle a_1, s\rangle \equiv n_1}{\langle a_0 \leq a_1, s\rangle \equiv false}$$ where n_0 is greater than n_1

Meaning of Boolean Expressions B - (3)

- Negation

$$\frac{\langle b, s\rangle \equiv true}{\langle \neg b, s\rangle \equiv false} \qquad \frac{\langle b, s\rangle \equiv false}{\langle \neg b, s\rangle \equiv true}$$

- Conjunction

$$\frac{\langle b_0, s\rangle \equiv t_0 \quad \langle b_1, s\rangle \equiv t_1}{\langle b_0 \wedge b_1, s\rangle \equiv t}$$ where t is logical conjunction of t_0, t_1

Example of Boolean Expr. meaning

$$\frac{\dfrac{\langle a,\ (a=5,b=6)\rangle \equiv 5 \quad \langle b,\ (a=5,b=6)\rangle \equiv 6}{\langle a=b,\ (a=5,b=6)\rangle \equiv false}}{\langle \neg a=b,\ (a=5,b=6)\rangle \equiv true}$$

$$\frac{\dfrac{\langle a,\ s\rangle \equiv 5 \quad \langle b,\ s\rangle \equiv 6}{\langle a \leq b,\ s\rangle \equiv true} \qquad \dfrac{\langle a,\ s\rangle \equiv 5 \quad \langle b,\ s\rangle \equiv 6}{\langle a=b,\ s\rangle \equiv false}}{\langle a \leq b \wedge a=b,\ s\rangle \equiv false}$$

where s is the state $(a=5,b=6)$

Meaning of Expressions

- Expressions evaluate to values in a given state.
- Therefore, the meaning of expressions are given by values.
  - boolean values for boolean expressions
  - numbers for arithmetic expressions
- Using the meaning of expressions, we can assign meaning to commands.

Meaning of Commands

- Execution of commands leads to a change of program state.
- Therefore the meaning of a command c is: If c is executed in some state s, how does it change s to s'.

$$\langle c, s \rangle \rightarrow s'$$

Syntax of Commands C - recap

$$C \rightarrow skip$$

$$C \rightarrow V := A$$

$$C \rightarrow C; C$$

$$C \rightarrow if\ B\ then\ C\ else\ C$$

$$C \rightarrow while\ B\ do\ C$$

Rules for commands - (1)

- Skip

$$\langle skip, s\rangle \rightarrow s$$

- Sequencing

$$\frac{\langle c_0, s\rangle \rightarrow s_{int} \quad \langle c_1, s_{int}\rangle \rightarrow s'}{\langle c_0; c_1, s\rangle \rightarrow s'}$$

Rules for commands - (2)

- Assignment

$$\frac{\langle a, s\rangle \equiv n}{\langle X := a, s\rangle \rightarrow s[X = n]}$$

where $s[X = n]$ is a state which is same as state s, except that the value of varible X in $s[X = n]$ is n.

Thus:

$(a = 5, b = 20, c = 2)[a = 7]$ is the state $(a = 7, b = 20, c = 2)$

$(a = 5, b = 20, c = 2)[a = 5]$ is the state $(a = 5, b = 20, c = 2)$

Rules for commands - (2)

- If-then-else

$$\frac{\langle b, s\rangle \equiv true \quad \langle c_0, s\rangle \rightarrow s'}{\langle if\ b\ then\ c_0\ else\ c_1, s\rangle \rightarrow s'}$$

$$\frac{\langle b, s\rangle \equiv false \quad \langle c_1, s\rangle \rightarrow s'}{\langle if\ b\ then\ c_0\ else\ c_1, s\rangle \rightarrow s'}$$

Rules for commands - (3)

- While

$$\frac{\langle b, s\rangle \equiv false}{\langle while\ b\ do\ c, s\rangle \rightarrow s}$$

$$\frac{\langle b, s\rangle \equiv true \quad \langle c, s\rangle \rightarrow s_{int} \quad \langle while\ b\ do\ c, s_{int}\rangle \rightarrow s'}{\langle while\ b\ do\ c, s\rangle \rightarrow s'}$$

Summary of rules

- The meaning of each commands specifies how an execution of the command chnages state.
- Roughly speaking, this is done by simulating the execution of the commands.
- For example, the rule for while essentially unfolds the iterations of the while loop.

What we did consider

- We studied the syntax and semantics of a toy language which models many of the language features we are comfortable with.
- Our language IMP contains:
  - program variables
  - arithmetic and boolean expressions
  - choice (if-then-else)
  - loops (while)
- Studying the formal semantics of the language features allows:
  - pinpointing the precise meaning of a program
  - reasoning about programs

. . . and what we did not

- IMP does not contain many language features we are comfortable with. For example:
  - different data types (not just integers)
  - arrays of data types
  - pointers
  - objects
  - procedures/functions
  - recursion
- Formally reasoning about programs with all these features is still an active topic of research.

AXIOMATIC SEMANTICS

- Operational semantics lets us understand the meaning of a program.
- Axiomatic semantics allows us to also prove correctness of programs.
- The central idea in axiomatic semantics is that of an assertion: some property of program state at a particular *control location*

Assertions: Example

{true }
if (a >= b)
    m := a ;
else
    m := b
{ m = max(a,b) }

If we can prove the assertions above, we have proved correctness of the above code fragment !!

Hoare Triple

- A tool for reasoning about code fragments.
- Of the form {Pre} C {Post}
- Pre and Post are assertions. C is a program.
- Proving such a Hoare triple amounts to proving
  - if we start in a state where Pre holds
  - then execution of C in such a state produces
  - a state where Post holds
  - provided the program C terminates
- Previous slide contained a triple for the *max* program.

Proving Hoare Triples

- Proving correctness of a program amounts to setting appropriate pre- and post-conditions and then proving the corresponding Hoare Triple.
- To prove a Hoare Triple, we will use certain rules, based on the structure of the program under question.
- These rules are stated in the same manner as operational semantics rules.

Rule for if-then-else

$$\frac{\{P \wedge B\}c_0\{Q\} \quad \{P \wedge \neg B\}c_1\{Q\}}{\{P\} \text{ if B then } c_0 \text{ else } c_1 \ \{Q\}}$$

$$\frac{\{a >= b\}m := a\{m = max(a,b)\} \quad \{a < b\}m := b\{m = max(a,b)\}}{\{true\} \text{ if } a >= b \text{ then } m := a \text{ else } m := b \ \{m = max(a,b)\}}$$

Now prove $\{a >= b\}m := a\{m = max(a,b)\}$ and
$\{a < b\}m := b\{m = max(a,b)\}$

Rule for assignment

$$\frac{true}{\{Q[x\backslash e]\}x := e\{Q\}}$$

We need to prove $\{a >= b\}m := a\{m = max(a, b)\}$

Using the above rule we can only prove:
$\{a = max(a, b)\}m := a\{m = max(a, b)\}$

Rule of Consequence

$$\frac{\mathsf{P} \Rightarrow \mathsf{P'} \quad \{\mathsf{P'}\}\ \mathsf{C}\ \{\mathsf{Q'}\} \quad \mathsf{Q'} \Rightarrow \mathsf{Q}}{\{\mathsf{P}\}\ \mathsf{C}\ \{\mathsf{Q}\}}$$

Set $P = (a >= b)$,

$P' = (a = max(a, b))$

$Q = Q' = (m = max(a, b))$

Then $P \Rightarrow P'$

Therefore $\{a >= b\}m := a\{m = max(a, b)\}$ holds.

Other rules

- Rule of sequences

$$\frac{\{\mathrm{P}\}\ \mathrm{C1}\ \{\mathrm{R}\} \qquad \{\mathrm{R}\}\ \mathrm{C2}\ \{\mathrm{Q}\}}{\{\mathrm{P}\}\ \mathrm{C1}\ ;\ \mathrm{C2}\ \{\mathrm{Q}\}}$$

- Rule of loop invariants

$$\frac{\{\mathrm{Cond} \wedge \mathrm{P}\}\ \mathrm{Body}\ \{\mathrm{P}\}}{\{\mathrm{P}\}\ \mathrm{while\ Cond\ do\ Body}\{\neg\ \mathrm{Cond} \wedge \mathrm{P}\}}$$

P is a loop invariant : true after every iteration of the loop.

Exercise

Reading : Chapter 3.3 of textbook (for axiomatic semantics)

Study the correctness of the factorial function to get yourself familiar with reasoning about while loops. More about this to follow in the tutorial.