CSE 307: Principles of Programming Languages
Statements and Control Flow

R. Sekar

Topics

1. If-Then-Else

Control Statements

- Structured Control Statements:
- Case Statements:
  - Implementation using if-then-else
  - Understand semantics in terms of the semantics of simple constructs
  - actual implementation in a compiler
- Loops
  - while, repeat, for
Section 1

If-Then-Else

If-then-else. It is in two forms:

- if cond then s1 else s2
- if cond then s1

evaluate condition: if and only if evaluates to true, then evaluate s1 otherwise evaluate s2.

Dangling else problem: if c1 then if c2 then s1 else s2

may be interpreted as:

```
if c1 then
  if c2 then s1
else s2
```

Or

```
if c1 then
  if c2 then s1 else s2
```

This ambiguity can be avoided by bracketing syntax:

- if cond then s1 fi
- if cond then s1 else s2 fi

The above intended statements can be written as:

```
if c1 then
  if c2 then s1 else s2 fi
fi
```

Or

```
if c1 then
  if c2 then s1 fi
else s2 fi
```

Another way to avoid ambiguity is to use: associate else with closest “if” that doesn’t have “else”. This is used in most programming languages (C, C++ etc)
Case Statement

- Case statement
  ```python
  switch(<expr>){
      case <value> :
      case <value> :
      ...
      default :
  }
  ```
- Evaluate “<expr>” to get value v. Evaluate the case that corresponds to v.
- Restriction:
  - “<value>” has to be a constant of an original type e.g., int, enum
  - Why?

Implementation of case statement

- Naive algorithm:
  - Sequential comparison of value v with case labels.
  - This is simple, but inefficient. It involves O(N) comparisons
  ```python
  switch(e){
  case 0:s0;
  case 1:s1;
  case 2:s2;
  case 3:s3;
  }
  ```
  can be translated as:
  ```python
  v = e;
  if (v==0) s0;
  else if (v == 1) s1;
  else if (v == 2) s2;
  else if (v == 3) s3;
  ```

Implementation of case statement (Continued)

- Binary search:
  - O(log N) comparisons, a drastic improvement
  - over sequential search for large N.
- Using this, the above case statement can be translated as
  ```python
  v = e;
  if (v<=1)
    if (v==0) s0;
    else if (v == 1) s1;
  else if (v>2)
    if (v==3) s3;
  ```
Implementation of case statement (Continued)

- Another technique is to use hash tables.
- This maps the value v to the case label that corresponds to the value v.
- This takes constant time (expected).

Control Statements (contd.)

- while:
  - let s1 = while C do S
  - then it can also be written as
  - s1 = if C then {S; s1}
- repeat:
  - let s2 = repeat S until C
  - then it can also be written as
  - s2 = S; if (!C) then s2
- loop
  - let s = loop S end
  - its semantics can be understood as S; s
  - S should contain a break statement, or else it won't terminate.

For-loop

- Semantics of for (S2; C; S3) S can be specified in terms of while:
  - S2; while C do { S; S3 }
- In some languages, additional restrictions imposed to enable more efficient code
  - Value of index variable can't change loop body, and is undefined outside the loop
  - Bounds may be evaluated only once
Unstructured Control Flow

- Unstructured control transfer statements (goto) can make programs hard to understand:
  
  ```
  40: if (x > y) then goto 10
      if (x < y) then goto 20
      goto 30
  10: x = x - y
     goto 40
  20: y = y - x
     goto 40
  30: gcd = x
  ```

Unstructured Control Flow (Continued)

- Unstructured control transfer statements (goto) can make programs hard to understand:
  
  ```
  40: if (x > y) then goto 10
      if (x < y) then goto 20
      goto 30
  10: x = x - y
     goto 40
  20: y = y - x
     goto 40
  30: gcd = x
  ```

- Equivalent program with structured control statements is easier to understand:
  
  ```
  while (x!=y) {
      if (x > y) then x=x-y
      else y=y-x
  }
  ```

Control Statements (contd.)

- goto should be used in rare circumstances
  - e.g., error handling.
- Java doesn't have goto. It uses labeled break instead:
  
  ```
  L1: for ( ... ) {
      while (...) {
          ...
          break L1
      }
  }
  ```

- break L1 causes exit from loop labeled with L1
Control Statements (contd.)

- Restrictions in use of goto:
  - jumps across procedures
  - jumps from outer blocks to inner blocks or unrelated blocks

```c
goto l1;
if (...) then {
  int x;
  x = 5;
  l1: y = x*x;
}
```

- Jumps from inner to outer blocks are permitted.

Statements

- What does the statement `y = x + 1;` do?
- The effect of a statement is to change the store.
- `eval_stmt: stmt * environment * store -> store`
- We will use a function `update_store` to change the store:
  ```
  update_store(s, l, v) gives a new store sn which is identical to s except that location l in sn contains value v.
  ```

Evaluating statements: The Program

```c
eval_stmt(stmt, env, store) =
  match stmt with
  | Assign(x, e) ->
    let l = binding_of(env, x)
    and v = eval_expr(e, env, store)
    in update_store(store, l, Intval(v))
  | If(c, s1, s2) ->
    if (eval_cond(c, env, store))
    then eval_stmt(s1, env, store)
    else eval_stmt(s2, env, store)
  | While(c, s) ->
    if (eval_cond(c, env, store))
    then let store' = eval_stmt(s, env, store)
       in eval_stmt(While(c, s), env, store')
    else store
...
```