316 10-13-09

Operational Semantics

- operational semantics by intermediate code and virtual machine
  design a simple, clear virtual machine and its assembly-like intermediate code. This is presumed to be fully understood

$$\begin{tikzcd}
\boxed{\text{High-level language statement } S} \arrow[r, "\text{translate}", "\text{compile}"'] & \boxed{\text{Intermediate code}} \\
 & \text{Semantics of } S \text{ is given by step-by-step operation of this code} \arrow[u]
\end{tikzcd}$$

Our intermediate code:

goto <label> – unconditional goto to level
if (<var> = <bool>) goto <label>
<bool> = 0 (false) or 1 (true)

1-branch conditional $\boxed{\text{if } (B)\ S}$

```
    code to evaluate B
    b = result of above evaluation
    if (b=0) goto Out
    code to execute S
Out:
```

2-branch conditional $\boxed{\text{if } (B)\ S_1 \text{ else } S_2}$

```
    code to evaluate B
    b = result of above evaluation
    if (b=0) goto Else
    code to execute S1
    goto Out
Else: code to execute S2
Out:
```

Alternative:

```
    if (b=1) goto L1
    code to execute S2
    goto Out
L1: code to execute S1
Out:
```

- Switch statements

  Switch (E) { $L1: S_1 \ldots\ldots Ln: S_n;\ \overset{\text{default case}}{\boxed{S_d}}$ }

① Fall-through semantics (C++, Java)

```
    code to evaluate E
    e = result of above evaluation
    if(e = L1) goto A1
    if(e = L2) goto A2
        ⋮
    if(e = Ln) goto An
    goto Ad
A1: code to execute S1; break;
A2: code to execute S2; break;
An:                 Sn; break;
Ad:                 Sd; break;
```

② Exclusive-case semantics (Pascal / Ada)

```
 Same code
A1: code to execute S1
      goto Out
A2: code to execute S2
      goto out
An: code to execute Sn
Ad: code to execute Sd
Out:
```

$\boxed{\text{while (B) S}}$	$\boxed{\text{do S while(B)}}$
Loop: code to execute B	Loop: code to execute S
b = result of above evaluation	code to evaluate B
if(b = 0) goto out	b = result of above evaluation
code to execute S	if(b = 1) goto Loop
goto Loop	
Out:	

$\boxed{\text{for(init\_statement; B; incr\_statement) S'}}$

```
    code to execute init_statement
Loop: code to evaluate B
        b = result of above evaluation
        if (b=0) goto Out
        code to execute S'
        code to execute incr_statement
        goto Loop
Out:
```

$\boxed{\text{break}}$

```
{
  ⋮ ] S1          code to execute S1
  break;          goto Out
  ⋮ ] S2          code to execute S2
}                 Out:
```

Let this be the immediately surrounding block

if(B₁) { while (B₂) S₁ } else S₂

Q = provide intermediate-code semantics

A =

```
    code to execute B1
    b1 = result of above evaluation
    if (b1 = 0) goto Else
Loop: code to evaluate B2
        b2 = result of above evaluation
        if (b2 = 0) goto Out
        code to execute S1          → inefficient and redundant
        goto Loop
Out: goto OutIF
Else: code to execute S2
OutIF:
```

```
while (B₁){if (B₂) S₁ else S₂}
Loop: code to evaluate B₁
      b1 = result of above evaluation
      if(b1=0) goto Out
      code to evaluate B₂
      b₂ = result of above evaluation
      if (b₂=0) goto Else
      code to execute S₁
      goto OutIF
Else: code to execute S₂
OutIF: goto Loop
Out:
```

Redundant and inefficient

```
switch (x)
{ L1: {if(B) S₁; break;}
  L₂: {S₂; break;}
  Ld: Sd;
}
```

```
if( x=L1) goto A1
if(x=L₂) goto A₂
goto Ad
A1: code to evaluate B
    b=result of above evaluation
    if(b=0) goto Out 1
    code to execute S1
Out1: goto Out
A2: code to execute S₂
    goto Out
Ad: code to execute Sd
Out :
```