COMP 181

Lecture 15

Code generation II

October 26, 2006

Prelude

- Who are these people?
  - Pierre and Pam Omidyar
- And we care because….?
  - Pierre Omidyar is the founder of eBay
  - Donated $100 million to Tufts
- But there are strings…
  - The money must be invested in ***microfinance***
- What is microfinance?
  - Small business loans, typically given to individuals in developing countries
  - Average: $600 (as low as $40)

The person who invented microfinance just got the Nobel prize in economics

Conference report

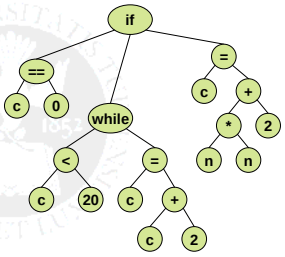
- Architectural Support for Programming Languages and Operating Systems
  - Mix of three areas
- Highlights:
  - Spatial scheduling for tiled architectures
  - GPU programming
  - Automatic generation of peephole superoptimizers
  - Combinatorial sketching for finite programs

Code generation

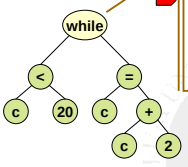
- Overview
  - Walk AST, generate code for each construct
  - Nested constructs:
    - Call generate on children as necessary
    - Use registers to pass values up tree
- Key: order is important
  - Emitted code goes into central list
  - Bottom-up post-order traversal

Example

```
if (c == 0) {
  while (c < 20) {
    c = c + 2;
  }
}
else
  c = n * n + 2;
```



Example

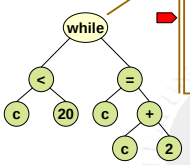



```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

Code

```
L1:
```

Example

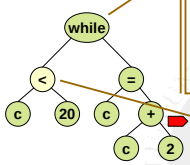



```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

Code

```
L1:
```


Example



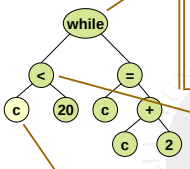

```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
t1 = generate(expr1)
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```

Code

```
L1:
```


Example




```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

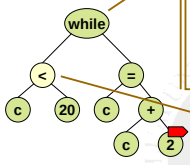
```
t1 = generate(expr1)
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```

```
r = new_temp() = R0
emit( r = load v )
return r
```

Code

```
L1:
R0 = load c
```


Example



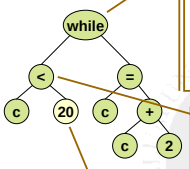

```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
t1 = generate(expr1)=R0
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```

Code

```
L1:
R0 = load c
```


Example




```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

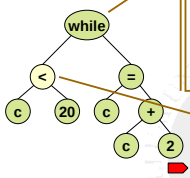
```
t1 = generate(expr1)=R0
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```

```
r = new_temp() = R1
emit( r = 20 )
return r
```

Code

```
L1:
R0 = load c
R1 = 20
```


Example



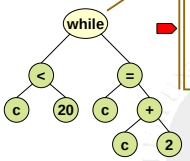

```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
t1 = generate(expr1)=R0
t2 = generate(expr2)=R1
r = new_temp() = R2
emit( r = t1 op t2 )
return r
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
```

Example

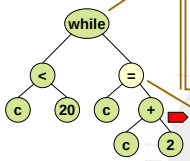



```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
```


Example



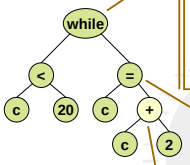

```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
r = generate(expr2)
l = lgenerate(expr2)
emit( store *l = r )
return r
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
```


Example




```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

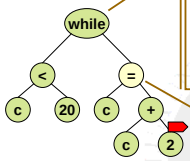
```
r = generate(expr2)
l = lgenerate(expr2)
emit( store *l = r )
return r
```

```
t1 = generate(expr1)=R3
t2 = generate(expr2)=R4
r = new_temp() = R5
emit( r = t1 op t2 )
return r
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
```


Example



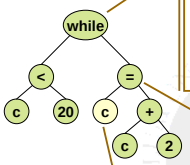

```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
r = generate(expr2)=R5
l = lgenerate(expr2)
emit( store *l = r )
return r
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
```


Example




```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
r = generate(expr2)=R5
l = lgenerate(expr2)
emit( store *l = r )
return r
```

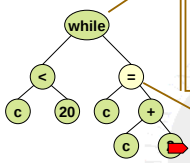
```
r = new_temp()= R6
emit( r = & v )
return r
```

Something like:
R6 = base + offset

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
```


Example



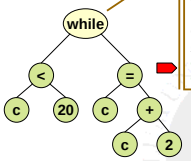

```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

```
r = generate(expr2)=R5
l = lgenerate(expr2)=R6
emit( store *l = r )
return r
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
```

Example

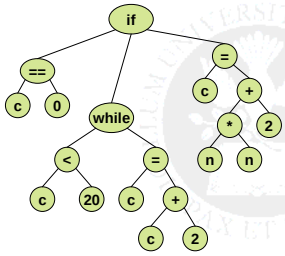



```
E = new_label() = L0
T = new_label() = L1
emit( T: )
t = generate(expr)=R2
emit( ifnot_goto t, E )
generate(statement)
emit( goto T )
emit( E: )
```

Code

```
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
```


Example



Code

```
R7 = load c
R8 = 0
R9 = R7 == R8
not_goto R9,L3
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
goto L4
L3:
 . . .
L4:
```


Nesting

while (c<20)
c = c + 2

c < 20

c = c + 2

c + 2

Code

```
R7 = load c
R8 = 0
R9 = R7 == R8
not_goto R9,L3
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
goto L4
L3:
 . . .
L4:
```


Registers only

- Assume all variables are in registers
  - Add loads and stores later
  - Change ***generate***(v)

```
r = new_temp()
emit( r = load v )
return r
```

Code

```
R7 = load c
R8 = 0
R9 = R7 == R8
not_goto R9,L3
L1:
R0 = load c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = load c
R4 = 2
R5 = R3 + R2
R6 = & c
store [R6]=R5
goto L1
L0:
goto L4
L3:
 . . .
L4:
```


Registers only

- Proliferation of registers and labels

Many CPUs have a fast c == 0 test

Can use accumulators: c = c + 2

Label leads to another goto; may have multiple labels

Code

```
R7 = c
R8 = 0
R9 = R7 == R8
not_goto R9,L3
L1:
R0 = c
R1 = 20
R2 = R0 < R1
not_goto R2,L0
R3 = c
R4 = 2
R5 = R3 + R2
c = R5
goto L1
L0:
goto L4
L3:
 . . .
L4:
```


Efficient lowering

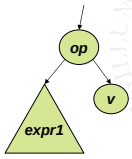
- Reduce number of temporary registers
  - Don't copy variable values unnecessarily
  - Accumulate values, when possible
  - Reuse temporaries, where possible
- Generate more efficient labels
  - Don't generate multiple adjacent labels
  - Avoid goto-label-goto

Avoiding extra copies

- Basic algorithm
  - Recursive generation traverses to leaves
  - At leaves, generate: R = v or R = c
- Improvement
  - Stop recursion one level early
  - Check to see if children are leaves
  - Don't call generate recursively on variables, constants

Avoiding copies

```
expr1 op expr2
```




```
if (expr1 is var)
  t1 = (Var) expr1
else
  t1 = generate(expr1)
if (expr2 is var)
  t2 = (Var) expr2
else
  t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```


Example

- ***Expr1*** is (a+b)
  - Not a leaf
  - Recursively generate code
  - Return temp
- ***Expr2*** is c
  - Return c
- ***Emit*** (R0 * c)

```
( a + b ) * c
```

Code

```
R0 = a + b
R1 = R0 * c
```


Use accumulation

- Idea:
  - We only need 2 registers to evaluate expr1 op expr2
  - Reuse temp assigned to one of the subexpressions

```
if (expr1 is var)
  t1 = (Var) expr1
else
  t1 = generate(expr1)
if (expr2 is var)
  t2 = (Var) expr2
else
  t2 = generate(expr2)
emit( t1 = t1 op t2 )
return t1
```


Example

- Combined:
  - Remove copies
  - Accumulate value
  - Only need one register
  - (Original version might have required as many as 6)

```
( a + b ) * c
```

Code

```
R0 = a + b
R0 = R0 * c
```


Reuse of temporaries

- Idea:

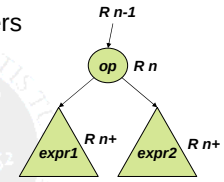
```
expr1 op expr2
```

```
t1 = generate(expr1)
t2 = generate(expr2)
r = new_temp()
emit( r = t1 op t2 )
return r
```

- Can *generate*(expr1) and *generate*(expr2) share temporaries?
  - Yes, except for t1 and t2
  - Observation: temporaries have a limited lifetime
  - Lifetime confined to a subtree

Reuse of temporaries

- Subtrees can share registers
- Algorithm:
  - Use a stack of registers
  - Start at # = 0
  - Each call to generate:
    - "Push" next number
    - Use any register > #
    - When done, "pop" back up




```
R# = expr1
R# = R# op expr2
```


Next time…

- Introduction to optimization
- Then: instruction selection