15-411: Compiler Design — Fall 2017

Recitation 1: The Backend — 08 September

Announcements

- L1 Tests due Tuesday (2000+ test cases will be released Wednesday morning)
- Written Assignment 1 due Thursday
- L1 Compiler due the following Tuesday
- Notolab is up! See your team name and submissions (info about your team name coming soon!)

Tracing Through the Backend

In this recitation, we're going to discuss an example of the processing done by the compiler backend from start to finish. Since you won't have to touch the frontend for Lab 1, we'll leave it for a future week. Here's the code and AST we'll use for the example:

```
1  int main() {
2     int x = 42;
3     int z;
4     if (x % 2 == 0) {
5        x++;
6        z = 1;
7     } else {
8        z = -1;
9     }
10    return z * x;
11 }
```

```
1  declare(x, seq(
2    assign(x, const(42)),
3    declare(z, seq(
4      if(equals(mod(x, const(2))), seq(
5        incr(x),
6        assign(z, const(1))
7      ),
8        assign(z, neg(const(1)))
9      ),
10     return(times(z, x))
11   ))
12 ))
```

Code Generation

As discussed in lecture last week, we use the "maximal munch" algorithm to generate abstract 3-address assembly from IR. For each line in the IR, we recursively pattern-match as deep as possible into each sub-expression and generate lines of assembly at each step.

Checkpoint 0

Translate the above AST into abstract 3-address assembly. NOTE: answers may differ depending on the exact rules you're using in your head to generate instructions.

Liveness Analysis

The first step in assigning registers is to determine which temps interfere with each other. Temps interfere with each other if they are both live on the same line. We use the following rules to determine which temps are live:

$$\frac{\mathsf{use}(l, x)}{\mathsf{live}(l, x)}\ K_1 \qquad \frac{\mathsf{succ}(l, l')\ \ \mathsf{live}(l', u)\ \ \neg\mathsf{def}(l, u)}{\mathsf{live}(l, u)}\ K_2$$

The judgments $\mathsf{use}(l, x)$ and $\mathsf{def}(l, x)$ are defined as you would expect according to the contents of the line l. The judgment $\mathsf{succ}(l, l')$ means that control can possibly transfer from the line l to the line l'.

Checkpoint 1

Analyze your 3-address assembly program from above to determine the liveness of each of the variables. Then draw the interference graph.

Maximum Cardinality Search

In order to color the interference graph using the greedy algorithm, we need to decide on an order in which to process the vertices. We do this using the Maximum Cardinality Search algorithm. We first assign a weight of 0 to each vertex. Then, at each step, we:

(a) Choose a vertex with maximal weight from the working set

(b) Add it to our ordering and remove it from the working set

(c) Increment the weights of all of its neighbors

This algorithm produces an ordering which is optimal for chordal graphs.

Checkpoint 2

Use Maximum Cardinality Search to generate an ordering of the vertices in the example above. Break ties by choosing the vertex that is lexicographically first.

Greedy Graph Coloring

Once we have an ordering, we can assign registers to each of the temps in our program. Ignoring pre-colored vertices, such as `%eax`, we can color the temps by assigning the lowest register that is not assigned to any of the vertex's neighbors.

Checkpoint 3

Perform Greedy Graph Coloring on the interference graph from above to assign registers `%r1, %r2, ...` to the temps in the program. Then rewrite the abstract assembly using the new registers.