Project 3: Building a Translator/Compiler

Due the last day of class (Dec 5, 2010)

In this project, you are required to perform some program analysis or optimization to the type-checked intermediate representation from project2 and then output the results of the analysis or optimization. In particular, the following provides several options that you could pick for this project.

1. **Control flow graph construction.** In particular, you need to build a control flow graph for an given input program, where each node of the CFG is a basic block, and edges in the CFG represent the flow of control between the basic blocks. The statements inside each basic block could be three-address instructions or statements/expressions in the higher level input language (i.e., internally, the high-level AST is still being used). The graph needs to be output to either standard output or a separate file. The output information must explicit enough so that the structure of the CFG is clear and it is trivial to validate its correctness. You can use the dot language (http://www.graphviz.org/doc/info/lang.html) to output the CFG, so that a nice graphical representation of the input program can be generated in a pdf or jpg file.

2. **Local redundancy elimination via value numbering.** In particular, you need to identify all basic blocks (or super blocks) in the input program, and for each of the basic blocks identified, build a value number table and use the table to eliminate redundant evaluations inside each basic block. The output of your project is the optimized code. Additionally, you may want to output what are the evaluations that you have found to be redundant (for debugging purposes).

3. **Machine code generation.** In particular, if you could generate machine assembly code that actually runs on a machine that I have access to, your project 3 is good. Here some form of instruction selection and register allocation is required.

4. **Local register allocation.** You can implement a local register allocation algorithm, which takes as parameter an arbitrary number of general purpose registers, and tries to map the collection of scalar variables in each basic block to registers. The output of your algorithm is an allocation scheme which maps each variable either to a register (an integer from 1-n, where n is the number of available registers) or to 0 (no register is available, spilling is required). Your allocation scheme must make sure that no register is shared among variables that are simultaneously alive.

If you have other ideas on how to build your project 3, let me know. If it is reasonable, I will add it to my list.

The following options, when implemented, will enable you to earn extra-credits for project3.

1. **value-numbering for superblocks (extra 20%)**. You can extend your local value numbering algorithm to support value numbering for extended basic blocks. Here the control flow graph needs to be fully built so that the superblock value numbering algorithm can be applied.

2. **Data-flow analysis (extra 30%).** Since data-flow analysis is built on top of a control flow graph, you need to first implement the control flow graph. You can choose a number of dataflow analysis problems, including live variable analysis, available expression analysis, reaching definition analysis, etc. Your output would include not only the control flow graph, and the data flow information at the entry (or exit) of each basic block.

3. **Regional/Global register allocation (extra 20%)**. You do not have to implement the graph coloring algorithm (which is worth 30% extra points if you choose to implement it). Extending your local allocation algorithm to work beyond basic blocks is sufficient. If you implment global register allocation, where global live variable analysis must be performed first, the combination is worth of 50% extra points (reg. allocation + dataflow analysis).

4. **SSA construction (extra 20%).** After the control-flow graph is built, you can convert your IR into SSA form. Your algorithm does not have to insert the minimal number of ϕ functions. However, it does need to make sure that every variable is defined only once, and each use of a variable refers to only a single definition. You can use the result of reaching definition analysis to support SSA construction, which means you are implementing two extra-credit components instead of one (which means you will be awarded 50% extra-credit if everything is done perfectly). Alternatively, you can use an ad-hoc approach which traces the use of each variable by following the control flow edges, and inserting ϕ functions whenever a join of CFG edges is encountered (here you will be awarded only 20% extra points).

5. **instruction scheduling or register allocation.** If you have picked machine code generation as your target, implementing advanced register allocation or instruction scheduling optimizations will earn you extra-credits.

Again, to implement the above capabilities, your IR could be either a high-level AST (abstract syntax tree) or a low-level three address code. The type information that you have collected for project 2 will be useful in code generation and some of the code optimizations. All three projects will be graded collectively together, so your project needs to print out the typed IR (AST or three-address code) together with any additional results that project 3 produces. The following code is again going to be used as the default input to test all your sub-projects.

```
float a[100][100], b[100][100], c[100][100];
int i, j, k;
i = 0;
while (i < 100) {
  j = 0;
  while (j < 100) {
    if (!(c[i][j] == 0))
       c[i][j] = 0;
    k = 0;
    while (k < 100) {
       c[i][j] = c[i][j] + a[i][k] * b[k][j];
       k = k + 1;
    }
    j = j + 1;
  }
  i = i + 1;
}
```

Depending what languages you chose to implement your project1, you may continue implementing project3 using C/C++/Java, POET, or any other mainstream programming language. Please document clearly your choice of languages as well as how to compile and run your program. Make sure that you make very clear which approach you have chosen to implement the project, especially if you are using an approach different from any of the options listed in this assignment. If you used any tool beyond those mentioned in this assignment, you also need to make sure that we have access to the same tool (letting us know how to download it). And if you have chosen a language different than C, you must provide your own test file which is equivalent to the one given.

Submit your project (combination of project1, project2, and project3) by packaging up your directory into a gzip file and then submitting it online at

```
http://www.cs.utsa.edu/~cs5363
```

Please submit using the username and password information I have sent you via email. Suppose your entire project is in a directory named "project3". Goto the parent directory of *project3* and type

```
> tar -cf project3.tar project3
> gzip project3.tar
```

A file *project3.tar.gz* will be generated. Test your package by copying it to another directory, say *tmp*, and then type

```
> tar -zxf project3.tar.gz
```

The package should unpack to a directory that contains your submission. After validating the package, submit it by uploading it online.

Extra-credit (20%). An extra 20% points will be awarded if all of your sub-projects (projects 1/2/3) support function definitions and function calls. In particular, you need to extend your project1 to parse a sequence of function definitions, and you need to add function calls into your syntax definitions for expressions. Then, for project 2, you need to extend your type checking to work for function calls. Finally, you project 3 (to be given) also needs to support the concept of function definitions and calls.