CS 536 Announcements for Thursday, May 5, 2022

Course evaluation – log into aefis.wisc.edu using your NetID

Last Time
- optimization overview
- peephole optimization
- loop optimizations
- copy propagation

Today
- wrap up optimization
  - copy propagation
- wrap up course / review

Optimization Review

Goal: Produce "better" code that does the "same thing" as the original code.
- better = faster code, fewer instructions

When?
- before code generation (i.e., on intermediate representation)
- after code generation (i.e., on generated machine code)

Important considerations
- performance/profitability – want to be sure optimization is "worth it"
- safety – original source code, non-optimized target code, and optimized target code all do the "same thing" / have the same "meaning"

Look at optimizations that
- are sound transformations
- recognize a behavior in a program & replace it with a "better" version
Copy propagation

Idea: Suppose we are at use $U$ of $x$ and a definition $D$ of $x$ (of the form $x = y$) reaches $U$
  • If
    1) no other definition of $x$ reaches $U$ and
    2) $y$ does not change between $D$ and $U$
  • then we can replace the use of $x$ at $U$ with $y$

Optimization opportunities
  • can create useless code (which can then be removed)
  • can create improved code
  • constant folding
  • if done before other optimizations, can improve results

To do copy propagation, we must make sure two properties hold:

Property 1) No other definition of $x$ reaches $U$
  • How? Do a reaching-definitions analysis
Example

```plaintext
x = 3;
y = 5;
p = x;

if (w*x > 9) {
    x = 4;
    z = x + w*y;
}
else {
    z = 2*y + x;
}

q = 5*p;
s = z + x;
t = s + y;
```
Copy Propagation (cont.)

Property 2) $y$ does not change between D and U
- If $y$ is a constant, then this is trivially true.

Optimization Wrap-up
Where have we been?
CS 536: Introduction to Programming Languages and Compilers

What does a programming language consist of?
- tokens
- grammar
- static semantic analysis

What else? What choices are made?
- scoping rules

- types

- parameter passing

- when do we check for things?
Where have we been?
CS 536: Introduction to Programming Languages and Compilers

How do we translate a PL into something a computer can run? i.e., compilers

- recognizing tokens
- recognizing languages
- enforcing scoping and typing rules
- developing data structures that assist our translation/representation/translation
- how do we organize and manage memory
- handling control flow within a program
  - interprocedural
  - intraprocedural

How can we make our translation better?

- intermediate representations
- IR optimizations
- MC optimizations
Course wrap-up

Covered a broad range of topics
- some formal concepts
- some practical concepts

What we skipped
- object-oriented language features
- dynamically-allocated memory management
- linking and loading
- interpreters
- register allocation
- dataflow analysis
- performance analysis
- proofs
Bring your UW Student ID

Reference material provided along with exam:
- copy of the minim grammar
- compiler class reference with selected class, methods, fields

Topic overview
Basic ideas of scanning & parsing
Symbol-table management / name analysis
- static scoping
- dynamic scoping
Type checking
Runtime storage management
- general storage layout
- activation records
- access to variables at runtime (parameters, locals, globals, non-locals)
Parameter-passing modes
Code generation
- numeric and control-flow approaches
Optimization
- goals
- optimization techniques (e.g., peephole optimization, copy propagation)
Extending
- grammar
- AST
- name analysis,
- type checking
- code generation
to handle new constructs