

Plan for today

Lvalues versus rvalues

Canonicalization or normalization

- reduce the set of cases the next phase of the compiler must handle
- make the IR more closely match the target backend
- program is still represented in same IR

Canonicalization for Tree IR

- expression evaluation order
- ExpCall nodes nested within other ExpCALL nodes will cause problems with argument registers
- StmCJUMP doesn't map to fall through that occurs in real machine ISAs

Basic blocks and traces

CS453 Lecture

Canonicalization

1

Lvalues versus Rvalues

Lvalue

- "result of an expression that can occur on the left of an assignment statement"
- examples: *(&a), b.membervar, p->membervar

Rvalue

- an expression whose result can only appear as a subexpression or on the rhs of a statement
- examples: &a, 3*4, new Foo()

Why?

- explains compiler errors you might see in the future, e.g. non-lvalue assignment
 - $x+3 = 5$
 - $4 = 5$
- emphasizes the difference between equality in math and assignment in programming languages
- think about where values are stored during the progression of a program
- this is why the lhs of a StmMOVE must be an ExpMEM or ExpTEMP
- what if we could pass around user defined types (not just references to them)?

CS453 Lecture

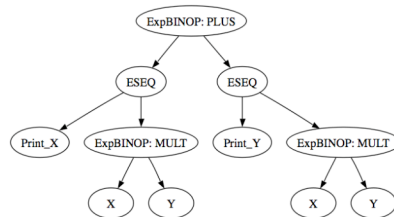
Canonicalization

2

Expression ordering

Problem:

- Can't evaluate subexpressions in any order if have ExpESEQs



Our solution:

- Don't use ExpESEQs
- Instead keep a statement list associated with each node in AST and synthesize the list while doing a bottom-up traversal on AST

CS453 Lecture

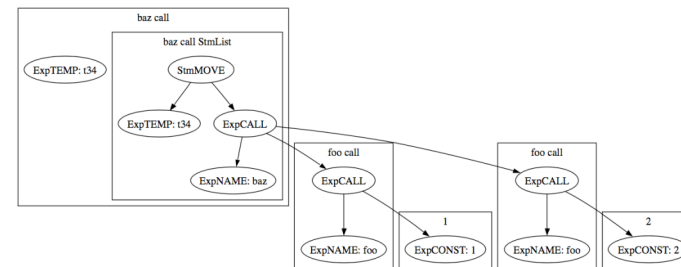
Canonicalization

3

ExpCALLs within ExpCALLs

Problem:

- Reusing argument registers causes argument values to be prematurely overwritten



CS453 Lecture

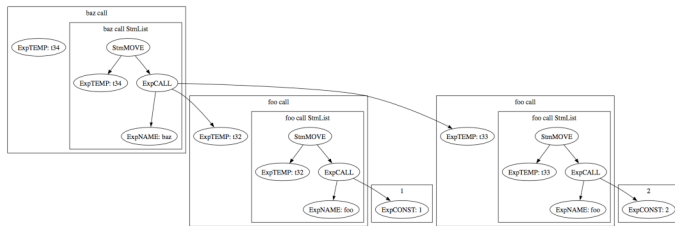
Canonicalization

4

Tree IR being generated in PA7

Our solution:

- Put the result of each ExpCALL into a temporary



CS453 Lecture

Canonicalization

5

Basic blocks and traces

Control flow

- what is the next instruction that can be executed after executing instr X?
- basic blocks are lists of statements with straight-line code control flow
- jumps and cjmps cause more interesting control flow

Canonicalization

- the false body needs to be after each StmCJUMP because of fallthrough

Optimization (could actually do some of this in PA8 to remove jumps)

- if a target basic block follows the block that unconditionally jumps to it, then can remove the jump
- in a trace, a target basic block is placed after a block that jumps to it
- hot paths should be put in the same trace so as to avoid frequent pipeline stalls and icache misses

CS453 Lecture

Canonicalization

6

Basic block and tracing example

```
f = new Foo();
y = new int [20];

if (this.otherFunc(y, f)) {
    local1 = 3;
    local2 = 4;
    local3 = 5 * local1 + y[77];
} else {
    local1 = 8;
    local2 = 9;
    local3 = 10 * local1;
}
```

CS453 Lecture

Canonicalization

7