Applied Static Analysis

Three Address Code

Software Technology Group
Department of Computer Science
Technische Universität Darmstadt
Dr. Michael Eichberg

Intermediate Representations

Goal: Facilitate Static Analyses

How:

- Nested Control-flow and complex expressions are unraveled.
- Intermediate values are given explicit names.
- The data-flow is made (more) explicit.
- The instruction set is limited (more orthogonal).
- …

Examples:

- 3-Address Code (TAC)
- Static Single Assignment (Form) (SSA)

Three-address Code

Three-address code is a sequence of statements (linearized representation of a syntax tree) with the general form:

```
x = y op z
```

where x,y and z are (local variable) names, constants (in case of y and z) or compiler-generated temporaries.

General Types of Three-Address Statements

- Assignment statements: `x = y op z` or `x = op z`
- Copy statements `x = y`
- Unconditional jumps: `goto l`
- Conditional jumps: `if (x rel_op y) goto l` (else fall through), `switch`
- Method call and return: `invoke(m, params)`, `return x`
- Array access: `a[i]` or `a[i] = x`
- *IR specific types.*

Converting Java Bytecode to Three-Address Code

(Syntax-directed Translation)

- Compute for each instruction the current stack layout by following the control flow; i.e., compute the types of values found on the stack before the instruction is evaluated.
  (This is required to correctly handle generic stack-manipulation instructions.)
- Assign each local variable to a variable where the name is based on the local variable index.
- Assign each variable on the operand stack to a corresponding local variable with an index based on the position on the stack.

Converting Java Bytecode to three-address code

```java
static int numberOfDigits(int i) {
    return ((int) Math.floor(Math.log10(i))) + 1;
}
```

PC	Code	Stack Layout	TAC
-	-	-	r_0 = i // init parameters
0	iload_0	<empty>	op_0 = r_0
1	i2d	0: Int Value, →	op_0 = (double) op_0
2	invokestatic log10 (double):double	0: Double Value , →	op_0 = log10(op_0)
5	invokestatic floor(double):double	0: Double Value, →	op_0 = floor(op_0)
8	d2i	0: Double Value, →	op_0 = (int) op_0
9	iconst_1	0: Int Value, →	op_1 = 1
10	iadd	0: Int Value, 1: Int Value, →	op_0 = op_0 + op_1
11	ireturn	0: Int Value, →	return op_0;

Optimizations to get "reasonable" three-address code

1. Peephole optimizations which use a *sliding window* over the cfg’s basic blocks to perform, e.g., the following optimizations:
   - copy propagation
   - elimination of redundant loads and stores
   - constant folding
   - constant propagation
   - common subexpression elimination
   - strength reduction (`x * 2` ⇒ `x + x`; `x / 2` ⇒ `x >> 1`)
   - elimination of useless instructions (`y = x * 0` ⇒ `y = 0`)
2. Intra-procedural analyses:
   - to type the reference variables
   - *standard optimizations to further minimize the code*

Static Single Assignment Form

When an intermediate (three-address code based) representation is in SSA (Form) then:

- each variable is assigned exactly once (i.e., it has only one static definition-site in the program text), and
- every variable is defined before it is used.

When two control-flow paths merge, a selector function ϕ is used that initializes the variable based on the control flow that was taken.

Example plain(naive) three-address code

```
static int max(int i, int j) {
  int max;
  if (i > j) max = i; else  max = j;
  return max;
}
```

```
0: if(i <= j) goto 3;
1: r_0 = i;
2: goto 4;
3: r_0 = j;
4: return r_0;
```

Example SSA three-address code

```
static int max(int i, int j) {
  int max;
  if (i > j) max = i; else  max = j;
  return max;
}

0: if(i <= j) goto 3;
1: t_1 = i;
2: goto 4;
3: t_2 = j;
4: t_3 = Φ(t_1,t_2); // <= Control-flow join
   return t_3;
```