CS 164 Programming Languages and Compilers **Spring 2019**

Discussion 4/29: Code Optimization

Instructor: Paul N. Hilfinger GSIs: Nikhil Athreya, Vivant Sakore

1 Optimizations

Consider the following basic block

```
p = 3
r = 10
s = p + r
t = 2*r + s
t = p
u = p + r
v = p + t
w = 3 + x
```

For each of the following blocks, state what optimization(s) were performed on the above block.

1.
```
p = 3
r = 10
s = p + r
t = p
u = p + r
v = p + t
w = 3 + x
```

2.
```
p = 3
r = 10
s = p + r
t = 2*r + s
t = p
u = s
v = p + t
w = 3 + x
```

3.
```
p = 3
r = 10
s = p + r
```

```
t = 2*r + s
t = p
u = p + r
v = p + p
w = 3 + x
```

4.

```
p = 3
r = 10
s = 13
t = 33
t = 3
u = 13
v = 36
w = 3 + x
```

2 Flow Analysis

Java supports subtype polymorphism by dynamically dispatching methods at execution time, depending on the dynamic type of the receivers. However, dynamic dispatches take longer than calls to known functions. We'd like to do a static flow analysis to improve programs by determining calls with unique, statically known dynamic types, so that these can be changed into direct method calls. Specifically, we'd like a conservative global flow analysis that allows us to determine if a call of the form `V.M(...)`, where `V` is a variable and `M` is a method name, must definitely refer to a particular, known method body. Aside from `if` conditionals, loops, and call statements, we'll restrict our attention to statements of the form:

1. `V1 = new T(...);` // Create a new `T`.
2. `V1 = (T1) V2;` // A cast to type `T1`.

Assume that all variable names are forward declared with types at the beginning of the scope, as in old-style parameter declarations in C.

For simplicity, assume that all types of the forward declarations are compatible with the `new` and `cast` statements. Also assume a cast to an undeclared variable is never performed, variables of the same name are never created with `new` more than once, and restrict the type of the cast to be in the type hierarchy of the original object. You do not need to handle method overriding in this question (if a child and parent class both implement a method `m`, then you do not need to select the child method over the parent method).

1. Describe a flow analysis that implements the above specification. What is a suitable set of abstract values for the purpose of this analysis? What are the initial values? What are the transfer rules of your flow analysis?