Compiler

Static Analysis: Overview

Static Analysis: Overview

- *Static*
  - performed at compile time, i.e., without executing the program
- *Analysis*
  - Determination of interesting properties of the run-time behavior of the program
- The goal is to
  - enable optimizations
  - can also be used for bug finding, testing, debugging, ...

Static Analysis: Motivation – Optimization

The optimization:

```
x = 1;        →        y = 1;
y = x;
```

is unsound if x is used in a later instruction:

```
x = 1;                 y = 1;
y = x;        →
...                    ...
z = x;                 z = x;
```

Static Analysis: Motivation – Checking for Unreachable Code

The following code are accepted by phases up to type checking:

```
int foo(int x) {
  return x;
  x = x + 1;
  x = x – 1;
}
```

```
void bar(boolean b) {
  if (b) { return; }
  else { return; }
  return;
}
```

We'd like to notify users about the above "ill-formed" code (similar to Eclipse IDE)!

Static Analysis: Motivation – Checking for Uninitialized Vars.

The following code are accepted by phases up to type checking:

```
void baz() {
  int x;
  int y;
  y = x;
}
```

```
int bazz(A a) {
  int x;
  if (a == null) { return 0; }
  return a.x + x;
}
```

We'd like to notify users about the above ill-formed code (similar to Eclipse IDE)!

Static Analysis: Motivation – Checking for Null Dereference

The following code is well-formed:

```
void bazzz(A a, int x) {
  if (a == null) {
    a.x = 5;
  }
}
```

```
void bazzzz(A a, int x) {
  A tmp = null;
  if (x > 0) { tmp = a; }
  tmp.x = 5;
}
```

However, we'd like to notify users that the above code may raise exception (not done in Eclipse IDE)!

Interesting Program Properties

- are nearly all undecidable, so the analysis computes a *conservative* approximation:
  - if we say *yes*, then the property definitely holds;
  - if we say *no*, then the property may or may not hold;
  - only the *yes* answer will help us to perform the optimization;
  - a trivial analysis will say *no* always; so
  - the art is to say *yes* as often as possible.
- Properties need not be simply *yes* or *no*, in which case the notion of *approximation* is more subtle.

When is Analysis Performed?

- Static analysis may take place:
  - at the source code level;
  - at some intermediate level; or
  - at the machine code level.
- Static analysis may look at:
  - basic blocks only;
  - an entire function (intraprocedural); or
  - the whole program (interprocedural).
- In each case, we are maximally pessimistic at the boundaries.
- The precision and cost of an analysis rises as we include more information.

Static Analysis: Topics

- Control Flow Graph (CFG)
- Reaching Definition Analysis
- Monotonic Data Flow (MDF) Framework
  - Reaching Definition Analysis
  - Live Variable Analysis
    - a.k.a. Dead Variable Analysis
  - Null Dereference Analysis