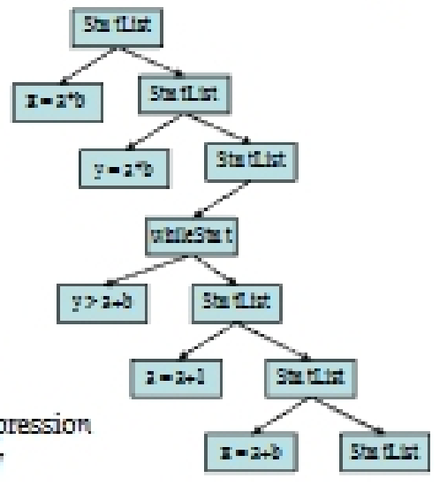
Data flow analysis

Abstract Syntax Trees

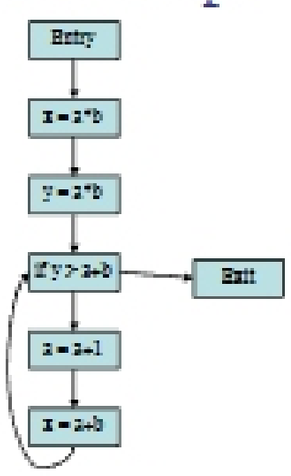
```
x = a*b;
y = a*b;
while (y > a+b) {
    a = a+1;
    x = a+b;
}
```



AST stopped at statement/expression level for brevity

Control Flow Graph

```
x = a*b;
y = a*b;
while (y > a+b) {
    a = a+1;
    x = a+b;
}
```



Choosing a representation

- Control flow graph is more general
- AST allows for more efficient algorithms
  - but new programming constructs require changing the algorithm
    - e.g., continue, break, switch, try-catch-finally, goto
  - program transformations may not leave the program in AST form
  - bytecode/machine code isn't in AST form
    - although you may be able to recover it

Data flow analysis

- A framework for proving facts about a program
  - reasoning about lots of little facts
  - little or no interaction between facts
  - based on all paths through program
    - including infeasible paths
  - e.g., which assignments to x can be seen at this read of x?

Reaching definitions

- Each assignment to a variable is a definition
- defs(v) represents the set of all definitions of v
- Assume all variables are scalars
  - no pointers or arrays

Gen and Kill

- Gen(S) = facts that are true after S, regardless of the facts true before S
- Kill(S) = facts that aren't true after S just because they were true before S
  - but might be true after S
- Out(S) = Gen(S) union (In(S) – Kill(S))

Initial conditions

- Out(Entry) needs to be separately defined
- What is appropriate for reaching definitions?

For reaching definitions

- Gen(d: v = exp) = { d }
- Kill(d: v = exp) = defs(v)