PURDUE
UNIVERSITY

Department of Computer Sciences

Program Analysis

CS502

- After machine code generation (w/o register allocation yet), the rest of this course will focus on *compiler-based program analysis*
- Such analysis is the foundation for many techniques that are aimed at
  - Program execution efficiency
  - Program memory use reduction
  - Program reliability enhancement
  - Program reliability enhancement
    - Debugging, testing, correctness proofs

Static analysis

- Analysis conducted at compile time is applied to the code only
  - This is called static analysis
- Static analysis discover program properties that are true under arbitrary input
  - It must make conservative assumptions
    - What does “conservative” mean depends on the goal of the analysis
- Static analysis may be applied to different levels of internal representation (IR)
  - However, the basic algorithms are usually the same for different levels
  - Our discussions in this course will alternately use the source level (AST) and the low tree level (3AC). NOTE: 3AC is a text dump of the low-level tree
  - At a higher level, more information about the data structure is available to the compiler
  - At a lower level, finer grain operations are exposed to the analysis, potentially yielding more “optimization” opportunities

Dynamic analysis

- Analysis based on information collected during program execution under specific input is called *dynamic analysis*
- Typically the compiler *instruments* the program by inserting information collecting operations
  - Such instrumented operations records events that are useful for the dynamic analysis
  - The level of details depends on the goal and the intended thoroughness of the analysis
  - Examples of collected information: the sequence of instructions executed, the memory locations visited by (load/store) instructions, the value changes to the variables.
- Dynamic analysis may be performed offline
  - The information is recorded in an *execution trace* (or trace in short) for post-execution analysis
- It may also be performed online
  - The information is analyzed during program execution, e.g. for security purpose or for “run-time optimization”

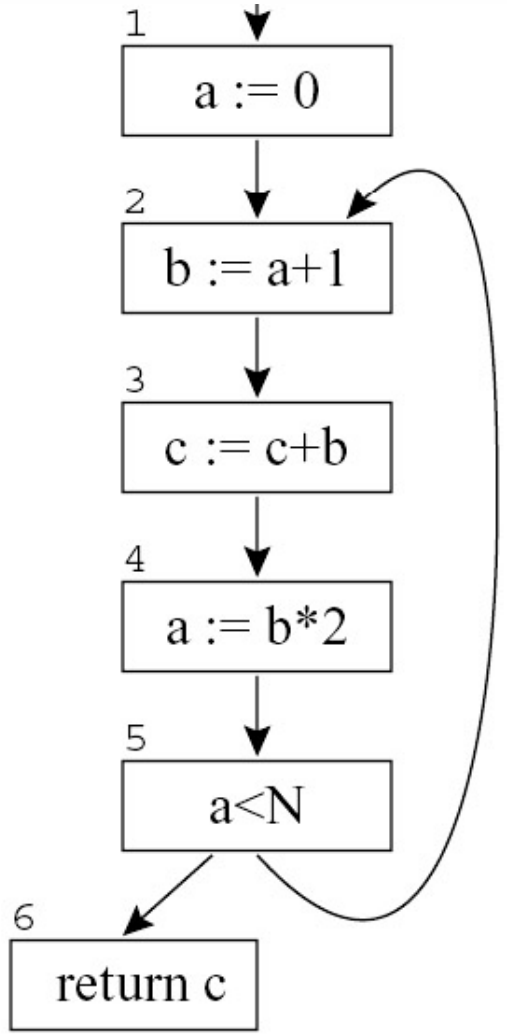
- Static analysis is more traditional than dynamic ones
- Dynamic analysis techniques are usually extensions of similar static techniques
- Therefore, we will begin by discussion of static analysis techniques and spend most of our time on them
- When an analysis is applied to individual functions independently to each other, it is called intra-procedural analysis
  - This analysis assumes no knowledge from other functions
  - It makes conservative assumptions about callees of the function being analyzed and the current function's input parameters.
- When the analysis is applied to multiple functions as a whole (sometime even the entire program), it is called inter-procedural analysis

Control flow graph

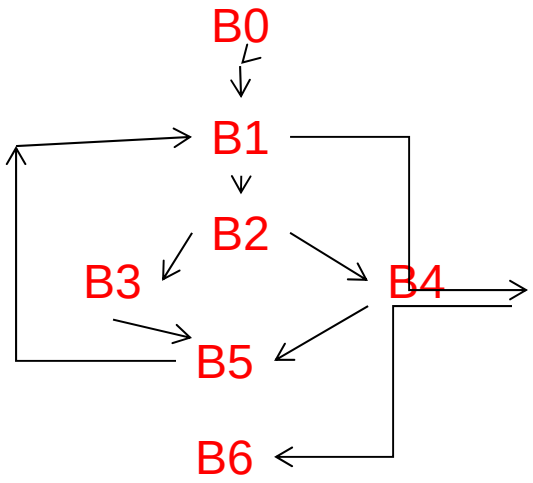
- To support program analysis, the compiler first partitions the program into *basic blocks* and build a *control flow graph* (or flow graph in short)
- A basic block is a sequence of statements (AST level) or instructions (3AC level) which contain no branch target except the first statement (instruction) and no branches except the last statement (instruction)
  - That is, a straight line of code
  - There are two extreme approach to basic-block partition, the maximal basic block (extends the block as far as possible), and the minimum block: single-statement/instruction basic block
  - With maximal basic blocks, the intra-procedural analysis normally requires two steps: local analysis and global analysis, but the minimum block approach requires only global analysis

Example of partitioning of a program into single-instruction basic blocks :

$$
\begin{aligned}
& a \leftarrow 0 \\
L_1 : \; & b \leftarrow a + 1 \\
& c \leftarrow c + b \\
& a \leftarrow b * 2 \\
& \text{if } a < N \text{ goto } L_1 \\
& \text{return } c
\end{aligned}
$$



Loop body is:
If readB
{ v = B[row,j]}
Else
{ v = A[row, j]}
C[row, j] = v;



B0
```
j = 0;
```
B1
```
L1: if j >= 100 goto L2
```
```
if (readB) goto L3
```
B2
```
t0 = row * size
t1= t0 + j
t2 = t1 * 4
t3 = baseA + t2
v = mem[t3]
goto L4
```
B3
```
L3: t4 = row * size
t5 = t4 + j
t6 = t5 * 4
t7 = baseB + t6
v = mem[t7]
```
B4
```
L4: t8 = row * size
t9  = t8 + j
t10 = t9 * 4
t11 = baseC + t10
mem[t11] = v
j = j + 1
goto L1
```
B5
```
L2:
```
B6

Example of partitioning a program into maximum basic blocks