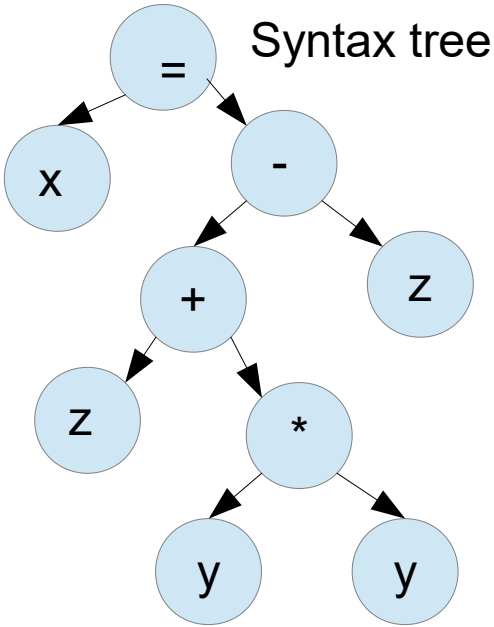
Intermediate representations

- Most compiler 'work' takes place after internal representation is built (analysis, transformations, optimizations, translations)
- Might be a sequence of different internal representations produced, manipulated, refined over time
- Want representations that are efficient to produce, traverse, search, manipulate
- May also want representations that effectively model either the source or target language
- Syntax trees and linear codes are two common representations

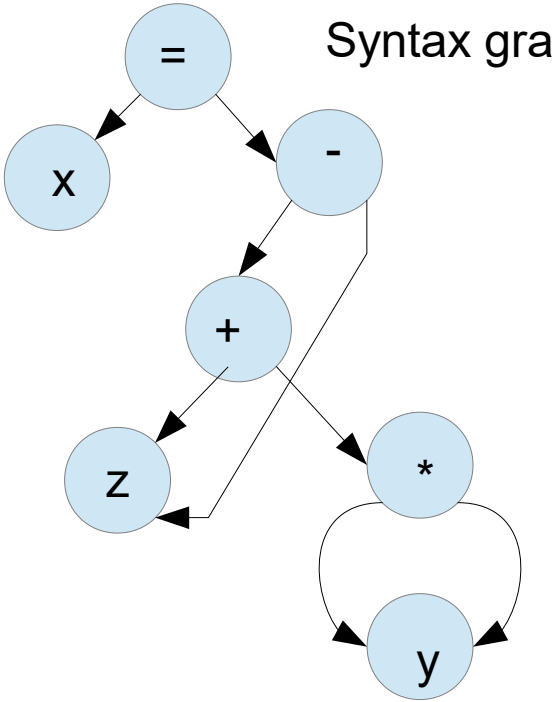
Trees, graphs, codes

X = Z+Y*Y–Z

Linear code
R0 = X
R1 = Y
R2 = Z
R3 = R1 * R1
R4 = R2 + R3
R5 = R4 - R2
X = R5



Syntax tree



Syntax graph

Trees/graphs vs linear codes

- Syntax trees are intuitive/natural representation from perspective of source language
- Wide variety of other graph-based representations of parts of programs (control flow, dependency graphs, call graphs, etc)
- Linear codes are good representation to reflect eventual translation to some form of assembly language
  - Various common forms of linear code (1-address, 3-address)

Combinations, implementations

- Often effective to use combinations of representations, e.g. linear codes plus control flow graphs
- Once a representation form is chosen, still many decisions on how to implement efficiently (which data structures optimize speed, size, usability)
- Also many decisions on the nature of the model of the program in question (choice of memory models, naming and re-naming choices and implications, opportunities for rearranging code segments in the model)