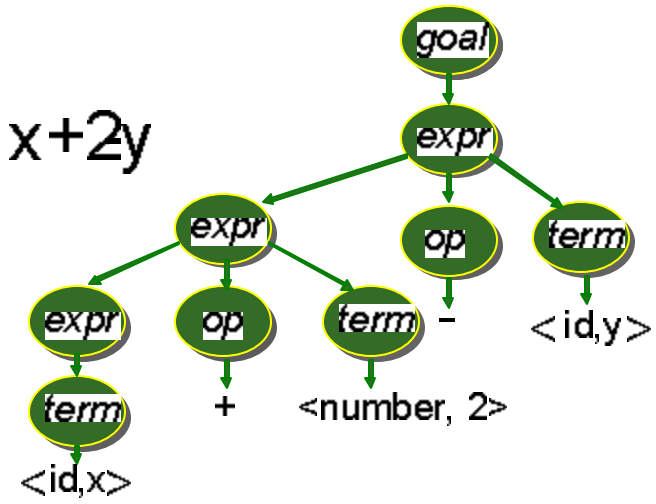
Lecture 3

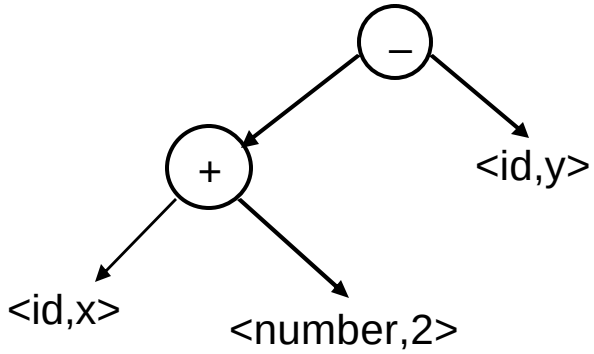
A parse can be represented by a tree: *parse tree* or *syntax tree*. For example, here is the parse tree for the expression `x+2-y`



The parse tree captures all rewrite during the derivation. The derivation can be extracted by starting at the root of the tree and working towards the leaf nodes.

Abstract Syntax Trees

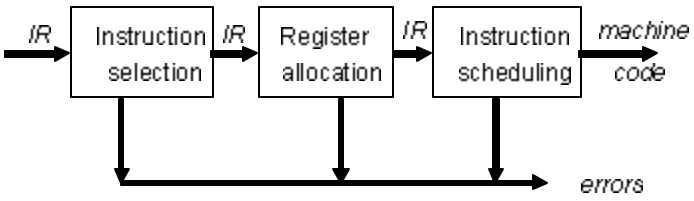
The parse tree contains a lot of unneeded information. Compilers often use an *abstract syntax tree (AST)*. For example, the AST for the above parse tree is



This is much more concise; AST summarizes grammatical structure without the details of derivation. ASTs are one kind of *intermediate representation (IR)*.

The Back End

The back end of the compiler translates IR into target machine code. It chooses machine (assembly) instructions to implement each IR operation. The back end ensure conformance with system interfaces. It decides which values to keep in registers in order to avoid memory access; memory access is far slower than register access.



The back end is responsible for instruction selection so as to produce fast and compact code. Modern processors have a rich instruction set. The back end takes advantage of target features such as addressing modes. Usually, instruction selection is viewed as a pattern matching problem that can be solved by dynamic programming based algorithms. Instruction selection in compilers was spurred by the advent of the VAX-11 which had a CISC (Complex Instruction Set Computer) architecture. The VAX-11 had a large instruction set that the compiler could work with.