Semantic Analysis Phases of Compilation

- Build an Abstract Syntax Tree (AST) while parsing
- Decorate the AST with type information (type checking/inference)
- Generate intermediate code from AST
- Optimize intermediate code
- Generate final code

Abstract Syntax Tree (AST)

Represents the syntactic structure of the input program, independent of peculiarities in the grammar.

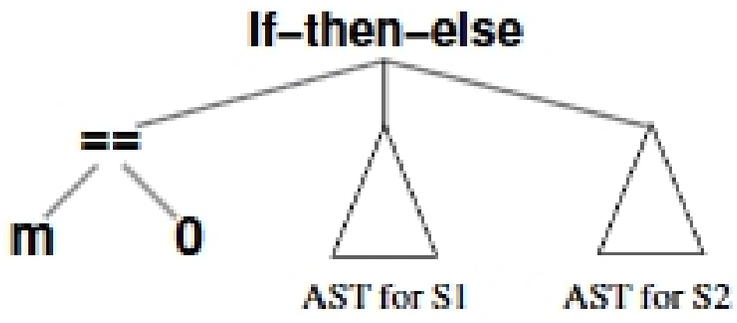
An Example:

Consider a statement of the form:

"`if (m == 0) S1 else S2`"

where S1 and S2 stand for some block of statements.

A possible AST for this statement is:



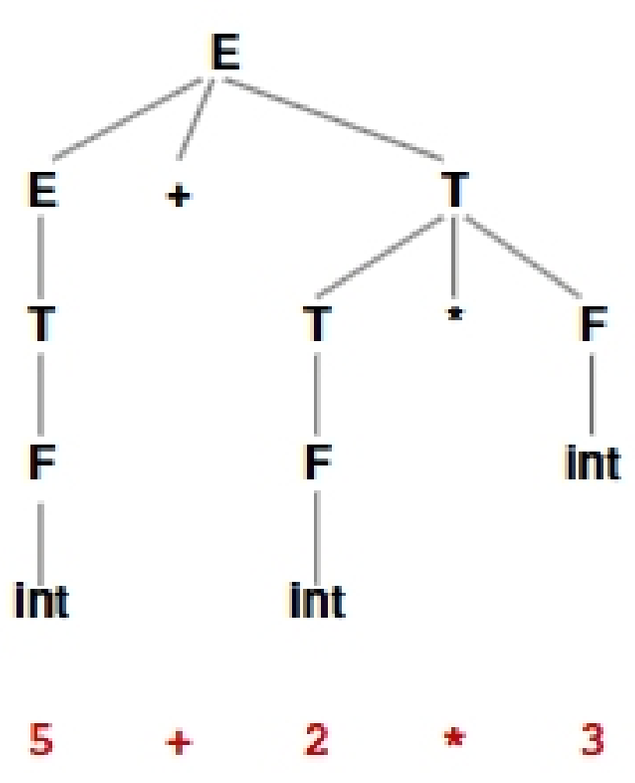
Construction of Abstract Syntax Trees

Typically done simultaneously with parsing

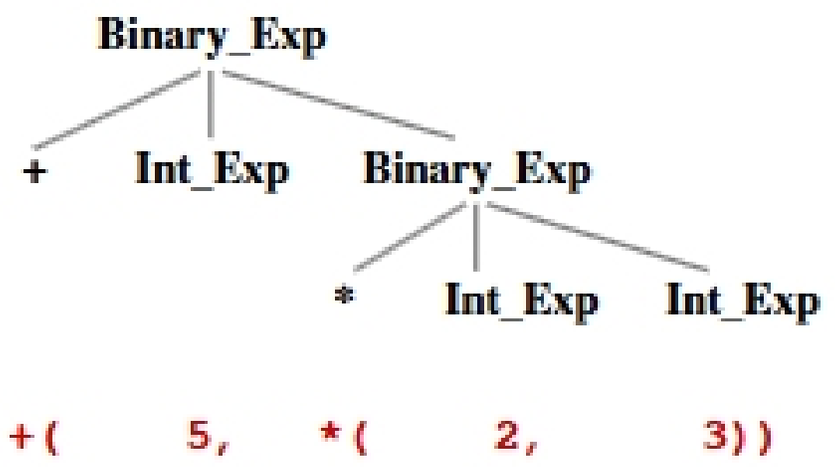
- ... as another instance of syntax-directed translation
- ... for translating *concrete* syntax (the parse tree) to *abstract* syntax (AST).
- ... with AST as a *synthesized attribute* of each grammar symbol.

Abstract Syntax Trees

Parse Tree



AST



Trees & Tree Grammars

Tree grammars can be used to specify a set of **trees**.

Example 1: The set of trees $\{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$ can be represented by the grammar:

$$\begin{array}{lcl} T & \longrightarrow & \texttt{a} \\ T & \longrightarrow & \texttt{f}(T) \end{array}$$

Example 2: The set of trees $\{b, g(b,b), g(g(b,b),b), g(b,g(b,b)), \ldots\}$ can be represented by:

$$\begin{array}{lcl} S & \longrightarrow & \texttt{b} \\ S & \longrightarrow & \texttt{g}(S,S) \end{array}$$

AST & Tree Grammars

The set of AST's that represent the set of (syntactically correct) programs can be described by tree grammars.

$$\begin{array}{lcl} \textit{Expression} & :: & \texttt{INT\_EXPR}(\textit{Int\_val}) \\ & | & \texttt{FLOAT\_EXPR}(\textit{float\_val}) \\ & \vdots & \vdots \\ & | & \texttt{METHOD\_EXPR}(\textit{Expression}, \textit{Expression}^*) \\ & | & \texttt{ARRAY\_EXPR}(\textit{Expression}, \textit{Expression}) \\ & | & \texttt{FIELD\_EXPR}(\textit{Expression}, \textit{Name}) \\ & | & \texttt{BINARY\_EXPR}(\textit{BinaryOp}, \textit{Expression}, \textit{Expression}) \\ & | & \texttt{UNARY\_EXPR}(\textit{UnaryOp}, \textit{Expression}) \\ & \vdots & \vdots \end{array}$$