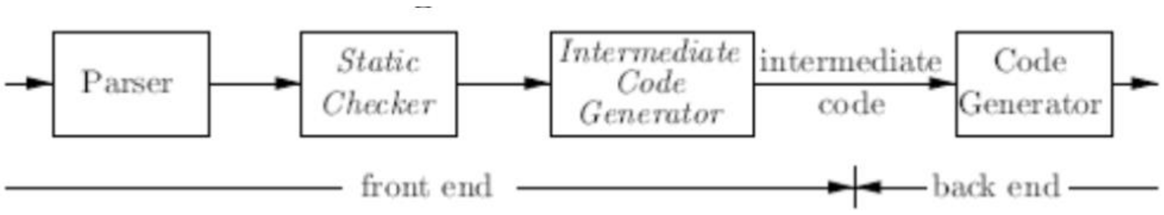
INTERMEDIATE LANGUAGES

In the analysis-synthesis model of a compiler, the front end analyzes a source program and creates an intermediate representation, from which the back end generates target code. Ideally, details of the source language are confined to the front end, and details of the target machine to the back end.

Benefits of using a machine-independent intermediate format are

1. Retargeting is facilitated; a compiler for a different machine can be created by attaching a back end for the new machine to an existing front end.
2. A machine-independent code optimizer can be applied to the intermediate representation.

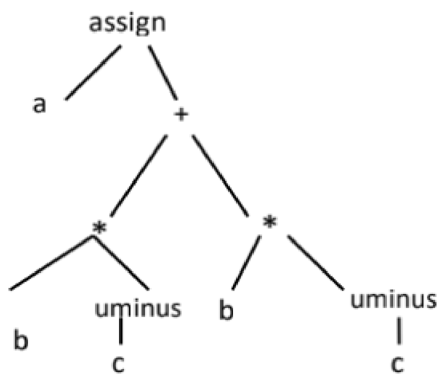


SYNTAX TREE

Graphical Representation

A syntax tree depicts the natural hierarchical structure of a source program. A dag gives the same information but in more compact way because common subexpression are identified. A syntax tree and dag for the assignment statement for

a := b * -c + b * -c



Variants of Syntax Trees

Nodes in a syntax tree represent constructs in the source program; the children of a node represent the meaningful components of a construct. A Directed Acyclic Graph (DAG) for an expression identifies the common sub-expressions (sub expressions that occur more than once) of the expression.

DAG for Expressions

Like the syntax tree for an expression, a DAG has leaves corresponding to atomic operands and interior codes corresponding to operators. The difference is that a node N in a DAG has more than one parent if N represents a common subexpression; in a syntax tree, the tree for the common sub expression would be replicated as many times as the sub expression appears in the original expression. Thus, a DAG not only represents expressions more succinctly, it gives the compiler important clues regarding the generation of efficient code to evaluate the expressions.

The DAG for the expression a + a * (b - c) + (b - c) * d