Compiler construction

In formal language theory, a **grammar** (when the context is not given, often called a formal **grammar** for clarity) describes how to form strings from a language's alphabet that are valid according to the language's syntax.

Derivation

Derivation a sequence of applications of the rules of a **grammar** that produces a finished string of terminals. A leftmost **derivation** is where we always substitute for the leftmost nonterminal as we apply the rules (we can similarly define a rightmost **derivation**). A **derivation** is also called a parse

Production

A **production** or **production rule** in computer science is a rewrite rule specifying a symbol substitution that can be recursively performed to generate new symbol sequences. A finite set of productions {P} is the main component in the specification of a formal grammar (specifically a generative grammar). The other components are a finite set {N}

of nonterminal symbols, a finite set (known as an alphabet) { Sigma } of terminal symbols that is disjoint from { N}

and a distinguished symbol {S € N}

that is the start symbol

Sentence

Grammar a set of rules by which valid **sentences** in a language are constructed. nonterminal a **grammar** symbol that can be replaced/expanded to a sequence of symbols. ... Such a string is called a **sentence**. In the context of programming languages, a **sentence** is a syntactically correct and complete program

Null symbol

null symbol ε it is sometimes useful to specify that a **symbol** can be replaced by nothing at all. To indicate this, we use the **null symbol** ε, e.g., A –> B | ε. BNF a way of specifying programming languages using **formal grammars** and production rules with a particular form of notation (Backus-Naur form)

Last modified: 12:32 pm