CS109B Notes for Lecture 4/28/95

Why Grammars?

- Useful for describing programming languages.
- First great use of a theory to design better software — a multi-person-year job (parsing in original Fortran compiler) became an afternoon's work using tools like YACC.

Grammars

A notation for inductive definition of certain languages.

- *Syntactic categories* = symbols that represent one of perhaps several recursively defined languages.
  - ◻ Denoted by triangular brackets and a descriptive term, e.g., <exp> for the syntactic category of strings that are arithmetic expressions.
- *Terminals* = symbols that may appear in the strings of the language(s) defined by the syntactic category(ies).
  - ◻ Represented by characters or italic words, e.g., `0` or *digit*.
- *Productions* = rules about how strings of terminals in the language of one SC are formed from constant strings and strings in certain SC's by concatenation.
  - ◻ Form is *head* → *body*. *head* is a SC and *body* is a sequence of zero or more terminals and SC's.

Example: The gross structure of ML matches can be described by the following grammar.

(1) <match> → <pat_exp> `|` <match>
(2) <match> → <pat_exp>

(3) <pat_exp> → *pattern* `=>` *exp*

- A more detailed description would make *pattern* and *exp* be SC's and give them suitable productions.

Languages

Each SC defines a language. These languages are defined recursively by:

Basis: If SC A is the head of a production with only terminals in the body, then the body is in $L(A)$.

Induction: Consider every production with at least one SC in its body. Replace the SC's of the body by strings known already to be in their language(s) in all possible ways.

- Each resulting string is in the language of the head.

Example: For ML match grammar:

Basis: (Round 1) The string "*pattern* => *exp*" (a string of length 4) is in $L(<pat\_exp>)$ by production (3).

Induction: Round 2: That string is also in $L(<match>)$ by production (2). Production (1) yields nothing.

Round 3: Production (1) yields

pattern => *exp* | *pattern* => *exp*

for $L(<match>)$, and so on.

- In round i, production (1) yields for $L(<match>)$ a string with $i-1$ pattern-expression pairs and $i-2$ bars.

Class Problem

We could define the language consisting of only the string 0^{1000} (one thousand 0's) by a single production

$$<\text{goal}> \rightarrow 00\cdots 0 \text{ (1000 of them)}$$

Can you propose a grammar that can be written down more succinctly, even if it is more "complex"?