

Introduction to Formal Language Theory

A formal language is simply a set of sequences.

Usually we restrict our selves to *possibly infinite* set of *finite sequences* over a *finite set* Σ.

Formal language theory considers *finite* descriptions of languages.

We are particularly interested in description methods that are

- easy to understand and use
- lead to algorithms for analyzing sequences
- suitable for automated processing

In particular we will look at

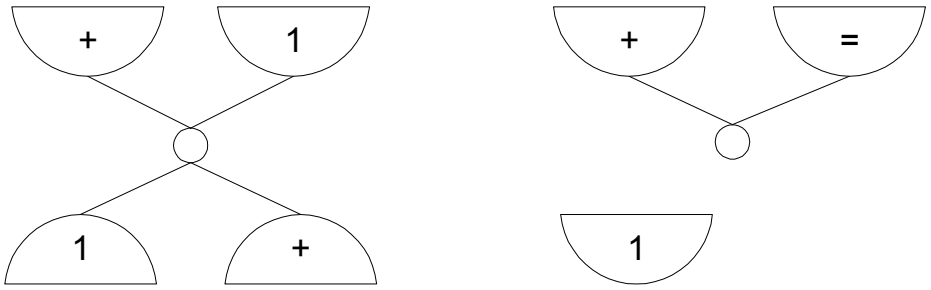
- Grammars
- Regular Expressions
- Finite State Automata

Grammars

Unrestricted Grammars

Game 1

We have an unlimited supply of puzzle pieces of each of 3 shapes



The puzzle starts with a sequence of pieces:

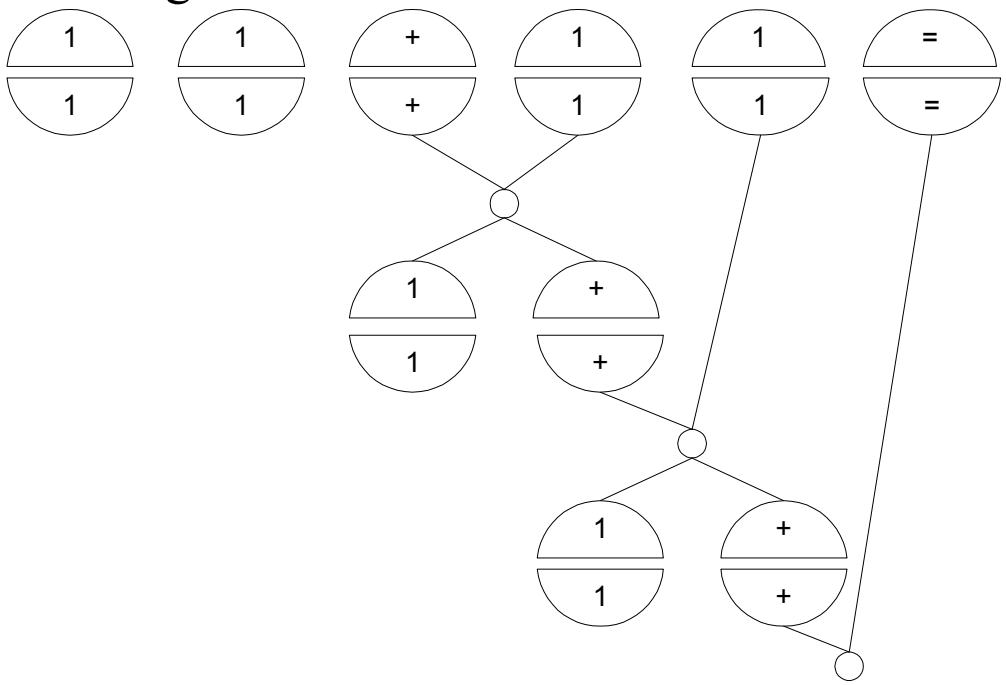
The idea is to close all the circles.

In each step we can add one pieces from our infinite supply.

We can stretch the pieces, but can not alter the sequence of symbols along their top or bottom and can not rotate the pieces.

We also can not cross lines.

In this case we get



Now looking at the sequence of closed circles (with no lines at the bottom), we get:

1111

Thus our puzzle pieces describe an algorithm for computing addition in unary (tally) notation.

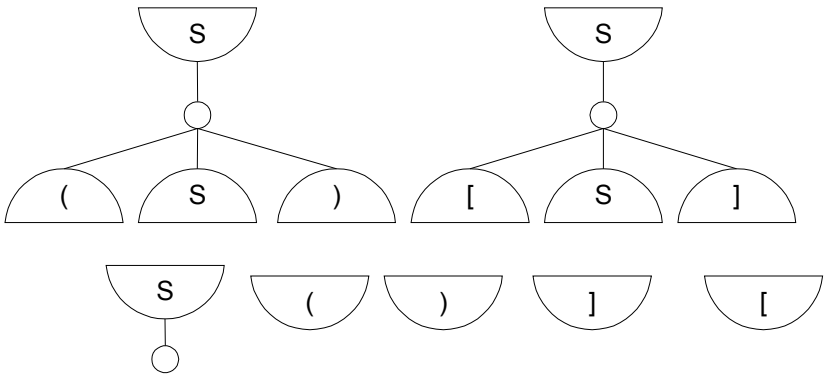
In general, a set of puzzle pieces defines a function from finite sequences to finite sequences.

Proposition 1 *Any function from finite sequences of strings to finite sequences that can be computed by an algorithm can be turned into a game like this.*

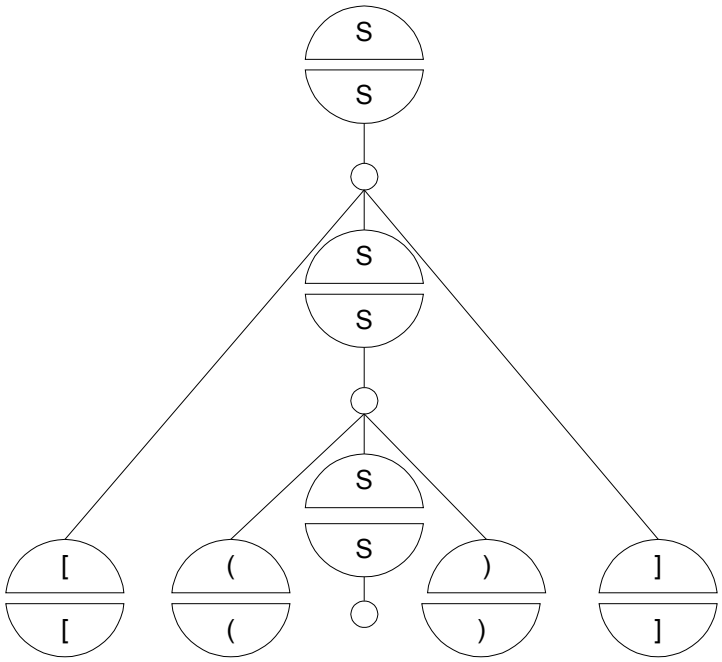
We call such a function a “computable function”.

Game 2

For this game we start always start with an S and the 7 piece kinds are



We can make a tree



The final sequence is [()] other sequences we can reach are [[[(([])]]] and the empty sequence.
This game defines an infinite set of finite sequences over the *alphabet* {‘(’, ‘)’, ‘[’, ‘]’}.

Proposition 2 *Any set of finite sequences that can be generated by an algorithm can be defined by a game like this.*

We call set a set a “computable set”.

Handier Notation

To save space, we will use a more compact notation.

Game 1 again

We define a finite set of "terminal symbols" $\Sigma = \{1\}$.

We define a finite set of "nonterminal symbols" $N = \{\text{“+”},\text{“=”}\}$.

We define a finite set of "productions"

$$P = \{\ + = \ \longrightarrow \epsilon,$$
$$+1 \longrightarrow 1+\ \}$$

(Note ϵ is the empty sequence, earlier written as $\langle\rangle$)

The above comprise a "grammar".

We play the game by starting with a sequence, say 11+111=, and replacing any occurrence of the right hand side of a production with its left hand side, stopping when only terminals remain

$$\begin{aligned} & 11{+}111{=} \\ \Longrightarrow\ & 111{+}11{=} \\ \Longrightarrow\ & 1111{+}1{=} \\ \Longrightarrow\ & 11111{+}{=} \\ \Longrightarrow\ & 11111 \end{aligned}$$

Game 2 again

We define a finite set of "terminal symbols" $\Sigma = \{\text{'('},\text{')'},\text{'['},\text{']'}\}$.

We define a finite set of "nonterminal symbols" $N = \{S\}$.

We define a finite set of "productions"

$$P = \{ \quad S \longrightarrow \epsilon, \\ S \longrightarrow (S), \\ S \longrightarrow [S] \ \}$$

We define a starting nonterminal S.

We play the game by starting with the starting nonterminal and replacing left hand sides with right hand sides until we only have terminals

$$\begin{aligned} & S \\ \Longrightarrow\ & (S) \\ \Longrightarrow\ & ((S)) \\ \Longrightarrow\ & (([S])) \\ \Longrightarrow\ & (([])) \end{aligned}$$

Unlike game 1, we are faced with some choices.

Formalizing a bit

A bit of notation

For this part of the course:

- I'll call finite sequences "strings"
- I'll write strings without angle brackets or commas, i.e. abc rather than $\langle a, b, c \rangle$.
- I'll write the empty string as ϵ rather than $\langle\rangle$.
- I'll write catenation as juxtaposition, i.e. $\alpha\beta$ rather than $\alpha // \beta$.
- Given a set T, I'll write T^* for the set of all finite strings over T.

Conventions

- α, β, γ, δ, η, and κ are strings of symbols (terminal or nonterminal).
- w, x, y, and z are strings of terminals.
- a, b, c, d, and e are terminals.
- A, B, C, D, E and S are nonterminals
- X, Y, Z are symbols (terminal or nonterminal)

Grammars

Definition: A "grammar" is a tuple $G = (N, \Sigma, P, S)$ or $G = (N, \Sigma, P)$ where

- N is a finite set of nonterminal symbols
- Σ is a finite set of terminal symbols (disjoint from N)
- P is a finite set of productions of the form $\alpha \longrightarrow \beta$ with at least one nonterminal in α.
- S is a member of N called the "start symbol"

Production

If we have a production rule $\alpha \longrightarrow \beta$ and then we say a string $\gamma\alpha\delta$ "can produce" a string $\gamma\beta\delta$.

Definition: More formally, given a grammar $G = (N, \Sigma, P, S)$ we say that η "can produce" κ exactly if there exist

- a production $(\alpha \longrightarrow \beta) \in P$
- and strings γ and δ such that $\eta = \gamma\alpha\delta$ and $\kappa = \gamma\beta\delta$.

We write $\eta \Longrightarrow \kappa$ to mean η "can produce" κ

Derivation

Definition: If there exists a finite sequence of strings α_0, α_1, ... , α_n such that

$$\alpha = \alpha_0 \Longrightarrow \alpha_1 \Longrightarrow \ldots \Longrightarrow \alpha_n = \beta$$

then we say that α "derives" β. In notation:

$$\alpha \stackrel{*}{\Longrightarrow} \beta$$

And we say that α, α_1, ... , β is a "derivation".

The language generated by a grammar

Definition: If a grammar G has a start symbol S, then the "language generated by" the grammar is

$$L(G) = \{w \mid S \stackrel{*}{\Longrightarrow} w\}$$

Note that $L(G) \subseteq \Sigma^*$. For example for G from game 1 we have

$$L(G) = \{\epsilon, (), [], (()), ([]), [()], [[]], \ldots\}$$

Context Free Grammars

Game 1 and Game 2 have a significant difference.

Game 2 only produces trees. This is because each puzzle piece has only one semicircle in its top row.

We call such a grammar "context free".

Definitions

Definition: A "context free grammar" is a grammar where each production rule is of the form

$$A \longrightarrow \beta$$

for some $A \in N$.

Definition: A "context free language" is a language generated by some context free grammar.

Significance

Con: There are computable languages that are not context free.

Pro: context free grammars

- Are easy to use and understand
- Given a grammar, there is always an algorithm to determine whether or not a string is in the language generated by the

grammar: $O(N^3)$
  * For common special cases there are fast algorithms: $O(N)$.

- Many useful and important languages are context free.
  * Example: The language of syntactically correct Java classes
  * Example: Well formed HTML documents.
  * Example: Correct usages of many communication protocols.
- For languages that are not context free, we can often start by defining a context free language and then restricting that language
  * Example: The language of compile-time error free Java classes.