Formal Languages

Grammar—Intro & Regular

Ryan Stansifer

Computer Sciences
Florida Institute of Technology
Melbourne, Florida USA 32901

http://www.cs.fit.edu/~ryan/

2 March 2024

Outline

1. Computation and Formal Languages
   - Examples
   - Universe
2. Grammar and Language
   - Definitions, Derivation
   - Different Kinds of Grammars
3. Regular Grammars
   - Definitions
   - Reflection and Conclusions

- Grammars are a model of computation and are a good place to start (equal to, and independent of, automata).
- The details of a formal treatment of grammars may obscure their place in the grand scheme of computation (and even their practical applications).
- We recap our overall motivation and introduction. In other words, we start all over again.

The big question is:
What is computation?

What do *Formal Languages* and Automata have to do with it?

Definition

A *symbol* is a sign or a mark that can be distinguished from any other symbols.

Definition

An *alphabet* is a finite set of symbols.

Definition

A *string* is a finite, ordered sequence of symbols from an alphabet.

Definition

A *formal language* is a set of strings (possibly infinite) made up of symbols from the same alphabet.

Definition

A *formal language* is a set of strings (possibly infinite) made up of symbols from the same alphabet.

Definition

A *formal language* is a set of strings (possibly infinite) made up of symbols from the same alphabet.

The significance is:

computational problem

=

formal language

How does one view a computational problem as a formal language?

Computation can be reduced to accepting or rejecting strings.

1. Computational input and data can be encoded as strings.
2. Computational results are reduced to yes and no.

Neither procedural nor data abstraction is convenient in string form.

However, the fundamental nature of computation can more easily be explored in a simple setting.

The universality of strings is scarcely in doubt in the digital world of 0's and 1's.

language	in the language	not in the language
second-to-last symbol is *a*	*aa* *bbbab* *bbbbbbaaabab*	*a* *aaaba* *bbbbbbbbbb*
equal number of *a*'s and *b*'s	*ba* *bbaaba* *aaabbbbbaaab*	*a* *bbbaaa* *abababababa*
palindromes	*a* *abba* *abaabaabaaba*	*ab* *bbbbaab* *abababababab*
contains the pattern *abba*	*abba* *ababbabbababbba* *bbbbbbbbbabbabb*	*abb* *bbabaab* *aaaaaaaaaaaa*
the number of *b*'s is divisible by 3	*bbb* *baaaaabaaaab* *bbbabbaabaabababa*	*bb* *abababab* *aaaaaaaaaaab*

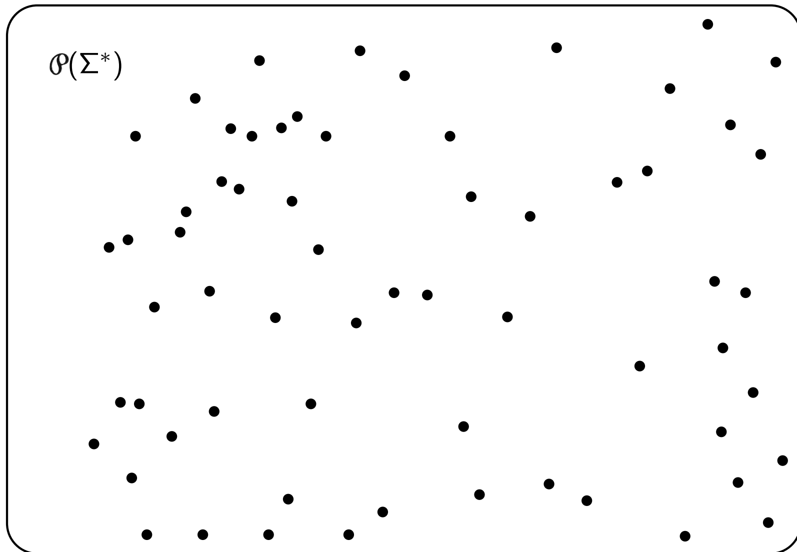
language	in the language	not in the language
second-to-last symbol is 0	00 11101 111111000101	0 00010 11111111111
equal number of 0's and 1's	10 110010 000111110001	0 111000 01010101010
palindromes	0 0110 010010010010	01 1111001 0101010101010101
contains the pattern 0110	0110 0101101101011110	011 1101001 0000000000000
	111111111011011	
the number of 1's is divisible by 3	111 100000100001 11101100100101010	11 01010101 00000000001

language	in the language	not in the language
second-to-last symbol is **£**	**££** **฿฿฿£฿** **฿฿฿฿฿฿£££฿£฿**	**£** **£££฿£** **฿฿฿฿฿฿฿฿฿฿฿**
equal number of **£**'s and **฿**'s	**฿£** **฿฿££฿£** **£££฿฿฿฿฿£££฿**	**£** **฿฿฿£££** **£฿£฿£฿£฿£฿£**
palindromes	**£** **£฿฿£** **£฿££฿££฿££฿£**	**£฿** **฿฿฿฿££฿** **£฿£฿£฿£฿£฿£฿£฿**
contains the pattern **£฿฿£**	**£฿฿£** **£฿£฿฿£฿฿£฿£฿฿฿£** **฿฿฿฿฿฿฿฿฿£฿฿£฿฿**	**£฿฿** **฿฿£฿££฿** **£££££££££££££**
the number of **฿**'s is divisible by 3	**฿฿฿** **฿£££££฿££££฿** **฿฿฿£฿฿££฿££฿£฿£฿£**	**฿฿** **£฿£฿£฿£฿** **££££££££££££฿**

$\wp(\Sigma^*)$

sets of strings over the alphabet Σ

The universe of formal languages



There are many different formal languages

$\wp(\Sigma^*)$

$\bullet\{a, aa, aaa, \ldots\}$

$\bullet\{a, b\}$

$\bullet\{a, b, aba, abba\}$

Each element (dot) is a formal language (over $\Sigma = \{a, b\}$)

$\wp(\Sigma^*)$

$\bullet \Sigma^* = \{\epsilon, a, b, aa, ab, \ldots\}$

$\bullet \Sigma^+ = \{a, b, aa, ab, ba, bb, \ldots\}$

$\bullet \{\}$

Special languages: the empty language of cardinality zero, Σ^*, and Σ^+

Incidentally, it should be clear from the meaning of powerset that for formal languages L:

$$L \in \wp(\Sigma^*) \qquad \text{is the same thing as} \qquad L \subseteq \Sigma^*.$$

Properties of formal languages give rise to sets or families of formal languages.

$$\mathcal{A} = \{\, L \in \wp(\Sigma^*) \mid \text{property } P(L) \text{ holds} \,\}$$

Properties like:

1. languages with strings of length exactly two, e.g., $L_1 = \{aa, bb\}$
2. languages with strings whose symbols are in alphabetical order, e.g., $L_2 = \{c, ac, abc, ac\}$
3. languages with a finite number of members, e.g., $L_3 = \{\epsilon, \text{฿}, \text{£฿฿£}\}$.
4. languages with strings consisting of only a's, e.g., $L_4 = \{\epsilon, aa, aaaa, \ldots\}$.

Using mathematical properties of formal languages to define language families

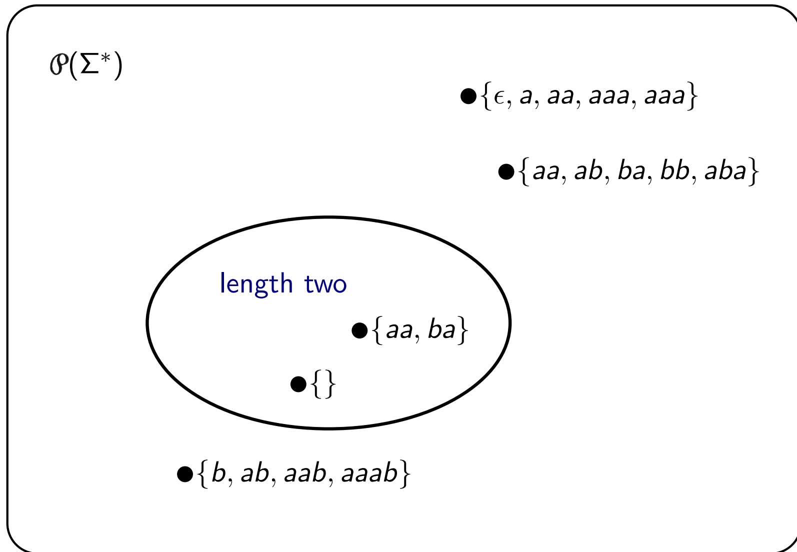
$$\mathcal{A} = \{\, L \in \wp(\Sigma^*) \mid \text{property } P(L) \text{ holds} \,\}$$

appears not to ensnare us with Russell's paradox.

Although the formal language $L \in \wp(\Sigma^*)$ is a *set*, a property $P(L)$ like $L \notin L$ is nonsense. The *types* do not work out. L is a set of strings; it cannot also be a set of formal languages (or can it?).

Nonetheless there are many reasons to reject mathematical properties as a means of description in favor of more means that are more explicit, simple, and algorithmic.

In the end, types do not prevent Russell's paradox because of encodings.



A set of formal languages, e.g., all members have length two

$\wp(\Sigma^*)$

$\bullet \{a, aa, aaa, \ldots\}$

languages not containing ϵ

languages with ϵ

$\bullet \{\epsilon, a, a, aa, \ldots\}$

$\bullet \{\epsilon, abba\}$

$\bullet \{b, ab, aab, aaab, \ldots\}$

$\bullet \{aa, ab, ba, bb\}$

Sets of formal languages, e.g., ϵ-free languages

$\mathcal{P}(\Sigma^*)$

$\bullet\{\epsilon, a, aa, aaa, \ldots\}$

infinite languages

finite languages

$\bullet\{a, b, aba, aabba\}$

$\bullet\{\epsilon, abba\}$

$\bullet\{b, ab, aab, aaab, \ldots\}$

Sets of formal languages, e.g., finite languages

The set of finite languages is far too small and exclusive a set to be a satisfactory characterization of computabilit.

Some infinite languages, for example $L_a = \{\epsilon, a, aa, aaa, \ldots\}$ seem intuitively computable.

We can easily construct L_a inductively

1. the empty string $\epsilon \in L_a$,
2. if $w \in L_a$, then $a : w \in L_a$.

Thus L_a is not much of a computatational challenge.

L_a is just the same as tally notation. We can capture this formal language in so many diferent versions in Haskell, not to mention Agda or Coq.

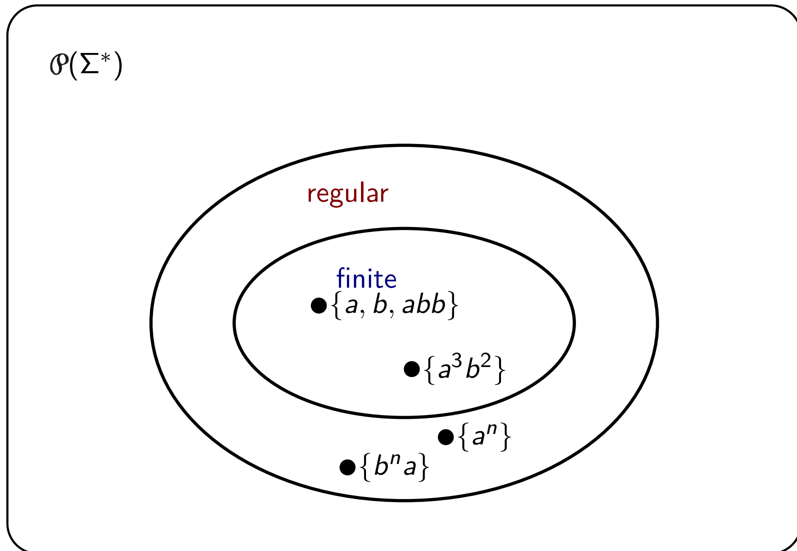
```
-- Peano Numbers, Tally notation
data Nat = Zero  | Succ Nat

data La  = Empty | ConsA La

La :: String
La = [] : [a:w | w <- La]
La = [] : (map ('a':) La)
La = iterate ('a':) La
```

The characteristic function for L_a written (recursively) in Haskell:

```
allA :: String -> Bool
allA [] = True
allA (c:cs) = (c == 'a') && (allA cs)
```

The "real" Haskell programmer would write simply:

```
allA = all (=='a')
```

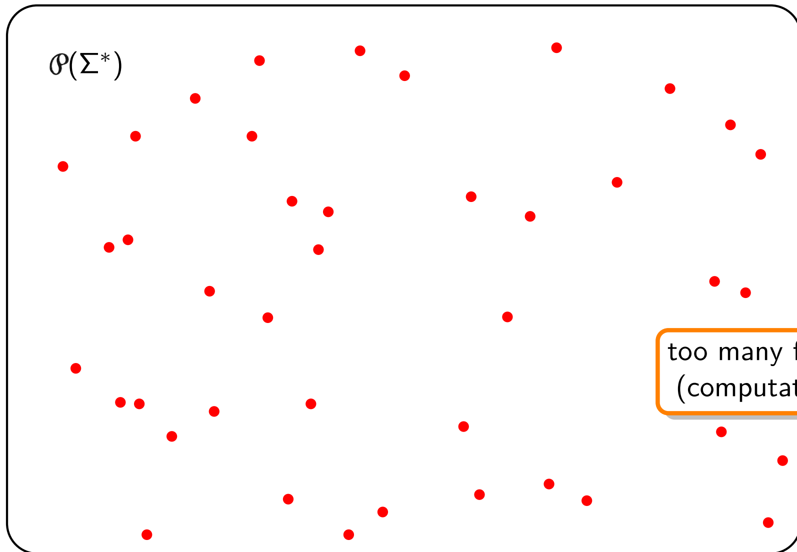


We will study important family of languages, e.g., regular languages

If the set of finite languages is too small and exclusive a set. Maybe all formal languages are computable. But the family of all languages is far too large and impossible as a characterization of computability.

There are more problems than there are programs to solve them.

There are more (true) facts than there are proofs to establish them.



Doomed from the outset by Cantor's theorem (diagonalization)

Theorem (Cantor's Theorem)

For any set X we cannot have a one-to-one correspondence between the elements of X and the elements of $\wp(X)$.

Put another way ...

Theorem (Cantor's Theorem)

A set X cannot be put into a one-to-one correspondence with $\wp(X)$.

In particular the cardinality of X is strictly less than that of $\wp(X)$.
The cardinality of any $L \subseteq \Sigma^*$ is $|L| < |\wp(L)|$.

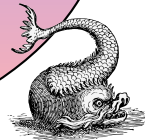
Nothing will describe all possible formal languages; not

- C++, Java, Haskell, Ada programs,
- Turing machines,
- Quantum computers, nor
- *any conceivable description of computation*

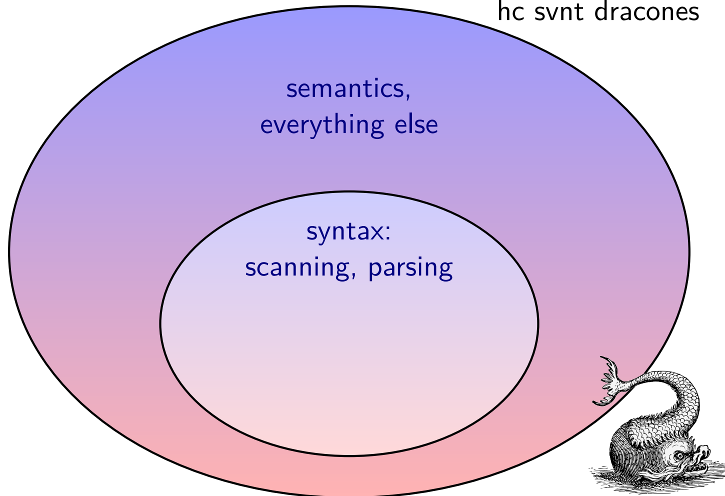
Because formalizable is going to entail countability, no resonable description mechanism is going to describe all languages; no computational system is going to solve all problems.

hc svnt dracones

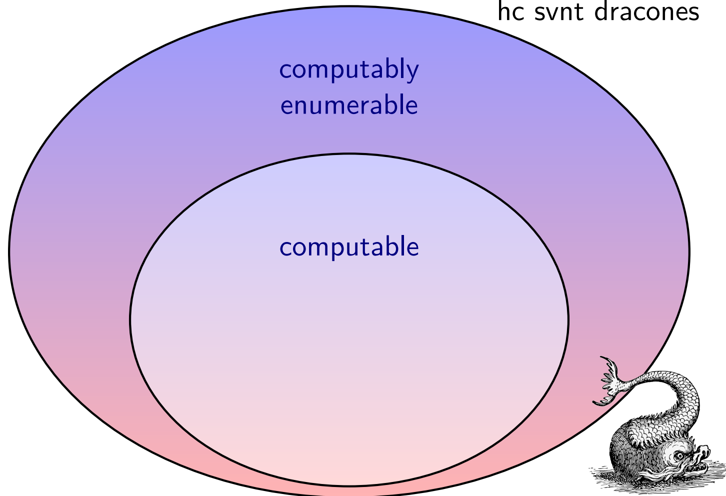
set of strings,
syntax



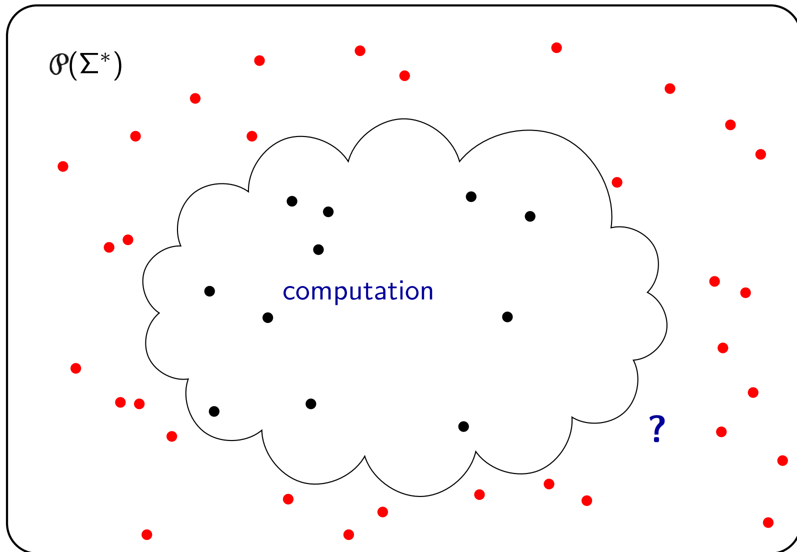
From a theoretical perspective, everything is syntax



From an implementation perspective, a sharp disctinction



Computably enumerable versus computable

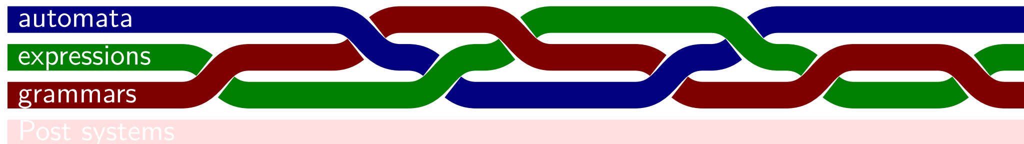


Can we characterize computation?

Four Paradigms of Formal Languages

- Ad-hoc mathematical descriptions using sets and set operations.

1. An automaton M accepts or recognizes a formal language $\mathscr{L}(M)$.
2. An expression x denotes a formal language $\mathscr{L}[\![x]\!]$.
3. A grammar G generates a formal language $\mathscr{L}(G)$.
4. A Post system P derives a formal language $\mathscr{L}(P)$.



A language may be described mathematically using predicates and set comprehension, as in $L = \{ w \in \Sigma^* \mid P(w) \}$. This means $w \in L$ if, and only, if $P(w)$ is true.

The strings of a language may be accepted by an automaton M.

$$\mathscr{L}(M) = \{w \in \Sigma^* \mid \langle q_0, w\rangle \vdash^* \langle q_f, \epsilon\rangle \text{ where } q_f \text{ is final state} \}$$

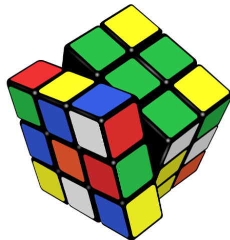
A language may be denoted by an expression x. We write $L = \mathscr{L}[\![x]\!]$.

The strings of a language may be generated by a grammar G.

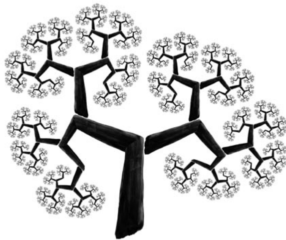
$$\mathscr{L}(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \}$$

A strings of a language may be derived by a Post system.

Automata compute



Expressions denote



Trees demonstrate

Grammars construct

Grammars

Grammar

A grammar is another formalism for defining and generating a formal language (a set of strings over an alphabet).

Grammars were developed independently of the question of computability and have their own vocabulary. In the context of grammars it is traditional to call the alphabet "a set of terminal symbols;" and to call a string "a sentence."

Grammars play an important role in defining programming languages and in the construction of compilers for programming languages.

Many different kinds of grammars have been studied.

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is a finite set of terminal symbols;

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S\rangle$:

- T is a finite set of terminal symbols;
- V is a finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is a finite set of terminal symbols;
- V is a finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is a finite set of terminal symbols;
- V is a finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;
- P is a finite set of productions.

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is a finite set of terminal symbols;
- V is a finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;
- P is a finite set of productions.

A production has the form $\alpha \rightarrow \beta$ where α and β are ordered sequences (strings) of terminals and nonterminals symbols. (We write $(T \cup V)^*$ for the set of ordered sequences of terminal and nonterminal symbols.) The LHS α of a production can't be the empty sequence, but β might be.

Definition of Grammar

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is a finite set of terminal symbols;
- V is a finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;
- P is a finite set of productions.

A production has the form $\alpha \to \beta$ where α and β are ordered sequences (strings) of terminals and nonterminals symbols. (We write $(T \cup V)^*$ for the set of ordered sequences of terminal and nonterminal symbols.) The LHS α of a production can't be the empty sequence, but β might be.

The set of terminal symbols is exactly the same concept as the alphabet no matter if we denote it by T or Σ.

Example

The following five productions are for a grammar with $T = \{ \mathtt{0}, \mathtt{1} \}$, $V = \{S\}$, and start symbol S.

1. $S \to \epsilon$
2. $S \to \mathtt{0}$
3. $S \to \mathtt{1}$
4. $S \to \mathtt{0}\, S\, \mathtt{0}$
5. $S \to \mathtt{1}\, S\, \mathtt{1}$

Common Notational Conventions

To more effectively communicate and spare tedious repetition, it is convenient to establish some notational conventions.

1. Lower-case letters near the beginning of the alphabet, a, b, and so on, are terminal symbols. We shall also assume that non-letter symbols, like digits, $+$, are terminal symbols.
2. Upper-case letters near the beginning of the alphabet, A, B, and so on, are nonterminal symbols. Often the start symbol of a grammar is assumed to be named S, or sometimes it is the nonterminal symbol on the left-hand side of the first production.
3. Lower-case letters near the end of the alphabet, such as w or z, are ordered sequences (possibly empty) of strings of terminal symbols
4. Upper-case letters near the end of the alphabet, such as X or Y, are a single symbol, either a terminal or a nonterminal symbol.
5. Lower-case Greek letters, such as α and β, are ordered sequences sequences (possibly empty) of terminal and nonterminal symbols.

Common Notational Conventions (Recap)

To more effectively communicate and spare tedious repetition, it is convenient to establish some notational conventions.

1. a, b, c, ... are terminals.
2. A, B, C, ... are nonterminals.
3. ..., X, Y, Z are terminal or nonterminal symbols.
4. ..., w, x, y, z are strings/sequences of terminals only.
5. α, β, γ, ... are strings/sequences of terminals or nonterminals.

Compact Notation

If the RHS of a production is a sequence with no symbols in it, we use ϵ to communicate that fact clearly. So, for example, $A \rightarrow \epsilon$ is a production.

Definition

A production in a grammar in which the RHS of a production is a sequence of length zero is called an ϵ-production.

If several productions have the same LHS like these: $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$, $\ldots$,$A \rightarrow \alpha_n$ it is convenient to write then using the following notation:

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

Definition

And, all the productions with the LHS nonterminal system A are sometimes called A-productions.

Compact Notation

It is convenient to think of a production [in a CFG] as "belonging" to the variable [nonterminal] of its head [LHS]. We shall often use remarks like "the productions for A" or "A-productions" to refer to the production whose head [LHS] is [the] variable A. We may write the productions for a grammar by listing each variable [nonterminal] once, and then listing all the [RHS] bodies of the productions for that variable [nonterminal], separated by vertical bars.

HMU 3rd, §5.1, page 175.

Derivation

A grammar $G = \langle T, V, P, S \rangle$ gives rise naturally to a method of constructing strings of terminal and nonterminal symbols by application of the productions very much like that of inductive sets.

Definition

If $\alpha, \beta \in (T \cup V)^*$, we say that α *derives* β *in one step*, or β *is derivable from* α *in one step* and we write

$$\alpha \overset{1}{\Rightarrow}_G \beta$$

if β can be obtained from α by replacing some occurrence of the substring δ in α with γ, where $\delta \to \gamma$ is a production of G.

Derivation

Perhaps we can recast the previous definition slightly more perspicuously as follows:

Definition

If $\delta \to \gamma$ is a production of G, then $\alpha_1\delta\alpha_2$ *derives* $\alpha_1\gamma\alpha_2$ *in one step*, or $\alpha_1\gamma\alpha_2$ *is derivable from* $\alpha_1\delta\alpha_2$ *in one step*. We write

$$\alpha_1\delta\alpha_2 \overset{1}{\Rightarrow}_G \alpha_1\gamma\alpha_2$$

for all $\alpha_1, \alpha_2 \in (T \cup V)^*$.

We usually omit the the subscript G for the grammar, and write simply:

$$\alpha \overset{1}{\Rightarrow} \beta$$

and leave it to the reader to figure out which grammar is meant.

Also, we sometimes omit the 1 above the arrow, by writing simply:

$$\alpha \Rightarrow \beta$$

though these may lead to confusion with the relation defined next.

Derivation

Definition

Let $\stackrel{*}{\Rightarrow}_G$ be the reflexive, transitive closure of the $\stackrel{1}{\Rightarrow}_G$ relation. That is:

$$\begin{aligned} \alpha &\stackrel{*}{\Rightarrow} \alpha && \text{for any } \alpha \\ \alpha &\stackrel{*}{\Rightarrow} \beta && \text{if } \alpha \stackrel{*}{\Rightarrow} \gamma \text{ and } \gamma \stackrel{1}{\Rightarrow} \beta \end{aligned}$$

This relation is sometimes called the "derives in zero or more steps" relation, or simple the "derives" relation.

Derivation – Inductively Defined Relation

The transitive closure is an inductively defined set (given below in a Post system) and so we can easily prove things about the derives relations.

$$\frac{\delta \to \gamma \in P}{\alpha_1 \delta \alpha_2 \overset{1}{\Rightarrow} \alpha_1 \gamma \alpha_2}$$

$$\frac{}{\alpha \overset{*}{\Rightarrow} \alpha} \qquad \frac{\alpha \overset{*}{\Rightarrow} \gamma \quad \gamma \overset{1}{\Rightarrow} \beta}{\alpha \overset{*}{\Rightarrow} \beta}$$

$\alpha, \alpha, \alpha, \beta, \delta, \gamma$ are arbitrary strings in $(T \cup V)^*$

Language Generated by G

A string in $(T \cup V)^*$ that is derivable from the start symbol S is called a *sentential form*. A sentential form is called a *sentence*, if it consists only of terminal symbols.

Definition

The *language generated by* G, denoted $\mathscr{L}(G)$, is the set of all sentences:

$$\mathscr{L}(G) = \{\, x \in T^* \mid S \overset{*}{\Rightarrow}_G x \,\}$$

Example

$$
\begin{array}{ll}
1 & S \rightarrow \epsilon \\
2 & S \rightarrow 0 \\
3 & S \rightarrow 1 \\
4 & S \rightarrow 0\,S\,0 \\
5 & S \rightarrow 1\,S\,1
\end{array}
$$

For this grammar G we have the following derivation:

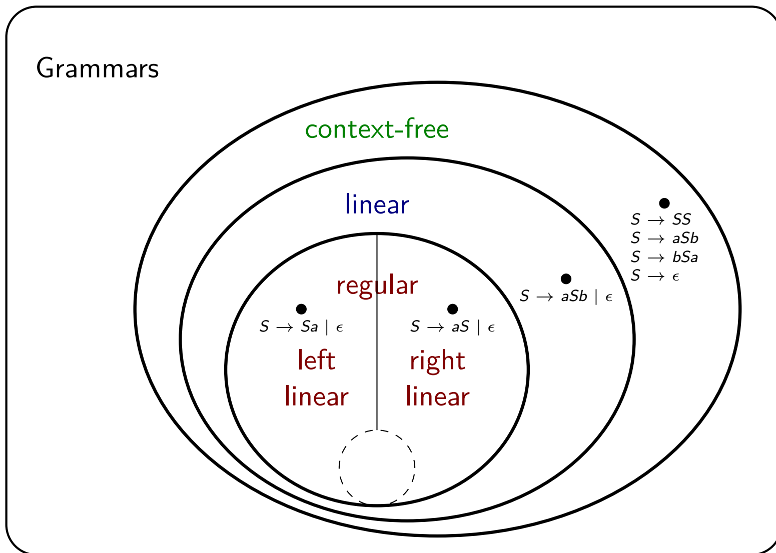
$$S \stackrel{1}{\Rightarrow} 0S0 \stackrel{1}{\Rightarrow} 01S10 \stackrel{1}{\Rightarrow} 01010$$

So, $S \stackrel{*}{\Rightarrow} 01010$, and we conclude that $01010 \in \mathscr{L}(G)$.

There are many different kinds of grammars with slightly different restrictions on the kinds of productions that are permitted.

Completely general grammars are rare. We see examples only briefly and only much later.

The previous example was that of a particular kind of grammar known as a context-free grammar.



Classification of Some Grammars (not languages)

Different Kinds of Grammars

Unrestricted grammar

- Context-sensitive grammar
- Context-free grammar
  - Grammar in Greibach normal form
    - simple-grammar, s-grammar
  - Grammar in Chomsky normal form
  - Linear grammar
    - Regular grammar
      - right-linear grammar.
      - left-linear grammar.
- Compiler theory: LL(k), LR(k) etc.

Grammars Definitions in Linz 6th

Grammar (Linz 6th, §1.2, definition 1.1, page 21).

- Context-sensitive grammar (Linz 6th, §11.4, definition 11.4, page 300).
- Context-free grammar (Linz 6th, §5.1, definition 5.1, page 130).
  - Greibach normal form (Linz 6th §6.2, definition 6.5, page 174).
    - simple-grammar, s-grammar (Linz 6th, §5.3, definition 5.4, page 144).
  - Chomsky normal form (Linz 6th, §6.2, definition 6.4, page 171).
  - Linear grammar (Linz 6th, §3.3, page 93).
    - Regular grammar (Linz 6th, §3.3, definition 3.3, page 92).
      - right-linear grammar (Linz 6th, §3.3, definition 3.3, page 92).
      - left-linear grammar (Linz 6th, §3.3, definition 3.3, page 92).
- Compiler theory: LL(k) (Linz 6th, §7.4, definition 7.5, page 210); LR(k) [Not defined in Linz 6th.]

Grammars Definitions in HMU 3rd

- Context-sensitive grammar [Not defined in HMU].
- Context-free grammar (HMU 3rd, §5.1.2, page 173).
  - Greibach normal form (HMU 3rd, §7.1, page 277).
    - simple-grammar, s-grammar [Not defined in HMU].
  - Chomsky normal form (HMU 3rd, §7.1.5, page 272).
  - Linear grammar [Not defined in HMU].
    - Regular grammar [Not defined in HMU].
      - right-linear grammar (HMU 3rd, §5.1.7, exercise 5.1.4, page 182).
      - left-linear grammar.
- Compiler theory: LL(k), LR(k) etc. [Not defined in HMU].

Restrictions on Grammars

- Context-sensitive grammar. Each production $\alpha \rightarrow \beta$ restricted to $\mathsf{len}(\alpha) \leq \mathsf{len}(\beta)$
- Context-free grammar. Each production $A \rightarrow \beta$ where $A \in N$
  - Grammar in Greibach normal form. $A \rightarrow a\gamma$ where $a \in T$ and $\gamma \in V^*$
    - simple-grammar or s-grammar. Any pair $\langle A, a \rangle$ occurs at most once in the productions
  - Grammar in Chomsky normal form. $A \rightarrow BC$ or $A \rightarrow a$
  - Linear grammar. RHS β contains at most one nonterminal
    - Regular grammar, either:
      • right-linear grammar. The nonterminal (if any) occurs to the right of (or after) any terminals
      • left-linear grammar. The nonterminal (if any) occurs to the left of (or before) any terminals

If $\epsilon \in L(G)$ we must allow a niggly exception $S \rightarrow \epsilon$.

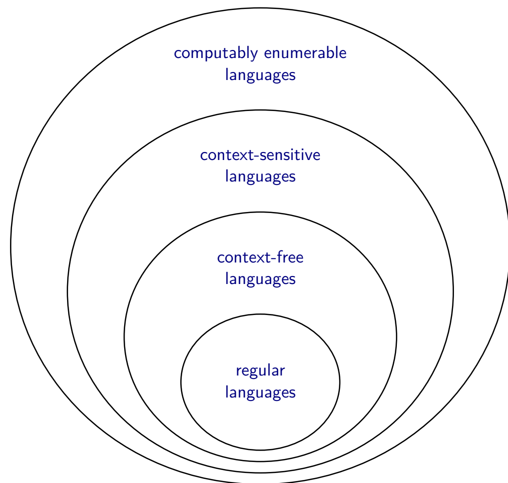
The s-languages are those languages recognized by a particular restricted form of deterministic pushdown automaton, called an s-machine. They are uniquely characterized by that subset of the standard-form grammars in which each rule has the form $Z \rightarrow aY_1 \dots Y_n, n \geq 0$, *and for which the pairs* (Z, a) *are distinct among the rules. It is shown that the s-languages have the prefix property, and that they include the regular sets with end-markers. Finally, their closure properties and decision problems are examined, and it is found that their equivalence problem is solvable.*

Korenja, Hopcroft, Simple deterministic languages, 1968.

right-linear	$A \to xB$ or $A \to x$	$A, B \in V,\ x \in T^*$
strongly right-linear	$A \to aB$ or $A \to \epsilon$	$A, B \in V,\ a \in T$
left-linear	$A \to Bx$ or $A \to x$	$A, B \in V,\ x \in T^*$
strongly left-linear	$A \to Ba$ or $A \to \epsilon$	$A, B \in V,\ a \in T$

Grammars Characterize The Chomsky Hierarchy

Like automata, grammars can be used used to characterize the formal languages of the Chomsky Hierarchy.



Grammars and Automata

defining	other

DFAs are equivalent to regular grammars

CFGs are equivalent to PDAs

LBAs are equivalent to context-sensitive grammars

TMs are equivalent to (unrestricted) grammars

Grammars and Automata

defining	other

DFAs are equivalent to regular grammars
CFGs are equivalent to PDAs
LBAs are equivalent to context-sensitive grammars
TMs are equivalent to (unrestricted) grammars

Grammars and Automata

defining	other

DFAs are equivalent to regular grammars
CFGs are equivalent to PDAs
LBAs are equivalent to context-sensitive grammars
TMs are equivalent to (unrestricted) grammars

Context-free grammars (CFG) are significant because they are:

1. natural and simple,
2. expressive enough for most purposes, and
3. easily parsed.

Grammars and Automata

defining	other

DFAs are equivalent to regular grammars
CFGs are equivalent to PDAs
LBAs are equivalent to context-sensitive grammars
TMs are equivalent to (unrestricted) grammars

Linear Bounded Automata (LBA)

Context-free grammars (CFG) are significant because they are:

1. natural and simple,
2. expressive enough for most purposes, and
3. easily parsed.

of lesser significance

Grammars and Automata

defining	other

DFAs are equivalent to regular grammars
CFGs are equivalent to PDAs
LBAs are equivalent to context-sensitive grammars
TMs are equivalent to (unrestricted) grammars

Turing machine (TM)

of lesser significance

Context-free grammars (CFG) are significant because they are:

1. natural and simple,
2. expressive enough for most purposes, and
3. easily parsed.

Definition of Grammar (Recap)

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

Definition of Grammar (Recap)

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is the finite set of terminal symbols;

Definition of Grammar (Recap)

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is the finite set of terminal symbols;
- V is the finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;

Definition of Grammar (Recap)

A grammar is a 4-tuple $\langle T, V, P, S \rangle$:

- T is the finite set of terminal symbols;
- V is the finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;

Definition of Grammar (Recap)

A grammar is a 4-tuple $\langle T, V, P, S\rangle$:

- T is the finite set of terminal symbols;
- V is the finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;
- P is the finite set of productions.

Definition of Grammar (Recap)

A grammar is a 4-tuple $\langle T, V, P, S\rangle$:

- T is the finite set of terminal symbols;
- V is the finite set of nonterminal symbols, also called variables or syntactic categories, $T \cap V = \emptyset$;
- $S \in V$ is a distinguished nonterminal symbol, called the start symbol;
- P is the finite set of productions.

A production has the form $\alpha \to \beta$ where α and β are ordered sequences (strings) of terminals and nonterminals symbols. (We write $(T \cup V)^*$ for the set of ordered sequences of terminal and nonterminal symbols.) The LHS α of a production can't be the empty sequence, but β might be.

Linear Grammars

Linear grammar. Context-free (LHS a single nonterminal); RHS β contains at most one nonterminal (β possibly ϵ).

- Regular grammar, either:
  - right-linear grammar. The nonterminal (if any) occurs to the right of (or after) any terminals
  - left-linear grammar. The nonterminal (if any) occurs to the left of (or before) any terminals

All linear grammars can put in a form with all RHS with exactly one terminal.

$$A \to abBcd \qquad \begin{aligned} A &\to aX \\ X &\to bY \\ Y &\to Zd \\ Z &\to Bc \end{aligned}$$

No epsilon productions (for $L - \{\epsilon\}$) and no unit productions.

Linz 6th, Theorem 3.3 and 3.4, page 94 and 96
HMU 3rd, Exercise 5.1.4, page 182.
Du & Ko, Theorem 3.10, page 96.

Theorem (Linz 6th, Theorem 3.3, page 94)

For all right-linear grammars G, the language $L(G)$ is regular.

Theorem (Linz 6th, Theorem 3.4, page 96)

If language L is regular, then there is a right-linear grammar G such that $L(G) = L$.

So the set of all languages generated by right-linear grammars is equal to the set of regular languages.

Proof Sketch

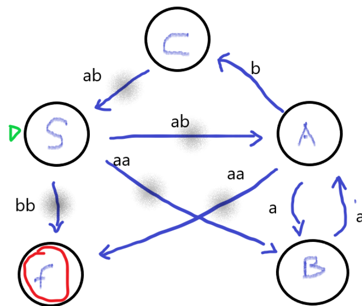
There is a two-way construction (with some niggly details) inking a right-linear grammar and a DFA. We equate (pretty much) the set of nonterminal symbols in the grammar with the states in the DFA.

Let $G = \langle \Sigma, Q, S, P \rangle$ be a right-linear grammar. Let $M = \langle \Sigma, Q \cup \{q_f\}, \delta, S, \{q_f\} \rangle$ be a NFA. Without loss of generality assume the productions in G have at most one terminal in them and M has a unique final state q_f reachable only by ϵ moves. We equate the set of grammar terminals with the alphabet; and the set of grammar nonterminals with the set of states of the automata. We associate the production $A \to aB$ with the transition $A \xrightarrow{a} B$. And we associate the production $A \to a$ with the transition $A \xrightarrow{a} q_f$ for a final state q_f.

The key insight is

$$S \overset{*}{\Rightarrow} wA \quad \text{iff} \quad \delta^*(S, w) = A$$

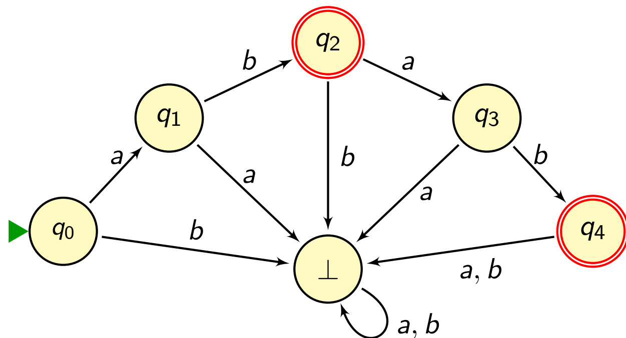
$S \rightarrow abA \mid aaB \mid bb$
$A \rightarrow aB \mid bC \mid aa$
$B \rightarrow aA \mid bB$
$C \rightarrow abS$



$$S \overset{*}{\Rightarrow} abA \overset{*}{\Rightarrow} abaB \overset{*}{\Rightarrow} abaaA \overset{*}{\Rightarrow} abaabC \overset{*}{\Rightarrow} abaababS \overset{*}{\Rightarrow} abaababbb$$

$$S \xrightarrow{ab} A \xrightarrow{a} B \xrightarrow{a} A \xrightarrow{b} C \xrightarrow{ab} S \xrightarrow{bb} f$$

Right Linear Grammars



$$
\begin{aligned}
q_0 &\to aq_1 \mid b\bot \\
q_1 &\to a\bot \mid bq_2 \mid b \\
q_2 &\to aq_3 \mid b\bot \\
q_3 &\to a\bot \mid bq_4 \mid b \\
q_4 &\to a\bot \mid b\bot \\
\bot &\to a\bot \mid b\bot
\end{aligned}
$$

Left-linear Grammars?

What about left-linear grammars?

The set of languages they generate is also the set of regular languages.

We prove it indirectly via reversal, using that fact that reversing a left-linear grammar is a right-linear grammar and vice versa.

Reversing an NFA

Theorem

If language L is regular, then L^R is regular.

Proof.

Given an NFA M for L, we construct an NFA M' by reversing the transitions. $L(M') = L^R$. □

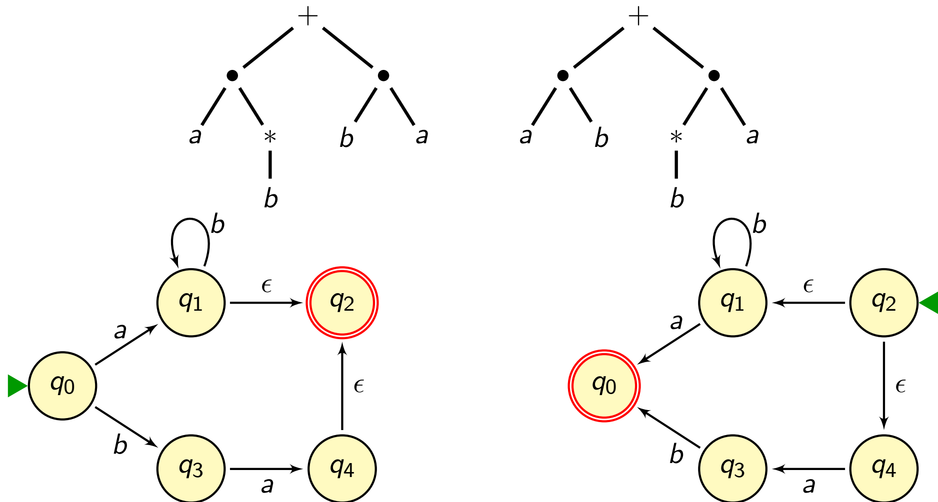
Theorem

If L^R is regular, then L is regular.

Proof.

Since by the previous theorem $(L^R)^R$ is regular and $(L^R)^R = L$, we have L is regular. □

Reversing Example



$L = \mathscr{L}[\![ab^* + ba]\!]$ and $L^R = \mathscr{L}[\![ab + b^*a]\!]$

Reversing a Linear Grammar

Theorem

For all right-linear grammars G, there is a left-linear grammar $G^{\triangleleft}$ such that $L(G^{\triangleleft}) = L(G)^R$

Proof.

Reversing the RHSs of the productions in G constructs a left-linear grammar $G^{\triangleleft}$. Since $S \overset{*}{\Rightarrow}_G \alpha$ iff $S \overset{*}{\Rightarrow}_{G^{\triangleleft}} \alpha^R$, we have $L(G) = L(G^{\triangleleft})^R$. □

Reversing a Linear Grammar

Theorem

For all left-linear grammars G, there is a right-linear grammar $G^{\triangleright}$ such that $L(G^{\triangleright}) = L(G)^R$

Proof.

Reversing the RHSs of the productions in G constructs a left-linear grammar $G^{\triangleright}$. Since $S \overset{*}{\Rightarrow}_G \alpha$ iff $S \overset{*}{\Rightarrow}_{G^{\triangleright}} \alpha^R$, we have $L(G) = L(G^{\triangleleft})^R$. □

Reversing Example

$$S \to bA \qquad S \to Ab$$
$$S \to aB \qquad S \to Ba$$
$$A \to aC \qquad A \to Ca$$
$$B \to bB \qquad B \to Bb$$
$$B \to F \qquad B \to F$$
$$C \to F \qquad C \to F$$
$$F \to \epsilon \qquad F \to \epsilon$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

Reversing Example

$S \to bA$	$S \to Ab$
$S \to aB$	$S \to Ba$
$A \to aC$	$A \to Ca$
$B \to bB$	$B \to Bb$
$B \to F$	$B \to F$
$C \to F$	$C \to F$
$F \to \epsilon$	$F \to \epsilon$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

S

S

Reversing Example

$$S \to bA \qquad S \to Ab$$
$$S \to aB \qquad S \to Ba$$
$$A \to aC \qquad A \to Ca$$
$$B \to bB \qquad B \to Bb$$
$$B \to F \qquad B \to F$$
$$C \to F \qquad C \to F$$
$$F \to \epsilon \qquad F \to \epsilon$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

$$S \stackrel{1}{\Rightarrow} aB$$

$$S \stackrel{1}{\Rightarrow} Ba$$

Reversing Example

$$S \to bA \qquad\qquad S \to Ab$$
$$S \to aB \qquad\qquad S \to Ba$$
$$A \to aC \qquad\qquad A \to Ca$$
$$B \to bB \qquad\qquad B \to Bb$$
$$B \to F \qquad\qquad B \to F$$
$$C \to F \qquad\qquad C \to F$$
$$F \to \epsilon \qquad\qquad F \to \epsilon$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

$$S \overset{1}{\Rightarrow} aB \overset{1}{\Rightarrow} abB$$

$$S \overset{1}{\Rightarrow} Ba \overset{1}{\Rightarrow} Bba$$

Reversing Example

$$S \to bA \qquad S \to Ab$$
$$S \to aB \qquad S \to Ba$$
$$A \to aC \qquad A \to Ca$$
$$B \to bB \qquad B \to Bb$$
$$B \to F \qquad B \to F$$
$$C \to F \qquad C \to F$$
$$F \to \epsilon \qquad F \to \epsilon$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

$$S \stackrel{1}{\Rightarrow} aB \stackrel{1}{\Rightarrow} abB \stackrel{1}{\Rightarrow} abbB$$

$$S \stackrel{1}{\Rightarrow} Ba \stackrel{1}{\Rightarrow} Bba \stackrel{1}{\Rightarrow} Bbba$$

Reversing Example

$$S \to bA \qquad S \to Ab$$
$$S \to aB \qquad S \to Ba$$
$$A \to aC \qquad A \to Ca$$
$$B \to bB \qquad B \to Bb$$
$$B \to F \qquad B \to F$$
$$C \to F \qquad C \to F$$
$$F \to \epsilon \qquad F \to \epsilon$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

$$S \overset{1}{\Rightarrow} aB \overset{1}{\Rightarrow} abB \overset{1}{\Rightarrow} abbB \overset{1}{\Rightarrow} abbF$$

$$S \overset{1}{\Rightarrow} Ba \overset{1}{\Rightarrow} Bba \overset{1}{\Rightarrow} Bbba \overset{1}{\Rightarrow} Fbba$$

Reversing Example

$$\begin{aligned} S &\to bA \\ S &\to aB \\ A &\to aC \\ B &\to bB \\ B &\to F \\ C &\to F \\ F &\to \epsilon \end{aligned} \qquad \begin{aligned} S &\to Ab \\ S &\to Ba \\ A &\to Ca \\ B &\to Bb \\ B &\to F \\ C &\to F \\ F &\to \epsilon \end{aligned}$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

$$S \overset{1}{\Rightarrow} aB \overset{1}{\Rightarrow} abB \overset{1}{\Rightarrow} abbB \overset{1}{\Rightarrow} abbF \overset{1}{\Rightarrow} abb$$

$$S \overset{1}{\Rightarrow} Ba \overset{1}{\Rightarrow} Bba \overset{1}{\Rightarrow} Bbba \overset{1}{\Rightarrow} Fbba \overset{1}{\Rightarrow} bba$$

Reversing Example

$$S \to bA \qquad S \to Ab$$
$$S \to aB \qquad S \to Ba$$
$$A \to aC \qquad A \to Ca$$
$$B \to bB \qquad B \to Bb$$
$$B \to F \qquad B \to F$$
$$C \to F \qquad C \to F$$
$$F \to \epsilon \qquad F \to \epsilon$$

$$L = \{ab^n \mid 0 \leq n\} \cup \{ba\} \quad \text{and} \quad L^R = \{ab\} \cup \{b^n a \mid 0 \leq n\}$$

$$S \stackrel{1}{\Rightarrow} aB \stackrel{1}{\Rightarrow} abB \stackrel{1}{\Rightarrow} abbB \stackrel{1}{\Rightarrow} abbF \stackrel{1}{\Rightarrow} abb$$

$$S \stackrel{1}{\Rightarrow} Ba \stackrel{1}{\Rightarrow} Bba \stackrel{1}{\Rightarrow} Bbba \stackrel{1}{\Rightarrow} Fbba \stackrel{1}{\Rightarrow} bba$$

So $aab \in L$, $bba \in L^R$, and $(abb)^R = bba$.

Theorem

A language L generated by a left-linear grammar is regular.

Proof.

- Given a left-linear grammar for L we can construct a right-linear grammar generating L^R.
- Since L^R is generated by a right-linear grammar, it is regular (Thm 3.3),
- Therefore L is also regular.

Theorem

If a language L is regular, then there is a left-linear grammar for it.

Proof.

- Since L is regular, then L^R is regular.
- Since L^R is regular, then there is a right-linear grammar for it (Thm 3.4)
- Then there is a left-linear grammar for $(L^R)^R = L$.

Theorem

If a language L is regular, then there is a regular grammar for it.

Theorem

A language L generated by a regular grammar is regular.

Theorem

If a language L is regular, then there is a regular grammar for it.

Theorem

A language L generated by a regular grammar is regular.

Theorem

A language L is regular iff it is generated by a regular grammar.

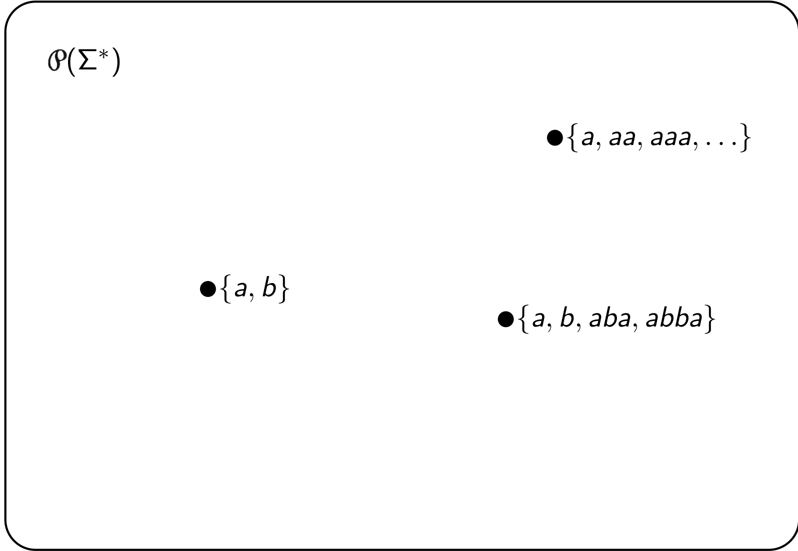
A language L is regular if, and only if,

- there is a DFA for it (by definition)
- there is an NFA for it
- there is a regular expression for it
- there is a right-linear grammar for it
- there is a left-linear grammar for it
- there is a regular grammar for it

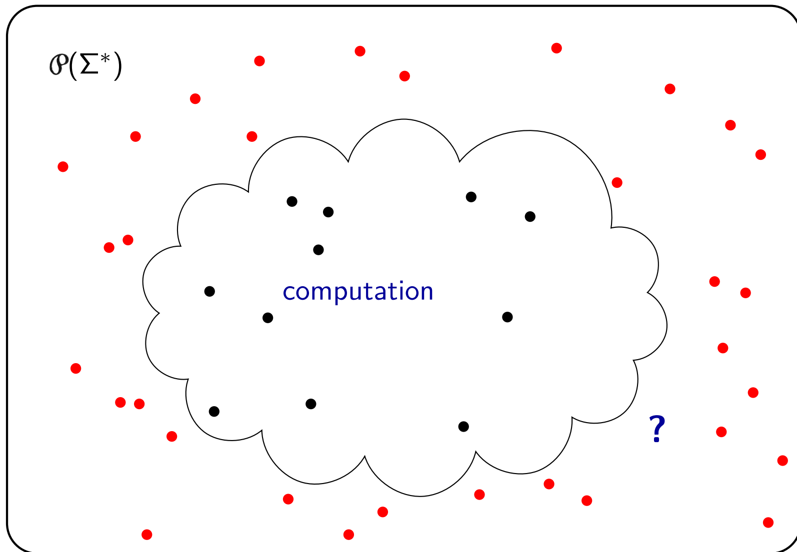
$\wp(\Sigma^*)$

$\bullet\{a, b, aba, abba\}$

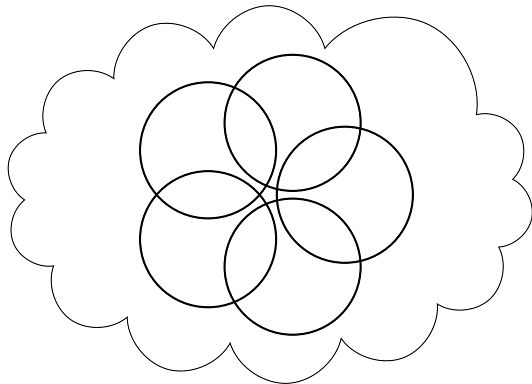
The universe of formal languages



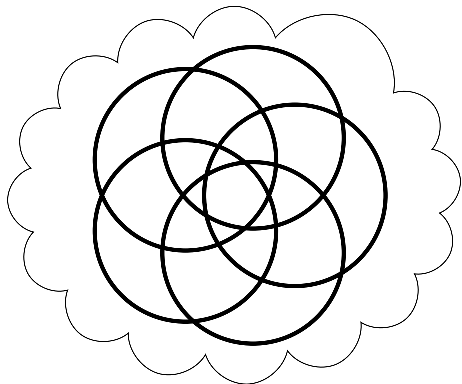
Each element (dot) is a formal language (over $\Sigma = \{a, b\}$)



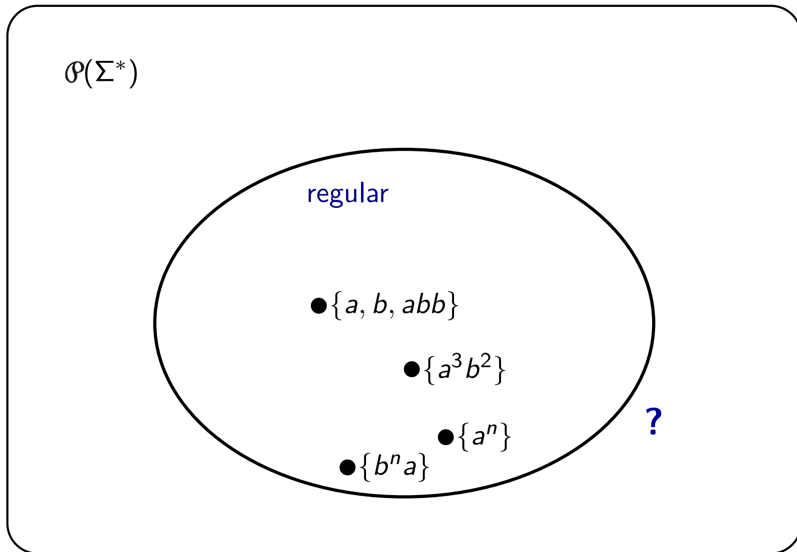
Can we characterize computation?



Many different models: automata, expression, grammars, and so on.



Models of computation might have little relationship to each other.



Regular languages seem important

$\wp(\Sigma^*)$

$\bullet \{a^n b^n \mid 0 \leq n\}$

other languages

regular languages

$\bullet \{a, aba\}$

$\bullet \{a^n b\}$

Need to keep exploring

Elsewhere

- Closure Properties of Regular Languages
- Decision Procedures for Regular Languages
- Pumping Lemma for Regular Languages