Context-Free Grammars

https://web.stanford.edu/class/archive/cs/cs103/cs103.1164/lectures/18/

Describing Languages

- We've seen two models for the regular languages:
  - ***Finite automata*** accept precisely the strings in the language.
  - ***Regular expressions*** describe precisely the strings inthe language.
- Finite automata ***recognize*** strings in the language.
  - Perform a computation to determine whether a specific string is in the language.
- Regular expressions ***match*** strings in the language.
  - Describe the general shape of all strings in the language.

Context-Free Grammars

A *context-free grammar* (or *CFG*) is an entirely different formalism for defining a class of languages.

Goal: Give a procedure for listing off all strings in the language.

CFGs are best explained by example...

Arithmetic Expressions

Suppose we want to describe all legal arithmetic expressions using addition,subtraction, multiplication, and division.

Here is one possible CFG:

E → `int`

E → **E Op E**

E → `(`**E**`)`

Op → `+`

Op → `-`

Op → `*`

Op → `/`

E

⇒ **E Op E**

⇒ **E Op** `int`

⇒ `int` **Op** `int`

⇒ `int / int`

Context -Free Grammars

- Formally, a context-free grammar is a collection of four objects:
  - A set of *nonterminalsymbols* (also called *variables*),
  - A set of *terminal symbols* (the *alphabet* of the CFG)
  - A set of *production rules* saying how each nonterminal can be replaced by a string of terminals and nonterminals, and
  - A *start symbol* (which must be a nonterminal) that begins the derivation.

E → int

E → E OpE

E → (E)

Op → +

Op → -

Op → *

Op → /

Some CFG Notation

- Capitalletters in **BoldRedUppercase** will represent nonterminals.
  - i.e. **A**, **B**, **C**, **D**
- Lowercase letters in **blue monospace** `willl` represent terminals.
  - i.e. **t**, **u**, **v**, **w**
- Lowercase Greek letters in *gray italics* will represent arbitrary strings of terminals and nonterminals.
  - i.e. α, γ, ω

A Notational Shorthand

$$E \rightarrow \texttt{int}$$

$$E \rightarrow E\ Op\ E$$

$$E \rightarrow \texttt{(}E\texttt{)}$$

$$Op \rightarrow \texttt{+}$$

$$Op \rightarrow \texttt{-}$$

$$Op \rightarrow \texttt{*}$$

$$Op \rightarrow \texttt{/}$$

A Notational Shorthand

$$E \rightarrow \text{int} \mid E\ Op\ E \mid (\ E\ )$$

$$Op \rightarrow + \mid - \mid * \mid /$$

Derivations

E → E Op E | int | (E)
Op → + | * | - | /

E
⇒ E Op E
⇒ E Op (E)
⇒ E Op (E Op E)
⇒ E * (E Op E)
⇒ int * (E Op E)
⇒ int * (int Op E)
⇒ int * (int Op int)
⇒ int * (int + int)

A sequence of steps where nonterminals are replaced by the right-hand side of a production is called a ***derivation***.

If string α derives string ω, we write $\alpha \Rightarrow^* \omega$.

In the example on the left, we see **E** $\Rightarrow^*$ `int * (int + int)`.

The Language of a Grammar

If G is a CFG with alphabet Σ and start symbol **S**, then the ***language of G*** is the set

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

That is, $\mathcal{L}(G)$ is the set of strings derivable from the start symbol.

Note: w must be in Σ^*, the set of strings made from terminals. Strings involving nonterminals aren't in the language.

Context-Free Languages

A language L is called a ***context-free language*** (or CFL) if there is a CFG G such that $L = \mathcal{L}(G)$. Questions:

What languagesare context-free?

How are context-free and regular languages related?

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$.

They do not have the regular expression operators * or □.

However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow a^*b$$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$.

They do not have the regular expression operators * or □.

However, we can convert regular expressions to CFGs as follows:

$S \rightarrow a^*b$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or □. However, we can convert regular expressions to CFGs as follows:

$S \rightarrow A\ b$

$A \rightarrow A\ a \mid \varepsilon$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$.

They do not have the regular expression operators * or □. However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow a(b \cup c^*)$$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$.

They do not have the regular expression operators * or □. However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow aX$$

$$X \rightarrow (b \cup c^*)$$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or □.

However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow aX$$

$$X \rightarrow b \cup c^*$$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or □.

However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow aX$$

$$X \rightarrow b \mid c^*$$

From Regexes to CFGs

CFGs consist purely of production rules of the form $A \rightarrow \omega$. They do not have the regular expression operators * or □. However, we can convert regular expressions to CFGs as follows:

$$S \rightarrow aX$$

$$X \rightarrow b \mid C$$

From Regexes to CFGs

FGs consist purely of production rules

f the form $A \rightarrow \omega$.

hey do not have the regular expression

perators * or □. However, we can convert

egular expressions to CFGs as follows:

$S \rightarrow aX$

$X \rightarrow b \mid C$

$C \rightarrow C\ c \mid \varepsilon$

Regular Languages and CFLs

Theorem : Every regular language is context-free.

Proof Idea: Use the construction from the previous slides to convert a regular expression for L into a CFG for L. ■

Problem Set Exercise: Instead, show how to convert a DFA/NFA into a CFG.

The Language of a Grammar

- Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

- What strings can this generate?

The Language of a Grammar

Consider the following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?

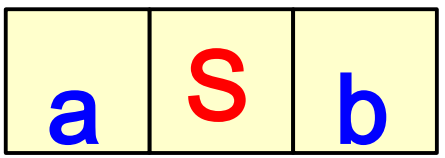
S

The Language of a Grammar

- Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

- What strings can this generate?



The Language of a Grammar

- Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

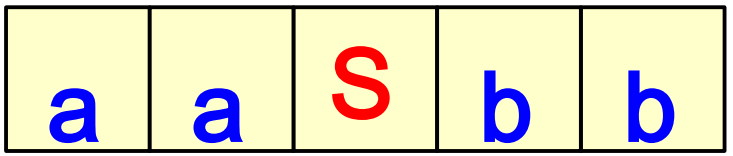
- What strings can this generate?

The Language of a Grammar

- Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

- What strings can this generate?



The Language of a Grammar

Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

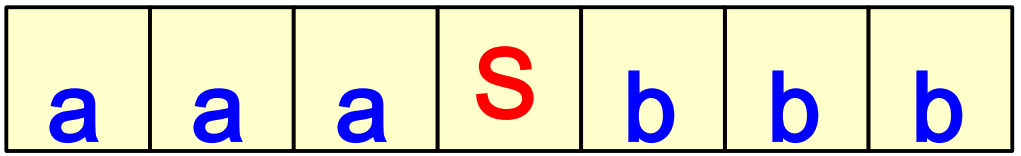
What strings can this generate?

The Language of a Grammar

Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?

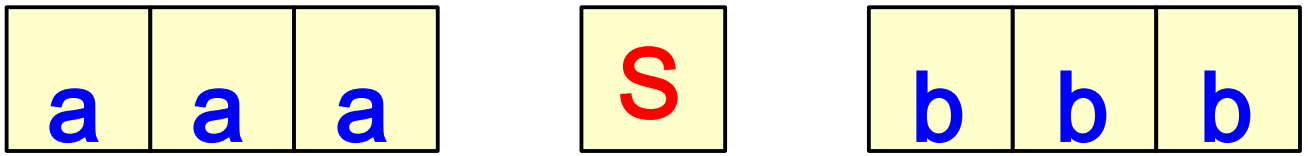


The Language of a Grammar

- Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?

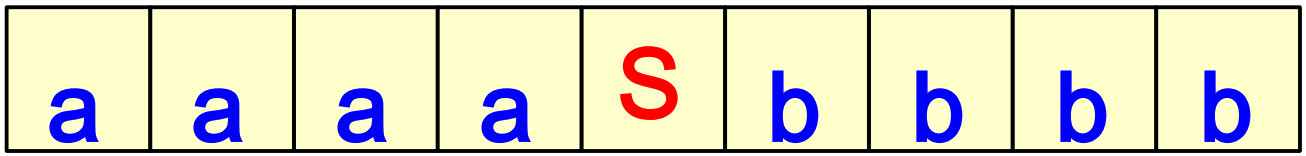


The Language of a Grammar

Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?

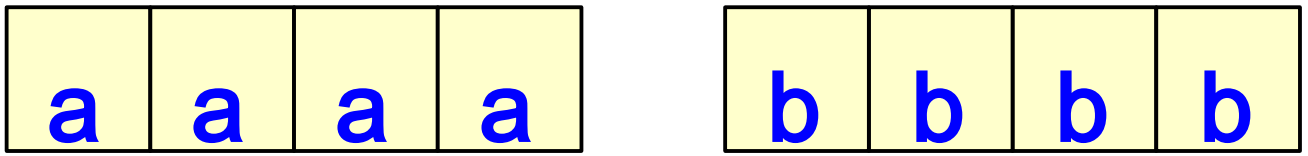


The Language of a Grammar

Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?



The Language of a Grammar

Considerthe following CFG *G*:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?

a	a	a	a	b	b	b	b

bv

The Language of a Grammar

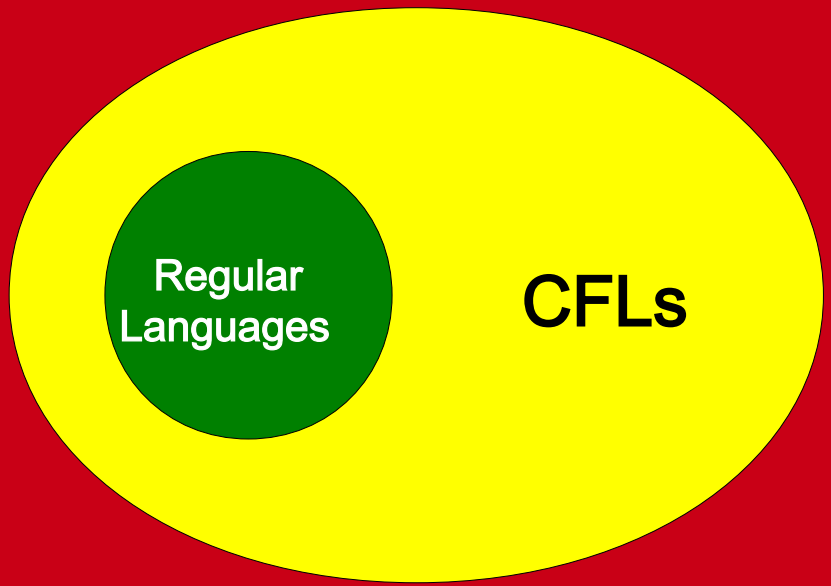
Considerthe following CFG G:

$$S \rightarrow aSb \mid \varepsilon$$

What strings can this generate?

a	a	a	a	b	b	b	b

$$\mathcal{L}(G) = \{A^n\ b^n \mid n \in \mathbb{N}\}$$



All Languages

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

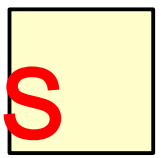
$$S \rightarrow aSb \mid \varepsilon$$

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$$S \rightarrow aSb \mid \varepsilon$$

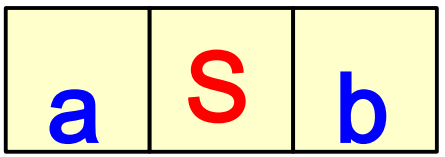


Why the Extra Power?

Whydo CFGshavemore power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$S \rightarrow aSb \mid \varepsilon$



Why the Extra Power?

Whydo CFGshavemore power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$$S \rightarrow aSb \mid \varepsilon$$

a S b

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$$S \rightarrow aSb \mid \varepsilon$$

a a S b b

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

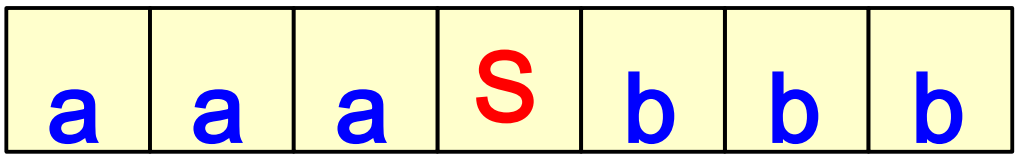
$S \rightarrow aSb \mid \varepsilon$

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$S \rightarrow aSb \mid \varepsilon$



Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

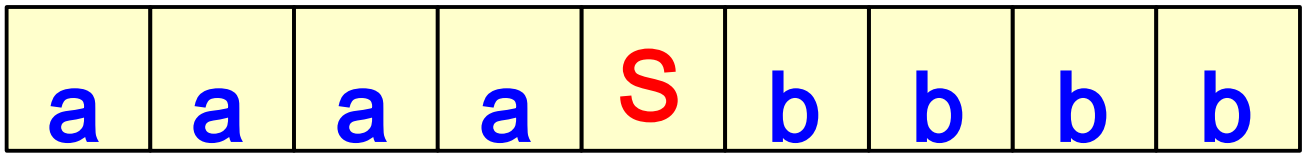
$S \rightarrow aSb \mid \varepsilon$

Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$S \rightarrow aSb \mid \varepsilon$

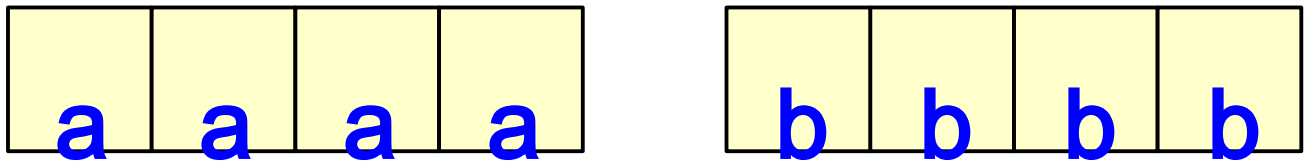


Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$S \rightarrow aSb \mid \varepsilon$



Why the Extra Power?

Why do CFGs have more power than regular expressions?

Intuition: Derivations of strings have unbounded "memory."

$S \rightarrow aSb \mid \varepsilon$