CFGs

David Kauchak
CS30 – Spring 2015

Grammars

What is a grammar?

Grammars

Language view:

A grammar is a set of structural rules that govern the composition of sentences, phrases and words.

Computational view:

A grammar (often called a "formal grammar") is a set of rules that describe what strings are valid in a formal language.

Grammars

What types of (formal) grammars have you heard of before?

Lots of different kinds of grammars:

- regular
- context-free
- context-sensitive
- recursively enumerable
- transformation grammars

Context Free Grammars (CFG)

How many people have heard of them?

What do you know about them?

Where are they used?

CFG production rules

S → NP VP

left hand side (single symbol)

right hand side (one or more symbols)

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

S

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

S

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

A B C

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

A B C

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

A really C

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

A really C

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

A really like cs

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

A really like cs

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

I really like cs

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

Grammars "generate" or "derive" strings:

S → A B C
A → I
B → really
B → really, B
C → like cs

I really like cs

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

No more rules apply, so we're done!

CFG example

S → A B C
A → I
B → really
B → really, B
C → like cs

Grammars "generate" or "derive" strings:

I really like cs

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

Is this the only string that can be derived?

CFG example

S → A B C
A → I
B → really
B → really, B
C → like cs

Grammars "generate" or "derive" strings:

A really, B C

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

S → A B C
A → I
B → really
B → really, B
C → like cs

Grammars "generate" or "derive" strings:

A really, really, B C

We can apply a rule by substituting the symbol on the left hand side with the symbols on the right

CFG example

S → A B C
A → I
B → really
B → really, B
C → like cs

Grammars describe a language, i.e. the strings (aka sentences) that are part of that language

I really, really, … like cs

CFGs formally

G = (**NT**, **T**, **P**, **S**)

NT: finite set of nonterminal symbols

T: finite set of terminal symbols, **NT** and **T** are disjoint

P: finite set of productions of the form
$A \rightarrow \alpha$, $A \in NT$ and $\alpha \in (T \cup NT)^*$

S $\in$ NT: start symbol