UNIT V
ANNOTATING LINGUISTIC STRUCTURE

By
Dr. S. S. Gharde
Dept. of Information Technology/ AIML
Government Polytechnic Nagpur

Contents

Introduction to CFG

- Context-free grammars are the backbone of many formal models of the syntax of natural language
- Syntactic parsing is the task of assigning a syntactic structure to a sentence.
- Parse trees can be used in applications such as grammar checking

Context-Free Grammars

- Context-free grammars are also called phrase-structure grammars, and the formalism is equivalent to Backus-Naur form, or BNF.
- A context-free grammar consists of a set of rules or productions and a lexicon of words and symbols.
- Context-free rules can be hierarchically embedded, so we can combine the previous rules with others
- For example, an NP (or noun phrase) can be composed of either a ProperNoun or a determiner (Det) followed by a Nominal; a Nominal in turn can consist of one or more Nouns.

NP → Det Nominal
NP → ProperNoun
Nominal → Noun | Nominal Noun

Det → a
Det → the
Noun → flight

Context-Free Grammars

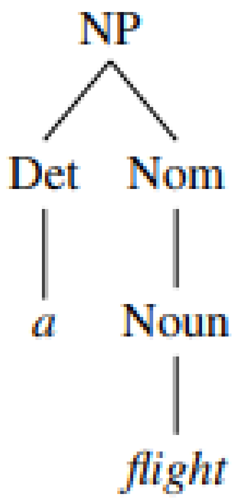
- The symbols that are used in a CFG are divided into two classes: Terminals and Nonterminals
- The symbols that correspond to words in the language are called terminal symbols
- The symbols that express abstractions over these terminals are called non-terminals.

Context-Free Grammars

- a CFG can be used to generate a set of strings. This sequence of rule expansions is called a derivation of the string of words.
- It is common to represent a derivation by a parse tree.

$$
\begin{aligned}
NP &\rightarrow Det\ Nominal \\
NP &\rightarrow ProperNoun \\
Nominal &\rightarrow Noun \mid Nominal\ Noun
\end{aligned}
$$

$$
\begin{aligned}
Det &\rightarrow a \\
Det &\rightarrow the \\
Noun &\rightarrow flight
\end{aligned}
$$

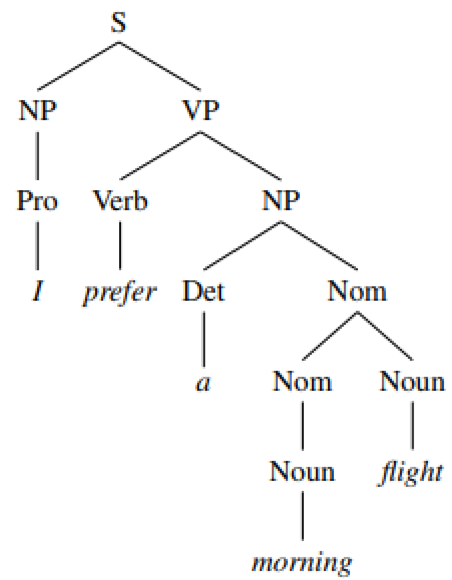


Noun →	*flights* | *flight* | *breeze* | *trip* | *morning*
Verb →	*is* | *prefer* | *like* | *need* | *want* | *fly* | *do*
Adjective →	*cheapest* | *non-stop* | *first* | *latest*
	| *other* | *direct*
Pronoun →	*me* | *I* | *you* | *it*
Proper-Noun →	*Alaska* | *Baltimore* | *Los Angeles*
	| *Chicago* | *United* | *American*
Determiner →	*the* | *a* | *an* | *this* | *these* | *that*
Preposition →	*from* | *to* | *on* | *near* | *in*
Conjunction →	*and* | *or* | *but*

Figure 17.2 The lexicon for $\mathcal{L}_0$.

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

Figure 17.3 The grammar for $\mathcal{L}_0$, with example phrases for each rule.



The parse tree for
“I prefer a morning flight”
according to grammar L0.

Formal Definition of Context-Free Grammar

- A context-free grammar G is defined by four tuples as

$$G = (N, \Sigma, R, S)$$

OR $G = (V, T, P, S)$

Where

- N is a set of non-terminal symbols (or variables V)
- Σ is a set of terminal symbols (T)
- R is a set of rules or productions (P)
  - each of the form $A \rightarrow \beta$
    - where A is a non-terminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$
- S is a designated start symbol and a member of N

Formal Definition of Context-Free Grammar

- General conventions for CFG
- Non-terminals - Capital letters like A, B, and S
- The start symbol - S
- Strings drawn from $(\Sigma \cup N)*$ - Lower-case Greek letters like α, β, and γ
- Strings of terminals - Lower-case Roman letters like u, v, and w

Formal Definition of Context-Free Grammar

- Derivation
- A language is defined through the concept of derivation.
- One string derives another one if it can be rewritten as the second one by some series of rule applications.
- if $A \rightarrow \beta$ is a production of R and α and γ are any strings in the set $(\Sigma \cup N)*$, then $\alpha A \gamma$ directly derives $\alpha\beta\gamma$, or $\alpha A \gamma \Rightarrow \alpha\beta\gamma$.

- Derivation is then a generalization of direct derivation.

- Let $\alpha 1, \alpha 2, ..., \alpha m$ be strings in $(\Sigma \cup N)*$,$m \geq 1$, such that

$$\alpha 1 \Rightarrow \alpha 2, \quad \alpha 2 \Rightarrow \alpha 3, \quad ..., \quad \alpha m{-}1 \Rightarrow \alpha m$$

- We say that $\alpha 1$ derives αm, or $\alpha 1$ derives $\Rightarrow \alpha m$.

Formal Definition of Context-Free Grammar

- Language LG generated by a grammar G can be formally define as the set of strings composed of terminal symbols that can be derived from the designated start symbol S.

$$\mathcal{L}_G = \{w | w \text{ is in } \Sigma^* \text{ and } S \stackrel{*}{\Rightarrow} w\}$$

- The problem of mapping from a string of words to its parse tree is called syntactic parsing.

Treebanks

- A corpus in which every sentence is annotated with a parse tree is called a treebank.
- Treebanks play an important role in parsing as well as in linguistic investigations of syntactic phenomena.
- Treebanks are generally made by parsing each sentence with a parse that is then hand-corrected by human linguists.
- The sentences in a treebank implicitly constitute a grammar of the language.

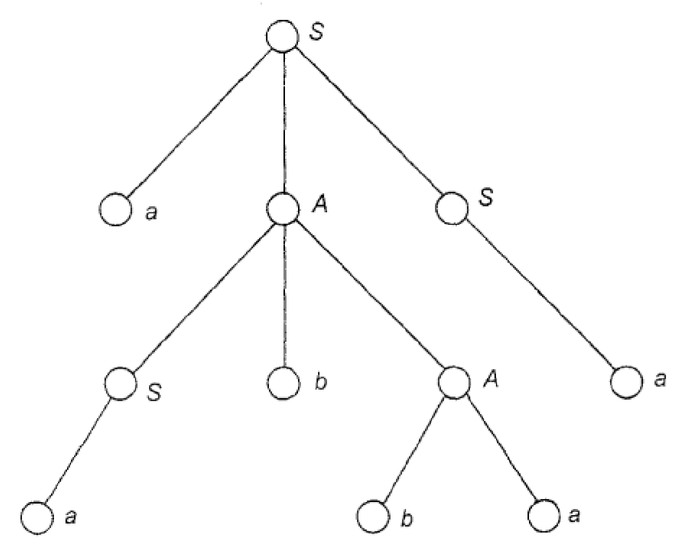
Examples

- 1. Show that S ➔ aabbaa and construct a derivation tree whose yield is aabbaa. Consider the G whose productions are

  S → aAS | a

  A → SbA | SS | ba

- Solution:
- S == > aAS
- == > aSbAS
- == > aabAS
- == > aabbaS
- == > aabbaa



ig. 6.8 The derivation tree with yield *aabbaa* for Example 6.2.

Examples

- 1. Let G be the grammar S → 0B |1A, A → 0 | 0S| 1AA, B → 1| 1S | 0BB. For the string 00110101, find (a) the leftmost derivation, (b) the rightmost derivation, and (c) the derivation tree.

Solution

(a) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011S$
$\Rightarrow 0^21^20B \Rightarrow 0^21^201S \Rightarrow 0^21^2010B \Rightarrow 0^21^20101$

(b) $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1S \Rightarrow 00B10B$
$\Rightarrow 0^2B101S \Rightarrow 0^2B1010B \Rightarrow 0^2B10101 \Rightarrow 0^2110101.$

(c) The derivation tree is given in Fig. 6.9.

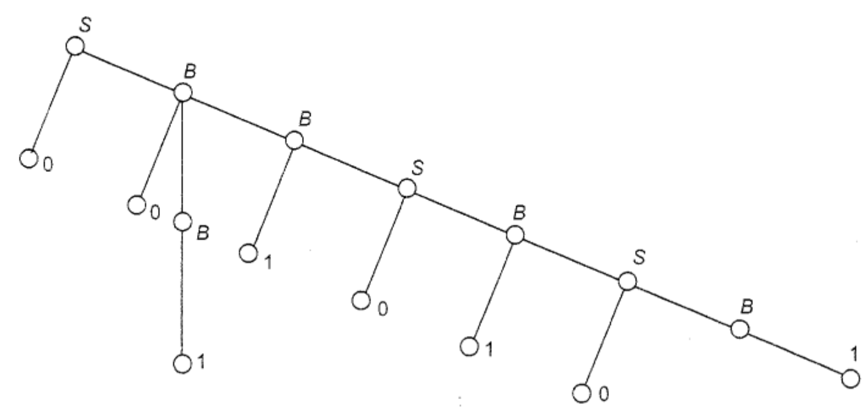


Fig. 6.9 The derivation tree with yield 00110101 for Example 6.3.

Examples

- Find the leftmost derivation for aaa bba bbba and construct parse tree for the same string. Consider the G whose productions are

$$S \rightarrow aB \mid bA$$
$$A \rightarrow a \mid aS \mid bAA$$
$$B \rightarrow b \mid bS \mid aBB$$

6.1 Find a derivation tree of $a * b + a * b$ given that $a * b + a * b$ is in $L(G)$, where G is given by $S \rightarrow S + S \mid S * S$, $S \rightarrow a \mid b$.

Examples

EXAMPLE 6.4

If G is the grammar $S \rightarrow SbS \mid a$, show that G is ambiguous.

6.5 Consider the following productions:

$$S \rightarrow aB \mid bA$$

$$A \rightarrow aS \mid bAA \mid a$$

$$B \rightarrow bS \mid aBB \mid b$$

For the string *aaabbabbba*, find
(a) the leftmost derivation,
(b) the rightmost derivation, and
(c) the parse tree.

6.6 Show that the grammar $S \rightarrow a \mid abSb \mid aAb$, $A \rightarrow bS \mid aAAb$ is ambiguous.

6.7 Show that the grammar $S \rightarrow aB \mid ab$, $A \rightarrow aAB \mid a$, $B \rightarrow ABb \mid b$ is ambiguous.

Grammar Equivalence and Normal Form

- **Eliminating Ɛ-productions**
- A CFG may have a production for A → Ɛ where A is a Non terminal.
- In CFG if there is production B and B→ Ɛ and B derives Ɛ in one or more steps then B is called Nullable Nonterminal.
- Ex. A→ a|Ɛ and B → a
- Here A is Nullable NT and B is not nullable NT.

Grammar Equivalence and Normal Form

- **Eliminating Ɛ-productions**
- Procedure
- Step 1: Delete all Ɛ-productions.
- Step 2: Identify all Nullable NT
- Step 3: For each Nullable NT, add subset of each nullable NT on RHS of productions.

Ex: Eliminate Ɛ-production from Grammar G whose productions are

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

Grammar Equivalence and Normal Form

- **Eliminating Ɛ-productions**

Ex 1: Eliminate Ɛ-production from Grammar G whose productions are

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

Solution:

Here S is Nullable NT since $S \rightarrow \varepsilon$

Eliminate it.

$$S \rightarrow aSa \mid bSb$$

Subset of $S \rightarrow aSa$ are $S \rightarrow aSa|aa$

Subset of $S \rightarrow bSb$ are $S \rightarrow bSb|bb$

By adding these subset, we get resultant Grammar as

$$\mathbf{S \rightarrow aSa \mid bSb \mid aa \mid bb}$$

Grammar Equivalence and Normal Form

- **Eliminating Ɛ-productions**

Ex 2: Eliminate Ɛ-production from Grammar G whose productions are

$$S \rightarrow Xa$$
$$X \rightarrow aX \mid bX \mid \varepsilon$$

Ex 3: Eliminate Ɛ-production from Grammar G whose productions are

$$S \rightarrow ABA$$
$$A \rightarrow aA \mid \varepsilon$$
$$B \rightarrow bB \mid \varepsilon$$

Grammar Equivalence and Normal Form

- It is sometimes useful to have a **normal form** for grammars
- A context-free grammar is in Chomsky normal form (CNF)

  if it is Ɛ-free and if in addition each production is either of the form

  $$A \rightarrow B\ C \text{ or } A \rightarrow a.$$
- That is, the right-hand side of each rule either has two non-terminal symbols or one terminal symbol.
- **Chomsky normal form** grammars are binary branching, that is they have binary trees.

- Any context-free grammar can be converted into equivalent Chomsky normal form grammar.

- For example, a rule of the form

  $$A \rightarrow B\ C\ D$$
- can be converted into the following two CNF rules

  $$A \rightarrow B\ X$$
  $$X \rightarrow C\ D$$

Ambiguity

- Structural ambiguity occurs when the grammar can assign more than one parse to a sentence.
- Two common kinds of ambiguity are attachment ambiguity and coordination ambiguity.
  - A sentence has an attachment ambiguity if a particular constituent can be attached to the parse tree at more than one place.
  - In coordination ambiguity phrases can be conjoined by a conjunction like *and*.

- Simple: For deriving input string w, if more than one derivation tree are produced, then the grammar is Ambiguous.

Ambiguity

- Example 1: Show that G is ambiguous if G is the grammar

  $S \rightarrow S + S \mid S * S \mid a \mid b$

Consider string : *a + a * b*

The leftmost derivations of $a + a * b$ induced by the two derivation trees are

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

Therefore, $a + a * b$ is ambiguous.

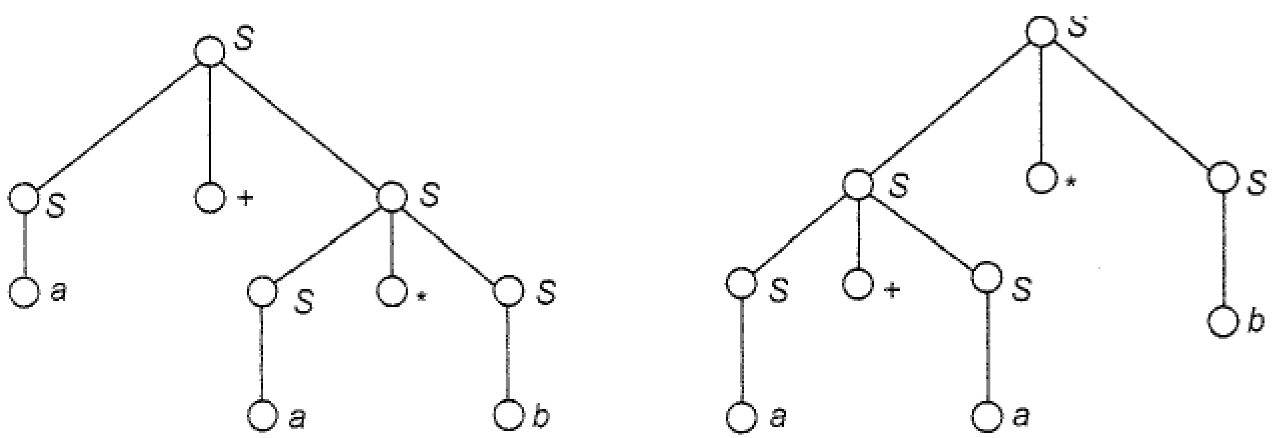
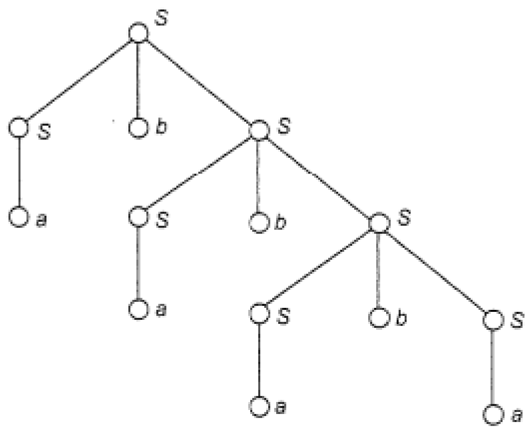


Fig. 6.10 Two derivation trees for $a + a * b$.

Ambiguity

- Example 2: Show that G is ambiguous if G is the grammar

  S → SbS|a



- Consider string : *abababa*

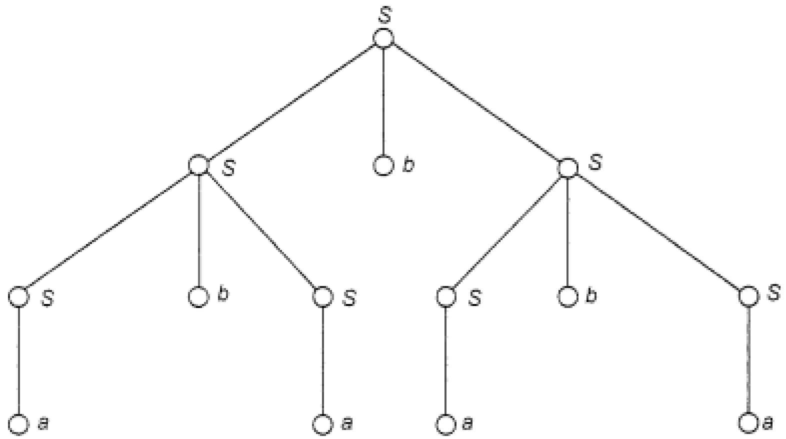


Fig. 6.11 Two derivation trees of *abababa* for Example 6.4.

Ambiguity

- Elimination of Ambiguity:
- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow id$

EXAMPLE 6.26

Show that a CFG G with productions $S \rightarrow SS \mid (S) \mid \Lambda$ is ambiguous.

Solution

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow \Lambda(S) \Rightarrow \Lambda(\Lambda) = (\Lambda)$$

Also,

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (\Lambda)S \Rightarrow (\Lambda)\Lambda = (\Lambda)$$

Hence G is ambiguous.

CKY Parsing

- Cocke-KasamiYounger (CKY) algorithm is the most widely used dynamic-programming based approach to parsing.
- Conversion to Chomsky Normal Form
- The CKY algorithm requires grammars to first be in Chomsky Normal Form (CNF)

- Eliminate unit productions
- Eliminate null productions

CKY Parsing

- Conversion to Chomsky Normal Form

- The entire conversion process can be summarized as follows:

  - 1. Copy all conforming rules to the new grammar unchanged.
  - 2. Convert terminals within rules to dummy non-terminals.
  - 3. Convert unit productions.
  - 4. Make all rules binary and add them to new grammar.

Grammar Equivalence and Normal Form

- Convert the given grammar into in CNF

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

- Convert the given grammar into in CNF

$$S \rightarrow bA \mid aB$$

$$A \rightarrow bAA \mid aS \mid a$$

$$B \rightarrow aBB \mid bS \mid b$$

Grammar Equivalence and Normal Form

Reduce the following grammar G to CNF. G is $S \to aAD$, $A \to aB \mid bAB$, $B \to b$, $D \to d$.

Solution

As there are no null productions or unit productions, we can proceed to step 2.

Step 2 Let $G_1 = (V'_N, \{a, b, d\}, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $B \to b$, $D \to d$ are included in P_1.

(ii) $S \to aAD$ gives rise to $S \to C_aAD$ and $C_a \to a$.

$A \to aB$ gives rise to $A \to C_aB$.

$A \to bAB$ gives rise to $A \to C_bAB$ and $C_b \to b$.

$V'_N = \{S, A, B, D, C_a, C_b\}$.

Step 3 P_1 consists of $S \to C_aAD$, $A \to C_aB \mid C_bAB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$.

$A \to C_aB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$ are added to P_2

$S \to C_aAD$ is replaced by $S \to C_aC_1$ and $C_1 \to AD$.

$A \to C_bAB$ is replaced by $A \to C_bC_2$ and $C_2 \to AB$.

Let

$$G_2 = (\{S, A, B, D, C_a, C_b, C_1, C_2\}, \{a, b, d\}, P_2, S)$$

where P_2 consists of $S \to C_aC_1$, $A \to C_aB \mid C_bC_2$, $C_1 \to AD$, $C_2 \to AB$, $B \to b$, $D \to d$, $C_a \to a$, $C_b \to b$. G_2 is in CNF and equivalent to G.

Grammar Equivalence and Normal Form

EXAMPLE 6.12

Find a grammar in Chomsky normal form equivalent to $S \rightarrow aAbB$, $A \rightarrow aA \mid a$, $B \rightarrow bB \mid b$.

Solution

As there are no unit productions or null productions, we need not carry out step 1. We proceed to step 2.

Step 2 Let $G_1 = (V'_N \{a, b\}, P_1, S)$, where P_1 and V'_N are constructed as follows:

(i) $A \rightarrow a$, $B \rightarrow b$ are added to P_1.

(ii) $S \rightarrow aAbB$, $A \rightarrow aA$, $B \rightarrow bB$ yield $S \rightarrow C_aAC_bB$, $A \rightarrow C_aA$, $B \rightarrow C_bB$, $C_a \rightarrow a$, $C_b \rightarrow b$.

$$V'_N = \{S, A, B, C_a, C_b\}.$$

Step 3 P_1 consists of $S \rightarrow C_aAC_bB$, $A \rightarrow C_aA$, $B \rightarrow C_bB$, $C_a \rightarrow a$, $C_b \rightarrow b$, $A \rightarrow a$, $B \rightarrow b$.

$$S \rightarrow C_aAC_bB \text{ is replaced by } S \rightarrow C_aC_1,\ C_1 \rightarrow AC_2,\ C_2 \rightarrow C_bB$$

The remaining productions in P_1 are added to P_2. Let

$$G_2 = (\{S, A, B, C_a, C_b, C_1, C_2\}, \{a, b\}, P_2, S),$$

where P_2 consists of $S \rightarrow C_aC_1$, $C_1 \rightarrow AC_2$, $C_2 \rightarrow C_bB$, $A \rightarrow C_aA$, $B \rightarrow C_bB$, $C_a \rightarrow a$, $C_b \rightarrow b$, $A \rightarrow a$, and $B \rightarrow b$.

G_2 is in CNF and equivalent to the given grammar.

Grammar Equivalence and Normal Form

EXAMPLE 6.22

Reduce the following grammar to CNF:

$$S \to ASA \mid bA, \quad A \to B \mid S, \quad B \to c$$

Solution

Step 1 *Elimination of unit productions:*
The unit productions are $A \to B$, $A \to S$.

$$W_0(S) = \{S\},\ W_1(S) = \{S\} \cup \emptyset = \{S\}$$

$$W_0(A) = \{A\},\ W_1(A) = \{A\} \cup \{S, B\} = \{S, A, B\}$$

$$W_2(A) = \{S, A, B\} \cup \emptyset = \{S, A, B\}$$

$$W_0(B) = \{B\},\ W_1(B) = \{B\} \cup \emptyset = \{B\}$$

The productions for the equivalent grammar without unit productions are

$$S \to ASA \mid bA,\ B \to c$$

$$A \to ASA \mid bA,\ A \to c$$

So, $G_1 = (\{S, A, B\}, \{b, c\}, P, S)$ where P consists of $S \to ASA \mid bA$, $B \to c$, $A \to ASA \mid bA \mid c$.

Step 2 *Elimination of terminals in R.H.S.:*

$S \to ASA$, $B \to c$, $A \to ASA \mid c$ are in proper form. We have to modify $S \to bA$ and $A \to bA$.

Replace $S \to bA$ by $S \to C_bA$, $C_b \to b$ and $A \to bA$ by $A \to C_bA$, $C_b \to b$.

So, $G_2 = (\{S, A, B, C_b\}, \{b, c\}, P_2, S)$ where P_2 consists of

$$S \to ASA \mid C_bA$$

$$A \to ASA \mid c \mid C_bA$$

$$B \to c,\ C_b \to b$$

Step 3 *Restricting the number of variables on R.H.S.:*

$$S \to ASA \text{ is replaced by } S \to AD,\ D \to SA$$

$$A \to ASA \text{ is replaced by } A \to AE,\ E \to SA$$

So the equivalent grammar in CNF is

$$G_3 = (\{S, A, B, C_b, D, E\}, \{b, c\}, P_3, S)$$

where P_3 consists of

$$S \to C_bA \mid AD$$

$$A \to c \mid C_bA \mid AE$$

$$B \to c,\ C_b \to b,\ D \to SA,\ E \to SA$$

Grammar Equivalence and Normal Form

6.12 Reduce the following grammars to Chomsky normal form:

(a) $S \to 1A \mid 0B, \quad A \to 1AA \mid 0S \mid 0, \quad B \to 0BB \mid 1S \mid 1$

(b) $G = (\{S\}, \{a, b, c\}, \{S \to a \mid b \mid cSS\}, S)$

(c) $S \to abSb \mid a \mid aAb, \quad A \to bS \mid aAAb.$

6.13 Reduce the grammars given in Exercises 6.1, 6.2, 6.6, 6.7, 6.9, 6.10 to Chomsky normal form.

1. Consider the grammer G which has the productions

$$A \to a \mid Aa \mid bAA \mid AAb \mid AbA$$

2. Consider the grammar G which has the following productions

$$S \to aB \mid bA, \; A \to aS \mid bAA \mid a, \; B \to bS \mid aBB \mid b.$$

6.1 Find a derivation tree of $a * b + a * b$ given that $a * b + a * b$ is in $L(G)$, where G is given by $S \to S + S \mid S * S$, $S \to a \mid b$.

6.6 Show that the grammar $S \to a \mid abSb \mid aAb$, $A \to bS \mid aAAb$ is ambiguous.

6.7 Show that the grammar $S \to aB \mid ab$, $A \to aAB \mid a$, $B \to ABb \mid b$ is ambiguous.

6.9 Find a reduced grammar equivalent to the grammar $S \to aAa$, $A \to bBB$, $B \to ab$, $C \to aB$.

Dependency Parsing

- **Dependency Relations**
- The traditional linguistic notion of grammatical relation provides the basis for the binary relations that comprise these dependency structures.
- The arguments to these relations consist of a **head** and a **dependent.**
- The head plays the role of the central organizing word, and the dependent as a kind of modifier.
- The head-dependent relationship is made explicit by directly linking heads to the words that are immediately dependent on them.

Dependency Parsing

- **Dependency Relations**
- Linguists have developed taxonomies of relations that go well beyond the familiar notions of subject and object.

- The **Universal Dependencies** (UD) project, an open community effort to annotate dependencies and other aspects of grammar across more than 100 languages, provides an inventory of 37 dependency relations.

- Fig. 18.2 shows a subset of the UD relations and Fig. 18.3 provides some examples.

Dependency Parsing

- **Dependency Relations**

Clausal Argument Relations	Description
NSUBJ	Nominal subject
OBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 18.2 Some of the Universal Dependency relations (de Marneffe et al., 2021).

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
OBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 18.3 Examples of some Universal Dependency relations.

Dependency Formalisms

- A dependency structure can be represented as a directed graph $G = (V, A)$
- consisting of a set of vertices V,
- and a set of ordered pairs of vertices A, which we'll call *arcs.*
- The set of arcs, A, captures the head dependent and grammatical function relationships between the elements in V.
- A **dependency tree** is a directed graph that satisfies the following constraints:
  - 1. There is a single designated root node that has no incoming arcs.
  - 2. With the exception of the root node, each vertex has exactly one incoming arc.
  - 3. There is a unique path from the root node to each vertex in V

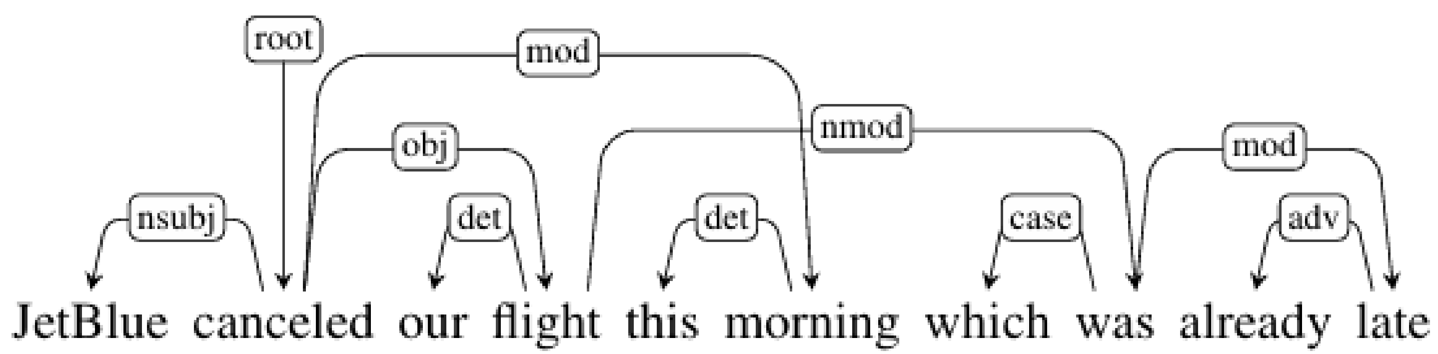
Projectivity

- The notion of projectivity imposes an additional constraint that is derived from the order of the words in the input.
- An **arc** from a head to a dependent **is said to be** projective if there is a path from the head to every word that lies between the head and the dependent in the sentence.
- A **dependency tree is then said to be** projective if all the arcs that make it up are projective.

Projectivity

- In this example, the arc from flight to its modifier was is non-projective since there is no path from flight to the intervening words this and morning.
- As we can see from this diagram, projectivity (and non-projectivity) can be detected in the way we've been drawing our trees.

Consider the following example.



Dependency Treebanks

- Treebanks play a critical role in the development and evaluation of dependency parsers.
- They are used for training parsers, they act as the gold labels for evaluating parsers, and they also provide useful information for corpus linguistics studies.
- Dependency treebanks are created by having human annotators directly generate dependency structures for a given corpus, or by hand-correcting the output of an automatic parser.
- The largest open community project for building dependency trees is the Universal Dependencies project, which currently has almost 200 dependency treebanks in more than 100 languages.

Transition-Based Dependency Parsing

- First approach to dependency parsing is called transition-based parsing.
- This architecture draws on shift-reduce parsing, a paradigm originally developed for analyzing programming languages.
- In transition-based parsing we'll have a stack on which we build the parse, a buffer of tokens to be parsed, and a parser which takes actions on the parse via a predictor called an oracle, as illustrated in Fig. 18.4.

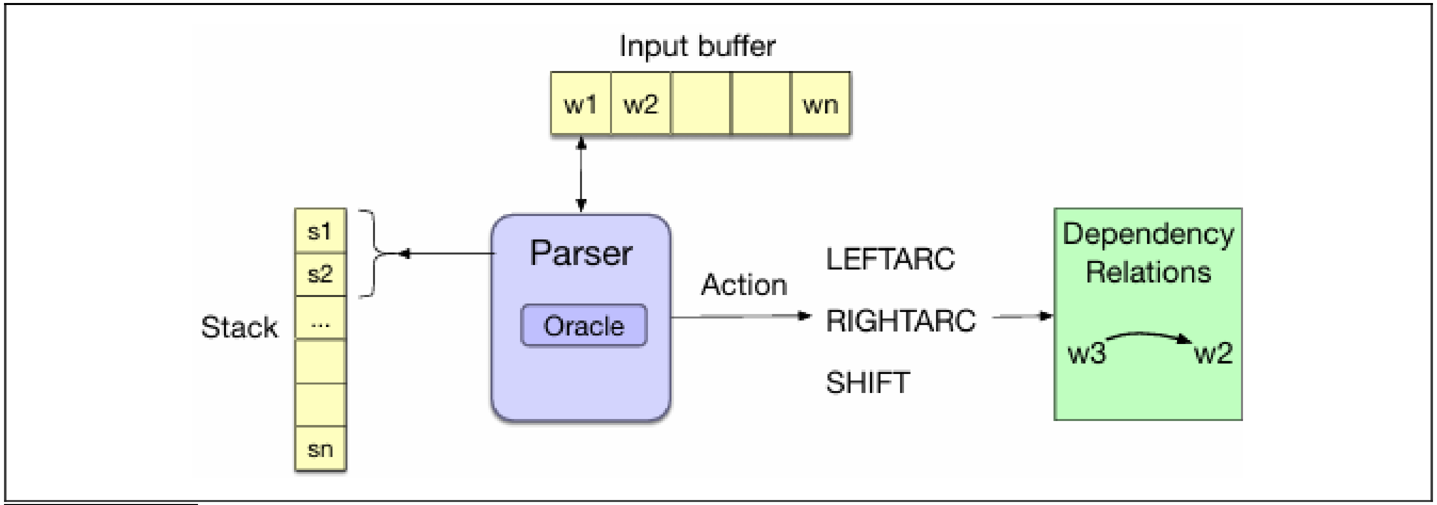


Figure 18.4 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action by consulting an oracle that examines the current configuration.

Transition-Based Dependency Parsing

- The parser walks through the sentence left-to-right, successively shifting items from the buffer onto the stack.
- At each time point we examine the top two elements on the stack, and the oracle makes a decision about what transition to apply to build the parse.

- The specification of a transition-based parser is quite simple, based on representing the current state of the parse as a configuration:
  - the stack,
  - an input buffer of words or tokens, and
  - a set of relations representing a dependency tree.
- We start with an initial configuration in which the stack contains the ROOT node, the buffer has the tokens in the sentence, and an empty set of relations represents the parse.
- In the final goal state, the stack and the word list should be empty, and the set of relations will represent the final parse.

Transition-Based Dependency Parsing

- Fig. 18.5 gives the algorithm.

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)          ; choose a transition operator to apply
    state ← APPLY(t, state)  ; apply it, creating a new state
  return state
```

Figure 18.5 A generic transition-based dependency parser

End of UNIT V