Introduction to Parsing

Parsing
ISCL-BA-06

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2020/21

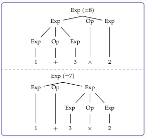
version: 2a6d9e0 @2020-11-30

What is parsing?

- Parsing is the task of assigning a structure to a given sentence
- It is related to recognition: typically we follow the steps taken during derivation to obtain the structure
- From a different perspective, parsing is the inverse of the generation task
- Note: we focus on context-free parsing – the structures we build/recover are trees

Why do we need parsing?

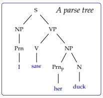
- The formal approach to languages as sets emphasizes recognition
  - a string is whether in the language or not
- Parsing is in general a step for semantics
  - we cannot assign semantics without structure



Overview

- Representation context-free analyses and parse trees
- Ambiguity
- Top-down parsing
- Bottom-up parsing
- General overview of the parsing methods
- Representing parsing methods: parse forests
- Parsing and semantics

Different ways to represent a context-free parse



A history of derivations

Sentential form	derivation
S	(start)
NP VP	S ⇒ → NP VP
Prn VP	NP ⇒ → Prn
I VP	Prn ⇒ → I
I V NP	VP ⇒ → V NP
I saw NP	V ⇒ → saw
I saw Prn_p N	NP ⇒ → Prn_p N
I saw her N	Prn_p ⇒ → her
I saw her duck	N ⇒ → duck

(Labeled) brackets: $\left[_S \left[_{NP} [_{Prn}\ I] \right] \left[_{VP} [_V\ saw] \left[_{NP} [_{Prn_p}\ her] [_N\ duck] \right] \right] \right]$

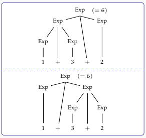
Relation between different representations

- The parse tree and the bracket representation is equivalent
  - parse trees are easier to read by humans
  - brackets are easier for computers
  - brackets are the typical representation for treebanks
- A parse tree (or bracket representation) can be obtained with a different order of production rules

Grammars and ambiguity

Exp → n
Exp → Exp + Exp
(terminal symbol 'n' stands for any number)

- If a grammar is ambiguous, some sentences produce multiple analyses
- If the resulting analysis lead to the same semantics, the ambiguity is *spurious*

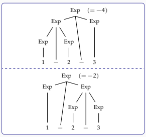


Grammars and ambiguity

Exp → n
Exp → Exp − Exp

(terminal symbol 'n' stands for any number)

- Is this ambiguity spurious?
- If different structures yield different semantics, the ambiguity is *essential*



Languages and ambiguity

- A language is ambiguous if there is no unambiguous grammar that can produce it
- For example, the language $a^n b^n c^m \cup a^p b^q c^q$ is ambiguous
  - The strings of the form $a^k b^k c^k$ could be generated by either part of the language definition
- Note: do not confuse ambiguity with different derivations leading to same analysis
  - Ambiguity results in different structures
  - Multiple derivations with the same structure is related to the mechanism used for obtaining the derivations

Ambiguity can be removed from a grammar

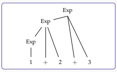
if the language is not ambiguous

Exp → n
Exp → Exp + n
(terminal symbol 'n' stands for any number)

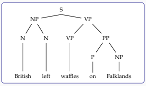
- This one does not have the ambiguity of

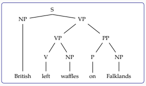
  Exp → n
  Exp → Exp + Exp

- Both grammars define the same language



Natural languages are ambiguous





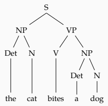
- The grammars we define have to distinguish between two different structures

Top-down parsing

general idea

- Start from S, find a sequence of derivations that yield the sentence
- This is simply the same as the generation procedure we discussed earlier
- Attempt to generate all strings from the parse grammar, but allow productions that only leads to the input string

Top-down: demonstration



S	→	NP VP
NP	→	Det N
VP	→	V NP
VP	→	V
Det	→	a
Det	→	the
N	→	cat
N	→	dog
V	→	bites

From demonstration to parsing

- There may be multiple production applicable
- We need an automatic mechanism to select the correct productions
- We have two actions:
  - predict generate a hypothesis based on the grammar
  - match when a terminal is produced, check if it matches with the terminal in the expected position
    - if matched, continue
    - otherwise, backtrack
- if we eliminate all non terminals, and the complete input string is matched, then parsing successful

Top-down parsing: another demonstration

the grammar

S	→	NP VP
NP	→	Det N
VP	→	V NP
VP	→	V
Det	→	a
Det	→	the
N	→	cat
N	→	dog
V	→	bites

parse: *the cat bites a dog*

matched	goal	production
	S	S ⇒ NP VP
	NP VP	NP ⇒ Det VP
	Det N VP	Det ⇒ a ✗
	Det N VP	Det ⇒ the ✓
the	N VP	N ⇒ dog ✗
the cat	N VP	N ⇒ cat ✓
the cat	VP	VP ⇒ V
the cat bites	V	V ⇒ bites ✓
the cat bites		(not at the end) ✗
the cat	V NP	VP ⇒ V NP
the cat bites	NP	V ⇒ bites ✓
the cat bites	Det N	NP ⇒ Det N
the cat bites a	N	Det ⇒ a ✓
the cat bites a dog		Det ⇒ dog ✓

Note that the valid productions yield the parse tree.

Top-down parsing: problems and possible solutions

- Trial-and-error procedure leads to exponential time parsing
- But lots of repeated work: dynamic programming may help avoid it
- What happens if we had a rule like

  NP → NP PP

  some rules may cause infinite loops
- Notice that if we knew which terminals are possible as the initial part of a non-terminal symbol, we can eliminate the unsuccessful matches earlier

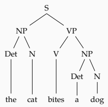
Bottom-up parsing

general idea

- Start from from the input symbol, and try to *reduce* the input to start symbol
- We need to match parts of the sentential form (starting from the input) to the RHS of the grammar rules
- While top-down process relies on *productions* the bottom-up process relies on *reductions*

production ↓						reduction ↑
	NP		V	NP		
	Det	N	V	Det	N	
	the	cat	bites	a	dog	

Bottom-up: demonstration



S	→	NP VP
NP	→	Det N
VP	→	V NP
VP	→	V
Det	→	a
Det	→	the
N	→	cat
N	→	dog
V	→	bites

A (first) introduction to shift-reduce parsing

- We keep two data structures:
  - a stack for the (partially) reduced sentential form
  - an input queue that contains only terminal symbols

  [NP V] [a dog]
- We use two operations:
  - shift shifts a terminal to stack

    [NP V] [a dog] —shift→ [NP V a] [dog]
  - reduce when top symbols on stack mach a RHS, replace them with the LHS of the rule

    [NP V] [a dog] —reduce→ [NP VP] [a dog]

Shift-reduce (bottom-up) parsing a demonstration

stack	input	rule
	the cat bites a dog	shift
the	cat bites a dog	Det ⇒ the
Det	cat bites a dog	shift
Det cat	bites a dog	N ⇒ cat
NP	bites a dog	NP ⇒ Det N
NP	bites a dog	shift
NP bites	a dog	V ⇒ bites
NP V	a dog	VP ⇒ V
NP VP	a dog	S ⇒ NP VP
S	a dog	shift
S a	dog	Det ⇒ A
S Det dog		N ⇒ dog
S Det N		NP ⇒ Det N
S NP		(stuck)

stack	input	rule
NP V	a dog	shift
NP V a	dog	Det ⇒ a
NP V Det	dog	shift
NP V Det dog		N ⇒ dog
NP V Det N		NP ⇒ Det N
NP V NP		VP ⇒ V NP
NP VP		S ⇒ NP VP
S		(done)

- All input reduced to S, accept
- Rules form the parse tree

Summary

- Parsing can be formulated as a top-down or bottom-up search (the search may also be depth-first or breadth first)
- Naive parsing algorithms are inefficient (exponential time complexity)
- There are some directions: dynamic programming, filtering
- Suggested reading for this part: Grune and Jacobs (2007, ch.3)

Next:

- Bottom-up chart parsing: CKY algorithm
- Suggested reading: Grune and Jacobs (2007, section 4.2), Jurafsky and Martin (2009, draft 3rd ed, section 13.2)

Acknowledgments, references, additional reading material

- Please read Grune and Jacobs (2007) chapter 3, a big part part of the lecture follows this chapter

Grune, Dick and Ceriel J.H. Jacobs (2007). *Parsing Techniques: A Practical Guide*. second. Monographs in Computer Science. The first edition is available at http://dickgrune.com/Books/PTAPG_1st_Edition/BookBody.pdf. Springer New York. ISBN: 9780387689548.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3. URL: http://web.stanford.edu/~jurafsky/slp3/.