

# Parsing

## Unger's Parser

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Summer 2020



# Table of contents

- 1 Introduction
- 2 The Parser
- 3 An Example
- 4 Optimizations
- 5 Conclusion

# Introduction (1)

Unger's parser (Grune and Jacobs, 2008) is a CFG parser that is

- a **top-down** parser: we start with S and subsequently replace lefthand sides of productions with righthand sides
- a **non-directional** parser: the expanding of non-terminals (with appropriate righthand sides) is not ordered; therefore we need to guess the yields of all non-terminals in a right-hand side at once

## Introduction (2)

$G = \langle N, T, P, S \rangle$ ,  $N = \{S, NP, VP, PP, V, \dots\}$ ,  $T = \{Mary, man, telescope, \dots\}$ , productions:  $S \rightarrow NP VP$ ,  $VP \rightarrow VP PP$ ,  $VP \rightarrow V NP$ ,  $NP \rightarrow Mary, \dots$

Input: *Mary sees the man with the telescope*

1.	<i>S</i>	Mary sees the man with the telescope	
2.	<i>NP</i>	Mary	$S \rightarrow NP VP$ (1.)
3.	<i>VP</i>	sees the man with the telescope	
4.	<i>NP</i>	Mary sees	$S \rightarrow NP VP$ (1.)
5.	<i>VP</i>	the man with the telescope	
⋮			
14.	<i>Mary</i>	Mary	$NP \rightarrow Mary$ (2.)
15.	<i>VP</i>	sees	$VP \rightarrow VP PP$ (3.)
16.	<i>PP</i>	the man with the telescope	
17.	<i>VP</i>	sees the	$VP \rightarrow VP PP$ (3.)
18.	<i>PP</i>	man with the telescope	

⋮

## Introduction (3)

Parsing strategy:

- The parser takes an  $X \in N \cup T$  and a substring  $w$  of the input.
- Initially, this is  $S$  and the entire input.
- If  $X$  and the remaining substring are equal, we can stop (success for  $X$  and  $w$ ).
- Otherwise,  $X$  must be a non-terminal that can be further expanded. We then choose an  $X$ -production and partition  $w$  into further substrings that are paired with the righthand side elements of the production.
- The parser continues recursively.

# The parser (1)

Assume CFG without  $\epsilon$ -productions and without loops  $A \xRightarrow{+} A$

```
function unger(w, X):  
  out := false;  
  if w = X, then out := true  
  else for all  $X \rightarrow X_1 \dots X_k$ :  
    for all  $x_1, \dots, x_k \in T^+$  with  $w = x_1 \dots x_k$ :  
      if  $\bigwedge_{i=1}^k \text{unger}(x_i, X_i)$   
        then out := true;  
  return out
```

The following holds:

$\text{unger}(w, X)$  iff  $X \xRightarrow{*} w$  (for  $X \in N \cup T, w \in T^*$ )

## The parser (2)

Extension to deal with  $\epsilon$ -productions and loops:

- Add a list of preceding calls
- pass this list when calling the parser again
- if the new call is already on the list, stop and return `false`

Initial call: `unger(w, S,  $\emptyset$ )`

## The parser (3)

```
function unger( $w, X, L$ ):  
  out := false;  
  if  $\langle X, w \rangle \in L$ , return out;  
  else if  $w = X$  or ( $w = \epsilon$  and  $X \rightarrow \epsilon \in P$ )  
    then out := true  
  else for all  $X \rightarrow X_1 \dots X_k \in P$ :  
    for all  $x_1, \dots, x_k \in T^*$  with  $w = x_1 \dots x_k$ :  
      if  $\bigwedge_{i=1}^k \text{unger}(x_i, X_i, L \cup \{\langle X, w \rangle\})$   
        then out := true;  
  return out
```



## The parser (4)

- So far, we have a recognizer, not a parser.
- To turn this into a parser, every call `unger(..)` must return a (set of) parse trees.
- This can be obtained from
  - ① the successful productions  $X \rightarrow X_1 \dots X_k$ , and
  - ② the parse trees returned by the calls `unger( $x_i, X_i$ )`.
- Note, however, that there might be a large amount of parse trees since in each call, there might be more than one successful production.
- We will come back to the compact presentation of several analyses in a parse forest.

## An example (1)

- Assume a CFG without  $\varepsilon$ -productions
- Production  $S \rightarrow NP VP$
- Input sentence  $w$  with  $|w| = 34$ :

*Mr. Sarkozy's pension reform, which only affects about 500,000 public sector employees, is the opening salvo in a series of measures aimed more broadly at rolling back France's system of labor protections.*

(New York Times)

## An example (2)

Partitions according to Unger's parser:

	<i>NP</i>	<i>S</i>	<i>VP</i>
1.	Mr.		Sarkozy's ... protections
2.	Mr. Sarkozy	's ...	protections
3.	Mr. Sarkozy's	pension ...	protections
		⋮	
33.	Mr. ...labor		protections

$|w| = 34$ , consequently we have 33 different partitions.

## An example (3)

- Consider the following partition for  $S \rightarrow NP VP$ :

---

$S$	$NP$	Mr. Sarkozy's pension reform, which ... employees,
	$VP$	is ... protections

---

- For  $NP \rightarrow NP S$ , there are 12 partitions of the  $NP$  part
- The partition above is just one partition for one production!
- In the worst case, parsing is exponential in the length  $n$  of the input string!

# A note about time complexity

## Time complexity

We say that an algorithm is of

- **polynomial time complexity** if there is a constant  $c$  and a  $k$  such that the parsing of a string of length  $n$  takes an amount of time  $\leq cn^k$ .

Notation:  $\mathcal{O}(n^k)$

- **exponential time complexity** if there is a constant  $c$  and a  $k$  such that the parsing of a string of length  $n$  takes an amount of time  $\leq ck^n$ .

Notation:  $\mathcal{O}(k^n)$

# Optimizations (1)

As an additional filter, we can constrain the set of partitions that we investigate:

- Check on occurrences of terminals in rhs.
- Check on minimal length of terminal string derived by a non-terminal.
- Check on obligatory terminals (pre-terminals) in strings derived by non-terminals, e.g., each *NP* contains an *N*, each *VP* contains a *V*, ...
- Check on the first terminals derivable from a non-terminal.

## Optimizations (2)

Furthermore, we can use tabulation (dynamic programming) in order to avoid computing several times the same thing:

- 1 Whenever  $\text{unger}(X, w, L)$  yields a result  $res$ , we store  $\langle X, w, res \rangle$  in our table of partial parsing results.
- 2 In every call  $\text{unger}(X, w, L)$ , we first check whether we have already computed a result  $\langle X, w, res \rangle$  and if so, we stop immediately and return  $res$ .

## Optimizations (3)

Results  $\langle X, w, res \rangle$  can be stored in a three-dimensional table (**chart**)  $\mathcal{C}$ :

- Assume  $k = |N + T|$  and non-terminals  $N$  and terminals  $T$  to have a unique index  $\leq k$ . Furthermore, assume  $|w| = n$  with  $w = w_1 \cdots w_n$ , then you can use a  $k \times n \times n$  table, the chart!
  - ① Whenever  $\text{unger}(X, w_i \cdots w_j, L)$  yields a result  $res$  and  $m$  index of  $X$ , then  $\mathcal{C}(m, i, j) = res$
  - ② In every call  $\text{unger}(X, w_i \cdots w_j, L)$ , we first check whether we have already a value in  $\mathcal{C}(m, i, j)$  and if so, we stop and return  $\mathcal{C}(m, i, j)$
- Advantage: access of  $\mathcal{C}(m, i, j)$  in constant time.
- Disadvantage: storing the Chart needs more memory.
- Assumption: grammar is  $\varepsilon$ -free – otherwise we need a  $k \times (n + 1) \times (n + 1)$  chart.



# Optimizations (4)

## Example

- $G = \langle N, T, P, S \rangle$ ,  $N = \{S, B\}$ ,  $T = \{a, b, c\}$  and productions  
 $S \rightarrow aSB \mid c$   $B \rightarrow bb$
- Input word  $w = acbb$ .
- We assume that, when guessing the span of a rhs element, we take into account that ...
  - ① each terminal spans only a corresponding single terminal
  - ② the span of an  $S$  has to start with an  $a$  or a  $c$
  - ③ the span of a  $B$  has to start with a  $b$
  - ④ the span of each  $X \in N \cup T$  contains at least one symbol (no  $\varepsilon$ -productions)

# Optimizations (5)

## Example continued

Chart obtained for  $w = acbb$

j					
4	$\langle S, t \rangle$		$\langle B, t \rangle$	$\langle b, t \rangle$	
				$\langle B, f \rangle$	
3		$\langle S, f \rangle$	$\langle b, t \rangle$		
2		$\langle S, t \rangle$			
		$\langle c, t \rangle$			
1	$\langle a, t \rangle$				
	1	2	3	4	i

$S \xRightarrow{*} acbb? \rightarrow t$   
 $a \xRightarrow{*} a? \rightarrow t$   
 $S \xRightarrow{*} c? \rightarrow t$   
 $c \xRightarrow{*} c? \rightarrow t$   
 $B \xRightarrow{*} bb? \rightarrow t$   
 $b \xRightarrow{*} b? \rightarrow t$   
 $b \xRightarrow{*} b? \rightarrow t$   
 $S \xRightarrow{*} cb \rightarrow f$   
 $B \xRightarrow{*} b \rightarrow f$

(Productions:  $S \rightarrow aSB \mid c$   $B \rightarrow bb$ )

## Optimizations (6)

In addition, we can tabulate entire productions with the spans of their different symbols. This gives us a compact presentation of the parse forest!

- In every call  $\text{unger}(X, w_i \cdots w_j)$ , we first check whether we have already a value in  $\mathcal{C}(m, i, j)$  and if so, we stop and return  $\mathcal{C}(m, i, j)$ .
- Otherwise, we compute all possible first steps of derivations  $X \xrightarrow{*} w$ : for every production  $X \rightarrow X_1 \dots X_k$  and all  $w_1, \dots, w_k$  such that the recursive Unger calls yield `true`, we add  $\langle X, w \rangle \rightarrow \langle X_1, w_1 \rangle \dots \langle X_k, w_k \rangle$  with the indices of the spans to the list of productions.
- If at least one such production has been found, we return `true`, otherwise `false`.

Example on handout.

# Conclusion

Unger's parser is

- a non-directional top-down parser.
- highly non-deterministic because during parsing, the yields of all non-terminals in righthand sides must be guessed.
- in general of exponential (time) complexity.
- of polynomial time complexity if tabulation is applied.

Grune, D. and Jacobs, C. (2008). *Parsing Techniques. A Practical Guide.*  
Monographs in Computer Science. Springer. Second Edition.