Parsing

Unger's Parser

Laura Kallmeyer, Magnus Roos
Heinrich-Heine-Universität Düsseldorf

Sommersemester 2014

Overview

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)

S $\rightarrow$ NP VP, VP $\rightarrow$ VP PP, VP $\rightarrow$ V NP, NP $\rightarrow$ Mary, ...

1.	S	Mary sees the man with the telescope	
2.	NP	Mary	S $\rightarrow$ NP VP from 1.
3.	VP	sees the man with the telescope	
4.	NP	Mary sees	S $\rightarrow$ NP VP from 1.
5.	VP	the man with the telescope	

...

6.	Mary	Mary	NP $\rightarrow$ Mary from 2.
7.	VP	sees	VP $\rightarrow$ VP PP from 3.
8.	PP	the man with the telescope	

...

Introduction (3)

- The parser takes a $X \in N \cup T$ 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 a 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 ϵ-productions and without loops $A \overset{+}{\Rightarrow} A$.

```
function unger(w,X):
    out := false;
    if w = X, then out := true
    else for all X → X_1 ... X_k:
        for all x_1,...,x_k ∈ T^+ with w = x_1 ... x_k:
            if ⋀_{i=1}^{k} unger(x_i,X_i)
            then out := true;
    return out
```

`unger(w,X)` iff $X \overset{*}{\Rightarrow} w$ (for $X \in N \cup T, w \in T^*$)

The parser (2)

Extension to deal with ϵ-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 $X \to \epsilon \in P$)

then out := true

else for all $X \to X_1 \dots X_k \in P$:

for all $x_1, \dots, x_k \in T^*$ with $w = x_1 \dots x_k$:

if $\bigwedge_{i=1}^{k}$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

- the succssful productions $X \rightarrow X_1 \dots X_k$, and
- 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 ϵ.

Production S $\rightarrow$ NP VP.

Input sentence w:

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:

S	
NP	VP
Mr.	Sarkozy's pension ... protections
Mr. Sarkozy	's pension ... protections
...	
Mr. Sarkozy's pension ... labor	protections

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

An example (3)

Take the following partition:

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

For NP $\rightarrow$ NP S, there are 12 partitions of the NP part.

In the worst case, parsing is exponential in the length n of the input string.

An example (4)

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)

Conditions on the partitions:

- 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 a N, each VP a V, ...)
- check on the first terminals derivable from a non-terminal

Optimizations (2)

Tabulation: avoid computing several times the same thing:

- whenever `unger(`X, w`)` yields a result res, we store $\langle X, w, res\rangle$ in our table of partial parsing results;
- in every call `unger(`X, w`)`, 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 can be stored in a three-dimensional table (chart) $\mathcal{C}$:

If $k = |N + T|$ and non-terminals and terminals X have a unique index $\leq k$ and $|w| = n$ with $w = w_1 \cdots w_n$, then use a $k \times n \times n$ table, the chart.

- Whenever `unger(`$X, w_i \cdots w_j$`)` yields a result *res* and m index of X, then $\mathcal{C}(m, i, j) = res$.
- In every call `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 $\mathcal{C}(m, i, j)$.

Advantage: access of $\mathcal{C}(m, i, j)$ in constant time.

(Assumption: grammar ε-free, otherwise we need a $k \times (n + 1) \times (n + 1)$ chart.)

Optimizations (4)

Example: CFG with start symbol S, $N = \{S, B\}$, $T = \{a, b, c\}$ and productions

$$S \rightarrow aSB \,|\, c \quad B \rightarrow bb$$

Input word $w = acbb$. We assume that, when guessing the span of a rhs element, we take into account that

- each terminal spans only a corresponding single terminal;
- the span of an S has to start with an a or an c;
- the span of a B has to start with a b.
- the span of each $X \in N \cup T$ contains at least one symbol (no ε-productions)

Optimizations (5)

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

$S \stackrel{*}{\Rightarrow} acbb? \rightarrow \mathrm{t}$

$a \stackrel{*}{\Rightarrow} a? \rightarrow \mathrm{t}$

$S \stackrel{*}{\Rightarrow} c? \rightarrow \mathrm{t}$

$c \stackrel{*}{\Rightarrow} c? \rightarrow \mathrm{t}$

$B \stackrel{*}{\Rightarrow} bb? \rightarrow \mathrm{t}$

$b \stackrel{*}{\Rightarrow} b? \rightarrow \mathrm{t}$

$b \stackrel{*}{\Rightarrow} b? \rightarrow \mathrm{t}$

$S \stackrel{*}{\Rightarrow} cb \rightarrow \mathrm{f}$

$B \stackrel{*}{\Rightarrow} b \rightarrow \mathrm{f}$

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

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 $\texttt{unger(}X, w_i \cdots w_j\texttt{)}$, 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 $X \stackrel{*}{\Rightarrow} w$: for every production $X \rightarrow X_1 \ldots X_k$ and all $w_1, \ldots, w_k$ such that the recursive Unger calls yield `true`, we add $\langle X, w\rangle \rightarrow \langle X_1, w_1\rangle \ldots \langle X_k, w_k\rangle$ 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: see 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 complexity;
- polynomial if tabulation is applied.

References

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