Parsing

Parsing as deduction

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2017/18

Heinrich Heine Universität Düsseldorf

Table of contents

Motivation (1)

- Algorithmic descriptions of parsing algorithms (e.g., in pseudo-code) introduce *data structures* and *control structures*
- The parsing strategy of the algorithm does not depend on them

Question: Can we separate the parsing strategy from the control strategy?

Answer: Parsing as Deduction Shieber et al. (1995); Sikkel (1997)

Motivation (2)

Advantages:

- Concentration on parsing strategy
- Facilitation of proofs (e.g., soundness and completeness of an algorithm):

  Soundness: If the algorithm yields *true* for w, then $w \in L(G)$.
  Completeness: If $w \in L(G)$, then the algo yields *true* for w .
- (Time) Complexity of an algorithm sometimes easier to determine

Parsing schemata (1)

How characterize a single parsing step?

During parsing, the parser produces trees (*parse trees*, partial results) and tries to combine them to new trees, until some tree rooted by the goal category (e.g. S) comes out

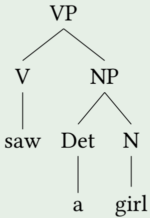
- We can characterize parse trees
- We can characterize how new parse trees can be deduced from existing ones
- We can fix a goal: We want to deduce a tree with root S that spans the entire input sentence

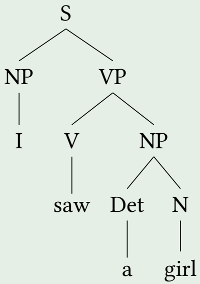
Parsing Schemata (2)

- We characterize a parse tree rooted by some nonterminal X by the terminals X spans.
- We write parse trees/partial parse results in the form of *items*: $[X, i, j]$, meaning that X derives the terminals between position i and position j

Items for parse trees

$_0\text{I}_1\text{saw}_2\text{a}_3\text{girl}_4$





Items: [VP,1,4] [S,0,4]

Parsing Schemata (3)

Sometimes, a category and its yield have been predicted but not yet recognized. To mark this, we can use dotted items or items with dotted productions:

1. $\bullet S$ signifies that S has been predicted.
2. $S\bullet$ signifies that S has been recognized.
3. $A \rightarrow A_1 \ldots A_i \bullet A_{i+1} \ldots A_n$ signifies that the rhs of the production $A \rightarrow A_1 \ldots A_n$ has been recognized up to A_i while the part from A_{i+1} to A_n has been predicted.

Parsing Schemata (4)

- Parsing Schemata understand parsing as a deductive process.
- Deduction of new items from existing ones can be described using inference rules.
- General form:

$$\frac{\textit{antecedent}}{\textit{consequent}}\ \textit{side conditions}$$

- Antecedent, consequent: (lists of) items.
- Application: if antecedent can be deduced and side conditions hold, then the consequent can be deduced as well.

Parsing Schemata (5)

- A parsing schema consists of
  1. Deduction rules
  2. An axiom (or axioms): can be written as a deduction rule with empty antecedent
  3. A goal item
- The parsing algorithm succeeds if, for a given input, it is possible to deduce the goal item.

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

Initial call: `unger(w, S)` Axiom

Example: Unger (2)

- An Unger item needs to caracterize
  1. A nonterminal category or a terminal symbol
  2. Its yield in the input string
  3. Whether the item is predicted or recognized
- Item form:
  $[\bullet X, i, j]$ or $[X\bullet, i, j]$ with $X \in N \cup T$, $i, j \in \mathbb{N}$, $i \leq j$.

Example: Unger (3)

- We start with the prediction that S yields the whole input (question $S \stackrel{*}{\Rightarrow} w$, $|w| = n$?).

$$\text{Axiom: } \frac{}{[\,\bullet S, 0, n\,]} \quad |w| = n$$

- The goal is to find an S that spans the whole input:

$$\text{Goal item: } [\,S\bullet, 0, n\,] \text{ where } n = |w|$$

- Whenever we encounter a terminal that matches the input, we can turn the predict item into a recognize item.

$$\text{Scan: } \frac{[\,\bullet a, i, i+1\,]}{[\,a\bullet, i, i+1\,]} \quad w_{i+1} = a$$

Example: Unger (4)

- Whenever we have predicted an $A \in N$ we can predict the RHS of any A-production while partitioning the input.

$$\text{Predict: } \frac{[\,\bullet A, i_0, i_k\,]}{[\,\bullet A_1, i_0, i_1\,], \ldots, [\,\bullet A_k, i_{k-1}, i_k\,]} \quad \begin{array}{l} A \to A_1 \ldots A_k \in P \\ i_j < i_{j+1} \end{array}$$

- Once all predictions for the rhs are true (turned into recognized items), we can turn also the A-item into a recognized item.

Complete:

$$\frac{[\,\bullet A, i_0, i_k\,], [\,A_1 \bullet, i_0, i_1\,], \ldots, [\,A_k \bullet, i_{k-1}, i_k\,]}{[\,A \bullet, i_0, i_k\,]} \quad A \to A_1 \ldots A_k \in P$$

Example: Unger (5)

Unger with deduction rules

Sample CFG: $S \to aSb \mid ab$, input word $w = aaabbb$.

Deduced items (only successful parse):

$[\bullet S, 0, n]$	axiom
$[\bullet a, 0, 1], [\bullet S, 1, 5], [\bullet b, 5, 6]$	predict
$[a\bullet, 0, 1], [\bullet a, 1, 2], [\bullet S, 2, 4], [\bullet b, 4, 5], [b\bullet, 5, 6]$	scan, predict, scan
$[a\bullet, 1, 2], [\bullet a, 2, 3], [\bullet b, 3, 4], [b\bullet, 4, 5]$	scan, predict, scan
$[a\bullet, 2, 3], [b\bullet, 3, 4]$	scan, scan
$[S\bullet, 2, 4]$	complete
$[S\bullet, 1, 5]$	complete
$[S\bullet, 0, 6]$	complete

Example: Unger (6)

Soundness and completeness of Ungers's algorithm:
Assume that we don't have the check on the terminals. Then for all $X \in N \cup T$, $i, j \in [0..n]$ with $i < j$:

- $[\bullet X, i, j]$ iff $S \stackrel{*}{\Rightarrow} \alpha X \beta$ for some $\alpha, \beta \in (N \cup T)^*$ such that $|\alpha| \leq i, |\beta| \leq n - j$;
- $[X\bullet, i, j]$ iff $S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} \alpha w_{i+1} \dots w_j \beta$ for some $\alpha, \beta \in (N \cup T)^*$ such that $|\alpha| \leq i, |\beta| \leq n - j$;

This can be shown by induction on the parsing schema:

1. Show that claim holds for axiom.
2. Show for every deduction rule that, if the claim holds for the antecedent, then it also holds for the consequent.

Example: Top-Down (1)

Assume CFG without ϵ-productions and without loops $A \overset{+}{\Rightarrow} A$.

```
def top-down(w,α):
   out = false
   if w = α = ε:
      out = true
   elif w = aw′ and α = aα′:
      out = top-down(w′,α′)                    Scan
   elif α = Xα′ with X ∈ N:
      for X → X_1 ... X_k in P:
         if top-down(w, X_1 ... X_k α′):     Predict
            out = true
   return out
```

Example: Top-Down (2)

The items must encode

- the remaining input or, alternatively, the position up to which the input has been parsed, and
- the remaining sentential form

$\Rightarrow$ item form $[\alpha, i]$ with $\alpha \in (N \cup T)^*, 0 \leq i \leq n$

Example: Top-Down (3)

Whenever we have the next input terminal as topmost symbol of the stack (left element of α), we can scan it:

$$\text{Scan: } \frac{[a\alpha, i]}{[\alpha, i+1]} \; w_{i+1} = a$$

Whenever the topmost stack symbol is A and there is an A-production $A \to \gamma$, we can predict this (here with check on length of sentential form):

$$\text{Predict: } \frac{[A\alpha, i]}{[\gamma\alpha, i]} \; A \to \gamma \in P, |\gamma\alpha| \leq n - i$$

Example: Top-Down (4)

Axiom is the whole input w as remaining input (i.e., only the part up to position 0 has been parsed) and a stack containing S:

$$\text{Axiom: } \frac{}{[S, 0]}$$

The goal item is an empty stack with the input up to position n already parsed:

$$\text{Goal: } [\epsilon, n] \text{ with } |w| = n$$

Example: Top-Down (5)

Top-Down

CFG $S \to aSb \mid ab$, input $w = aaabbb$:

Deduced items (only successful ones are listed):

$[S, 0]$	axiom
$[aSb, 0]$	predict
$[Sb, 1]$	scan
$[aSbb, 1]$	predict
$[Sbb, 2]$	scan
$[abbb, 2]$	predict
$[bbb, 3]$, $[bb, 4]$, $[b, 5]$, $[\epsilon, 6]$	scan

Example: Top-Down (6)

How about soundness and completeness?

There is a direct correspondence between leftmost derivations of a w and parses in a top down parser:

- Soundness: If $[\alpha, i]$, then $S \stackrel{*}{\Rightarrow} w_1 \dots w_i \alpha$.
- Completeness: If $S \stackrel{*}{\Rightarrow} w_1 \dots w_i \gamma$ is a leftmost derivation where $\gamma \in (N \cup T)^*$ such that $\gamma = A\gamma'$, $A \in N$ or $\gamma = \epsilon$, then $[\gamma, i]$.

Implementation issues (1)

When dealing with natural languages, we are in general faced with highly ambiguous grammars.

1. On the one hand, strings can have more than one analysis. Consequently, we need to find some way to branch and pursue all of them.

2. On the other hand, different analyses can have common sub-analyses for certain substrings. In order to avoid computing these sub-analyses several times, we need to find some way to reuse (partial) parse trees that we have already found.

$\Rightarrow$ we have to store intermediate parsing results, make sure we pursue all of them and retrieve them if needed in order to reuse them in a different context.

Implementation issues (2)

Computation sharing (tabulation) is particularly easy when using parsing schemata:

- During parsing, we deduce new trees from already existing trees (partial results), present as items.
- The same item can be used in different deductions (and has to be calculated only once).

Related notion: chart parsing. The chart is the structure that contains all intermediate results computed so far.

Implementation issues (3)

Chart parsing:
We have two structures,

- the chart $\mathcal{C}$
- and an agenda $\mathcal{A}$.

Both are initialized as empty.

- We start by computing all items that are axioms, i.e., that can be obtained by applying rules with empty antecedents.
- Starting from these items, we extend the set $\mathcal{C}$ as far as possible by subsequent applications of the deduction rules.
- The agenda contains items that are waiting to be used in further deduction rules. It avoids multiple applications of the same instance of a deduction rule.

Implementation issues (4)

General algorithm for chart parsing (recognizer)

Chart parsing

```
𝒞 = 𝒜 = ∅
for all items I resulting form a rule applica-
tion with empty antecedent set:
    add I to 𝒞 and to 𝒜
while 𝒜 ≠ ∅:
    remove an item I from 𝒜
    for all items I′ deduced from I and items
    from 𝒞 as antecedents:
        if I′ ∉ 𝒞:
            add I′ to 𝒞 and to 𝒜
if there is a goal item in 𝒞:
    return true
else return false
```

Implementation issues (5)

Example: Unger Parsing: The chart is a $(n+1) \times (n+1)$ table where n is the length of the input.

- Whenever an item $[\bullet X, i, j]$ is predicted, we enter it into the chart.
- Whenever an item $[X\bullet, i, j]$ is completed, we replace the predicted item by the completed one.

Implementation issues (5)

Chart parsing: Unger

Sample CFG: $S \rightarrow aSb \mid ab$, input word $w = aaabbb$ (with terminal filter). [a]

	0	1	2	3	4	5	6
6	•S•					•b•	
5		•S•			•b•		
4			•S•	•b•			
3			•a•				
2		•a•					
1	•a•						
0							

[a] Note that for this slide, in handout mode, i.e., without overlays, we have two dots per item, which is of course not correct, we first predict (e.g., •S) and then replace by the recognized item (e.g., S•).

Complexity (1)

For a given grammar G and an input sentence $w \in T^*$, we call the recognition problem the task to decide whether $w \in L(G)$ or not.

- Fixed recognition problem: Assume a given grammar G (fixed). Then decide for a given input word w if $w \in L(G)$. In this case, the complexity of the problem is given only with respect to the size of the input sentence w, i.e., the size of the grammar is taken to be a constant. This is also sometimes called the word recognition problem.

- Universal recognition problem: Decide for an input grammar G and an input word w if $w \in L(G)$. In this case, we have to investigate the complexity of the problem in the size of the input sentence w and the grammar G.

Complexity (2)

- In real natural language applications, we often deal with very large grammars: Grammars extracted from treebanks for instance can easily have much more than $10,000$ productions.
- The average sentence length in natural languages is somewhere between 20 and 30.
- Therefore, for natural language processing, the complexity of the universal recognition problem is an important factor.

Complexity (3)

We distinguish between the time and the space complexity.
We distinguish the following different complexity classes:

- **P (PTIME)**: problems that can be solved deterministically in an amount of time that is polynomial in the size of the input. I.e., there are constants c and a k such that the problem can be solved in an amount of time $\leq cn^k$ where n the size of the input.
  Notation: $\mathcal{O}(n^k)$.
- **NP**: problems whose positive solutions can be verified in polynomial time given the right information, or equivalently, whose solutions can be non-deterministically found in polynomial time.

Complexity (4)

- **NP-complete**: the hardest problems in NP. A problem is NP-complete if any problem in NP can be transformed into it in polynomial time.

The question whether the two classes P and NP are equal or not is an open question. Most people think however that NP is larger.

Complexity (5)

- The specification of parsing algorithms via deduction rules facilitates the computation of the complexity of an algorithm.
- In order to determine the time complexity, we have to calculate the maximal number of (different) rule applications that is possible.
- This depends on the most complex deduction rule in our parsing schema.

Unger: complexity

Most complex rule: complete

$$\frac{[\bullet A, i_0, i_k], [A_1\bullet, i_0, i_1], \dots, [A_k\bullet, i_{k-1}, i_k]}{[A\bullet, i_0, i_k]} \; A \to A_1 \dots A_k \in P$$

Complexity $\mathcal{O}(n^{k+1})$ where k the maximal length of a righthand side in the grammar.

Conclusion

Parsing Schemata

- characterize partial parsing results via items;
- characterize parsing as a deductive process;
- allow to separate the proper algorithm from data structures and control structures;
- facilitate the proof of soundness and completeness of an algorithm;
- facilitate comparisons between different algorithms;
- make the complexity of an algorithm more visible;
- facilitate tabulation and computation sharing.

Bibliography

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1 and 2):3–36.

Sikkel, K. (1997). *Parsing Schemata*. Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York.