Parsing
Shift Reduce Parsing

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Winter 2016/17

Table of contents

1. Introduction
2. Shift and reduce
3. The algorithm
4. Soundness and completeness
5. Control structures

Introduction (1)

CFG parser that is

- a bottom-up parser: we start with the terminals and subsequently replace righthand sides of productions with lefthand sides.
- a directional parser: the replacing of righthand sides with lefthand sides is ordered corresponding to a rightmost derivation.
- a LR-parser: we process the input from left to right while constructing a rightmost derivation.
- a Shift-reduce-parser: the two operations of the parser are shift and reduce.

Introduction (2)

This parser corresponds to the CYK with dotted productions and with more or less an on-line order for filling the chart:

- read input from left to right,
- at every input position i, complete as much as possible

But: instead of a chart, we use a stack that contains the sentential form that we have already found.

Shift and reduce (1)

The parser consists of

- a stack (initially empty) $\Gamma \in (N \cup T)^*$
- the remaining input (initially w).

Idea:

- w is shifted on the stack while, whenever the top of the stack is the rhs of a production in reverse order, this is replaced with the lhs.
- Success if $\Gamma = S$ and remaining input ϵ.

Shift and reduce (2)

For convenience we write the stack with its top on the right.

Example

$S \to ABC, A \to a \mid Aa, B \to b \mid Bb, C \to c$
$w = aabbbc$.
(only successful moves of the parser are listed)

	$aabbbc$	
a	$abbbc$	shift
A	$abbbc$	reduce, $A \to a$
Aa	$bbbc$	shift
A	$bbbc$	reduce, $A \to Aa$
Ab	bbc	shift
AB	bbc	reduce, $B \to b$
ABb	bc	shift

Shift and reduce (3)

Example continued

$S \to ABC, A \to a \mid Aa, B \to b \mid Bb, C \to c$

ABb	bc	
AB	bc	reduce, $B \to Bb$
ABb	c	shift
AB	c	reduce, $B \to Bb$
ABc		shift
ABC		reduce, $C \to c$
S		reduce, $S \to ABC$

If we apply the productions in reverse order we obtain a rightmost derivation:
$S \Rightarrow ABC \Rightarrow ABc \Rightarrow ABbc \Rightarrow ABbbc \Rightarrow Abbbc \Rightarrow Aabbbc \Rightarrow aabbbc$

Shift and reduce (4)

In general, this parsing strategy is non-deterministic.
Non-determinism can arise if there are two productions such that the rhs of one of them is a prefix of the rhs of the other, i.e., if there are different productions $A \to \alpha$, $B \to \alpha\beta$ with $\alpha \in (N \cup T)^+$ and $\beta \in (N \cup T)^*$.

To see this assume that we have such productions. In a situation $\Gamma = \ldots \alpha$ we might have the possibility to either reduce to $\Gamma = \ldots A$ or continue with a sequence of shift and reduce steps leading to $\Gamma = \ldots \alpha\beta$ and then reducing to $\Gamma = \ldots B$.

If parsing is deterministic, we always try reduce first. Only if it is not possible, we perform a shift.

Shift and reduce (5)

In the non-deterministic case, problems can be caused by

- ϵ-productions and
- loops $A \overset{+}{\Rightarrow} A$.

Both can lead to infinite loops of the parser.

The algorithm (1)

Assume a grammar without ϵ-productions and without loops.

```
function bottom-up(w,Γ):
    if w = ϵ and Γ = S then true
    else reduce(w,Γ) or shift(w,Γ)
```

```
function shift(w,Γ):
    if w = ϵ then false
    else if w = aw′, a ∈ T
        then bottom-up(w′,Γa)
```

The algorithm (2)

```
function reduce(w,Γ):
    out := false;
    for every A → α ∈ P:
        if Γ = Γ′α and bottom-up(w,Γ′A)
        then out := true;
    return out
```

Initial call: `bottom-up(`w`,` ϵ`)`

The algorithm (3)

Shift reduce parsing schema

Parsing schema for shift-reduce parsing:
Item form $[\Gamma, i]$ (w has been shifted up to position i).

Axiom: $\dfrac{}{[\epsilon, 0]}$

Reduce: $\dfrac{[\Gamma\alpha, i]}{[\Gamma A, i]}\ A \to \alpha \in P$

Shift: $\dfrac{[\Gamma, i]}{[\Gamma a, i+1]}\ w_{i+1} = a$

Goal item $[S, n]$.

The algorithm (4)

Shift-reduce parsing is exactly what is done by the following PDA constructed from a CFG:

- start with stack Z_0 and q_0;
- $\langle q_0, aZ\rangle \in \delta(q_0, a, Z)$ for all $a \in T, Z \in N \cup T \cup \{Z_0\}$ (shift);
- $\langle q_0, A\rangle \in \delta(q_0, \epsilon, \alpha^R)$ for all $A \to \alpha$ (reduce);
- $\langle q_1, \epsilon\rangle \in \delta(q_0, \epsilon, S)$;
- $\langle q_f, \epsilon\rangle \in \delta(q_1, \epsilon, Z_0)$.

(LR PDA construction in JFLAP for a given CFG)

The algorithm (5)

In the non-deterministic case, the number of items can be quite large.
Example: $S \to aB \mid bA$, $A \to a \mid aS \mid bAA$, $B \to b \mid bS \mid aBB$
$w = ab$ yields 8 items:

1.	$[\epsilon, 0]$	axiom
2.	$[a, 1]$	shift
3.	$[A, 1]$	reduce from 2.
4.	$[ab, 2]$	shift from 2.
5.	$[Ab, 2]$	shift from 3.
6.	$[aB, 2]$	reduce from 4.
7.	$[AB, 2]$	reduce from 5.
8.	$[S, 2]$	reduce from 6.

$w = abba$ yields 49 items! (At some point, 11 possibilities are pursued in parallel.)

Soundness and completeness

To prove that our algorithm is correct (sound and complete), we have to show that $[\Gamma, i]$ iff $\Gamma \stackrel{*}{\Rightarrow} w_1 \dots w_i$.
We split this into two parts:

1. Soundness:
   If $[\Gamma, i]$ then $\Gamma \stackrel{*}{\Rightarrow} w_1 \dots w_i$.
   (Can be shown with an induction over the deduction rules.)
2. Completeness:
   If $\Gamma \stackrel{l}{\Rightarrow} w_1 \dots w_i$ then $[\Gamma, i]$.
   (Can be shown with an induction over l assuming a rightmost derivation.)

Control structures (1)

As in the LL-parsing (top-down) case, there are two possibilities:

- either proceed depth-first (try one reduce, pursue as far as possible, backtrack if parsing not successful),
- or proceed breadth-first (try all possible reduce and shift operations in parallel).

Control structures (2)

Advantages and disadvantages are similar as in the top-down case.
Breadth-first:

- Needs a lot of memory.
- Better for on-line parsing. (At every moment, *all* analyses for the input that has been seen so far have been computed.)

Depth-first (backtracking):

- Does not need much memory.
- Preferable in a probabilistic setting when we search only for the best solution.

Conclusion

Important features of directional bottom-up parsing:

- LR-parsing: input processed from left to right, constructs a rightmost derivation;
- parsing steps shift and reduce;
- non-deterministic in general;
- different control structures (breadth-first, depth-first);
- does not work for grammars with loops or ϵ-productions;
- no chart parser.