xLR(k): deterministic bottom-up parsing

Parsing
ISCL-BA-06

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2020/21

version: ff0149d+ @2021-01-25

Recap: bottom-up parsing

- Start from the input symbols, try to *reduce* the input to the start symbol
- Unlike top-down parsing where *productions* drive the parsing, in bottom-up parsing *reduction* is the main operation
- Reduction matches RHS of a grammar rule, and replaces it with its LHS
- A typical bottom-up parser has two basic operations

reduce	replace one more more symbols in the sentential form with their LHS non-terminal
shift	move the next unprocessed symbol from the input to the sentential form

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
	dnvan	shift

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
d	nvan	shift

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llllll} S & \rightarrow NP\ VP & NP & \rightarrow d\ AN & NP & \rightarrow AN \\ VP & \rightarrow v\ NP & AN & \rightarrow a\ AN & AN & \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dn	van	shift

shift/reduce conflict

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dnv	an	shift

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

Sent. Form	Input	Action
dn	van	r: AN → n
dnva	n	shift

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dnvan		r: AN → n

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

Sent. Form	Input	Action
dn	van	r: AN → n
dnva AN		r: AN → a AN
dnva AN		r: NP → AN

reduce/reduce conflict

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dnva AN		r: AN → a AN
dnva NP		reject

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dnv AN		r: AN → a AN

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dnv AN		r: NP → AN

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dnv NP		r: VP → v NP

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llll} \text{S} \rightarrow \text{NP VP} & \text{NP} \rightarrow \text{d AN} & \text{NP} \rightarrow \text{AN} \\ \text{VP} \rightarrow \text{v NP} & \text{AN} \rightarrow \text{a AN} & \text{AN} \rightarrow \text{n} \end{array}$$

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n
dn VP		reject

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
dn	van	r: AN → n

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

Sent. Form	Input	Action
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN	van	shift

shift/reduce conflict

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN v	an	shift

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llllll} S & \rightarrow NP\ VP & NP & \rightarrow d\ AN & NP & \rightarrow AN \\ VP & \rightarrow v\ NP & AN & \rightarrow a\ AN & AN & \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN va	n	shift

Bottom-up (shift-reduce) parsing: an example

S → NP VP	NP → d AN	NP → AN
VP → v NP	AN → a AN	AN → n

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN van		r: AN → N

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN va AN		r: AN → a AN
d AN va AN		r: NP → AN

reduce/reduce conflict

Bottom-up (shift-reduce) parsing: an example

S → NP VP	NP → d AN	NP → AN
VP → v NP	AN → a AN	AN → n

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN va AN		r: AN → a AN
d AN va NP		reject

Bottom-up (shift-reduce) parsing: an example

S → NP VP   NP → d AN   NP → AN
VP → v NP   AN → a AN   AN → n

Sent. Form	Input	Action
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN v NP		r: VP → v NP

Bottom-up (shift-reduce) parsing: an example

$$
\begin{array}{lll}
S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\
VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n
\end{array}
$$

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d AN	van	r: NP → AN
d AN VP		reject

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llllll} S & \rightarrow NP\ VP & NP & \rightarrow d\ AN & NP & \rightarrow AN \\ VP & \rightarrow v\ NP & AN & \rightarrow a\ AN & AN & \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
d AN	van	r: NP → d AN
d NP	van	reject

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP	van	shift

Bottom-up (shift-reduce) parsing: an example

$$
\begin{array}{llllll}
S & \rightarrow NP\ VP & NP & \rightarrow d\ AN & NP & \rightarrow AN \\
VP & \rightarrow v\ NP & AN & \rightarrow a\ AN & AN & \rightarrow n
\end{array}
$$

SENT. FORM	INPUT	ACTION
NP v	an	shift

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP va	n	shift

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP van		r: AN → n

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP va AN		r: AN → a AN
NP va AN		r: NP → AN

reduce/reduce conflict

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{llllll} S & \rightarrow NP\ VP & NP & \rightarrow d\ AN & NP & \rightarrow AN \\ VP & \rightarrow v\ NP & AN & \rightarrow a\ AN & AN & \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP va AN		r: AN → a AN
NP va NP		reject

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP v AN		r: NP → AN

Bottom-up (shift-reduce) parsing: an example

$$\begin{array}{lll} S \rightarrow NP\ VP & NP \rightarrow d\ AN & NP \rightarrow AN \\ VP \rightarrow v\ NP & AN \rightarrow a\ AN & AN \rightarrow n \end{array}$$

SENT. FORM	INPUT	ACTION
NP v NP		r: VP → v NP

Bottom-up (shift-reduce) parsing: an example

S → NP VP	NP → d AN	NP → AN
VP → v NP	AN → a AN	AN → n

SENT. FORM	INPUT	ACTION
NP VP		r: S → NP VP

Bottom-up (shift-reduce) parsing: an example

S → NP VP NP → d AN NP → AN
VP → v NP AN → a AN AN → n

SENT. FORM	INPUT	ACTION
S		accept

Two issues with a backtracking shift-reduce parser

- Obvious one: reduce/reduce and shift/reduce conflicts mean non-determinism
- Not-so-obvious one: recognizing ‘handles’:
  - The rule that we locate at the right edge of the active sentential form is called a *handle*
  - For variable RHS, we need to search the grammar to determine which rule applies (if any)
- In a efficient parser we want to avoid both

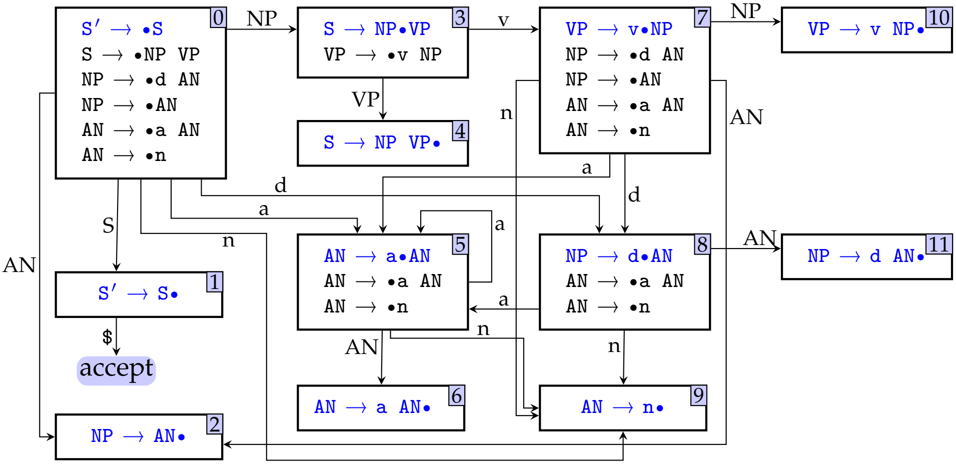
Table driven bottom-up parsing

- The extra work done by a backtracking shift-reduce parser can be eliminated for a large class of grammars
- The general idea is the same with LL(k) grammars: preprocess the grammar to construct a table
- The class of LR(k) (scanning from ***L****eft-to-right*, producing a ***R****rightmost derivation*) grammars can be parsed deterministically using k lookahead symbols
- $k = 1$ is most common, $LR(0)$ parser are also useful in some cases, larger k allows expressive grammars
- LL(k) grammars are a subset of LR(k) grammars
- Most practical programming language compilers are LR(1) parsers
- LR(k) parsers are difficult to build manually, but tools that take a CF grammar and construct and LR(1) parser are in common user (e.g., `yacc`)

Dotted rules, or ‘items’, (again) and augmented grammars

- An LR parser keeps a set of states (actually a finite-state automaton) to represent the current parser state during parsing
- An LR parser’s states are sets of ‘dotted rules’ similar to Early or chart parsers we discussed earlier
  - $\mathtt{A} \rightarrow \bullet\alpha$
  - $\mathtt{A} \rightarrow \alpha\bullet\beta$
  - $\mathtt{A} \rightarrow \alpha\bullet$
- We also introduce a new start symbol, with a single production $S^{'} \rightarrow S$
- This rule helps parser to determine when to stop: the parser accepts the input only when reducing S to $S^{'}$

LR(0) automaton



0
S′ → •S
S → •NP VP
NP → •d AN
NP → •AN
AN → •a AN
AN → •n

3
S → NP•VP
VP → •v NP

7
VP → v•NP
NP → •d AN
NP → •AN
AN → •a AN
AN → •n

10
VP → v NP•

4
S → NP VP•

1
S′ → S•

accept

5
AN → a•AN
AN → •a AN
AN → •n

8
NP → d•AN
AN → •a AN
AN → •n

11
NP → d AN•

6
AN → a AN•

9
AN → n•

2
NP → AN•

Shift-reduce parsing with LR(0) automaton

- The simplest version of the LR parsers uses LR(0) automaton to guide the parsing decisions
  - Use a stack to keep track of active states
  - Start with state 0
  - If there is an outgoing edge labeled with the current input, shift: push the target state to the stack
  - Otherwise reduce based on contents of the current state. For example, if the current state contains `S` $\rightarrow$ `NP VP•`,
    - pop two symbols (for NP and VP) from the stack
    - push the state reachable through S from the state on the top of the stack

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S$'$ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Example

Parsing with LR(0) automaton 1

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce $S' \rightarrow S$								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0		d n v a n $	shift

Example

Parsing with LR(0) automaton 2

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce $S' \rightarrow S$								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 8	d	n v a n $	shift

Example

Parsing with LR(0) automaton 3

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

STACK	SENT. FORM	INPUT	ACTION
0 8 9	d n	v a n $	AN → n

Example

Parsing with LR(0) automaton 4

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 8 11	d AN	v a n $	NP → d AN

Example

Parsing with LR(0) automaton 5

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 3	NP	v a n $	shift

Example

Parsing with LR(0) automaton 6

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 3 7	NP v	a n $	shift

Example

Parsing with LR(0) automaton 7

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 3 7 5	NP v a	n $	shift

Example

Parsing with LR(0) automaton 8

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 3 7 5 9	NP v a n	$	shift

Example

Parsing with LR(0) automaton 9

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 3 7 5 6	NP v a AN	$	AN → n

Example

Parsing with LR(0) automaton 10

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

STACK	SENT. FORM	INPUT	ACTION
0 3 7 2	NP v AN	$	AN → a AN

Example

Parsing with LR(0) automaton 11

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S' → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

Stack	Sent. Form	Input	Action
0 3 7 10	NP v NP	$	NP → AN

Example

Parsing with LR(0) automaton 12

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

STACK	SENT. FORM	INPUT	ACTION
0 3 4	NP VP	$	VP → v NP

Example

Parsing with LR(0) automaton 13

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

STACK	SENT. FORM	INPUT	ACTION
0 1	S	$	S → NP VP

Example

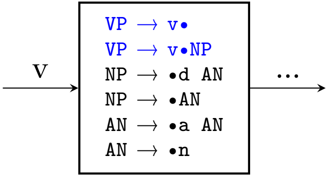
Parsing with LR(0) automaton 14

state	ACTION	GOTO							
		a	d	n	v	S	NP	VP	AN
0	shift	5	8	9	e	1	3	e	2
1	reduce S′ → S								
2	reduce NP → AN								
3	shift	e	e	e	7	e		4	
4	reduce S → NP VP								
5	shift	5	e	9	e				6
6	reduce AN → a AN								
7	shift	5	8	9	e		10		2
8	shift	5	e	9	e				11
9	reduce AN → n								
10	reduce VP → v NP								
11	reduce NP → d AN								

STACK	SENT. FORM	INPUT	ACTION
0 1	S	$	accept

Limitations of LR(0)

- Assume we have an additional rule: VP → v
- This would lead to a LR(0) automaton entry



VP → v•
VP → v•NP
NP → •d AN
NP → •AN
AN → •a AN
AN → •n

- We have a shift/reduce conflict
- A simple solution (SLR): shift if possible, otherwise reduce
- In general LR(0) parsers/grammars are limited, for most purposes we need more powerful parsers

LR parsers with lookahead

- LR(k): parsers augment the chart entries (items) with lookahead
- Lookahead allows LR(k) parser to parse a larger class of grammars
- The disadvantage is much larger chart sizes
- Another option is the LALR(k) parsers which use a smaller automaton
- LALR(1) parsers and parser generators are commonly used in practice

Why use xLR(k) parsers?

- LR(k) parsers general, efficient (non-backtracking) shift-reduce parsers
- LR(k) parsers can be constructed for (almost) any formal/programming language constructs
- In general LR(k) grammars are more expressive. LL(k) is a subset of LR(k)
- LR(k) parsers can detect syntax errors as soon as it is possible to detect them

LR grammars and ambiguity

- LR(k) parsers cannot handle ambiguity
- If a grammar is ambiguous we cannot construct an LR(k) parse table for it
- In general, determining whether a grammar is ambiguous is intractable
- This is sometimes used for a test for ambiguity:
  - If we can build a LR(k) parser for a grammar, then it is not ambiguous
  - If we cannot, it is inconclusive

What about natural language parsing

- Natural languages are inherently ambiguous
- As a result, we cannot use these parsers for parsing natural languages
- Nevertheless, the techniques are useful
  - We can use LR-like parsers to reduce the non-determinism: GLR parsers (also known as Tomita parser)
  - Instead of a table-driven parser, we can predict the action with a machine learning method: transition-based dependency parsers do that

Summary

- xLR(k) parsers are powerful bottom-up deterministic parsers
- LR grammars are more general than LL grammars
- These parsers are difficult to build manually, but automatic parser generators exist
- Although they cannot handle ambiguity, the similar ideas are also used in natural language parsers to reduce the non-determinism
- Understanding the concepts here is useful for building parser generators and understanding the related natural language parsers
- Reading suggestion: Grune and Jacobs (2007, ch.9), Aho et al. (2007, Section 4.5–4.7)

Acknowledgments, references, additional reading material

Aho, Alfred V., Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman (2007). *Compilers: Principles, Techniques, & Tools*. Pearson/Addison Wesley. ISBN: 9780321486813.

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.

blank

blank

blank

blank

blank

blank