LR(k) Grammars

L: read input Left-to-right
R: generate rightmost derivation ("backwards")
k: # of lookahead symbols at each step

LR parsing is bottom-up:

We try to generate a derivation tree starting at the bottom of the tree, reducing terminal symbols and sentential forms to their nonterminal equivalents, until we get the initial nonterminal back on the root of the tree.

Let G = (N, ∑, P, S) be a CFG.

Suppose $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta w \underset{rm}{\overset{*}{\Rightarrow}} xw$ is a rightmost derivation.

We say the sentential form $\alpha\beta w$ can be <u>reduced</u> under the production A -> β to the sentential form $\alpha A w$. We call the substring β a <u>handle</u> of $\alpha\beta w$.

So, we will try to find handles and reduce them until we can get the initial nonterminal! A handle is ***any*** string of symbols (terminal or nonterminal), as long as it appears on the right-hand side of a grammar.

<u>Example</u>:

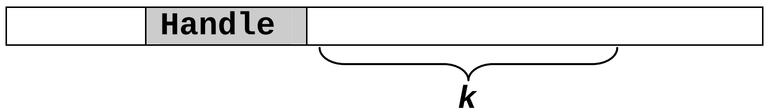
Given the following CFG and a rightmost derivation:
(1) S -> SaSb
(2) S -> ε

$$S \underset{rm}{\overset{1}{\Rightarrow}} Sa\underline{S}b \underset{rm}{\overset{1}{\Rightarrow}} SaSaSbb \underset{rm}{\overset{2}{\Rightarrow}} SaSabb \underset{rm}{\overset{2}{\Rightarrow}} Saabb \underset{rm}{\overset{2}{\Rightarrow}} aabb$$

We say that aabb can be reduced under S -> ε to Saabb, Saabb can be reduced under S -> ε to SaSabb, SaSabb can be reduced under S -> ε to SaSaSbb, SaSaSbb can be reduced under S -> SaSb to SaSb, which can then be reduced under S -> SaSb to the root S.

Intuitively, a grammar is LR(k) if, given a rightmost derivation $S \underset{rm}{\Rightarrow} \alpha_0 \underset{rm}{\Rightarrow} \alpha_1 \underset{rm}{\Rightarrow} \alpha_2 \ldots \underset{rm}{\Rightarrow} \alpha_m = z$, we can isolate the handle of each sentential form and determine which nonterminal is to replace the handle by scanning α_i from left to right, but only going at most k symbols past the right end of the handle of α_i.

α_i:



Definition: Let $G = (N, \Sigma, P, S)$ be a CFG and let G' be its augmented grammar. $G' = (N \cup \{S'\}, \Sigma, P \cup \{S' \to S\}, S')$.

We say that G is LR(k) if

1.) $S' \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta w$,

2.) $S' \overset{*}{\underset{rm}{\Rightarrow}} \gamma B x \underset{rm}{\Rightarrow} \alpha\beta y$, and

3.) $FIRST_k(w) = FIRST_k(y)$

imply that $\alpha A y = \gamma B x$ (i.e. $\alpha = \gamma$, $A = B$, and $y = x$).

Example:

S' -> S
S -> 0S1 | A
A -> 1A | 1 this is not LR(1)

Proved by definition: Consider the following two rightmost derivations:

S' -> S -> 0S1 -> 0A1 -> 01A1 -> 0111
(α:01; A:A; w:1; β:1)
S' -> S -> 0S1 -> 0A1 -> 01A1 -> 011A1 -> 01111
(γ:011; B:A; x:1; y:11)

$FIRST_1(w) = \{1\} = FIRST_1(y) = \{1\}$

However, $\alpha A y = 01A11 \neq \gamma B x = 011A1$

This is called a Shift-Reduce conflict.

Example: S -> Ab | Bc
A -> Aa | ε
B -> Ba | ε   this is not LR(k) for any k.

For bottom-up parsing, we know that the ε on top of the stack is the handle at the beginning of the process. However, in order to choose the correct nonterminal for ε to reduce to, we need to see the last character (which will either be a 'b' or a 'c'). This could be any number of lookaheads away — the string of a's could be of arbitrary length — so there is no fixed value k.

This is called a Reduce-Reduce conflict.

Example: (0) S' -> S
(1) S -> SaSb
(2) S -> ε

{V stands for viable prefix, which is the set of prefixes of sentential forms in the rightmost derivation that can appear on top of the stack during parsing ∀ input strings.}

$A_0 = V_1(\varepsilon)$

```
I:   [S' -> .S, ε]  //things to the right of the dot
                    //need to be processed
                    //ε is the local follow set of S'

                    //S is after dot: 2 S-productions
                    //create 2 more entries (see the
                    //rule below to add items)
     [S -> .SaSb, ε]
     [S -> . , ε]
                    //S is after dot: 2 more entries
     [S -> .SaSb, a]
     [S -> . , a]
                    //Since we have sets that are the
                    //same, but with different follow
                    //sets, we combine them:

     1. [S' -> .S, ε]
     2. [S -> .SaSb, ε | a]
     3. [S -> . , ε | a]
```

CHECK FOR CONSISTENCY:

$\varepsilon \mid a \in ?\ EFF_1(S\varepsilon) = \phi$

$EFF_1(SaSb\varepsilon) = \phi$

$EFF_1(SaSba) = \phi$

If ε or a is ***not*** in these sets, we can keep going...
If it ***is*** in the EFF sets, that means one symbol lookahead is not enough and we stop! We are comparing the lookahead of ***shift items*** with the lookahead ***of*** the ***reduce item***. (See below for more information on EFF.)

Rules to add new items to a table A_i: If $[A \to .B\alpha, u]$ is in $V_k(\gamma)$ and $B \to \beta$ is a production, then $\forall\ x \in FIRST_k(\alpha u)$, add to $V_k(\gamma)$ the item $[B \to .\beta, x]$.

We also need to add new tables until all symbols to the right of dot have been shifted to the left, for all items. So, we shift S in items 1. and 2. above, to get table A_1:

$A_1 = V_1(S)$ = goto(A_0, S)
I: [S' -> S., ε]
I: [S -> S.aSb, ε | a]

//No nonterminals after the dot, so we don't create more items.

$\varepsilon \in ?\ EFF_1(aSb\varepsilon) = \phi$ No
$EFF_1(aSba) = \phi$ No
So, consistent.

$A_2 = V_1(Sa)$ = goto(A_1, a)   //Sa are on top of stack
I: [S -> Sa.Sb, ε | a]
[S -> .SaSb, b]
[S -> . , b]
[S -> .SaSb, a]
[S -> ., a]
//we should keep going, but we would get the same
//answers, so we stop here and combine:

1. [S -> Sa.Sb, ε | a]
2. [S -> .SaSb, b | a]
3. [S -> . , b | a]

$b \mid a \in ?\ \mathrm{EFF}_1(Sb\varepsilon) = \phi$ No
$\mathrm{EFF}_1(Sba) = \phi$ No
$\mathrm{EFF}_1(SaSbb) = \phi$ No
$\mathrm{EFF}_1(SaSba) = \phi$ No
So, consistent.

$A_3 = V_1(SaS)$ = goto(A_2, S)
I: [S -> SaS.b, ε | a]
I: [S -> S.aSb, b | a]
All shift items, consistent.

$A_4 = V_1(SaSb)$ = goto(A_3, b)
I: [S -> SaSb., ε | a]
One item, consistent.

$A_5 = V_1(SaSa)$ = goto(A_3, a)
I: [S -> Sa.Sb, b | a]
[S -> .SaSb, b | a]
[S -> . , b | a]

$b \mid a \in ?\ \mathrm{EFF}_1(Sbb) = \phi$ No
$\mathrm{EFF}_1(Sba) = \phi$ No
$\mathrm{EFF}_1(SaSbb) = \phi$ No
$\mathrm{EFF}_1(SaSba) = \phi$ No
So, consistent.

$A_6 = V_1(SaSaS)$ = goto(A_5, S)
I: [S -> SaS.b, b | a]
I: [S -> S.aSb, b | a] ------->{shift a, goto A_5}
All shift items, consistent.

$A_7 = V_1(SaSaSb)$ = goto(A_6, b)
I: [S -> SaSb., b | a]
One item, consistent.

Since everything is consistent, G is LR(1)!

Rules to check for consistency: A set $V_K(\alpha)$ of LR(k) items is consistent if there do not exist items

$[A \to \beta., u]$ and //reduce item
$[B \to \gamma_1 . \gamma_2, v]$ //shift item if $\gamma_2 \neq \varepsilon$ or
//reduce item if $\gamma_2 = \varepsilon$

where $u \in EFF_K(\gamma_2 v)$.

The e-free first function, denoted by $EFF_k(\alpha)$, is defined:

1.) if α begins with a terminal, $EFF_K(\alpha) = FIRST_K(\alpha)$
2.) if α begins with a nonterminal, then
    $EFF_K(\alpha) = \{ w \mid w \in FIRST_K(\alpha)$ and
    there is a derivation $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta \underset{rm}{\Rightarrow} wx$
    where $\beta \neq Awx$, for any nonterminal A.$\}$

Note: $EFF_K(x_1x_2...x_m) = EFF_K(x_1) \oplus_K FIRST_K(x_2) ... \oplus_K FIRST_K(x_m)$.

Example: $S \to AB$
$A \to Ba \mid \varepsilon$
$B \to Cb \mid C$
$C \to c \mid \varepsilon$

$FIRST_2(S) = \{\varepsilon, c, b, a, cb, ba, ab, ac, ca\}$
$EFF_2(S) = \{cb, ca\}$

EFF_k does not include any part of the parse tree where strings were generated by replacing the leading nonterminal with ε! (It is helpful to see the parse tree.)

Example: $S \to aAb$
$A \to \varepsilon$
$EFF_2(S) = \{ab\}$

LR(1) Parser for the grammar S -> SaSb | ε from page 3:

|lookahead| ≤ 1 $\qquad\qquad$ $N \cup \sum$

	f: Parsing Action Section			g: Go To Section		
	a	**b**	**ε**	**S**	**a**	**b**
A_0	r, 2		r, 2	A_1		
A_1	s		r, 0 (acc)		A_2	
A_2	r, 2	r, 2		A_3		
A_3	s	s			A_5	A_4
A_4	r, 1		r, 1			
A_5	r, 2	r, 2		A_6		
A_6	s	s			A_5	A_7
A_7	r, 1	r, 1				

f(u) = <u>shift</u> if $[A \to \beta_1 .\ \beta_2,\ v]$
where $\beta_2 \neq \varepsilon$ and $u \in EFF_K(\beta_2 v)$

f(u) = <u>reduce</u> if $[A \to \beta_1 .,\ u]$

f(ε) = <u>accept</u> if $[S' \to S.,\ \varepsilon]$

f(u) = <u>error</u> otherwise

LR(1) Parsing for "aabb":

	Stack	Input	Output
	A_0	aabb	<empty>
r,2	A_0SA_1	aabb	2
s	$A_0SA_1aA_2$	abb	"
r,2	$A_0SA_1aA_2SA_3$	abb	2, 2
s	$A_0SA_1aA_2SA_3aA_5$	bb	"
r,2	$A_0SA_1aA_2SA_3aA_5SA_6$	bb	2, 2, 2
s	$A_0SA_1aA_2SA_3aA_5SA_6bA_7$	b	"
r,1	$A_0SA_1aA_2SA_3$	b	2,2,2,1
s	$A_0SA_1aA_2SA_3bA_4$	ε	"
r,1	A_0SA_1	ε	2,2,2,1,1
r,0	A_0S'	ε	222110

accept and output 0, 1, 1, 2, 2, 2

When do you augment the grammar? We must augment the given grammar G = (N, ∑, P, S) when S appears on the right-hand side of any production. With S' as the new initial nonterminal, we always know whether we should accept the present input string or continue parsing.

Example:
Prove that the following grammar is LR(0). S never appears on the right side of any production, so we don't need to augment it.

1.) S -> C
2.) S -> D
3.) C -> aC
4.) C -> b
5.) D -> aD
6.) D -> c

$A_0 = V_0(\varepsilon)$

```
I:   [S -> .C]          //these are all shift items
I:   [S -> .D]          //so it is consistent
     [C -> .aC]
     [C -> .b]
     [D -> .aD]
     [D -> .c]
```

$A_1 = V_0(C)$ = goto (A_0, C):

```
I:   [S -> C.]          //only 1 item; successful
```

$A_2 = V_0(D)$ = goto (A_0, D):

```
I:   [S -> D.]          //only 1 item; consistent
```

$A_3 = V_0(a)$ = goto (A_0, a):

```
I:   [C -> a.C]         //all shift items
I:   [D -> a.D]
     [C -> .aC] {goto A3}
     [C -> .b] {goto A4}
     [D -> .aD] {goto A3}
     [D -> .c] {goto A5}
```

$A_4 = V_0(b)$ = goto (A_0, b):

```
I:   [C -> b.]          //one item, consistent
```

$A_5 = V_0(c)$ = goto (A_0, c):

```
I:   [D -> c.]          //one item, consistent
```

$A_6 = V_0(aC)$ = goto (A_3, C):
I: [C -> aC.] //one item, consistent

$A_7 = V_0(aD)$ = goto (A_3, D):
I: [D -> aD.] //one item, consistent

No inconsistencies, so the grammar is LR(0).

The LR(0) parser:

	ε	S	C	D	a	b	c
A_0	s		A_1	A_2	A_3	A_4	A_5
A_1	r,1 accept						
A_2	r,2 accept						
A_3	s		A_6	A_7	A_3	A_4	A_5
A_4	r,4						
A_5	r,6						
A_6	r,3						
A_7	r,5						

LR(0) Parsing for "aab":

	Stack	Input	Output
	A_0	aab	<empty>
s	A_0aA_3	ab	"
s	$A_0aA_3aA_3$	b	"
s	$A_0aA_3aA_3bA_4$	ε	"
r,4	$A_0aA_3aA_3CA_6$	"	4
r,3	$A_0aA_3CA_6$	"	4,3
r,3	A_0CA_1	"	4,3,3
r,1	A_0S	"	4,3,3,1

accept and output 1, 3, 3, 4

This grammar cannot be LL(k) for any fixed value of k to skip a’s. But LR parser can delay the decision and keep shifting a’s into stack until either ‘b’ or ‘c’ appears, then decide which production to be used for reduce correctly. So, LR(0).

Example:
Is the following grammar LR(2)?

1.) S -> 0S1
2.) S -> A
3.) A -> 1A
4.) A -> 1

We need to augment this grammar! Add the following production:

0.) S' -> S

$\underline{A_0 = V_2(\varepsilon)}$

I: [S' -> .S, ε] //all shift items: consistent
[S -> .0S1, ε]
[S -> .A, ε]
[A -> .1A, ε]
[A -> .1, ε]

$\underline{A_1 = V_2(S)}$ = goto (A_0, S)

I: [S' -> S., ε] //one item: consistent

$\underline{A_2 = V_2(0)}$ = goto (A_0, 0)

I: [S -> 0.S1, ε] //all shift items: consistent
[S -> .0S1, 1]
[S -> .A, 1]
[S -> .1A, 1]
[S -> .1, 1]

$\underline{A_3 = V_2(A)}$ = goto (A_0, A)

I: [S -> A., ε] //one item

$\underline{A_4 = V_2(1)}$ = goto (A_0, 1)

I: [A -> 1.A, ε]
I: [A -> 1., ε]
[A -> .1A, ε] //goto A_4
[A -> .1, ε] //goto A_4

$\varepsilon \in ?\ EFF_2(A\varepsilon) = \{1, 11\}$ No
$EFF_2(1A\varepsilon) = \{11\}$ No
$EFF_2(1\varepsilon) = \{1\}$ No

Therefore, it is consistent.

$A_5 = V_2(0S)$ = goto (A_2, S)
I: [S -> 0S.1, ε] //one item: consistent

$A_6 = V_2(00)$ = goto (A_2, 0)
I: [S -> 0.S1, 1] //all shift items: consistent
[S -> .0S1, 11]
[S -> .A, 11]
[A -> .1A, 11]
[A -> .1, 11]

$A_7 = V_2(0A)$ = goto (A_2, A)
I: [S -> .A, 1] //one item

$A_8 = V_2(01)$ = goto (A_2, 1)
I: [A -> 1.A, 1]
[A -> 1., 1]
[A -> .1A, 1]
[A -> .1, 1]

1 ∈? EFF_2(A1) = {11} No
EFF_2(1A1) = {11} No
EFF_2(11) = {11} No

It is consistent.

$A_9 = V_2(1A)$ = goto (A_4, A)
I: [A -> 1A., ε] //one item

$A_{10} = V_2(0S1)$ = goto (A_5, 1)
I: [S -> 0S1., ε] //one item

$A_{11} = V_2(00S)$ = goto (A_6, S)
I: [S -> 0S.1, 1] //one item

$A_{12} = V_2(000)$ = goto (A_6, 0)
I: [S -> 0.S1, 11] //all shift: consistent
[S -> .0S1, 11]
[S -> .A, 11]
[A -> .1A, 11]
[A -> .1, 11]

$A_{13} = V_2(00A)$ = goto (A_6, A)
I: [S -> A., 11] //one item

$\underline{A_{14} = V_2(001)}$ = goto (A_6, 1)

I: [A -> 1.A, 11]
I: [A -> 1., 11]
[A -> .1A, 11]
[A -> .1, 11]

11 $\in$? EFF_2(A11) = {11} Yes
EFF_2(1A11) = {11} Yes
EFF_2(111) = {11} Yes

Therefore, it is <u>not</u> consistent!
This grammar is <u>not</u> LR(2).