LL(1) grammars

A grammar whose parsing table has no multiply-defined entries is said to be **LL(1)** which stands for: scanning the input from **L**eft to right producing a **L**eftmost derivation and using **1** input symbol of lookahead at each step to make parsing action decisions.

Example: the following grammar:

E → T E’
E’ → + T E’ | λ
T → F T’
T’ → * F T’ | λ
F → (E) | **id**

whose parsing table M is

N/I	**id**	+	*	(	)	$
E	E→TE’			E→TE’		
E’		E’→+TE’			E’→ λ	E’→ λ
T	T→FT’			T→FT’		
T’		T’→ λ	T’→*FT’		T’→ λ	T’→ λ
F	F→**id**			F→(E)		

is an **LL(1)** grammar

LL(1) grammars enjoys several nice properties: for example they are not ambiguous and not left recursive.

Example: the following grammar:

S → iEtSS’ | a
S’ → eS | λ
E → F b

whose parsing table M is

N/I	a	b	e	i	t	$
S	S→a			S→iEtSS’		
S’			S’→ λ S’→eS			S’→ λ
E		E→b				

is not **LL(1)** grammar because the table element M[S’,e] has the two entries S’→ λ and S’→eS

Bottom-up parsing

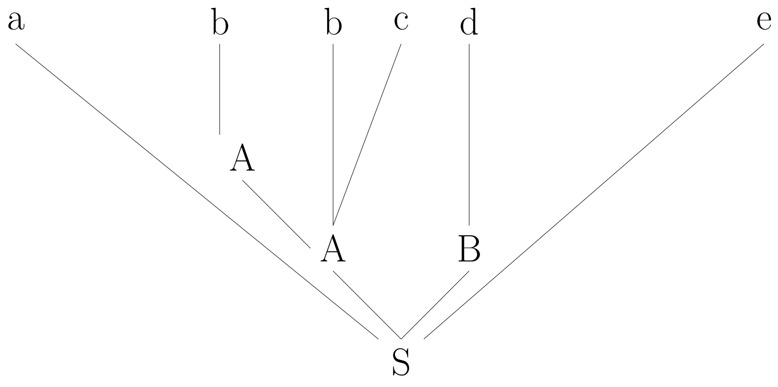
Here we study two types of bottom-up parsing: *shift-reduce* parsing and *LR* parsing which is more general.

Shift-reduce parsing

Shift-reduce parsing attempts to construct a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top)
Example: Consider the following grammar:

S → aABe
A → Abc | b
B → d

the parse tree for the input string abbcde can be formed (bottom-up) as follows:



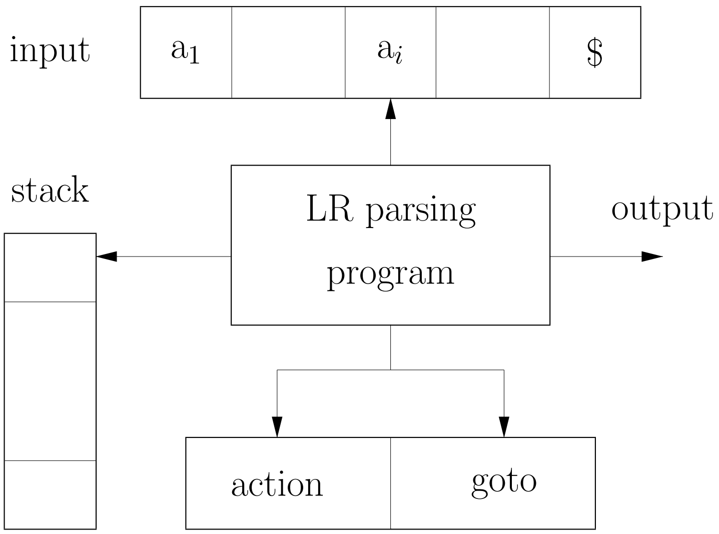
this is similar to rightmost derivation but in reverse order:
abbcde ⇐ aAbcde ⇐ aAde ⇐ aABe ⇐ S

LR parsing

LR parsing is an efficient, bottom-up syntax analysis technique that can be used to parse a large class of context-free grammars.

LR means: scanning the input string from **L**eft-to-right constructing a **R**ightmost derivation in reverse.

The following diagram models the **LR** parser:



LALR parsing: it stands for **L**ook**A**head-**LR**. This parsing method is often used in practice because it produce a smaller tables.