LL(1)のアルゴリズム
「自然言語解析の基礎（産業図書）」p.106~108 を超初心者用に解説

LL(ｋ)の解析は次のステップで行われる。
Step1:文法から LL 解析表を作る。
Step2:LL 解析表を元に解析を行う。

次の生成規則からなる CFG 例を考える。
(1) S → NP　VP
(2)NP → DET*　N*
(3)NP → PRON*
(4)VP → VT*　NP
(5)VP → VI*
(6)DET* → the
(7)N* → girl
(8)N*→ boy
(9)PRON* → you
(10)VT → love
(11)VI → walk
｛注：上の生成規則で*の付いた記号は、品詞（前終端記号）であることを示す｝

Step1 の LL 解析表は次の通りである。上記生成規則の(1)~(5)の横に{……,……,…}という中カッコが付いている。
(1)S → NP　VP　　{DET*, PRON*}
(2)NP → DET*　N*　{DET*}
(3)NP → PRON*　　{PRON*}
(4)VP → VT*　NP　{VT*}
(5)VP → VI*　　{VI*}

(6)以下は品詞を左辺に持つもので、それ以上の展開が無いので解析表には必要ない。
｛….,…,…｝の中カッコの中は、生成規則を最左導出していった時に、最初に現れる品詞である。
例えば、(1)について見てみると、最左なので一番左の非終端記号 NP に生成規則(2)か(3)が適用されるはずである。そのため、(1)が展開されていくと、最初に（(2)なら）DET*か（(3)なら）PRON*が来るはずである。
そのため、(1)S →　NP　VP　　{DET*, PRON*}　となっている。

同様に(2)～(5)の{……}が求まる。

解析例
上の LL 解析表を用いた解析例を示す。
解析する文は「you love the girl」である。これを品詞で表すと「PRON*　VT*　DET*　N*」である。

解析は次の様に進む。以下の解析の進行図では、生成規則の右に（解析を行う）入力文が示してある。
(a) S=> **NP** VP　　pron*　VT*　DET*　N*
最左導出なので NP が展開対象になり、入力の先頭 you は PRON*なので生成規則(3)が適用され、次の(b)に進む。
(b) S=>**PRON***　VP
=>you **VP**　　vt*　DET*　N*
VP の展開では、(4)と(5)が候補であるが、love は VT*なので次の様になる。
(c) => you **VT*** NP
=> you love **NP**　　det* N*
次は NP の展開で、入力文字が det*なので(4)が適用され次の様になる。
(d) =>you love **DET***　　N*
=>You love the **N***　　n*
=>you love the girl
となって、解析が完了する。

以上言葉で説明してきた処理をコンピュータで行うには、表形式の構文解析表とスタックを使った方法が知られている。

下の表が、表形式の構文解析表です（参照「言語と構文解析（共立出版）」）。
一番上の行は、解析のために入力される品詞である。
一番左の列は、TOS（トップオブスタック）である。
表中の記号は、TOS と（入力列の先頭の）入力品詞の交差するセルに、TOS を書き換える記号が書かれている。
例えば、TOS が NP の時、入力品詞が DET*だったら、NP をポップして DET* と N*をプッシュします。この時、DET*の方が後にプッシュする（TOS は DET*になる）。
pop と書かれていれば TOS をポップして、入力を 1 記号ずらします（先頭の記号を捨てる）。
表中の数値は、生成規則番号である。

	DET*	N*	PRON*	VT*	VI*	$
S	NP VP,1		NP VP,1			
NP	DET* N*,2		PRON*,3			
VP				VT* NP,4	VI*,5	
DET*	pop					
N*		pop				
PRON*			pop			
VT*				pop		
VI*					pop	
⊥						accept

「you love the girl」を解析する時のスタックの変化を示す。
TOS は左端で、⊥がスタックの底。
入力の最後が$.

スタック	入力列
S⊥	PRON* VT* DET* N*$
NP VP⊥	PRON* VT* DET* N*$
PRON* VP⊥	PRON* VT* DET* N*$
（PRON*と PRON*で PRON*を pop）	
VP⊥	VT* DET* N*$
VT* NP⊥	VT* DET* N*$
（VT*と VT*で VT*を pop）	
NP⊥	DET* N*$
DET* N*⊥	
（上同様 DET*を pop）	
N* ⊥	N*$
（上同様 N*を pop）	
⊥	$
(accept)で解析成功	

LL(k)は、これを拡張して考える。

付録（ネットで拾ったおまけ）

	下降型構文解析 LL(k)	**上昇型構文解析** LR(k)
利点	文法と解析器との対応が良く 人手で解析器を作るのが容易	扱える文法が幅広く 実用的なプログラミング言語に向く
欠点	扱える文法が少ない	解析器を人手で作るのが困難であり、 生成系の手助けが必要である

解析木と構文木

構文解析は、トークン列を入力し、**解析木**を出力します。**解析木**とよく似た構造に**構文木**があります。これらはよく混同されますので気を付けましょう。表1にそれぞれの特徴を示します。

表 1解析木と構文木

	葉	**節**
解析木	終端記号	非終端記号
構文木	被演算子	演算子

構文木は**解析木**から計算の実質に関係のないものを取り除くことにより得られます。たとえば、
・**解析木**から導出過程を表す非終端記号を取り除く
・**解析木**から演算順序を表す括弧を取り除く
などを行ないます。

図1は4．2節のLR解析の最初の例で入力記号列を $i*(i+i)$ として解析した**解析木**とそれを**構文木**で表現した例です。

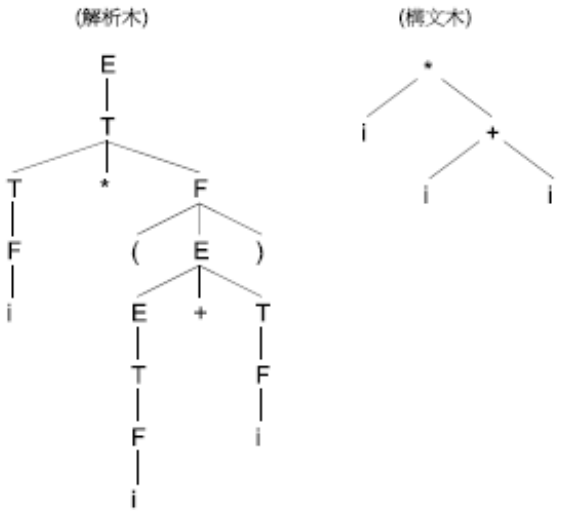

図 1解析木と構文木