文法と言語
ー文脈自由文法とLR構文解析ー

和田俊和

資料保存場所

http://vrl.sys.wakayama-u.ac.jp/SS/

前回までの復習

言語と文法

- 言語とは，ルールに従う記号列の無限集合である.
- 文法を与えることで言語が定義できる.
- 文法にはタイプ0からタイプ3までのレベルがあり，
  - プログラム言語としてはタイプ2「文脈自由文法」
  - プログラム中の字句の表現にはタイプ3「正規文法」

  が用いられる.

文脈自由文法

- 文脈自由文法(Context Free Grammar：CFG)は，前後の記号に関係なく「非終端記号１つ」を「非終端記号と終端記号から成る記号列」に置き換えるという生成規則　A→t のみを持つ文法
- 出発記号に対して生成規則の要素を何度か適用して終端記号列を得ることを「導出」と呼ぶ.
- 終端記号列と文法が与えられたとき，生成規則がどのように適用されたのか，つまり，導出の過程を求めることを「構文解析」と呼び，その結果，導出木が得られる.
- 導出木からそれを簡略化した「構文木（演算子木）」が求められ，それを利用した式の計算などが可能である.

正規文法

- 正規文法は，「非終端記号1つ」を「終端記号」もしくは「終端記号　非終端記号」に置き換えるという生成規則　A→a or B→bB　のみを持つ文法
- 正規文法で表現できるのは，「整数」「実数」「識別子（変数名）」「キーワード」など，比較的単純な記号の並びである.
- 正規文法によって規定される言語は，正規表現によって表現することができる.
- 正規表現から，それに唯一に対応付けられる「非決定性有限状態オートマトン（NFA）」が機械的に対応付けられる.

字句解析オートマトンの生成

- 機械的に求められたNFAは，ε-closureによる新たな状態の導入により計算機で実行可能なDFAに変換することができ，さらに状態数の最適化などが行われ，字句解析に用いられる.
- このような字句解析オートマトンを生成するプログラムに lex がある.
- lex は拡張された正規文法と，C言語プログラムを与えることで，簡単にオートマトンが生成できる.

文脈自由文法における導出

- 導出には最左導出と，最右導出があり，文法および構文解析法によっては，これらの導出が無限ループになることがある.
- 算術式の評価に於いては必ず，最右導出を行う必要がある.
- 最左導出は算術式以外の文を対象とした構文解析に用いることが可能であり，構文解析表とスタックを用いた「LL構文解析」，再帰呼び出しを用いた「再帰下降型構文解析」などがある.
- これら最左導出の構文解析アルゴリズムは，導出木を上から下へと辿る「下降型」構文解析法である.

確認の問題

- push(‘a’,S), push(‘b’,S), pop(S), push(‘c’,S), pop(S),pop(S)
- 上の操作で，pop(S)によって取り出される記号列は？

LL構文解析問題（前回の問題）

- S→F
- S→－F+S
- F→1

上記文法の元で，－１＋１をLL構文解析で構文解析するとどうなるか？構文解析表を求め，導出木を示しなさい．

構文解析表の作り方(例)

1. S→F
2. S→-F+S
3. F→1

	-	1	+	$
S	2	1	-	-
F	-	3	-	-

- **Fi**(F)=φ ， **Fi**(S)=φ
- **Fi**(S)={-}, **Fi**(F)={1}

S→Fというルールから**Fi**(S):=**Fi**(S)∪**Fi**(F)とする

- **Fi**(S)={-,1}, **Fi**(F)={1}

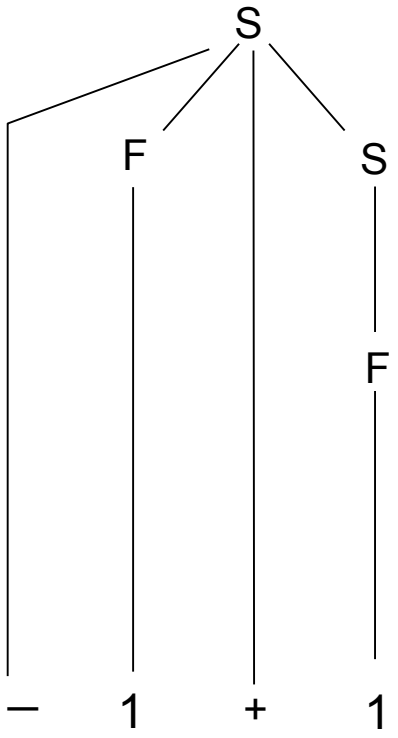
LL構文解析の結果

1. S→F
2. S→-F+S
3. F→1

	-	1	+	$
S	2	1	-	-
F	-	3	-	-

- [S,$], 入力:- ルール2の適用
- [-,F,+S,$], スタックと入力の‘-’を削除.
- [F,+,S,$], 入力:1
- [1,+,S,$],入力:1, スタックと入力の1を削除
- [+,S,$],入力:+スタックと入力の+を削除
- [S,$],入力:1 ルール1の適用
- [F,$],入力:1 ルール3の適用
- [1,$],入力:1 スタックと入力の1を削除

意味:S→-F+S→-1+S→-1+F→-1+1

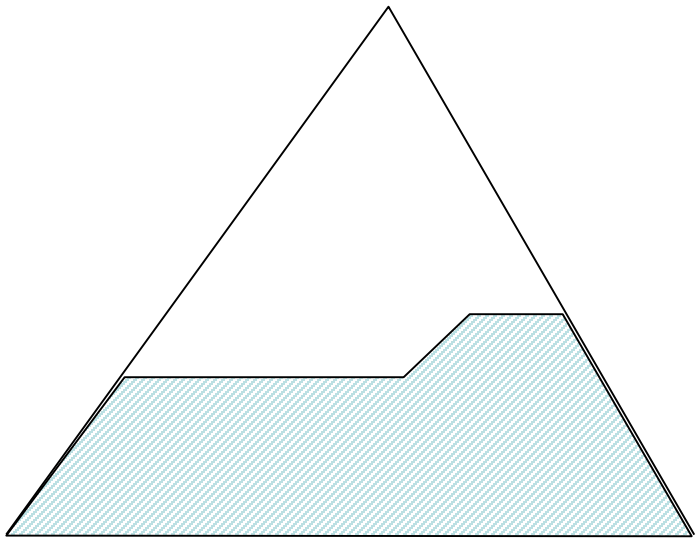


最右構文解析法

LR構文解析

最右導出と上昇型構文解析

- 最右導出を前提とした場合，上昇型の構文解析がしばしば用いられる.
- 上昇型構文解析では生成規則の右辺にマッチする部分を見つけ，それを左辺の非終端記号に置き換える「還元(reduction)」という操作を繰り返し，出発記号に到達する



導出 $\longrightarrow$

$$A \rightarrow \alpha\beta\gamma$$

還元 $\longleftarrow$

還元操作を行うには手がかりが必要

- <算術式>→<算術式><加減演算子><項>
  という生成規則の右辺にマッチする部分を
  １－２－３－４
  という記号列から見つける問題を考える.
- 明らかに”１－２－３”が右辺の＜算術式＞に，次の”－”が＜加減演算子＞に，そして”4”が＜項＞に対応付けられる.
- これをどうやって見つけるかが問題

LR法例題

文法の例

(1) <E> →<E>*<B>

(2) <E> →<E>+<B>

(3) <E> →<B>

(4) <B> → 0

(5) <B> → 1

	アクション					GOTO	
状態	*	+	0	1	$	E	B
0			s1	s2		3	4
1	r4	r4	r4	r4	r4		
2	r5	r5	r5	r5	r5		
3	s5	s6			acc		
4	r3	r3	r3	r3	r3		
5			s1	s2			7
6			s1	s2			8
7	r1	r1	r1	r1	r1		
8	r2	r2	r2	r2	r2		

アクション表と，GOTO表

アクション表は状態と終端記号でインデックス付けされている（終端記号 $ は入力の終わりを示す）。各マスには以下のようなアクションが記述されている:

– *shift*（s*n*）:　次の状態遷移先を *n* とする.

– *reduce*（r*m*）: *m*番の文法規則を実行する.

– *accept*（acc）: 入力バッファにある文字列を受理する.

GOTO表は状態と非終端記号でインデックス付けされており、各マスには非終端記号を解釈し終えた次に遷移する状態を示す.

LR法のアルゴリズム

- スタックを [**0**] と初期化する.（現在の状態は常にスタックトップの数字で表される）
- 現在の状態と入力にある記号からアクション表を参照する。ここで以下の4つのケースがある.
  - s*n* に従って状態遷移する.（これは，還元を行う文字列を一つ右に伸ばす）
    - 現在の終端記号を入力バッファから取り除く.
    - 状態 *n* をスタックにプッシュする.
  - r*m* に従って文法規則を適用する。（これは還元操作）
    - *m* 番の規則の右側にある各記号についてスタックから状態を削除する（例えば E * B なら3個ポップする）。
    - その時点のスタックのトップを状態とし、それと *m* 番の文法規則の左側からGOTO表を参照し、そこにある状態をスタックにプッシュして新たな状態とする.
  - 受理（acc）の場合、文字列の構文解析が完了。
  - アクションが指定されていない場合、文法エラーとなる.
- 上のステップを「受理」となるか「文法エラー」となるまで繰り返す。

1+1$ の構文解析例

状態	入力	スタック	アクション	意味	
0	1+1$	[0]	s2	1の部分を還元の候補にする	(1)+1
2	+1$	[2,0]	r5	1を<B>に還元する	<B>+1
4	+1$	[4,0]	r3	<B>を<E>に還元する	<E>+1
3	+1$	[3,0]	s6	+の部分を還元の候補にする	<E>(＋)1
6	1$	[6,3,0]	s2	1の部分を還元の候補にする	<E>(+)(1)
2	$	[2,6,3,0]	r5	1を<B>に還元する	<E>(+)<B>
8	$	[8,6,3,0]	r2	<E>+<B>を<E>に還元する	<E>
3	$	[3,0]	acc	受理	

導出木

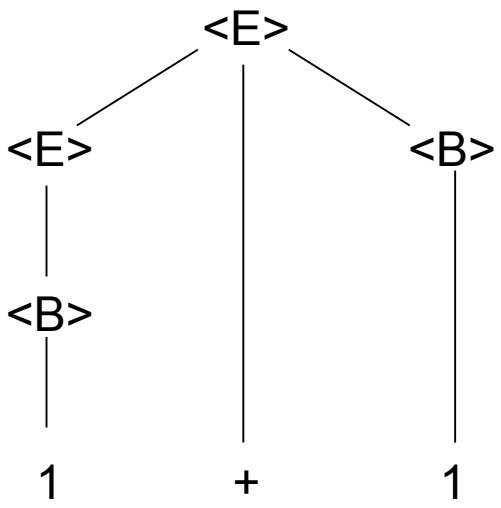
(1) <E> →<E>*<B>

(2) <E> →<E>+<B>

(3) <E> →<B>

(4) <B> → 0

(5) <B> → 1



問題

- 「０＋１＊１」の構文解析を上述の文法の下で行いなさい．導出木も示すこと.