自然言語処理論 I

4.文法2(構文解析)　その2

一般化LR法(GLR法)

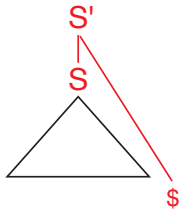
- LR法
  - 決定的な構文解析アルゴリズム
  - プログラミング言語などの解析
  - LR(k)文法
    - ✦ k語の先読みにより，決定的になる文法
    - ✦ 全てのLR(k)文法はLR(1)文法に変換できる
    - ✦ CFGはLR(k)文法ではない
- 一般化LR法(generalized LR Method)
  - LR法を非決定的アルゴリズムに拡張

LR法による構文解析の流れ

- LR表の作成
  - FIRST関数，FOLLOW関数の計算
  - gotoグラフの計算
  - gotoグラフをLR表に変換
    - ✦ 構文解析前に事前に行う
- LRパーザによる構文解析

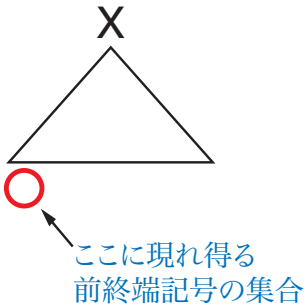
準備

- 以後、入力は前終端記号列とする
  - 辞書規則は用いない
- 文法の修正
  - 入力文の一番最後に特別な前終端記号$を加える
  - 文法規則に S'→S $ という規則を追加する



FIRST関数

- FIRST(X)
  - Xは非終端記号または前終端記号
  - Xから導出される前終端記号列の先頭になり得る前終端記号の集合



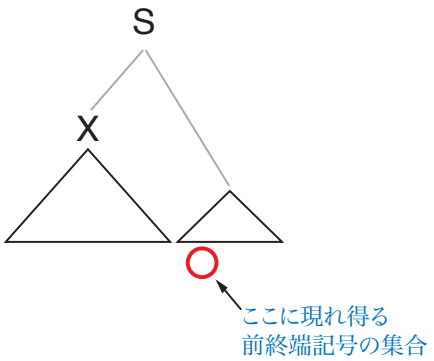
  - $FIRST(X) = \{a | X \stackrel{*}{\Rightarrow} a\beta, a \in V_P, \beta \in V_P{}^*\}$
    - ✦ V_Pは前終端記号の集合

FIRST関数

- 計算アルゴリズム
  - Xが前終端記号なら、FIRST(X) = { X }
  - Xが非終端記号なら、Xを左辺とする全ての規則 X→Y β について、FIRST(Y)をFIRST(X)に加える
    - ✦ 再帰的定義であることに注意
    - ✦ 計算可能なXから順にFIRSTを求める
    - ✦ Y ≠ X とする
      - ○ X → X Y Z のような左再帰規則は無視してよい

FOLLOW関数

- FOLLOW(X)
  - Xは非終端記号
  - Xから導出される記号列の直後に現れ得る前終端記号の集合



  - $FOLLOW(X) = \{a | S \stackrel{*}{\Rightarrow} \beta X a \gamma, a \in V_P, \beta\gamma \in (V_N + V_P)^*\}$

FOLLOW関数

- 計算アルゴリズム
  - FOLLOW(S) = { $ } とする
  - Xが非終端記号のとき
    - ✦ Z → α X Y β という規則があれば、FIRST(Y)をFOLLOW(X)に加える
    - ✦ Y → α X という規則があるとき、FOLLOW(Y)をFOLLOW(X)に加える
    - ✦ 再帰的に定義されていることに注意
    - ✦ 計算可能な非終端記号から順にFOLLOWを決定する
    - ✦ Y ≠ X とする
      - ○ X → Y X のような右再帰規則は無視してよい

FIRST関数,FOLLOW関数の例

- 例

S→NP VP (1)
NP→pron (2)
NP→det n (3)
VP→v (4)
VP→v NP (5)
VP→VP PP (6)
PP→prep NP (7)

	FIRST	FOLLOW
S	det, pron	$
NP	det, pron	$,v,prep
VP	v	$,prep
PP	prep	$,prep

FIRST関数,FOLLOW関数の例

- FIRSTの計算
  - FIRST(pron) = { pron }, FIRST(det) = { det }, FIRST(n) = { n }, FIRST(v) = { v }, FIRST(prep) = { prep }
  - FIRST(NP) = FIRST(pron)∪FIRST(det) = { pron, det }
    - ✦ 規則(2),(3)より
  - FIRST(VP) = FIRST(v)∪FIRST(v) = { v }
    - ✦ 規則(4),(5)より
    - ✦ 規則(6)のような左再帰規則は無視してよい
  - FIRST(PP) = FIRST(prep) = { prep }
    - ✦ 規則(7)より
  - FIRST(S) = FIRST(NP) = { pron, det }
    - ✦ 規則(1)より

FIRST関数,FOLLOW関数の例

- FOLLOWの計算
  - FOLLOW(S) = { $ }
  - FOLLOW(VP) = FOLLOW(S)∪FIRST(PP) = { $, prep }
    - ✦ 規則(1),(6)より
  - FOLLOW(PP) = FOLLOW(VP) = { $, prep }
    - ✦ 規則(6)より
  - FOLLOW(NP) = FOLLOW(VP)∪FOLLOW(PP)∪FIRST(VP) = { $, v, prep }
    - ✦ 規則(5),(7),(1)より

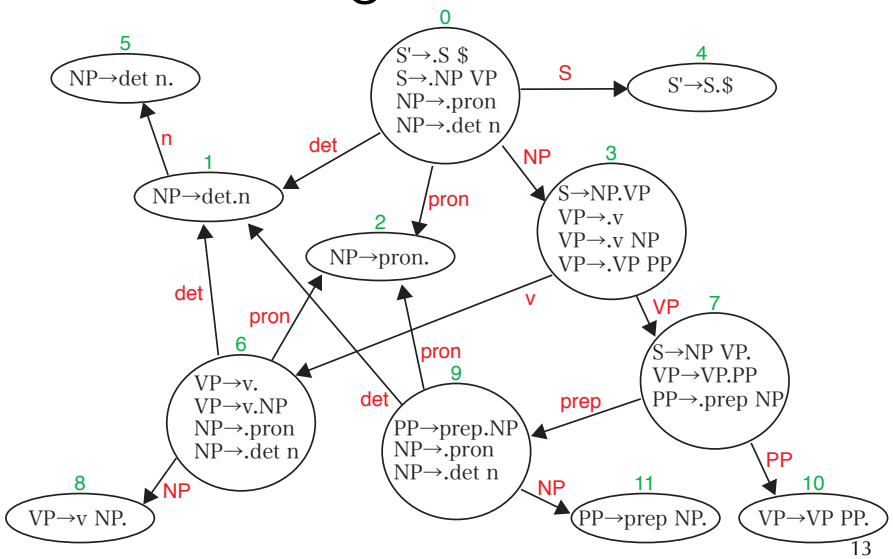
gotoグラフの作成

- 構文解析過程を表わす有限オートマトン
  - 状態は項の集合として定義される
- 作成アルゴリズム

1. 状態0に，S'→.S $ という項を追加
2. 各状態 i について
   - ✦ 項X→.Y βがあれば，Yを左辺とする全ての規則について，項Y→.αを追加する (クロージャー展開)
   - ✦ 項X→.Y βがあれば、新しい状態jを作り、Yをラベルとした弧を張り、状態jに項X→Y.βを加える
3. 同じ状態が生成されればマージする

※ 2,3を繰り返す

作成されたgotoグラフ



LR表

- 構文解析時に参照する状態遷移表
  - 行: 状態番号
  - 列: 前終端記号 (action部)
    非終端記号 (goto部)
  - 要素: shift動作/reduce動作 (action部)
    遷移先状態 (goto部)

	(action part)						(goto part)			
	det	n	pron	v	prep	$	S	NP	VP	PP
0										
1										
2										
⋮										

LR表の作成

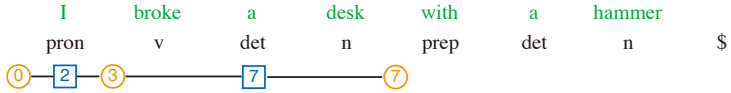
- gotoグラフより作成
- (i) —P→ (j) という弧があるとき
  - Pが前終端記号なら、(i,P)=sh j (shift動作)
  - Pが非終端記号なら、(i,P)=j
- 状態iに X→α. という項があるとき
  - (i,P)=re j (reduce動作)
    - ✦ jはX→αの規則番号
    - ✦ PはFOLLOW(X)の要素
- 状態iに S'→S . $ があるとき
  - (i,$)=acc(受理)

作成されたLR表

	det	n	pron	v	prep	$	S	NP	VP	PP
0	sh1		sh2				4	3		
1		sh5								
2				re2	re2	re2				
3				sh6					7	
4						acc				
5				re3	re3	re3				
6	sh1		sh2		re4	re4		8		
7					sh9	re1				10
8					re5	re5				
9	sh1		sh2					11		
10					re6	re6				
11					re7	re7				

LRパーザ

- 構文解析を行うシステム
  - LR表
    - LRパーザの動作を決める
  - スタック
    - 解析途中に生成される文の部分的な構造を保存
    - LR表の状態(◯で表わす)と履歴(□で表わす)を交互に積む
    - 状態の位置も重要
      - 解析がどこまで進んだか? / 部分木の範囲



  - 履歴
    - 完成した部分木を保存

LRパーザ

- アルゴリズム
  1. スタックの先頭の状態と先読み記号でLR表を参照し、次の動作を決める
     - 先読み記号: スタックの先頭の次にある前終端記号
  2. 動作を実行
     - shift動作 または reduce動作
  3. 1,2を繰り返す
     - accに到達すれば解析に成功
     - それ以外は失敗

- LRパーザの動作
  - shift動作 (sh i)
    - ID[前終端記号 単語]を履歴に追加
    - ID番号と状態番号iをスタックに積む
      - スタックの先頭の位置をひとつ右へ
  - reduce動作 (re i)
    - 規則iをX→αとする
    - スタックから，αの要素数だけ状態番号とID番号を取り出す(取り出した後のスタックの先頭の状態をjとする)
    - ID[X [ID番号の列]]を履歴に追加
    - ID番号をスタックに積む
    - LR表(j,X)を見て、次の状態番号をスタックに積む
      - スタックの先頭の位置は変わらない

LR法

- 解析例
  - 文法

S→NP VP	(1)		VP→v NP	(5)
NP→pron	(2)		VP→VP PP	(6)
NP→det n	(3)		PP→prep NP	(7)
VP→v	(4)			

  - LR表
    - 先ほどの例と一緒
  - 解析文
    - I broke a desk with a hammer
  - 解析過程→黒板、別紙資料(後日配布)
  - 履歴から構文木も復元できる

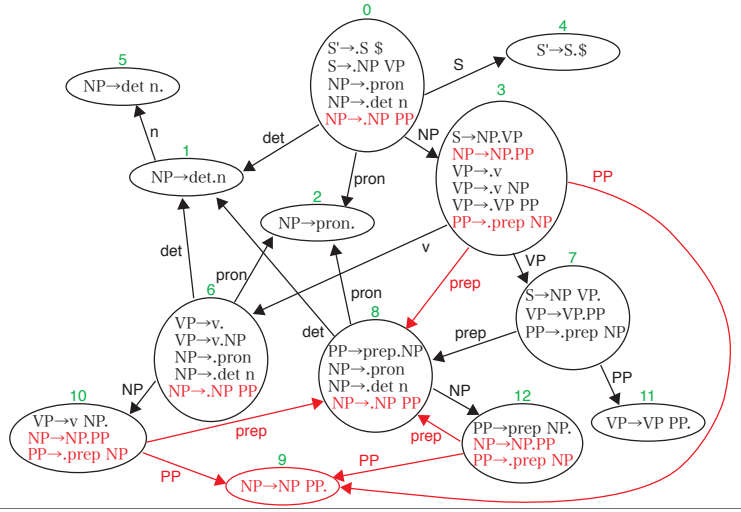
LR表の作成(一般化LR法の場合)

- 一般の文脈自由文法の場合，LR表は非決定的になる
- 非決定的なLR表を使えるようにLR法を拡張する
- 文法の例

S→NP VP	(1)	VP→v NP	(5)
NP→pron	(2)	VP→VP PP	(6)
NP→det n	(3)	PP→prep NP	(7)
VP→v	(4)	NP→NP PP	(8)

gotoグラフの作成

- アルゴリズムは全く一緒



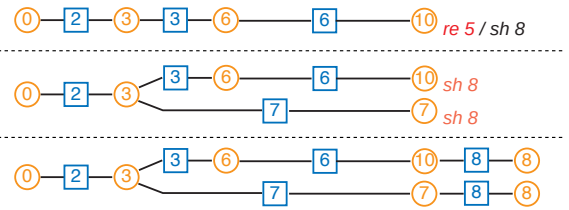
LR表の作成

- 競合(コンフリクト)が起こる可能性あり
  - shift/reduceコンフリクト
  - reduce/reduceコンフリクト

	det	n	pron	v	prep	$	S	NP	VP	PP
0	sh1		sh2				4	3		
1		sh5								
2				re2	re2	re2				
3				sh6	sh8				7	9
4						acc				
5				re3	re3	re3				
6	sh1		sh2		re4	re4		10		
7					sh8	re1				11
8	sh1		sh2					12		
9				re8	re8	re8				
10					re5/sh8	re5				9
11					re6	re6				
12				re7	re7/sh8	re7				9

一般化LR法による解析

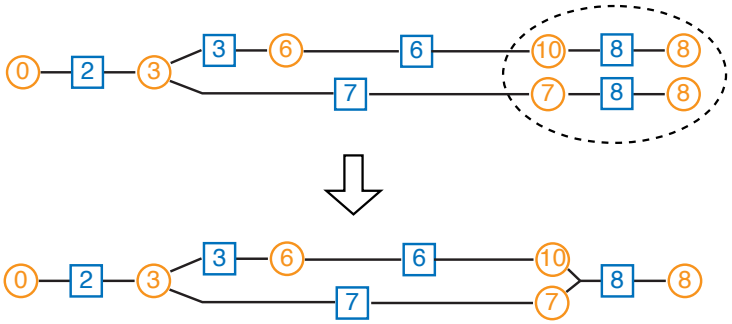
- LR法からの主な拡張点
  - 競合が起こったら、スタックを分岐させる
    - ✦ グラフ構造化スタック



  - 主な探索戦略
    - ✦ shift/reduceコンフリクト→reduce動作を優先
    - ✦ reduce/reduceコンフリクト→pop後の状態の位置が一番左にある動作を優先

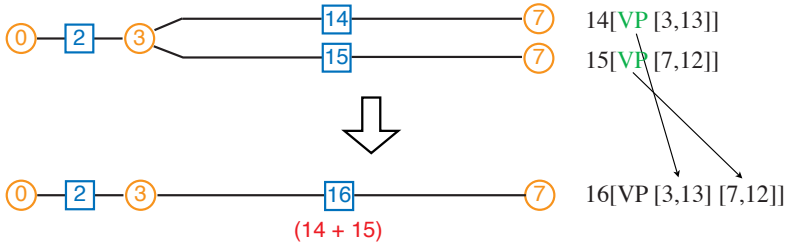
一般化LR法による解析

- スタック操作の工夫による効率化
- sharing
  - スタック上の同じ状態、IDは共有する



一般化LR法による解析

- packing
  - 同じ単語列をカバーし、同じ非終端記号で始まるIDはまとめる



※ 分岐・統合したスタックはグラフ構造化スタックと呼ばれる

一般化LR法の特徴

- ボトムアップアルゴリズム
- 計算量はnの指数オーダー
  - スタックの分岐・統合のため
- 実際には，他の構文解析アルゴリズムに比べて高速
  - LR表をあらかじめ作成することにより，解析途中の計算を省略できるため