言語理論(Language Theory)

- 言語の構文解析の基礎となる理論
- 記号列の集合としての言語の定義
- 計算モデルとしてのオートマトン
- ソフトウェア開発ツールの基礎
  - 語句解析プログラムの生成ツール(LEX)
  - 構文解析プログラムの生成ツール(YACC)

参考書

- Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, Addison-Wesley. 1979.
- Aho, A.V., Ullman, J.D.: Principles of Compiler Design, Addison-Wesley. 1977.
- Hunter, R.: Compilers- Their Design and Construction Using Pascal, Wiley. 1985.

言語の定義

- **集合**(set)**の概念**
  - **元**(element),**和集合**(union),**積集合**(intersection)
  - **部分集合**(subset),**空集合**(empty set)
  - **述語による集合の定義**：{ x | P(x) }
- **列**(sequence)**の集合の概念**
  - **空列**(empty sequence) **ε**
  - **連接**(concatenation)
    - $AB=\{ ab \mid a \in A, b \in B\}$
  - **長さn の列と列の集合の表記法**：a^n　A^n
  - **閉包**(closure): (**空列を含む**)**任意の有限列の集合**
    - $A^* = \{ \varepsilon \} \cup A \cup A^2 \cup A^3 \cdots$
    - $A^+ = A \cup A^2 \cup A^3 \cdots$

演習問題(言語の例)

- [LT01] 0個以上の文字 0 のうしろに同数の文字 1 が続いているような列の全体からなる集合を記述する方法を示せ。

- [LT02] 0個以上の文字 0 のうしろに0個以上の文字 1 が続いているような列の全体からなる集合を記述する方法を示せ。

文法(grammar)

- **語彙**(alphabet, vocabulary)**の列の集合の定義法**
  - **述語による集合の定義では表現が困難**
- **文法**: (V_T, V_N, P, S)
  - **終端記号**(terminal symbol)**の集合**: V_T
  - **非終端記号**(nonterminal symbol)**の集合**: V_N
  - **生成規則**(production rule)**の集合**: P
  - **文記号**(sentence symbol): $S \in V_N$
  - $V_T \cap V_N = \{\ \}$
  - **記号**: $V = V_T \cup V_N$
  - $P = \{\alpha \rightarrow \beta \mid \alpha \in V^+,\ \beta \in V^*\}$

導出(derivation)と言語(language)

- **導出**:**記号列の中に現れる記号列で生成規則の左辺に現れるものを、その規則の右辺の記号列で置き換える操作を繰り返すこと**
- **文形式**(sentential form): **文記号から導出される記号列**
- **文**(sentence): **終端記号のみで構成される文形式**
- **言語**(language): **文の集合**
  - **文法**Gによる言語をL(G)と書く

文法の例と導出

言語 $\{\mathbf{0}^n\mathbf{1}^n \mid n \geq 0\}$ を定義する文法 $G_0 = (V_T, V_N, P, S)$:

$V_T = \{\mathbf{0}, \mathbf{1}\}$

$V_N = \{S\}$

$P = \{S \to \mathbf{0}S\mathbf{1},\ S \to \varepsilon\}$

導出の例:

$$S \Rightarrow \mathbf{0}S\mathbf{1} \Rightarrow \mathbf{00}S\mathbf{11} \Rightarrow \mathbf{000}S\mathbf{111} \Rightarrow \mathbf{000111}$$

演習問題(文法の定義)

- [LT03] 0個以上の文字 a のうしろに0個以上の文字 b が続いているような列の全体からなる言語を記述する文法

  $G_1=(\{a, b\}, \{S, A, B\}, P, S)$

  の生成規則Pを示せ。
- [LT04] 文法$G_3=(\{a\},\{S,N,Q,R\},P,S)$,

  P: $S \rightarrow QNQ$, $QN \rightarrow QR$, $RN \rightarrow NNR$,

  $RQ \rightarrow NNQ$, $N \rightarrow a$, $Q \rightarrow \varepsilon$

  によって定義される言語の文はどのようなものか。

等価な文法(equivalent grammar)

- **同一の言語を定義する文法は等価**
  - **演習問題**[LT03]**の文法**

    $G_1=(\{a, b\}, \{S, A, B\}, P, S)$

    と等価な文法

    $G_2=(\{a, b\}, \{S, T\}, P, S)$

    の生成規則P：

    $\{S \rightarrow a\,S,\ S \rightarrow \varepsilon,\ S \rightarrow bT,\ T \rightarrow bT,\ T \rightarrow \varepsilon\}$

Chomskyによる文法のクラス階層

- **生成規則 α → β の制限による分類**
  - 0**型文法**
  - 1**型文法(文脈依存**, context sensitive)：
    |α|≦|β|
  - 2**型文法(文脈自由**, context free):
    A→ **β**
  - 3**型文法(正規**, **正則**, regular):
    A→a, A→aB　(**右線形**, right linear)
    A→a, A→Ba　(**左線形**, left linear)
- **言語についても同様**

演習問題(文法･言語のクラス)

- [LT05] **言語の階層間には包含関係が成り立ち、実際、真の包含関係であることを例によって示せ。**
- [LT06] **既出の文法**G_0, G_1 , G_2 , G_3**によって定義される言語はそれぞれどのクラスに属するか。**
  - **上の3型文法の定義では空の列 ε を生成することはできないが、そのような生成規則を含めることも一般的。**

正規表現(regular expression)

- **語彙Σ上の正規表現**:
  - x∈ Σ は正規表現
  - 空列は正規表現
  - P, Qが正規表現のとき以下の構成は正規表現
    - P Q　　[連接]
    - P | Q
    - P*
- **正規集合**:正規表現によって生成される集合

正規表現と正規文法

- 正規表現：連接、｜、* を演算子とする「式」
  - 演算子 | は可換で結合性をもつ
  - 連接演算は結合性をもつ
  - 結合の強さは * , 連接, | の順
  - 例： a* b | c a*　　(a a b | a b)*
- 任意の正規集合を生成する正規文法が存在する
- 記号列が正規集合に属するかどうかを決定するアルゴリズムが存在する
- 2つの正規表現が同一の正規集合を生成するかどうかを決定するアルゴリズムが存在する

有限オートマトン(finite state automaton)

(K, **Σ**, **δ**, S, F)

- **状態の有限集合**: K
- **入力記号の集合**: **Σ**
- **状態遷移の集合**: **δ**
- **開始状態**: S∈K
- **最終状態の集合**: F⊆K

- **状態遷移の性質により**
  - **決定性** (deterministic): DFA
  - **非決定性** (nondeterministic): NFA

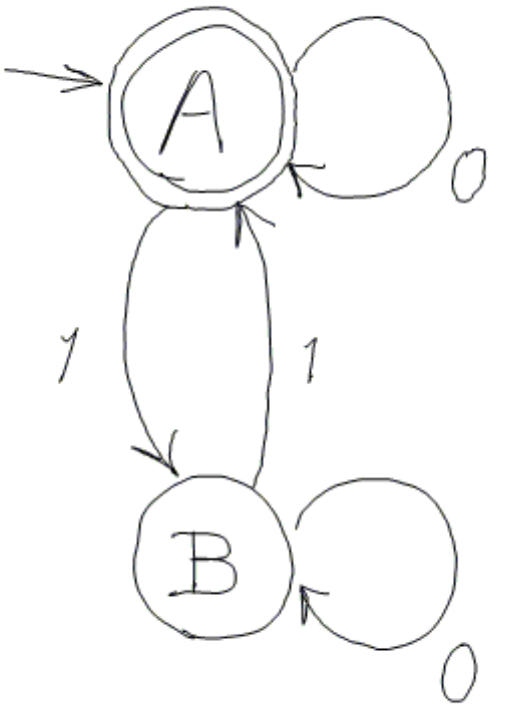
- **有限オートマトンは開始状態から入力記号によって状態遷移により最終状態 (の一つ) に到達することにより入力記号列を受理する**

有限オートマトンの例(DFA) M

- K = {A, B}
- **Σ** = {0, 1}
- **δ**:

  δ(A,0)=A, **δ**(A,1)=B

  δ(B,0)=B, **δ**(B,1)=A
- S= A
- F= {A}

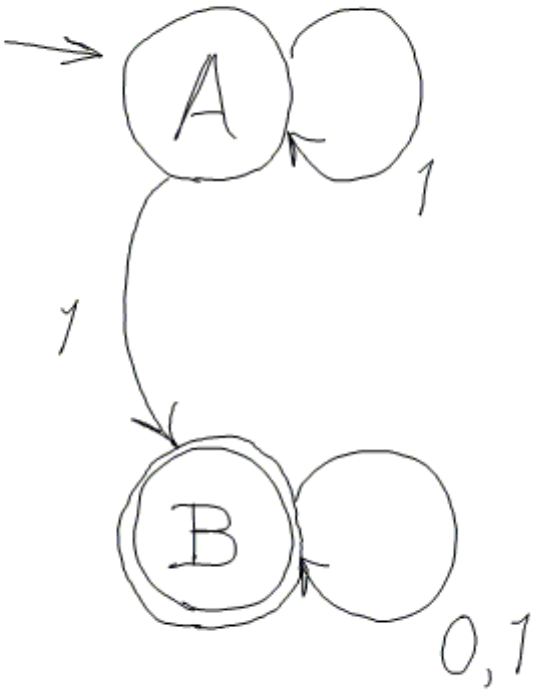


有限オートマトンの例(NFA) M_1

- K = {A, B}
- **Σ** = {0, 1}
- **δ**:

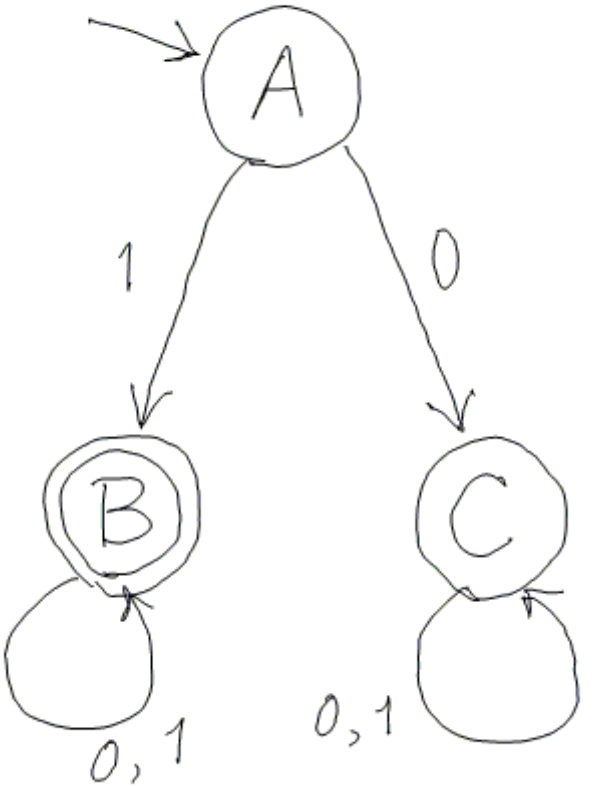
  δ(A,1)=A, **δ**(A,1)=B

  δ(B,0)=B, **δ**(B,1)=B
- S= A
- F= {B}



有限オートマトンの例(DFA) M_2

- K = {A, B, C}
- **Σ** = {0, 1}
- **δ**:
  δ(A,0)=C, **δ**(A,1)=B
  δ(B,0)=B, **δ**(B,1)=B
  δ(C,0)=C, **δ**(C,1)=C
- S= A
- F= {B}

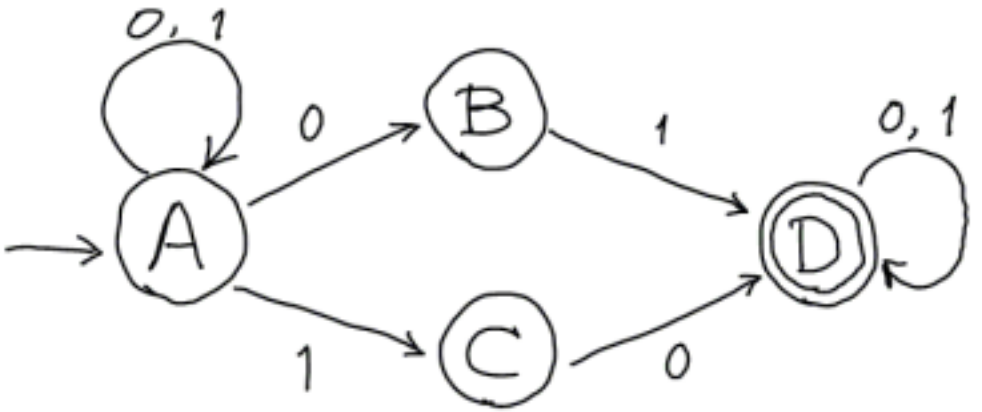


決定性/非決定性オートマトン

- 非決定性オートマトン(NFA)
  - 入力なしの遷移を許容
  - 同一の入力記号による遷移の可能性が複数
- 非決定性オートマトン(NFA)の受理する言語を受理する決定性オートマトン(DFA)が存在する
  - NFAをDFAに変換するアルゴリズムが存在
  - NFAで同じ入力記号によって遷移する先の状態の集合をDFAの状態とする

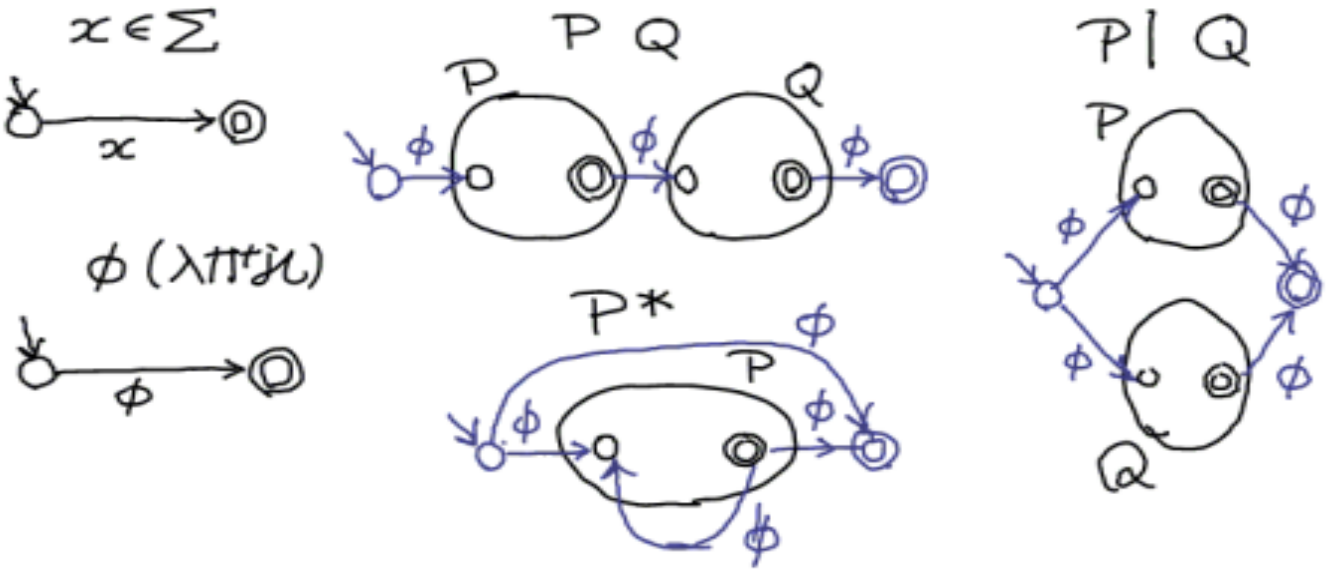
演習問題(NFAからDFAへの変換)

- [LT07] **下図のNFAの受理する言語を受理するDFAを示せ。これらのオートマトンで受理される言語の文はどのようなものか。**



正規表現とオートマトン

- 正規表現で定義される言語を受理する有限オートマトンが存在する
  - 正規表現の構成法に応じて、NFAを構成



語句解析プログラム生成システム

- LEX: Unix**上の**Lexical analyser generator
  - **名前**(identifier): Letter (Letter|Digit)*
  - **予約語**: if for ・・・
  - **定数表記**:

    (+|-) Digit Digit* . Digit Digit* (e (+|-|) Digit Digit* |)
  - **記号**: { } += ・・・
- **正規表現で定義された語句**(token)**を受理するNFAを構成し、**DFA**に変換して状態遷移表を生成する**

演習問題(語句解析とオートマトン)

- [LT08] **下記の正規表現で定義される数の表記を受理するDFAをそれぞれ示せ。ただし、dは個々の数字を表わすものであるがここでは単一の記号として扱ってよい。**

  ① d d*

  ② d d* . d d* (e (+|-|) d d* |)

語句解析オートマトンの例

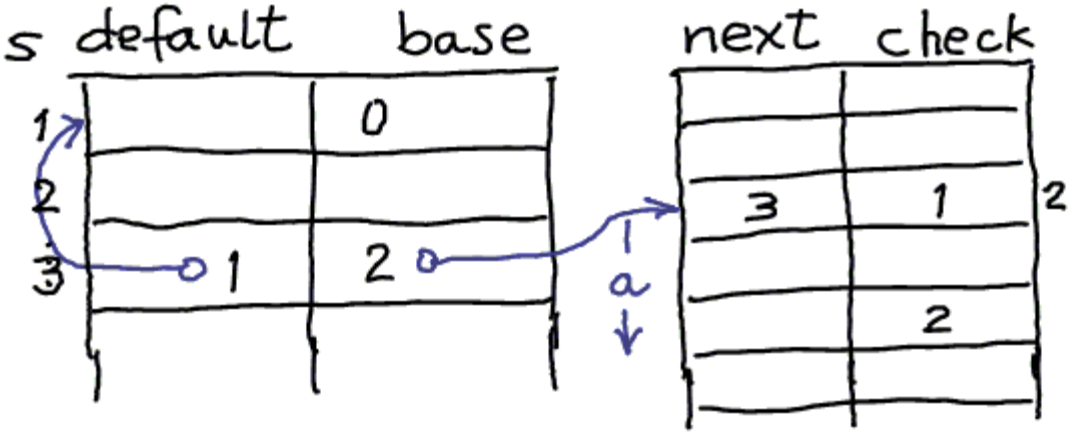
- **言語**Pascal**の語句の種類**62
  - begin, end, …
  - (0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*
  - (a|b|…|z)(a|b|…|z|0|1|2|3|4|5|6|7|8|9)*
  - ・・・
- **オートマトン状態数**: 163
- **入力文字の種類**: 96
- 2**次元の状態遷移表**: 163×96=15,648 bytes
  - **状態**s, **入力**a**による次の状態** t=nextstate[s,a]

状態遷移表の縮小化の例

– 語句解析オートマトンでは同一の入力による遷移先が同じであるような遷移が多い

for(k=s; check[base[k]+a]≠k; k=default[k]);

t= next[base[k]+a];



Pascalの例:
next-check:
463 (うち31が空)
default-base
163
全体で1415 bytes

正規文法と文脈自由文法

- 「任意の深さに入れ子になった括弧構造」は正規表現では定義できない

  S→ (S)

  S→ S S

  S→ ε

- プログラム言語の構文の多くは文脈自由文法で定義される

文脈自由文法(CFG)の正規形

任意のCFGと等価な以下のような正規形の文法が存在する

- Chomsky Normal Form: **生成規則が**

  $A \rightarrow B\ C$

  $A \rightarrow a$

  の形の文法

- Greibach Normal Form: **生成規則が**

  $A \rightarrow b\ \alpha$ $(\alpha \in V_N^*)$

  の形の文法

CFGの自己埋込み性

- **自己埋込み性**(self-embedding):
  - **文法**G**による導出過程において、**
    $A \Rightarrow \cdots \Rightarrow \alpha A \beta \qquad (\alpha, \beta \in V^{+})$
    が存在するような文法
- **自己埋込み性をもたない**CFG**は正規文法と等価**
  - **正規文法は自己埋込み性をもたない**

Pumping Lemma

- **任意のCFGには、以下のような定数kが存在する：**
  - **その言語の長さ k 以上の任意の文 z を**

    $z=uvwxy,\ |vwx| \leqq k,\ |vx| \neq 0$

    の形に書くと、このときのu,v,w,x,yによる列

    $uv^iwx^iy\ ,\ (i \geqq 0)$

    もその言語の文である。

Pumping Lemmaによる非CFG性の証明例

- {αα| α∈{0,1}* }はCFGの言語ではない
  - k個の0の後にk個の1が続いたものをαとする
    - z=αα=00・・・011・・・100・・・11・・・1
  - zをLemmaのようなuvwxyに分解できるとする
  - |vwx|≦kであるから下記のいずれかが成立
    - v,xはともにzの前半、あるいは後半
    - vxはzの前半の1と後半の0からなる
  - いずれの場合にもzからuwyを作ると、それが言語の文にはならないことがいえる

CFGとプッシュダウンオートマトン

- **プッシュダウンオートマトン**(push-down automaton)：**スタックを有する有限オートマトン**
- **一動作は下記のいずれか**
  - **入力記号を読み、スタック最上段の要素を記号列(空でありうる)で置き換え、状態を変える**
  - **入力記号は読まないで、上と同様の動作を行なう**
- **文脈自由言語はプッシュダウンオートマトンによって認識できる**

プッシュダウンオートマトン

$(K, \Sigma, \Gamma, \delta, q_0, Z_0)$

- **状態の有限集合**: K
- **入力記号の集合**: **Σ**
- **プッシュダウン記号**: **Γ**
- **状態遷移の集合**: **δ**
- **開始状態**: $q_0 \in K$
- **スタック初期記号**: $Z_0 \in \Gamma$

- **入力記号が終了したときにスタックが空になれば受理**

(例) 括弧対応認識

$K=\{A\}, \Sigma=\{[,]\}$

$\Gamma=\{0,1\}, q_0=A$

$Z_0=1$

$\delta(A,1,[)=(A,10)$

$\delta(A,0,[)=(A,00)$

$\delta(A,0,])=(A,\varepsilon)$

$\delta(A,1,\varepsilon)=(A,\varepsilon)$

演習問題(CFGとプッシュダウンオートマトン)

- [LT09] 言語{ $\alpha\,\alpha^r \mid \alpha \in \{0,1\}^*$ }を定めるCFGを定義せよ。α^rはαを反転させた列を表わす。
- [LT10] 非決定性プッシュダウンオートマトン

$$M=(\{A,B\},\{0,1\},\{X,0,1\},\delta,A,X)$$

$\delta(A,X,0)=\{(A,X0)\}$, $\delta(A,X,1)=\{(A,X1)\}$,
$\delta(A,0,0)=\{(A,00),(B,\varepsilon)\}$, $\delta(A,1,1)=\{(A,11),(B,\varepsilon)\}$,
$\delta(A,0,1)=\{(A,01)\}$, $\delta(A,1,0)=\{(A,10)\}$,
$\delta(B,0,0)=\{(B,\varepsilon)\}$, $\delta(B,1,1)=\{(B,\varepsilon)\}$,
$\delta(B,X,\varepsilon)=\{(B,\varepsilon)\}$, $\delta(A,X,\varepsilon)=\{(A,\varepsilon)\}$

が[LT09]の言語を受理することを確かめよ。

文脈自由言語の決定性/非決定性

- 決定性オートマトンでは認識できないCFGを認識する非決定性オートマトンが存在する
  - 正規文法に対する有限オートマトンの決定性/非決定性の関係とは異なる
- コンパイラにおける構文解析では決定性が望まれる
  - 決定性の(逆戻りなしの)解析ができるCFG言語を決定性言語(deterministic language)という

演習問題(文脈自由言語の決定性/非決定性)

- [LT11] **以下の言語がそれぞれ決定性であるかどうかを答えよ。**
  - $\{ \alpha c \alpha^r \mid \alpha \in \{0,1\}^* \}, c \neq 0,1$
  - $\{ c \alpha \alpha^r \mid \alpha \in \{0,1\}^* \}, c \neq 0,1$
  - $\{ 0^n 1^n \mid n \geqq 0 \}$
  - $\{ 0^n 1^{2n} \mid n \geqq 0 \}$
  - $\{ 0^n 1^n \mid n \geqq 0 \} \cup \{ 0^n 1^{2n} \mid n \geqq 0 \}$

文脈自由言語の構文解析(parsing)

- **下降型解析**(top-down parsing): **解析対象の文を生成するような文形式の導出過程を求める。**
  - **再帰下降型解析法**(recursive descent)
  - **表駆動型解析法**(table-driven parsing)
- **上昇型解析**(bottom-up parsing): **解析対象の文の部分を構成するような生成木を作り、右辺に合致する形式を左辺で置き換えることを繰返し、文記号まで還元する。**

再帰下降型(recursive descent)構文解析

A→a B の導出に従って文を解析する手続きA

```
void A(){
    if(x==a){
        x= nextsymbol();
        B();
    }
}
```

B→･･･ についても同様

- 非終端記号に相当する再帰的手続きの呼び出しによって導出を実現する

s-文法(s-grammar)

- 生成規則の制限
  - 右辺はどれも終端記号で始まる
    - Greibach標準形に類似
  - 左辺に複数回現れる非終端記号に対する右辺の先頭の終端記号はすべて異なる
    - 決定性の下降型解析を可能にする条件
- 文法がs-文法であるかどうかの決定は容易

s-文法の文の下降型解析の例

s-**文法**: S→pX, X→aXb, X→x

解析の例

paaxbb	S
p.aaxbb	⇒p.X
pa.axbb	⇒pa.Xb
paa.xbb	⇒paa.Xbb
paaxbb.	⇒paaxbb.

LL(1) **文法**

- s-**文法の一般化**
  - **決定性解析が可能**
- L: Left-to-right, L: Leftmost derivation (1): **生成規則の選択を1つの記号で判断**
- **生成規則の右辺は必ずしも終端記号で始まらなくてもよいが、1個の先読み記号で規則の選択が可能な文法**

LL(1)文法と左端記号集合(starter symbols)

- S(α): **記号列αから導出される記号列の左端に現れる終端記号の集合**
  - **例**: P→Ac, P→Bd, A→a, A→aA, B→b, B→bB
    - S(P)={a,b}, S(A)={a}, S(B)={b}
  - **例**: P→AB, P→BG, A→aA, A→ε , B→c, B→bB
    - S(AB)={a,b,c}, S(BG)={b,c}
- LL(1)**の必要条件:同一の非終端記号を左辺にもつ生成規則の右辺に対する左端記号集合が互いに排他的であること**

演習問題(s-文法, LL(1)文法)

- [LT12] s-**文法の生成規則と**Greibach**標準形の異なるところはどこか。**
- [LT13] **文法**

  T→AB, A→PQ, A→BC, P→pP, P→ **ε** ,
  Q→qQ, Q→ **ε** , B→bB, B→e, C→cC, C→f

  から、S(PQ), S(BC)**を求め、これが**LL(1)**文法の必要条件を満たしていることを確認せよ。**

  また、PQ**が空列を生成しうるので、決定性の解析を行なうことができないことを示せ。**

LL(1)文法と先導記号集合(director symbols)

- DS(P, α): P→αに対して
  - αがεを生成しないとき DS(P, α)=S(α)
  - αがεを生成するとき
    DS(P, α)=S(α)∪{Pの直後に現れうる記号}
- LL(1)文法の定義：

  複数の生成規則の左辺に現れる非終端記号に対して、対応する右辺の先導記号集合が互いに排他的であるような文法

LL(1)言語

- 文法がLL(1)であるかどうかを判定するアルゴリズムが存在する
  - すべてのCFGはLL(1)をもつのか→NO
  - 言語がLL(1)文法で定義できるかどうかを決定するアルゴリズムは存在しない
    - 任意の文法をLL(1)文法に変換するような規則は存在しない
    - 言語を定義する文法から、LL(1)文法の性質を満たしていない部分を取り除くことは可能

LL(1)文法への変換

- **左再帰性の除去**

  A→A **α**, A→a **から左再帰性を除去して**

  A→a A', A'→ **α** A', A'→ **ε に変換する**

  – **間接的に左再帰になっているものには、まず、直接の再帰性に変換する**

- **共通記号列のくくりだし**(factorization)

  – **同一の左端記号をもつ複数の規則の右辺部分をくくりだす**

演習問題(LL(1)言語)

- [LT14] くくりだしによって、文法

  S→a S b, S→a S c, S→c

  をLL(1)文法に変換せよ。
- [LT15] 次の文法をLL(1)に変換せよ。

  E→E＋T，E→T，T→T×F，T→F，

  F→(E)，F→x，F→y

上昇型構文解析法

- **解析の手順**
  - shift(**読込み**): **入力記号をスタックに入れる**
  - reduce(**還元**): **スタック上部の終端記号・非終端記号の並びに一致する生成規則の右辺に対応する左辺で置き換える**
- **解析の決定性**
  - shift/reduce confllict: shift**と**reduce**のどちらも可能なときにはどちらを行なうのか？**
  - reduce/reduce conflict: **複数の**reduce**の可能性があるとき、どの規則によるのか？**

LR(k)文法

- shift/reduce, reduce/reduce conflict **をすでに読み込んだ入力記号と** k **個の先読み記号によって解消できるような文法**
- L: Left-to-right, R: Rightmost parse
- **実用的なものは**LR(0), LR(1)
  - LR(k)**の任意の任意の言語は、文の終端に特別な記号があるものとすると、それは**LR(1)**文法で生成することができる。さらに、これらは**LR(0)**である。**
  - LR(1)**ではなく**LR(2)**であるような文法は存在するが、**LR(2)**であって**LR(1)**でない言語は存在しない。**

LR(1)解析法(LR(1) parsing)

- LR(1)解析の方法は、決定性の解析法で解析できる任意の言語に適用可能
- すべてのLL(1)言語はLR(1)で解析可能
  - LL(1)は非終端記号と先読み1記号で規則を選択
  - LR(1)は生成規則の右辺全体と先読み1記号を利用して規則を選択しており、LL(1)よりも多くの情報を利用

LR解析表

規則： (1) S→ r L　(2) L→ L , X　(3) L→X　(4) X→x

状態	S	L	X	r	,	x	$
1	Halt			S2			
2		S5	S4			S3	
3					R4		R4
4					R3		R3
5					S6		R1
6			S7			S3	
7					R2		R2

文末記号

LR解析表による解析

入力	記号	状態	動作
r x,x$	$	$1	(1,r)S2
x,x$	$r	$1 2	(2,x)S3
,x$	$r x	$1 2 3	(3, ,)R4
X,x$	$r	$1 2	(2,X)S4
,x$	$r X	$1 2 4	(4, ,)R3
L,x$	$r	$1 2	(2,L)S5
,x$	$r L	$1 2 5	(5, ,)S6
・・・			

Sの後は状態番号

還元時は左辺を入力へ

Rの後は規則番号

記号スタック

状態スタック

LR解析表の構成法(状態の表現)

- **オートマトンの状態：構文規則上での解析の進行状況**
- **配置**(configuration): _ **で示す**
  - 「(1) S→ _ r L　(2) L→ L , X　(3) L→X　(4) X→x」**は、配置**(1,0)**を表示**
  - 「(1) S→ r L　(2) L→ L , X _　(3) L→X　(4) X→x」**は、配置**(2,3)**を表示**
- **核配置**(core)**の閉包**(closure)**で状態を表現**
  - **核配置**(1,1)**の閉包**{(1,1),(2,0),(3,0),(4,0)}**が一つの状態に対応**

LR解析表の構成法(shift動作)

- **状態を埋め込んだ文法**

  (1) S→ [1] r [2] L [5]

  (2) L→ [2] L [5] , [6] X [7]

  (3) L→ [2] X [4]

  (4) X→ [2,6] x [3]

- Shift **動作**
  - **状態1で入力 r のときの動作は** S2
  - **状態2で入力 L のときの動作は** S5

LR解析表の構成法(reduce動作)

- **状態を埋め込んだ文法**

(1) S→ [1] r [2] L [5]　　(2) L→ [2] L [5] , [6] X [7]

(3) L→ [2] X [4]　　(4) X→ [2,6] x [3]

- Reduce**動作が可能な状態は**3,4,5,7
- LR(0)**文法では先読み記号を使わずに還元**
  - **状態** 3,4,5,7 **では、どの入力にも**R4, R3, R1, R2
  - **状態**5, **入力**',' **のところは** shift/reduce conflict
    (**この文法は**LR(0)**でない**)

LR(0)⊂SLR(1)⊂LALR(1)⊂LR(1)

- LR(0)文法におけるconflictを先読み記号1個で解決する：?LR(1)
- SLR(1): shift/reduce conflictの状態でreduceするときの入力記号の集合を利用して解消
  - 状態数はLR(0)と同じ
  - 状態5のときに先読み記号$ならばreduce
- LALR(1): reduceのための記号集合を求める際に左側の文脈を考慮
  - UNIXのYACC parser generator は LALR(1)解析表を作成
- LR(1)は左側の文脈に依存して同一の配置に対しても異なる状態とするので、状態数が極めて多くなり、実用的でない