

コンパイラ

第2回 形式言語と形式文法
 — 有限オートマトンの復習 —

<http://www.info.kindai.ac.jp/compiler>

E館3階E-331 内線5459
 takasi-i@info.kindai.ac.jp

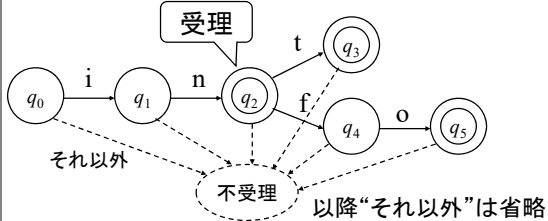


コンパイラの構造

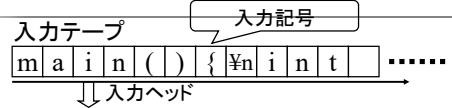
- 字句解析系 ⇒ 有限オートマトン
- 構文解析系
- 制約検査系
- 中間コード生成系
- 最適化系
- 目的コード生成系

有限(状態)オートマトン (finite (state) automaton)

- 特定の入力列を受理する状態遷移機械
- 例: 文字列 “in”, “int”, “info” を受理するオートマトン



有限オートマトンの概念図



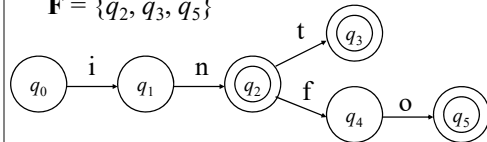
有限状態制御部
入力ヘッドの位置の入力記号を読み取り状態を遷移、入力ヘッドを1つ進める

有限オートマトンの5つ組

- $M = (Q, \Sigma, \delta, q_0, F)$
 - Q : 状態の有限集合
 - Σ : 入力記号の有限集合
 - δ : 状態遷移関数
 - $\delta(p, a) = q, (p, q \in Q, a \in \Sigma)$
 - q_0 : 初期状態
 - $q_0 \in Q$
 - F : 最終状態の有限集合
 - $F \subseteq Q$

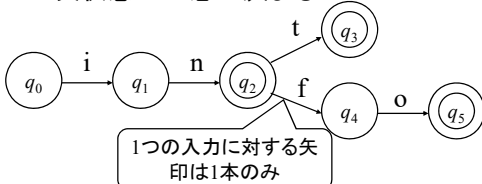
有限オートマトンの例

- $M = (Q, \Sigma, \delta, q_0, F)$
 - $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
 - $\Sigma = \{a, b, c, \dots, z\}$
 - $\delta(q_0, i) = q_1, \delta(q_1, n) = q_2, \delta(q_2, t) = q_3,$
 $\delta(q_2, f) = q_4, \delta(q_4, o) = q_5$
 - $F = \{q_2, q_3, q_5\}$



決定性有限オートマトン (deterministic finite automaton)

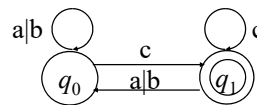
- 現在の状態と入力が決ま
⇒次状態が一意に決まる



⇔非決定性オートマトン

決定性オートマトンの 入力の受理

- 状態遷移で最終状態に到達すれば受理
- 例: c で終わる文字列を受理 ($\Sigma = \{a, b, c\}$)



| |
|---|
| <h3>決定性オートマトンの 入力の受理</h3> |
| <ul style="list-style-type: none"> ■ 状態遷移で最終状態に到達すれば受理 <p>例 : c で終わる文字列を受理 ($\Sigma=\{a,b,c\}$)</p> <p style="text-align: right;">入力 acc 受理</p> |

| |
|--|
| <h3>非決定性有限オートマトン (non-deterministic finite automaton)</h3> |
| <ul style="list-style-type: none"> ■ 現在の状態と入力が決まると次状態が複数あり得る <p style="text-align: right;">同じ入力に対する次状態が複数</p> |

| |
|---|
| <h3>非決定性オートマトンの 入力の受理</h3> |
| <ul style="list-style-type: none"> ■ 状態遷移のうちどれか一つが最終状態に到達すれば受理 <p>例 : c で終わる文字列を受理 ($\Sigma=\{a,b,c\}$)</p> |

| |
|--|
| <h3>非決定性オートマトンの 入力の受理</h3> |
| <ul style="list-style-type: none"> ■ 状態遷移のうちどれか一つが最終状態に到達すれば受理 <p>例 : c で終わる文字列を受理 ($\Sigma=\{a,b,c\}$)</p> <p style="text-align: right;">入力 acc</p> |

| |
|---|
| <h3>非決定性オートマトンの 入力の受理</h3> |
| <ul style="list-style-type: none"> ■ 状態遷移のうちどれか一つが最終状態に到達すれば受理 <p>例 : c で終わる文字列を受理 ($\Sigma=\{a,b,c\}$)</p> <p style="text-align: right;">入力 acc 受理</p> |

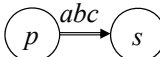
| |
|--|
| <h3>オートマトンの状態遷移</h3> |
| <ul style="list-style-type: none"> ■ $M = (Q, \Sigma, \delta, q_0, F)$ $-\delta(p, a) = q$ <p style="text-align: center;">以下 M, ○は省略</p> $p \xrightarrow{a} q$ |

拡張状態遷移

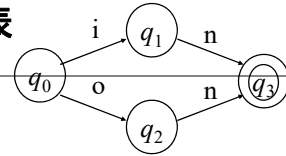
■ 入力列に対する状態遷移

$$\delta(p, a) = q, \delta(q, b) = r, \delta(r, c) = s$$

$$p \xrightarrow{a} q \xrightarrow{b} r \xrightarrow{c} s$$

$$\delta(p, abc) = s \quad p \xrightarrow{abc} s$$


状態遷移表



| 現状態 | 入力 | 次状態 |
|----------------|----|----------------|
| q ₀ | i | q ₁ |
| q ₀ | o | q ₂ |
| q ₁ | n | q ₃ |
| q ₂ | n | q ₃ |

本来なら $|Q| * |\Sigma|$ の表が必要

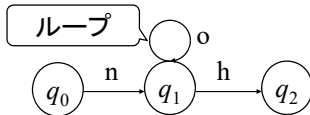
しかし大部分の行は次状態無し(不受理)

表に無い入力は不受理

状態遷移表

| 現状態 | 入力 | 次状態 |
|----------------|----|----------------|
| q ₀ | n | q ₁ |
| q ₁ | o | q ₁ |
| q ₁ | h | q ₂ |

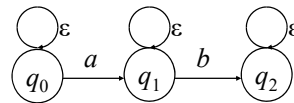
現状態=次状態
⇔ループ



空記号列, 空系列

■ ϵ : 空記号列, 空系列(長さ0の記号列)

$$- ab = \epsilon ab = a\epsilon b = ab\epsilon \quad \text{注意: } \epsilon \notin \Sigma$$



ループ以外の ϵ -動作がある

= 入力が無くても状態遷移

⇒非決定性オートマトンになる

繰り返し

■ $a \in \Sigma$ に対して

$$- a^0 = \epsilon, a^1 = a, a^2 = aa, a^3 = aaa, \dots$$

■ $\Sigma = \{0, 1\}$ のとき Σ に対して

$$- \Sigma^0 = \varnothing,$$

$$- \Sigma^1 = \{0, 1\},$$

$$- \Sigma^2 = \{00, 01, 10, 11\},$$

$$- \Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\},$$

- ...

閉包

$$\begin{aligned} & \{ \epsilon, a, aa, aaa, \dots \} \\ & \{ \epsilon, a, a^2, a^3, a^4, \dots \} \\ & \{ a, aa, aaa, \dots \} \\ & \{ \epsilon, a^2, a^4, a^6, a^8, a^{10}, \dots \} \\ & \{ \epsilon, 01, 0101, 010101, \dots \} \\ & \{ \epsilon, a^2, a^4, a^6, a^8, a^{10}, \dots \} \\ & \{ \epsilon, 01, 0101, 010101, \dots \} \end{aligned} \quad (\Sigma = \{0, 1\} \text{ のとき})$$

集合表現

- $\{a, aa, aaa, aaaa, \dots\} = \{a^i \mid i \geq 1\} = a^+$
- $\{\epsilon, a, aa, aaa, aaaa, \dots\} = \{a^i \mid i \geq 0\} = a^*$
- $\{ab, aab, abb, aaab, aabb, abbb, \dots\}$
 $= \{a^i b^j \mid i \geq 1, j \geq 1\} = a^+ b^+$
- $\{ab, aabb, aaabbb, aaaabbbb, \dots\}$
 $= \{a^i b^i \mid i \geq 1\}$
- $\{ab, abab, ababab, abababab, \dots\}$
 $= \{(ab)^i \mid i \geq 1\} = (ab)^+$

コンパイラの構造

- 字句解析系 \Rightarrow 有限オートマトン
- 構文解析系 \Rightarrow 形式言語と形式文法
- 制約検査系
- 中間コード生成系
- 最適化系
- 目的コード生成系

文法

■ 文法の例 日本語の場合

- 「文」は「主語」「述語」から成る
- 「主語」は「名詞」「助詞」から成る
- 「述語」は「動詞」から成る
- 「名詞」は「私」または「君」である
- 「助詞」は「が」または「は」または「も」である
- 「動詞」は「笑う」または「歌う」である

私が歌う 私は踊る 「踊る」は「動詞」ではない
 君も笑う 君も私も歌う 「主語」「主語」「述語」は「文」ではない
 歌う君が 「述語」「主語」は「文」ではない

形式文法(formal grammer)

- 形式文法 $G = (N, T, S, P)$
 - N : 非終端記号の有限集合
 - T : 終端記号の有限集合 ($N \cap T = \emptyset$)
 - S : 開始記号 ($S \in N$)
 - P : 生成規則の有限集合
 - $\alpha \rightarrow \beta$ ($\alpha, \beta \in (N \cup T)^*$)
 (非終端記号と終端記号とから成る文字列)

形式文法の例

- $G = (N, T, \text{文}, P)$
 - $N = \{\text{文}, \text{主語}, \text{述語}, \text{名詞}, \text{助詞}, \text{動詞}\}$
 - $T = \{\text{私}, \text{君}, \text{が}, \text{は}, \text{も}, \text{笑う}, \text{歌う}\}$
 - $P = \{\text{文} \rightarrow \text{主語述語},$
 主語 \rightarrow 名詞助詞,
 述語 \rightarrow 動詞,
 名詞 \rightarrow 私, 名詞 \rightarrow 君,
 助詞 \rightarrow が, 助詞 \rightarrow は, 助詞 \rightarrow も
 動詞 \rightarrow 笑う, 動詞 \rightarrow 歌う}

形式文法の例

- $G = (N, T, S, P)$
 - $N = \{S, B\}$
 - $T = \{a, b, c\}$
 - $P = \{S \rightarrow abc, S \rightarrow aBSc, Ba \rightarrow aB, Bb \rightarrow bb, S \rightarrow \epsilon\}$

生成規則 $\alpha \rightarrow \beta$: 文字列 α を文字列 β に置き換える

S
 $S \rightarrow aBSc$
 $aBSc \rightarrow aBabcc$
 $aBabcc \rightarrow aaBbcc$
 $aaBbcc \rightarrow aabbcc$

| |
|---|
| 形式文法の例 |
| <ul style="list-style-type: none"> ■ $G = (N, T, \text{文}, P)$ <ul style="list-style-type: none"> - $N = \{\text{文}, \text{主語}, \text{述語}, \text{名詞}, \text{助詞}, \text{動詞}\}$ - $T = \{\text{私}, \text{君}, \text{が}, \text{は}, \text{も}, \text{笑う}, \text{歌う}\}$ - $P = \{\text{文} \rightarrow \text{主語述語}, \text{主語} \rightarrow \text{名詞助詞}, \text{述語} \rightarrow \text{動詞},$ $\text{名詞} \rightarrow \text{私}, \text{名詞} \rightarrow \text{君},$ $\text{助詞} \rightarrow \text{が}, \text{助詞} \rightarrow \text{は}, \text{助詞} \rightarrow \text{も}$ $\text{動詞} \rightarrow \text{笑う}, \text{動詞} \rightarrow \text{歌う}\}$ <p>文 \rightarrow 主語述語 \rightarrow 名詞助詞動詞 \rightarrow 私が笑う 文 \rightarrow 主語述語 \rightarrow 名詞助詞動詞 \rightarrow 君は歌う</p> |

| |
|---|
| 導出(derivation) |
| <ul style="list-style-type: none"> ■ 導出 <ul style="list-style-type: none"> - ある $\alpha \in (N \cup T)^*$ から $\beta \in (N \cup T)^*$ を生成規則を有限回用いて導き出すこと - 書式: $\alpha \Rightarrow \beta$ <ul style="list-style-type: none"> ■ $\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3 \rightarrow \dots \rightarrow \alpha_n$ のとき $\alpha_1 \Rightarrow \alpha_n$ <p>$S \rightarrow aBSc \rightarrow aBabcc \rightarrow aaBbcc \rightarrow aabbcc$ $S \Rightarrow aabbcc$ S から $aabbcc$ を導出</p> |

| |
|---|
| 形式言語(formal language) |
| <ul style="list-style-type: none"> ■ 形式言語 $L(G)$ <ul style="list-style-type: none"> - $\{\omega \mid \omega \in T^*, S \Rightarrow \omega\}$ - 形式文法 G により開始記号 S から導出される終端記号から成る文字列の集合 |

| |
|---|
| 形式言語の例 |
| <ul style="list-style-type: none"> ■ $G = (N, T, \text{文}, P)$ <ul style="list-style-type: none"> - $N = \{\text{文}, \text{主語}, \text{述語}, \text{名詞}, \text{助詞}, \text{動詞}\}$ - $T = \{\text{私}, \text{君}, \text{が}, \text{は}, \text{も}, \text{笑う}, \text{歌う}\}$ - $P = \{\text{文} \rightarrow \text{主語述語}, \text{主語} \rightarrow \text{名詞助詞}, \text{述語} \rightarrow \text{動詞},$ $\text{名詞} \rightarrow \text{私}, \text{名詞} \rightarrow \text{君},$ $\text{助詞} \rightarrow \text{が}, \text{助詞} \rightarrow \text{は}, \text{助詞} \rightarrow \text{も}$ $\text{動詞} \rightarrow \text{笑う}, \text{動詞} \rightarrow \text{歌う}\}$ <p>$L(G) = \{\text{私が笑う}, \text{私が歌う}, \text{君が笑う}, \text{君が歌う},$ $\text{私は笑う}, \text{私は歌う}, \text{君は笑う}, \text{君は歌う},$ $\text{私も笑う}, \text{私も歌う}, \text{君も笑う}, \text{君も歌う}\}$</p> |

| |
|--|
| 形式言語の例 |
| <ul style="list-style-type: none"> ■ $G = (N, T, S, P)$ <ul style="list-style-type: none"> - $N = \{S, B\}$ - $T = \{a, b, c\}$ - $P = \{S \rightarrow abc, S \rightarrow aBSc, Ba \rightarrow aB, Bb \rightarrow bb, S \rightarrow \epsilon\}$ <p>$S \rightarrow \epsilon$ $S \rightarrow abc$ $S \rightarrow aBSc \rightarrow aBabcc \rightarrow aaBbcc \rightarrow aabbcc$ $S \rightarrow aBSc \rightarrow aBaBScc \rightarrow aBaBabccc$ $\rightarrow aBaaBbccc \rightarrow aBaabbccc \rightarrow aaBabbccc$ $\rightarrow aaaBbbccc \rightarrow aaabbccc$ $L(G) = \{a^n b^n c^n \mid n \geq 0\}$</p> |

| |
|--|
| 最左導出(left most derivation) |
| <ul style="list-style-type: none"> ■ 一番左にある非終端記号から置き換える <p>例: $N = \{S, A, B, C, D\}$ $T = \{a, b, c, d\}$ $P = \{S \rightarrow ABC, A \rightarrow a, B \rightarrow bD, C \rightarrow c, D \rightarrow d\}$</p> <p>$S \rightarrow ABC$ $ABC \rightarrow aBC$ $aBC \rightarrow abDC$ $abDC \rightarrow abdC$ $abdC \rightarrow abdc$</p> <p>\Leftrightarrow 最右導出(right most derivation)</p> |

最左導出の利点

- 左から右に順に置き換えていけばいい
 - 左に戻る必要が無い

abcd EFGhIjK
 ↑ これを変換
 abcde FGhIjK
 ←
 これより前は変換済

⇒変換場所を左に戻す必要が無い

チョムスキー階層 (Chomsky hierarchy)

- 形式文法のクラスの包含階層

| 階層 | 文法 | 生成規則 |
|--------|--------|--|
| Type-0 | | 制限無し |
| Type-1 | 文脈依存文法 | $\alpha A \beta \rightarrow \alpha \gamma \beta$ |
| Type-2 | 文脈自由文法 | $A \rightarrow \gamma$ |
| Type-3 | 正規文法 | $A \rightarrow a$ or $A \rightarrow aB$ |

$A, B \in N, a \in T,$
 $\alpha, \beta, \gamma \in (N \cup T)^*$ (終端記号と非終端記号で構成される文字列)

チョムスキー階層と 受理可能な文法

| 文法 | 生成規則 | 受理可能な文法例 |
|--------|--|----------------------------|
| 文脈依存文法 | $\alpha A \beta \rightarrow \alpha \gamma \beta$ | $a^n b^n c^n \ (n \geq 0)$ |
| 文脈自由文法 | $A \rightarrow \gamma$ | $a^n b^n \ (n \geq 0)$ |
| 正規文法 | $A \rightarrow a$ or $A \rightarrow aB$ | $a^n, b^n \ (n \geq 0)$ |

$A, B \in N, a, b, c \in T,$
 $\alpha, \beta, \gamma \in (N \cup T)^*$ (終端記号と非終端記号で構成される文字列)

文脈自由文法 (context-free grammar)

$A \rightarrow \gamma \ (A \in N, \gamma \in ((N-S) \cup T)^*)$

| | |
|----|----------------------|
| 左辺 | 非終端記号1つ |
| 右辺 | S以外の非終端記号と 終端記号の列 |

例外: 開始記号 S のみ $S \rightarrow \epsilon$ も可

文脈自由文法の例

- $G = (N, T, S, P)$

- $N = \{S, A\}$

- $T = \{a, b\}$

- $P = \{S \rightarrow A, A \rightarrow ab, A \rightarrow aAb, S \rightarrow \epsilon\}$

$S \rightarrow \epsilon$

$S \rightarrow A \rightarrow ab$

$S \rightarrow A \rightarrow aAb \rightarrow aabb$

$S \rightarrow A \rightarrow aAb \rightarrow aaAbb \rightarrow aaabbb$

左辺は非終端記号1個

右辺は終端記号と
非終端記号の列

ϵ は
 $S \rightarrow \epsilon$
のみ可

$L(G) = \{a^n b^n | n \geq 0\}$

正規文法, 正則文法 (regular grammar)

$A \rightarrow a, A \rightarrow aB \ (A \in N, B \in (N-S), a \in T)$

| | |
|----|---------------------------|
| 左辺 | 非終端記号1つ |
| 右辺 | 終端記号 or 終端記号・S以外の非終端記号 |

例外: 開始記号 S のみ $S \rightarrow \epsilon$ も可

正規文法で生成される言語は
有限オートマトンで受理可能

正規文法の例

■ $G = (N, T, S, P)$

- $N = \{S, B\}$

- $T = \{a, b\}$

- $P = \{S \rightarrow a, S \rightarrow b, S \rightarrow aA, S \rightarrow bB, A \rightarrow a, A \rightarrow aA, B \rightarrow b, B \rightarrow bB, S \rightarrow \epsilon\}$

$S \rightarrow \epsilon$

$S \rightarrow a \quad S \rightarrow b$

$S \rightarrow aA \rightarrow aa \quad S \rightarrow bB \rightarrow bb$

$S \rightarrow aA \rightarrow aaA \rightarrow aaa \quad S \rightarrow bB \rightarrow bbB \rightarrow bbb$

$L(G) = \{a^n, b^n | n \geq 0\}$

左辺は非終端記号1個

右辺は終端記号1個 or
終端記号1個・非終端記号1個

正規文法の例

in, int, info を受理する正規文法

■ $G = (N, T, S, P)$

- $N = \{S, N, T, F, O\}$

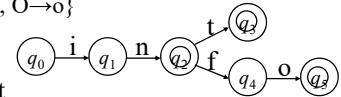
- $T = \{i, n, t, f, o\}$

- $P = \{S \rightarrow iN, N \rightarrow n, N \rightarrow nT, N \rightarrow nF, T \rightarrow t, F \rightarrow fO, O \rightarrow o\}$

$S \rightarrow iN \rightarrow in$

$S \rightarrow iN \rightarrow inT \rightarrow int$

$S \rightarrow iN \rightarrow inF \rightarrow infO \rightarrow info \quad L(G) = \{in, int, info\}$



正規言語, 正則言語 (regular language)

■ 有限オートマトンで受理される言語

$$L(M) = \{ \alpha \in \Sigma^* \mid q_0 \xRightarrow{\alpha} r, r \in F \}$$

$$= \{ \alpha \in \Sigma^* \mid \delta(q_0, \alpha) \in F \}$$

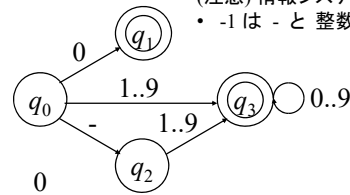
例1: $\{a^{2n+1} \mid n \geq 1\}$ {a, aaa, aaaaa, aaaaaa, ...}

例2: $a\{b\}c^*$ {a, ab, ac, abb, abc, acb, acc, ...}

正規言語の例

■ 整数を受理するオートマトン

(注意) 情報システムプロジェクトでは
・ -1 は - と 整数1 と判別



0 1 2 3 4 5 6 7 8 9 -1 -2 -3 -4 -5 -6 -7 -8 -9
10 11 12 13 14 ... -10 -11 -12 -13 -14 ...
100 101 102 103 104...

正規言語の例

■ 整数を生成する正規文法

- $N = \{S, P, D\}$

- $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -\}$

- $P = \{S \rightarrow 0, S \rightarrow P,$

$S \rightarrow 1, S \rightarrow 2, S \rightarrow 3, \dots, S \rightarrow 9,$

$S \rightarrow 1D, S \rightarrow 2D, S \rightarrow 3D, \dots, S \rightarrow 9D,$

$P \rightarrow 1, P \rightarrow 2, P \rightarrow 3, \dots, P \rightarrow 9,$

$P \rightarrow 1D, P \rightarrow 2D, P \rightarrow 3D, \dots, P \rightarrow 9D,$

$D \rightarrow 0, D \rightarrow 1, D \rightarrow 2, D \rightarrow 3, \dots, D \rightarrow 9,$

$D \rightarrow 0D, D \rightarrow 1D, D \rightarrow 2D, S \rightarrow 3D, \dots, D \rightarrow 9D\}$

BNF記法(Buckus Naur form)

■ 文法の記述法

| | 生成規則 | BNF記法 |
|-------|---------------|-------|
| 生成則 | \rightarrow | $::=$ |
| 終端記号 | $a \in T$ | “文字列” |
| 非終端記号 | $A \in N$ | <文字列> |

例: $E \rightarrow T+T, \langle \text{exp} \rangle ::= \langle \text{term} \rangle + \langle \text{term} \rangle$

$T \rightarrow 0 \quad \langle \text{term} \rangle ::= \langle 0 \rangle$

非終端記号

終端記号

BNF記法の例

または

```

<Integer> ::= "0" | <Pdeclist> | "-" <Pdeclist>
<Pdeclist> ::= <Pdec> | <Pdec> <Declist>
<Declist> ::= <Dec> | <Dec> <Declist>
<Pdec> ::= "1" | "2" | "3" | "4" | "5"
           | "6" | "7" | "8" | "9"
<Dec> ::= "0" | <Pdec>
    
```

再帰

BNF記法では繰り返しの定義には再帰が必要

EBNF記法

(extended Backus Naur form)

■ BNF記法の拡張

| EBNF記法 | 意味 |
|-----------------|----------------------|
| α, β | 記号列 $\alpha\beta$ |
| $\alpha \beta$ | α または β |
| $[\alpha]$ | 省略可能 (0回または1回) |
| $\{\alpha\}$ | 省略可能な繰り返し (0回以上) |

EBNF記法の例

0回または1回

0回以上

```

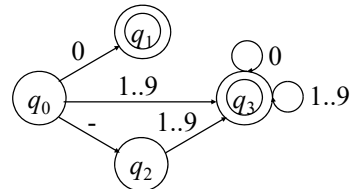
<Integer> ::= "0" | ["-"] <Pdec> { <Dec> }
<Pdec> ::= "1" | "2" | "3" | "4" | "5"
           | "6" | "7" | "8" | "9"
<Dec> ::= "0" | <Pdec>
    
```

0
 1 2 3 4 5 6 7 8 9 -1 -2 -3 -4 -5 -6 -7 -8 -9
 10 11 12 13 14 ... -10 -11 -12 -13 -14 ...
 100 101 102 103 104...
 1000 1001 1002 1003 1004...

EBNF記法の例

```

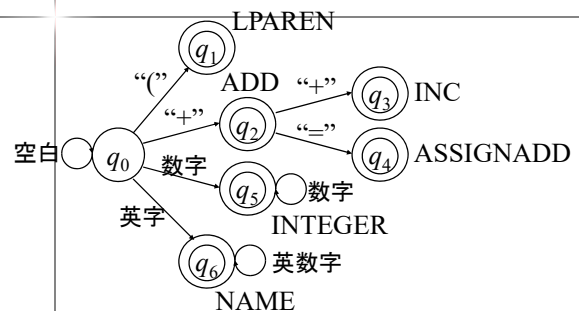
<Integer> ::= "0" | ["-"] <Pdec> { <Dec> }
<Pdec> ::= "1" | "2" | "3" | "4" | "5"
           | "6" | "7" | "8" | "9"
<Dec> ::= "0" | <Pdec>
    
```



正規表現とEBNF記法

| 意味 | 正規表現 | EBNF記法 |
|----------------------|------|---------|
| 連接 a の後に b | ab | a , b |
| 選択 a または b | a b | a b |
| 省略可能 (0回または1回) | a? | [a] |
| 省略可能な繰り返し (0回以上) | a* | { a } |
| 省略不可能な繰り返し (1回以上) | a+ | a { a } |

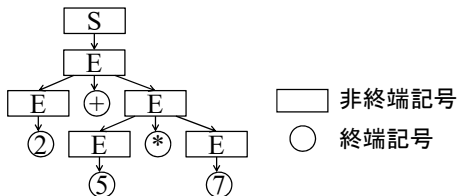
字句解析オートマトン(一部)



導出木(derivation tree)

■ 導出を木の形で表わしたもの

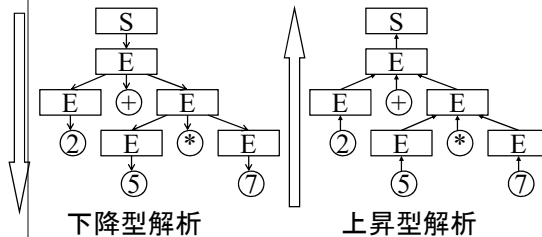
例 : $N = \{S, E\}$ $T = \{2, 5, 7, +, *\}$
 $P = \{S \rightarrow E, E \rightarrow E+E, E \rightarrow E * E, E \rightarrow 2, E \rightarrow 5, E \rightarrow 7\}$
 $S \rightarrow E \rightarrow E+E \rightarrow 2+E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$



構文解析

■ $\omega \in T^*$ に対して

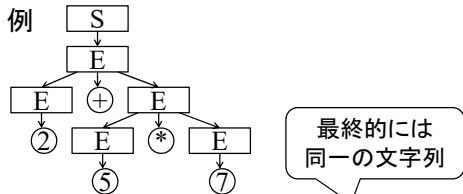
$S \Rightarrow \omega$ であるか判定, その導出木を得る



導出木

■ 同一の導出木

⇒ 同一の文字列が得られる導出



$S \rightarrow E \rightarrow E+E \rightarrow 2+E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$
 $S \rightarrow E \rightarrow E+E \rightarrow E+E * E \rightarrow E+E * 7 \rightarrow E+5 * 7 \rightarrow 2+5 * 7$

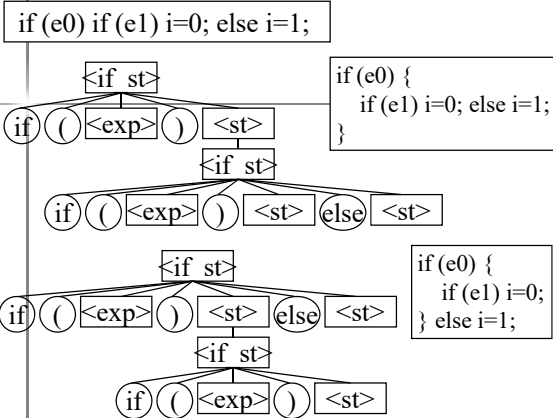
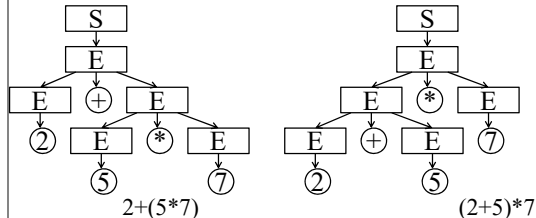
導出木

■ 同一の文字列が得られる導出

曖昧な文法

$S \rightarrow E \rightarrow E+E \rightarrow E+E * E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$

$S \rightarrow E \rightarrow E * E \rightarrow E+E * E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$



曖昧な文法(ambiguous)

■ 曖昧な文法

- 同一の文字列に対して異なる導出木

例 : $N = \{S, E\}$ $T = \{2, 5, 7, +, *\}$

$P = \{S \rightarrow E, E \rightarrow E+E, E \rightarrow E * E, E \rightarrow 2, E \rightarrow 5, E \rightarrow 7\}$

$S \rightarrow E \rightarrow E+E \rightarrow E+E * E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$

$S \rightarrow E \rightarrow E * E \rightarrow E+E * E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$

$2+(5 * 7)$ と $(2+5) * 7$ の区別ができない

⇒ 曖昧性の除去が必要

曖昧性の除去

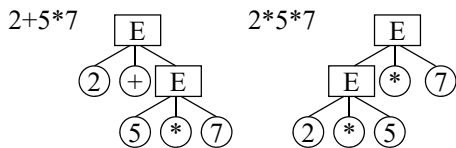
- 手法1
 - 曖昧性が無いように文法を書き換える
- 手法2
 - 新たな制約を加える
 - 演算子の優先順位, 結合性等

曖昧性の除去

- 手法1
 - 曖昧性が無いように文法を書き換える
- 例: $N = \{S, E\}$ $T = \{2, 5, 7, +, *\}$
 $P = \{S \rightarrow E, E \rightarrow E+E, E \rightarrow E * E, E \rightarrow 2, E \rightarrow 5, E \rightarrow 7\}$
- $N = \{S, E, T, F\}$ $T = \{2, 5, 7, +, *\}$
 $P = \{S \rightarrow E, E \rightarrow T, E \rightarrow T+E, T \rightarrow F, T \rightarrow T * F, F \rightarrow 2, F \rightarrow 5, F \rightarrow 7\}$

曖昧性の除去

- 手法2
 - 新たな制約を加える
 - 演算子の優先順位, 結合性等
- 例: 優先順位: * は + より先に計算
 結合性: * 同士, + 同士は左から先に計算

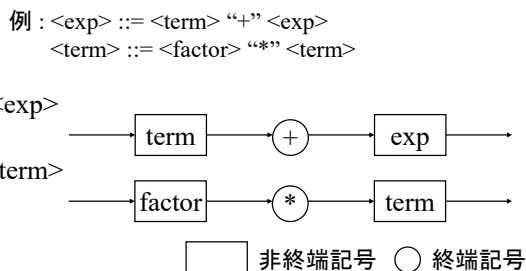


導出木と最左導出

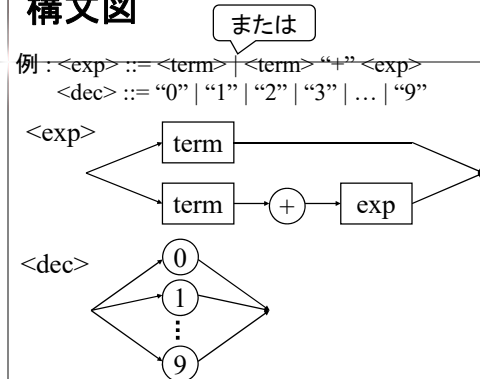
- 同一の導出木 = 同一の最左導出
- $S \rightarrow E \rightarrow E+E \rightarrow 2+E \rightarrow 2+E * E \rightarrow 2+5 * E \rightarrow 2+5 * 7$
 $S \rightarrow E \rightarrow E+E \rightarrow E+E * E \rightarrow E+E * 7 \rightarrow E+5 * 7 \rightarrow 2+5 * 7$
-
- 導出が異なっても同じ導出木
 一つの導出木には一つの最左導出が対応

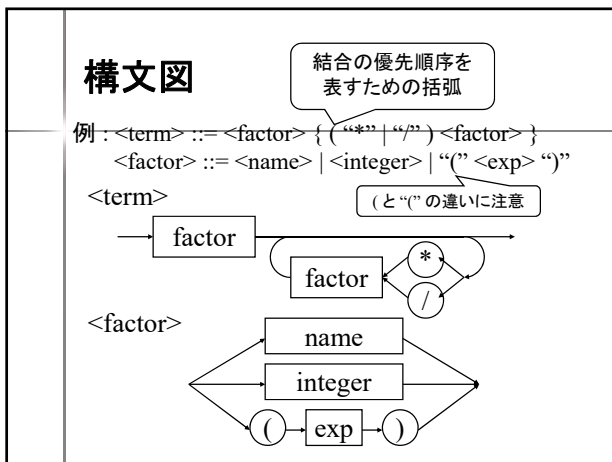
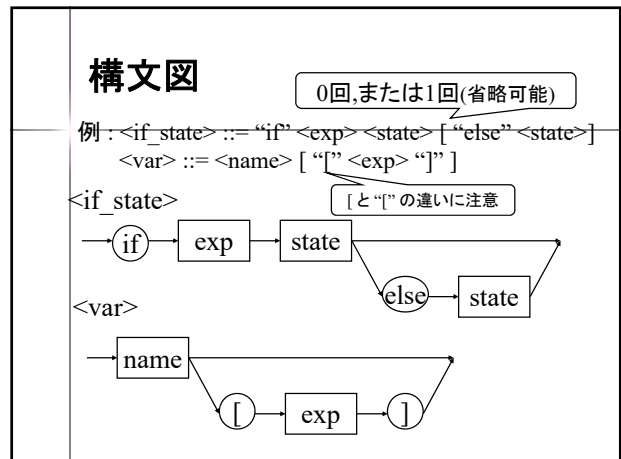
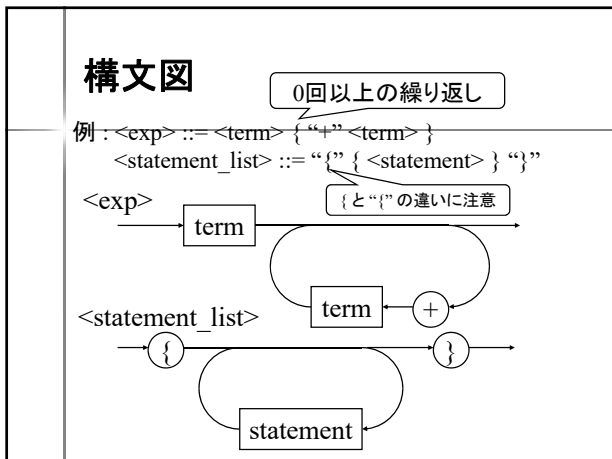
構文図

- 構文解析の流れを示した図



構文図





課題テスト

- 毎週 GoogleClassroom上で課題テストを行う
 - 授業後~翌週の授業開始まで
- GoogleClassroomで
 - コンパイラ
 - ⇒授業
 - ⇒その回の課題と辿る

