コンパイラ

湯淺太一

第３章

文　　法

構文，制約，意味

構文(syntax)：プログラムの構造を定義
制約(constraint)：構文以外の規則
意味(semantics)：実行を定義

例：if文
　構文：`if` ($expr$) $stat_1$ `else` $stat_2$
　制約：$expr$は真偽値を表す型
　意味：$expr$を計算し，その値が真なら$stat_1$を，
　　偽なら$stat_2$を実行する

文法：構文＋制約
　構文解析は「文法解析」ではない．

構文解析の話をするときは，制約には言及しない．
　→ 構文＝文法

構文の記法

ALGOL 60 の BNF

⟨program⟩ `::=`
  `program` ⟨identifier⟩ `(` ⟨identifier-list⟩ `)` `;` ⟨block⟩ `.`
  `|` `program` ⟨identifier⟩ `;` ⟨block⟩ `.`

⟨identifier⟩ は非終端記号 (nonterminal symbol) あるいは構文変数 (syntax variable)

`program` は，終端記号 (terminal symbol)

⟨identifier-list⟩ `::=` ⟨identifier⟩ `|` ⟨identifier-list⟩ `,` ⟨identifier⟩

識別子リストとは:

1. 識別子であるか,
2. 識別子リストと識別子をカンマで区切ったもの

例：“`x, y, z`”

1. `x` は識別子なので識別子リスト
2. `x` が識別子リストなので “`x, y`” も識別子リスト
3. “`x, y`” が識別子リストなので，“`x, y, z`” は識別子リスト

生成規則 (production)

`::=` と縦棒は，メタ記号 (meta-symbol)

拡張BNFの例：

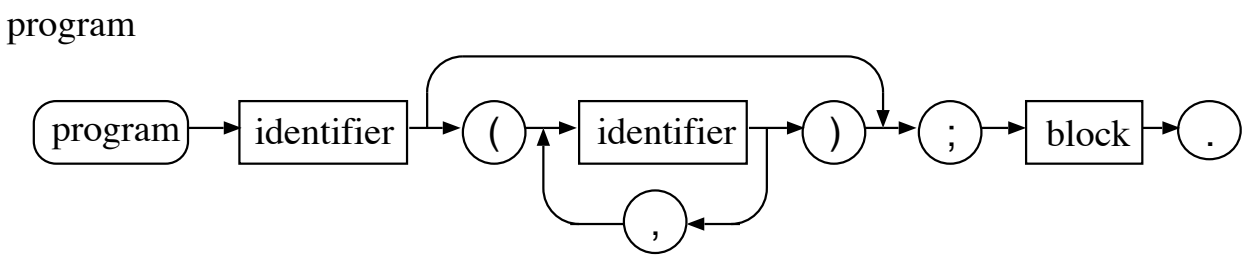
⟨identifier-list⟩ `::=` { ⟨identifier⟩ `,` } ⟨identifier⟩

C言語の構文記法：

identifier-list:
  identifier
  identifier-list `,` *identifier*

`for` `(` $expr_{opt}$ `;` $expr_{opt}$ `;` $expr_{opt}$ `)` *stat*

Pascalの構文図式(syntax diagram)の例：



文法

生成規則：$A \to \alpha$　(Aは記号，αは記号列)

ε規則：$A \to \varepsilon$

文法：組$\langle P, S\rangle$

P：生成規則の集合

S：出発記号(start symbol)

語彙(vocabulary) V：Pに現れる記号全体の集合

非終端記号V_N：生成規則の左辺に現れる記号

終端記号V_T：非終端記号以外の記号

$$V = V_N \cup V_T$$
$$V_N \cap V_T = \phi$$
$$S \in V_N$$

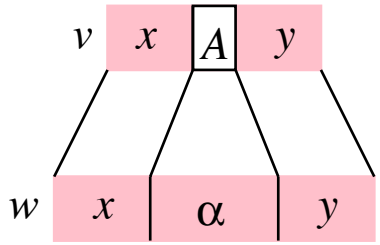
例：文法G1 $= \langle \mathrm{P1}, E\rangle = \langle V_N, V_T, \mathrm{P1}, E\rangle$

$$\begin{aligned}\mathrm{P1} = \{\ & E \to E + T \\ & E \to T \\ & T \to T * F \\ & T \to F \\ & F \to (\ E\ ) \\ & F \to i\ \}\end{aligned}$$

$$V_N = \{E, F, T\}$$
$$V_T = \{+, *, (,), i\}$$
$$\begin{aligned}V &= V_N \cup V_T \\ &= \{E, F, T, +, *, (,), i\}\end{aligned}$$

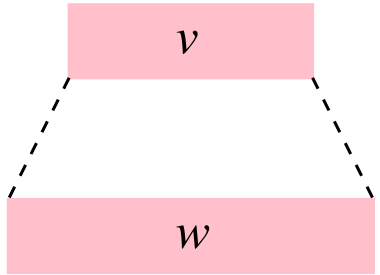
$$\begin{aligned}\mathrm{P1} = \{\ & E \to E + T \mid T \\ & T \to T * F \mid F \\ & F \to (\ E\ ) \mid i\ \}\end{aligned}$$

$v = xAy$, $w = x\alpha y$,
$A \to \alpha \in P$のとき，$v \Rightarrow w$
　vはwを直接生成する．
　wはvに直接還元される．



$v \Rightarrow u_1 \Rightarrow u_2 \cdots \Rightarrow u_n \Rightarrow w$ のとき，$v \overset{+}{\Rightarrow} w$

$v \overset{+}{\Rightarrow} w$または$v = w$のとき，$v \overset{*}{\Rightarrow} w$
　vはwを生成(produce)する．
　wはvに還元(reduce)される．



$S \overset{*}{\Rightarrow} x$のとき，$x$は$G$の文形式(sentential form)
　特にxが終端記号列ならxはGの文(sentence)

$L(G)$：Gの文全体の集合，Gの定義する言語

例：$\mathrm{P1} = \{\ E \to E+T \mid T$
　　$T \to T*F \mid F$
　　$F \to (\,E\,) \mid i\ \}$

$E \Rightarrow E+T \Rightarrow T+T \Rightarrow T+T*F \Rightarrow F+T*F$
　$\Rightarrow F+T*i \Rightarrow F+F*i \Rightarrow F+i*i \Rightarrow i+i*i$

$i+i*i$だけが文

導出(derivation)：

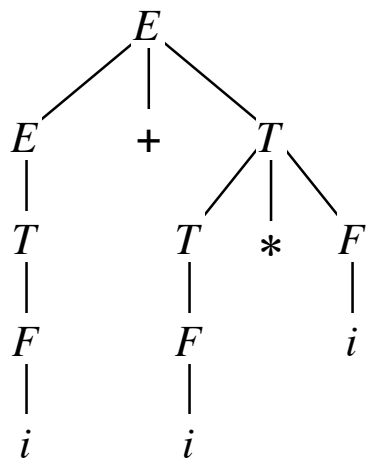
vから$v \overset{*}{\Rightarrow} w$である記号列$w$を求めること

最左(left-most)導出

$$E \underset{\mathrm{lm}}{\Longrightarrow} E+T \underset{\mathrm{lm}}{\Longrightarrow} T+T \underset{\mathrm{lm}}{\Longrightarrow} F+T \underset{\mathrm{lm}}{\Longrightarrow} i+T \underset{\mathrm{lm}}{\Longrightarrow} i+T*F$$
$$\underset{\mathrm{lm}}{\Longrightarrow} i+F*F \underset{\mathrm{lm}}{\Longrightarrow} i+i*F \underset{\mathrm{lm}}{\Longrightarrow} i+i*i$$

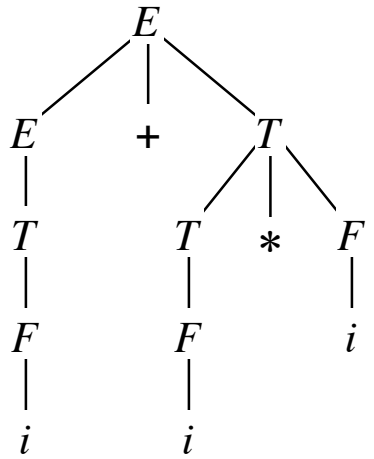
最右(right-most)導出

$$E \underset{\mathrm{rm}}{\Longrightarrow} E+T \underset{\mathrm{rm}}{\Longrightarrow} E+T*F \underset{\mathrm{rm}}{\Longrightarrow} E+T*i \underset{\mathrm{rm}}{\Longrightarrow} E+F*i$$
$$\underset{\mathrm{rm}}{\Longrightarrow} E+i*i \underset{\mathrm{rm}}{\Longrightarrow} T+i*i \underset{\mathrm{rm}}{\Longrightarrow} F+i*i \underset{\mathrm{rm}}{\Longrightarrow} i+i*i$$

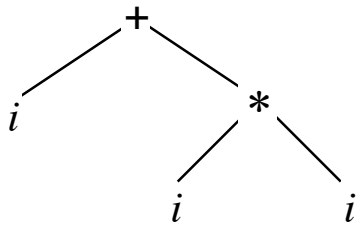


解析木とあいまい性

解析木



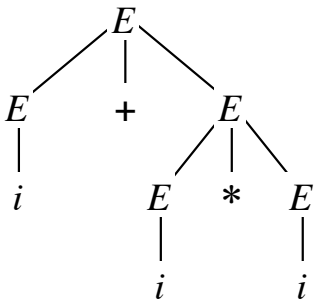
構文木

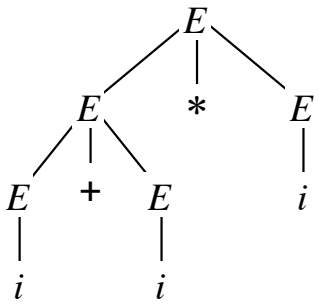


あいまいな文：二つの異なる解析木が考えられる文
あいまいな文法：あいまいな文を生成する文法

例：文法 $\mathrm{G2} = \langle \mathrm{P2}, E \rangle$，$\mathrm{P2} = \{\ E \to E + E \mid E * E \mid (\ E\ ) \mid i\ \}$

文 $i + i * i$ に対する二つの解析木





あいまいな英文

Time flies.

「主語＋動詞」→「光陰矢の如し」
「動詞＋目的語」→「蝿の速度計測をしろ」

あいまいな日本語文

ここではきものをぬぎなさい

「ここで履物を脱ぎなさい」
or「ここでは着物を脱ぎなさい」？

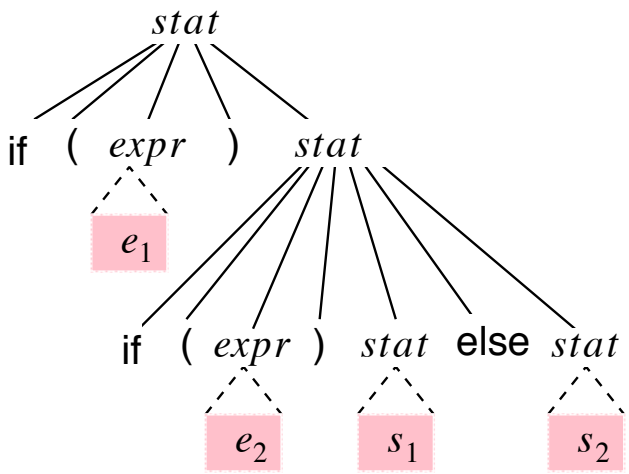
if 文の構文

stat:
  `if` (*expr*) *stat*
  `if` (*expr*) *stat* `else` *stat*
  other

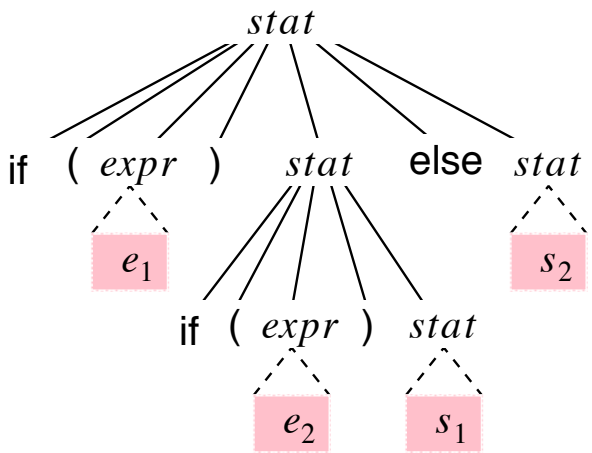
あいまいな if 文

`if` (e_1) `if` (e_2) s_1 `else` s_2

`if` (e_1) { `if` (e_2) s_1 `else` s_2 }



`if` (e_1) { `if` (e_2) s_1 } `else` s_2



C 言語の場合は左側

if 文のあいまい性除去

開いた（open）文：直後に「`else`＋文」をつけても
　文になり得る文
閉じた（closed）文：open でない文

stat:
　closed-stat
　open-stat
closed-stat:
　`if` (*expr*) *closed-stat* `else` *closed-stat*
　other
open-stat:
　`if` (*expr*) *stat*
　`if` (*expr*) *closed-stat* `else` *open-stat*

演算子の優先順位と結合性

演算子の優先順位

$$x + y * z = x + (y * z)$$

演算子の結合性

$$x - y - z = (x - y) - z$$

なので左結合的(left associative)

$$x = y = z = x = (y = z)$$

なので右結合的

優先順位と結合性

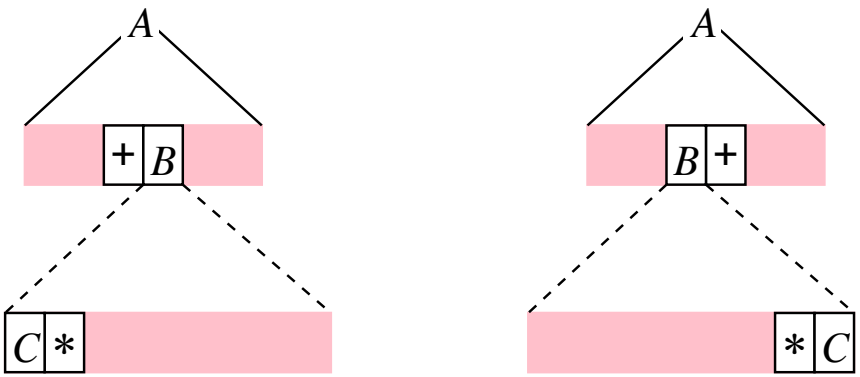
$$x + y - z = (x + y) - z$$

$op_1 \lessdot op_2$：$\cdots op_1\ expr\ op_2 \cdots$のとき op_2を先に計算

1. 生成規則 $A \rightarrow \cdots op_1\, B \cdots$が存在
2. ある非終端記号Cに対して $B \overset{+}{\Rightarrow} C\ op_2 \cdots$

$op_1 \gtrdot op_2$：$\cdots op_1\ expr\ op_2 \cdots$のとき op_1を先に計算

1. 生成規則 $A \rightarrow \cdots B\ op_2 \cdots$が存在
2. ある非終端記号Cに対して $B \overset{+}{\Rightarrow} \cdots op_1\, C$



例：文法G1

1. $E \to E + T$が存在し，
2. $E \Rightarrow E + T$

$\to\ + \dot{>} +$

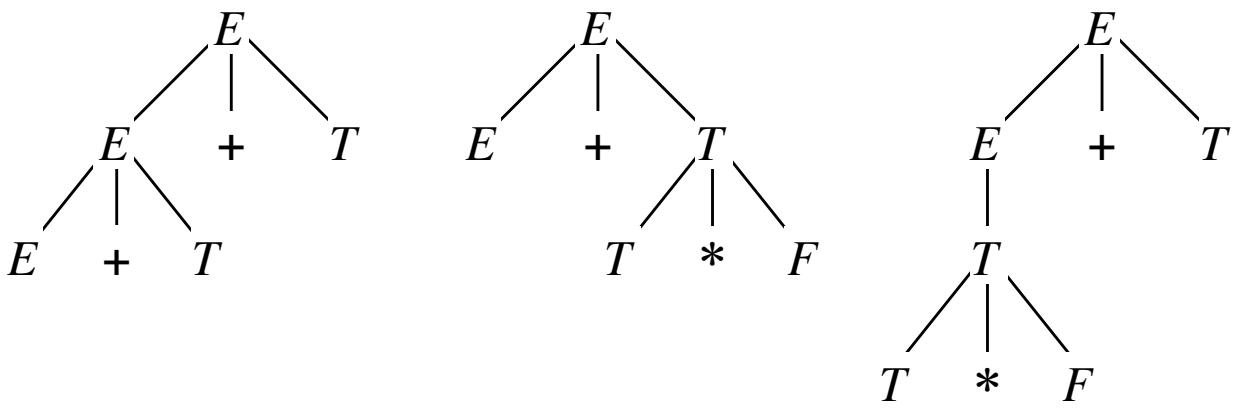
1. $E \to E + T$が存在し
2. $T \Rightarrow T * F$

$\to\ + \dot{<} *$

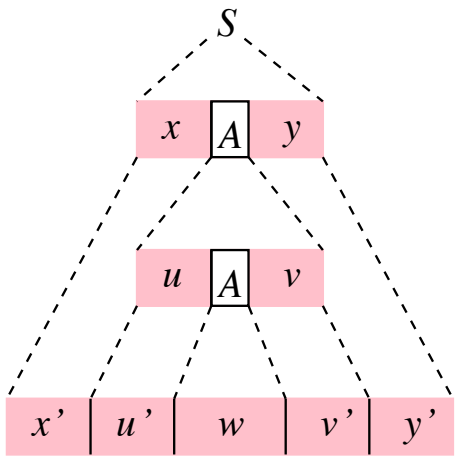
1. $E \to E + T$が存在し，
2. $E \Rightarrow T \Rightarrow T * F$

$\to\ * \dot{>} +$

$+ \dot{<} *$かつ$* \dot{>} +$なので，$*$は$+$より優先順位が高い．



文脈自由文法とその限界



Aが文形式に現れていれば，Aの前後（文脈）に関係なく，Aを左辺とする生成規則を適用可能

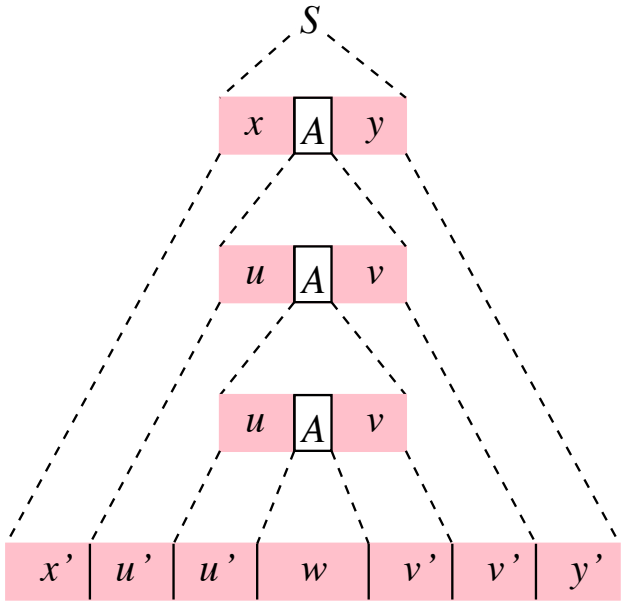
文脈自由文法で表現できない記号列集合

$$Y = \{a^i\, b^i\, c^i \mid i \geq 1\} = \{abc,\ aabbcc,\ aaabbbccc,\ \ldots\}$$

$L(G) = Y$なる文脈自由文法Gが存在したと仮定

次の$a^j b^j c^j$が存在する．

$$S \overset{*}{\Rightarrow} xAy \overset{+}{\Rightarrow} xuAvy \overset{+}{\Rightarrow} a^j b^j c^j$$



Gが生成する記号列

$$S \overset{*}{\Rightarrow} xAy \overset{+}{\Rightarrow} xuAvy \overset{+}{\Rightarrow} xuuAvvy \overset{+}{\Rightarrow} x'u'u'wv'v'y'$$

は，Yの要素でない．$\rightarrow$矛盾

Y を生成する文脈自由でない文法

$$
\begin{aligned}
&S \to aB\,b\,c \\
&B \to aB\,b\,C \mid \varepsilon \\
&Cb \to b\,C \\
&Cc \to c\,c
\end{aligned}
$$

$$
\begin{aligned}
S &\Rightarrow aBbc \Rightarrow aaBbCbc \Rightarrow aaaBbCbCbc \\
&\Rightarrow aaabCbCbc \Rightarrow aaabCbbCc \Rightarrow aaabCbbcc \\
&\Rightarrow aaabbCbcc \Rightarrow aaabbbCcc \Rightarrow aaabbbccc
\end{aligned}
$$