言語プロセッサ2014
ー 第6回目(11月10日) ー

Tokyo University of Technology
School of computer Science
Hiroyuki KAMEDA

今日の内容

- 構文解析
  - 構文解析技術の基盤理論（言語学から）
  - First集合とFollow集合　など
- 構文解析作成用ツール
  - 紹介
  - レポート課題（AntlrWorks）

LL(1)文法

- LL(1)文法のイメージ:

  $A \rightarrow \alpha \mid \beta$

  という規則で、αかβのどちらの書換えを選ぶかを決めるとき、入力の先頭記号1個を見ることにより、バックトラックが起きないような選択が可能な文法。

  つまり、適応するべき文法規則を、1文字先読みすれば決定できるということ。

つまり、迷路を迷わず駆け抜けることができる、ということ。凄いと思いませんか？

形式文法(復習)

もう聞き飽きた？

文法 G=(V, N, P, S),

ただし、

- V: 終端記号の集合(語彙)
- N: 非終端記号の集合(構文構造記述用語集)
- P: 書換え規則の集合
- S: 開始記号

* ちょっと一言

現在は変化の速い時代です.
小学生の頃のことを思い起こしてください.
あの時と今は、どう違っていますか？

- **15世紀**: Gutenberg（印刷技術）
- **19～20世紀**: Marconi（無線通信）
- **20世紀**: 機械式計算機
  電子式計算機（digital/analog computer）
  電話（自動車・ポケベル・携帯・Skype）
  ファクシミリ・電子メール・WWW
  電子マネー・ファミコンゲーム
- **21世紀**: Suica・PASMO，電子マネー（スマートカード）
  iPhone, iPad, 3Dテレビ，8Kテレビ，
  Virtual Reality(OculusなどのHMD)

- 多くのものが、生まれては消えています。
  しかし,…

  これから本授業で学ぶことは、50年後でも
  役立つ知識です。頑張って学びましょう！

ステップアップの
ための復習

より深い理解を求めて

オートマトンと言語
Automaton & Languages

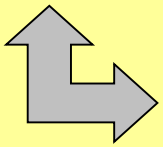
平成16年度開講科目
3回目より
(一部書き換えありBy　H.KAMEDA 2005/12/21, 2006/12/15
2007/12/27）

確認事項

- 人間の頭の中には、言語処理装置がある。
- すべての文を記憶しているわけではない。
- 文法として記憶している。

Chomskyの意見

- 文法とは何か？
  - 規範文法(Prescriptive Grammar)
  - 記述文法(Descriptive Grammar)
- 形式文法と形式言語



形式言語 v.s. 自然言語

まずは，頭の整理から

- 言語処理を考えたい
- 処理の対象は？　言語
- 自然言語と人工言語
- 言語の本質を切り出して整理
  　＝＞形式言語学
- 言語とは文法的に正しい文の集合L
- 言語Lは文法Gによって定義
- それでは文法Gとは…

また出てきた！

形式文法と形式言語

- 文法G = (V_n, V_t, S, P) :
  - ただし、
    - V_n（非終端記号の集合）: $0 < \#V_n < +\infty$
    - V_t（終端記号の集合）: $0 < \#V_t < +\infty$
    - S 開始記号(S∈Vn)
    - P（書き換え規則の集合）:

      $\{\alpha \rightarrow \beta \mid \alpha, \beta \in (Vn \cup Vt)^*\}$

- 言語L = L(G) = { x | S =*=> x }
  - ただし、S => ・・・ => x ∈ V_t

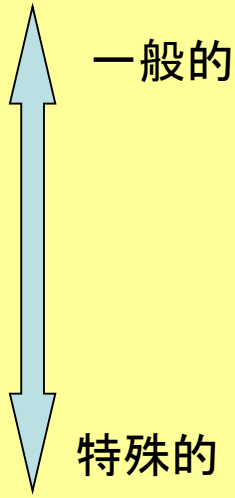
形式文法と形式言語(例)

- 文法G = (V_n, V_t, S, P) :
  - V_n ={S, B}　非終端記号の集合(文構造記述用語)
  - V_t={a, b, c}　終端記号の集合(単語の集合 = 語彙)
  - 開始記号S　書き換えの*種*(構文木の根)
  - P={ S → aBc, B → b | bc}　書き換え規則群

  - 言語L = L(G) = { α | S =*> α }

言語の階層(重要)

- 言語(文法)は階層構造をなしている。
  - 句構造言語 ⇔ 句構造文法
  - 文脈依存言語 ⇔ 文脈依存文法
  - 文脈自由言語 ⇔ 文脈自由文法
  - 正規言語 ⇔ 正規文法



Chomsky階層
(Chomsky Hierarchy)
とも言う。

句構造文法 (Phrase-Structure Grammar; PSG)

- 文法G = (V_n, V_t, P, S):
  - V_n(非終端記号の集合): $0 < {}^{\#}V_n < +\infty$
  - V_t: 終端記号の集合: $0 < {}^{\#}V_t < +\infty$
  - P: 書き換え規則の集合

    $\{\alpha \rightarrow \beta \mid \alpha, \beta \in (Vn \cup V_t)^*\}$
  - S: 開始記号($S \in V_n$)

句構造文法
(Phrase-Structure Grammar; PSG)

- 文法G = (V_n, V_t, P, S):
  - V_n(非終端記号の集合): $0 < {}^{\#}V_n < +\infty$
  - V_t: 終端記号の集合: $0 < {}^{\#}V_t < +\infty$
  - P: 書き換え規則の集合
    $\{\alpha \rightarrow \beta \mid \alpha, \beta \in (Vn \cup V_t)^*\}$
  - S: 開始記号($S \in V_n$)

- 言語L=L(G)

句構造文法
(Phrase-Structure Grammar; PSG)

- 文法G = (V_n, V_t, P, S):
  - V_n(非終端記号の集合): $0 < \#V_n < +\infty$
  - V_t: 終端記号の集合: $0 < \#V_t < +\infty$
  - P: 書き換え規則の集合
    $\{\alpha \rightarrow \beta \mid \alpha, \beta \in (Vn \cup V_t)^*\}$
  - S: 開始記号($S \in V_n$)
- 言語L=L(G): 句構造言語

ここに制限が付くと他の文法になる。

文脈依存文法
(Context-Sensitive Grammar; CSG)

- 文法G = (V_n, V_t, P, S):
  - V_n(非終端記号の集合): $0 < \#V_n < +\infty$
  - V_t: 終端記号の集合: $0 < \#V_t < +\infty$
  - P: 書き換え規則の集合
    $\{\alpha X\beta \rightarrow \alpha\gamma\beta \mid \alpha, \beta \in (V_n \cup V_t)^*, X \in V_n, \gamma \in (V_n \cup V_t)^+ \}$
  - S: 開始記号($S \in V_n$)

- 言語L=L(G):文脈依存言語

文脈自由文法
(Context-Free Grammar; CFG)

- 文法G = (V_n, V_t, P, S):
  - V_n(非終端記号の集合): $0 < {}^{\#}V_n < +\infty$
  - V_t: 終端記号の集合: $0 < {}^{\#}V_t < +\infty$
  - P: 書き換え規則の集合
    $\{ X \rightarrow \alpha \mid \alpha \in (Vn \cup V_t)^* \}$
  - S: 開始記号($S \in V_n$)

- 言語L=L(G):文脈自由言語

正規文法
(Regular Grammar; RG)

- 文法G = (V_n, V_t, P, S):
  - V_n(非終端記号の集合): $0 < {}^{\#}V_n < +\infty$
  - V_t: 終端記号の集合: $0 < {}^{\#}V_t < +\infty$
  - P: 書き換え規則の集合
    $\{X \rightarrow aY,\ X \rightarrow b \mid X,Y \in Vn,\ a,b \in V_t^*\}$
  - S: 開始記号($S \in V_n$)

- 言語L=L(G): 正規言語

生成規則部分の比較

- PSG: $\alpha \rightarrow \beta$
- CSG: $\alpha X \beta \rightarrow \alpha \gamma \beta$
- CFG: $X \rightarrow \alpha$
- RG: $X \rightarrow aY,\ X \rightarrow b$

– ただし、
  - $\alpha, \beta \in V^*$ ・$\gamma \in V^+$
  - $X, Y \in V_n$ ・$a, b \in V_t$ ・$V = V_n \cup V_t$

つまり，ギリシア文字は文字列を，ローマ字は1個の文字を表しています

生成規則部分の比較

- PSG: $\alpha \rightarrow \beta$
- CSG: $\alpha X \beta \rightarrow \alpha \gamma \beta$
- CFG: $X \rightarrow \alpha$
- RG: $X \rightarrow aY,\ X \rightarrow b$

  – ただし、
  - $\alpha, \beta \in V^*$ ・$\gamma \in V^+$
  - $X, Y \in V_n$ ・$a, b \in V_t$ ・$V = V_n \cup V_t$

生成規則部分の比較

- PSG: $\alpha \rightarrow \beta$
- CSG: $\alpha X \beta \rightarrow \alpha \gamma \beta$
- CFG: $X \rightarrow \alpha$
- RG: $X \rightarrow aY,\ X \rightarrow b$

  - ただし、
    - $\alpha, \beta \in V^*$ ・$\gamma \in V^+$
    - $X, Y \in V_n$ ・$a, b \in V_t$ ・$V = V_n \cup V_t$

生成規則部分の比較

- PSG: $\alpha \rightarrow \beta$
- CSG: $\alpha X \beta \rightarrow \alpha \gamma \beta$
- CFG: $X \rightarrow \alpha$
- RG: $X \rightarrow aY,\ X \rightarrow b$

  – ただし、
  - $\alpha, \beta \in V^*$ ・$\gamma \in V^+$
  - $X, Y \in V_n$ ・$a, b \in V_t$ ・$V = V_n \cup V_t$

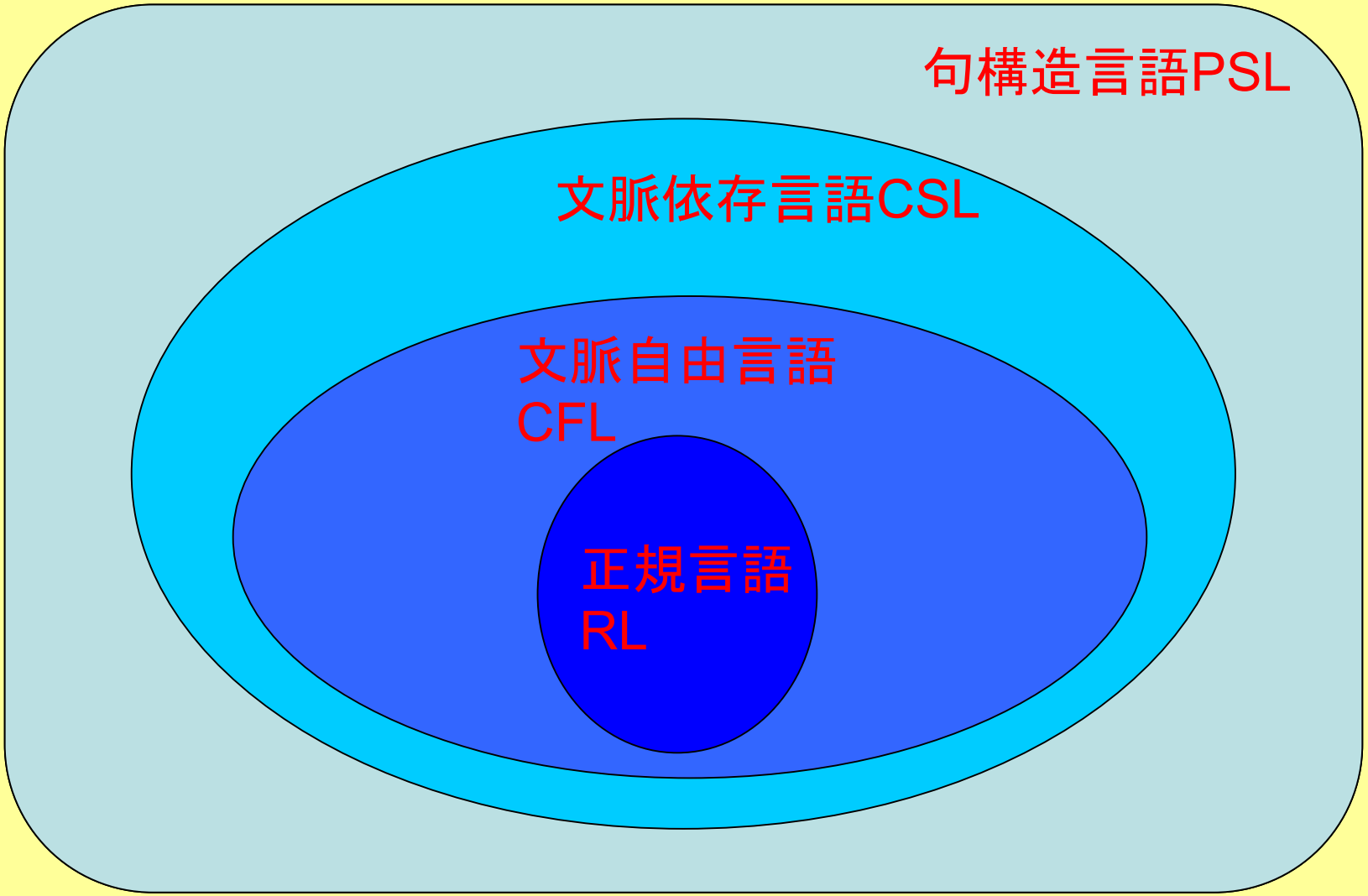
生成規則部分の比較

- PSG: $\alpha \rightarrow \beta$
- CSG: $\alpha X \beta \rightarrow \alpha \gamma \beta$
- CFG: $X \rightarrow \alpha$
- RG: $X \rightarrow aY,\ X \rightarrow b$

  – ただし、
  - $\alpha, \beta \in V^*$ ・$\gamma \in V^+$
  - $X, Y \in V_n$ ・$a, b \in V_t$ ・$V = V_n \cup V_t$

Chomsky階層

重要



言語の包含関係

L（PSG) ⊃ L(CSG) ⊃ L(CFG) ⊃ L(RG)

このうち、大切なのはCFGとRG。

CFGとRG

- CFG（文脈自由文法）:
  - プログラミング言語設計
  - コンパイラの構文解析
  - 自然言語処理（機械翻訳・仮名漢字変換）

3年次後期月曜1限開講科目です！

- RG（正規文法）:
  - 正規表現（検索・コンパイラの字句解析）

CFGの特徴

1. CFGには標準形がある。
2. 導出の過程を木で表現できる(導出木の存在)。
3. 解析手法が豊富に知られている。
4. 自然言語処理に部分的に適用できる。
5. プログラミング言語設計に利用されている。

プログラミング言語とその原理

1．CFGの標準形

- Chomskyの標準形
- Greibachの標準形

標準形があるということは、
一般論で議論しやすいですよね。

Chomskyの標準形

- 任意のCFGにおける書き換え規則群Pは、A→BC　または　A→a という形だけで表現できる。

Greibachの標準形

- 任意のCFGにおける書き換え規則群Pは、$A \rightarrow a\alpha$ という形だけで表現できる。ただし、$X \in Vn$, $a \in Vt$, $\alpha \in Vn^*$。

ドイツ語圏の名前なので、「グライバッハ」と読んでもいいが、英語読みで「グライバック」と読む人も多い。本人が何と読んでいるのが分かればいいのですが…

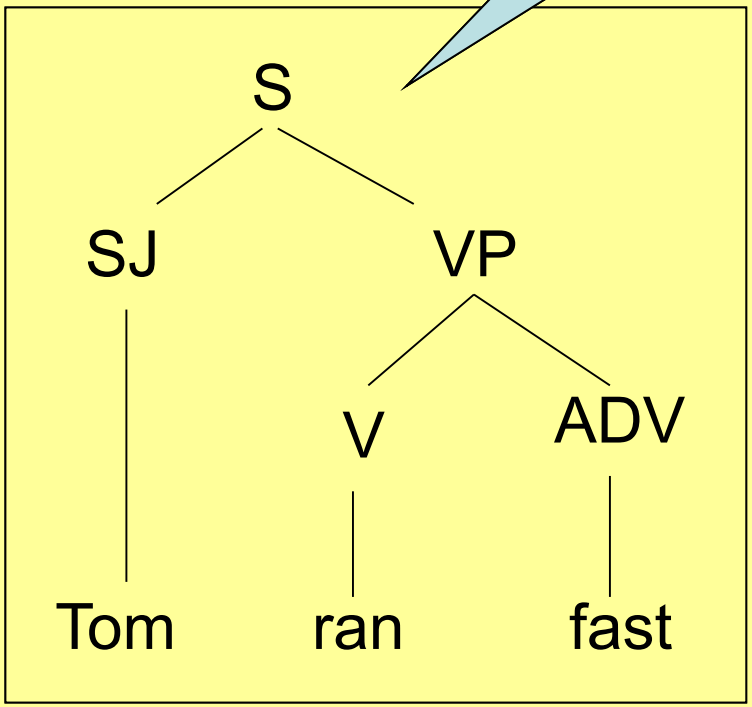
2. 導出木

- 導出木とは
  - 導出の過程を木構造で表現したもの。

- 例:

S => SJ VP
=> Tom V ADV
=> Tom ran fast

導出過程



３．解析手法

- CKY法(Cocke-Kasami-Younger　method)
- Early法(Early’s algorithm)
- Chart法(Chart algorithm)
- 優先順位文法法
- LR(k) 法
- LALR(k) 法
- SLR(k) 法
- LL(k) 法　　などなど

3. 解析手法

- CKY法(Cocke-Kasami-Younger method)
- Early法(Early's algorithm)
- Chart法(Chart algorithm)
- 優先順位文法法
- LR(k) 法
- LALR(k) 法
- SLR(k) 法

} Bottom up構文解析用

- LL(k) 法

} Top down 構文解析用
再帰的下向き構文解析用

解析手法は重要です。

（後日あらためて取り上げます。そして、試験範囲です）

- 機械翻訳・通訳電話などの自然言語処理
- コンパイラ，インタープリタ

などで応用されている。

言語プロセッサの授業では、まさにこの部分をいまやっています。

参考文献

- 文法:
  - 英語学概論　ー三大文法の流れと特徴ー, 松井千枝, 朝日出版(1980).

そもそも「文法」とは何か、を考える人には参考になると思います。
比較的気楽に読める本です。

ここまでのまとめ

- 言語には階層がある（Chomsky階層）
- 正規言語（正規文法）は
  字句解析に深く関わっている。
- 文脈自由言語（文脈自由文法）は
  構文解析に深く関わっている。

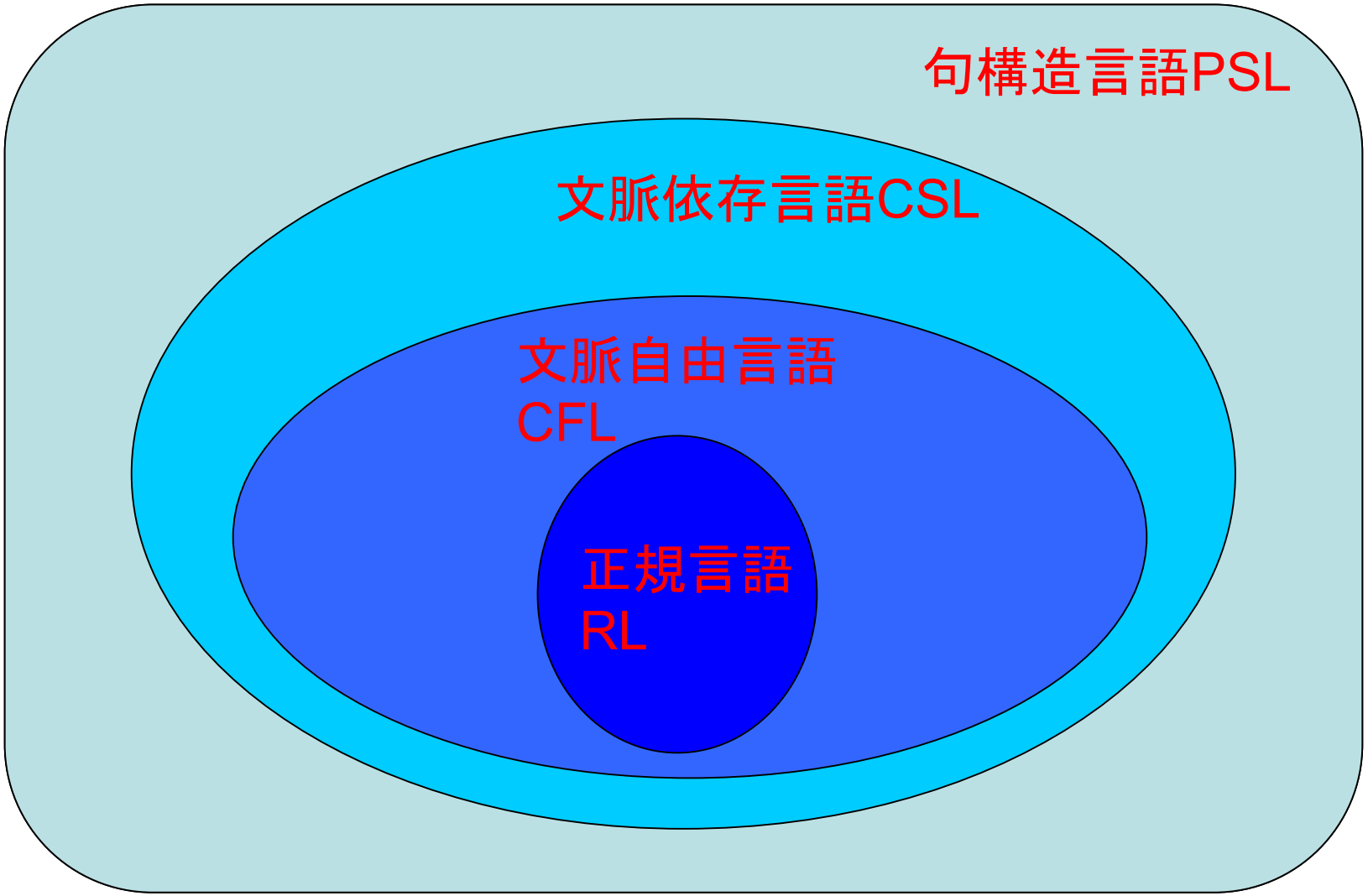
（再開）

- （以上の話、思い出したでしょうか。）

LL(1)文法の話に戻りましょう！

Chomsky階層

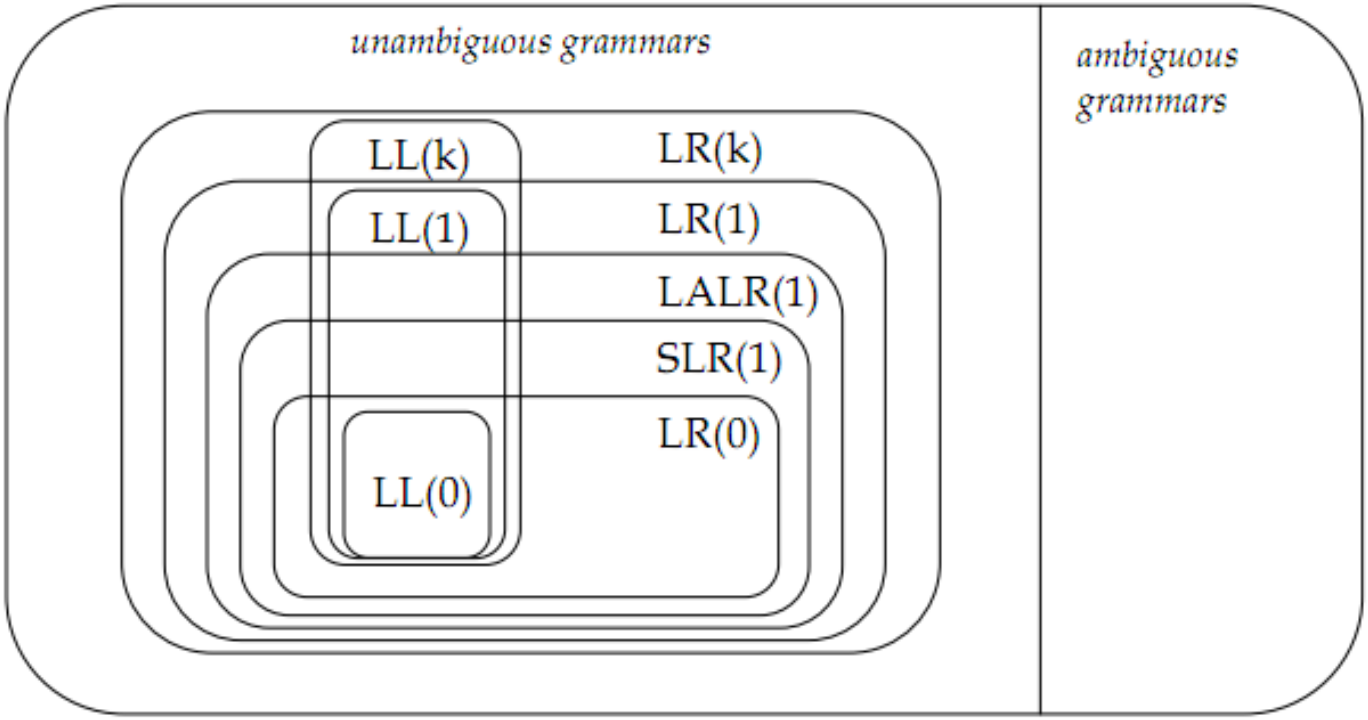
重要



LL(k)とLR(k)の関係図

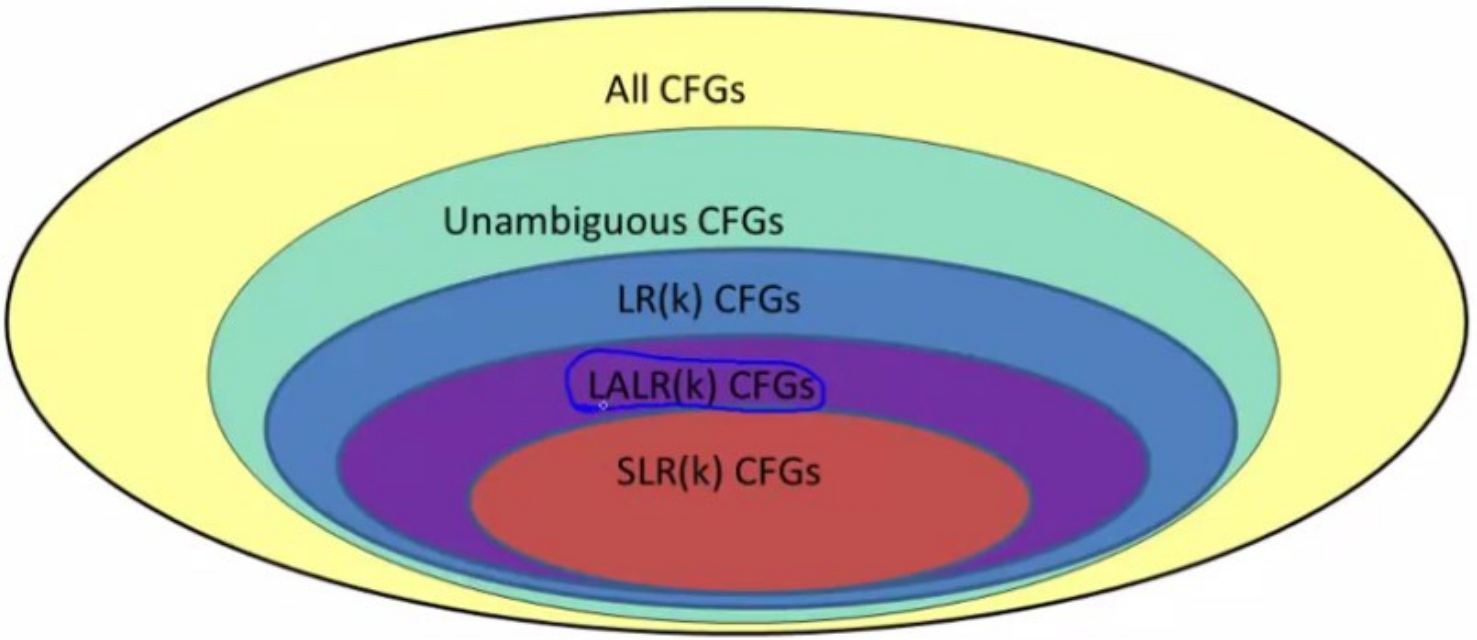
LL(1) versus LR(k)

A picture is worth a thousand words:



（出典）http://stackoverflow.com/questions/6379937/what-about-theses-grammars-and-the-minimal-parser-to-recognize-it

文脈自由文法とLR(k)の関係



（出典）http://www.wikipendium.no/TDT4205_Compiler_Technology

LL(1)文法

- LL(1)文法のイメージ:

  $A \rightarrow \alpha \mid \beta$

  という規則で、αかβのどちらの書換えを選ぶかを決めるとき、入力の先頭記号1個を見ることにより、バックトラックが起きないような選択が可能な文法。

  つまり、適応するべき文法規則を、1文字先読みすれば決定できるということ。

LL(1)文法の条件

1. 文法への制限
   - 左再帰性の除去
   - 括りだし(factoring)

2. 構文解析方法
   - Top down
   - 再帰呼び出し
   - 1文字先読み

1文字先読み十分性の条件は？

- First集合
- Follow集合
  がその条件に深くかかわっている！

（実践的観点からも重要。
ひと頑張りしてみましょう。）

First集合

【定義】

First(α)={a | a ∈ Vt, α=*=>a… }

ただし、α=*=>εならば、ε∈First(α)

First集合

【First集合を求めるアルゴリズム】

どのFirst集合にも新たに追加するものがなくなるまで、以下の作業を繰り返す。

1. First(ε)={ε}
2. First(aα)={a} if a∈Vt　　（*Vtは終端記号の集合*）
3. if(First(Y) /∋ ε)　　*Yは空文字列にならない。*
   First(Yα)= First(Y)
   else
   First(Yα)= (First(Y) ― {ε})∪ First(α)
4. if(X→α) First(X)= First(X) ∪First(α)

Follow集合

【定義】

Follow(X)={a | a ∈ Vt,　S =*=> …Xa… }

Follow集合

【Follow集合を求めるアルゴリズム】

どのFollow集合にも新たに追加するものがなくなるまで、以下の作業を繰り返す。

1. Follow(S)に$を加える。
2. 規則 A → αBβ (B∈N) に対して、
   （ア）First(β)をFollow(B)に加える。
   ただし、ε∈First(β) のときはεは加えない。
   （イ）ε∈First(β)またはβ=εならば、
   Follow(A)をFollow(B)に加える。

練習してみよう！

First集合とFollow集合（確認）

【定義】

1. First(α)={a | a ∈ Vt, α=*=>a… }

   ただし、α=*=>εならば、ε∈First(α)

2. Follow(X)={a | a ∈Vt, S =*=> …Xa… }

練習問題：First集合とFollow集合

【例】

文法G=(V, N, P, E)

P={ $E \rightarrow TE'$,

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid i$ }

教科書p.86より

求めてみよう！

- First(F)=
- First(T)=
- First(E)=
- Follow(E)=

- First(E) = First(T) = First(F) = { (, i }
- First(E') = { +, ε}
- First(T') = { *, ε}
- Follow(E) = Follow(E') = {), $}
- Follow(T) = Follow(T') = { +,), $}
- Follow(F) = { +, *,), $}

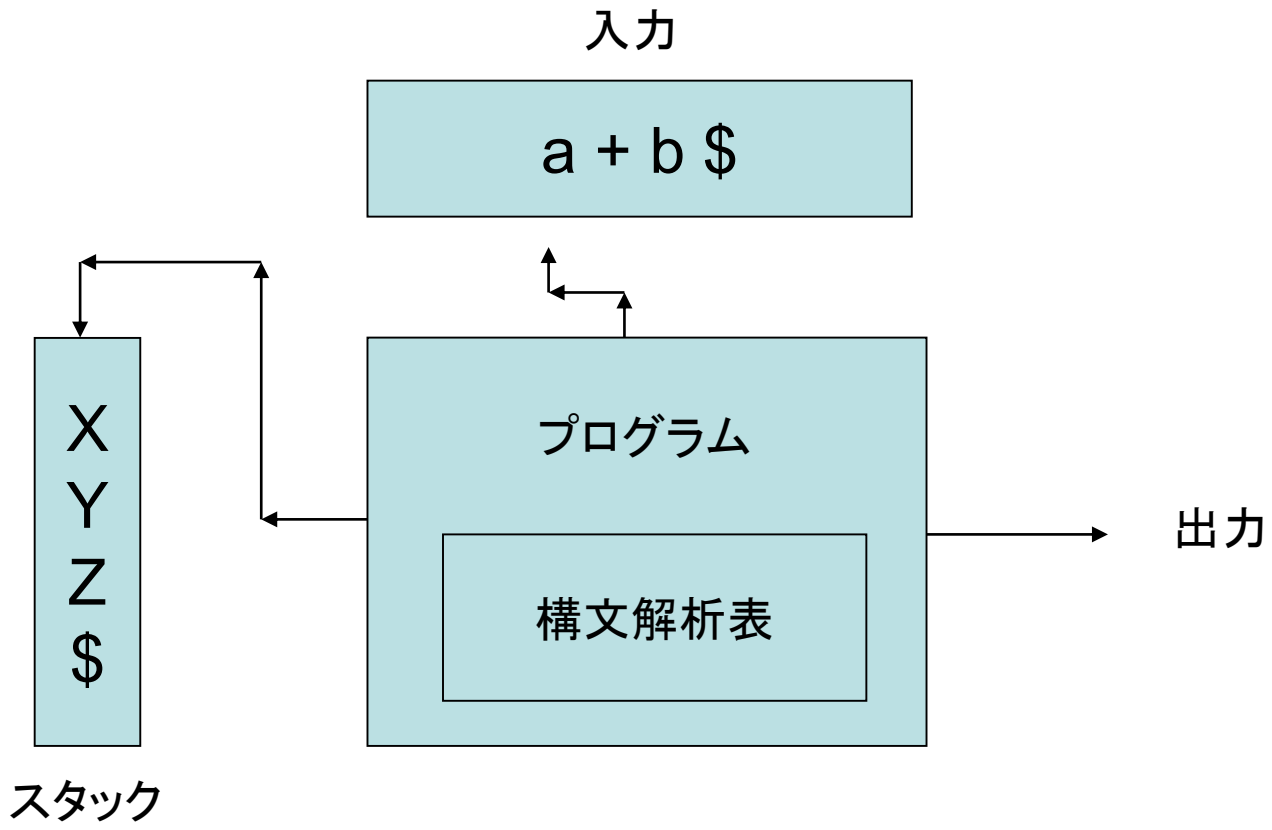
これでFirstとFollowが求まるようになった

- 「１文字先読み十分性」の条件を理解するために、もう少し先まで話をします。
- 次は、構文解析法（実際のアルゴリズム）の話しです。

構文解析表による構文解析法

- 予測的構文解析のモデル
- 構文解析表の作り方
- 構文解析のアルゴリズム

予測的構文解析のモデル



構文解析表の作り方

- 入力: 文法G
- 出力: 構文解析表M
- 手順:

1. 文法の各規則A→αに対して、ステップ2と3を行う。
2. 各終端記号a∈First(α)に対して、M[A, a]にA→αを記入する。
3. ε∈First(α)ならば、各終端記号∈Follow(A)に対して、M[A, b]にA→αを記入する。
   ε∈First(α)かつ$∈Follow(A)ならば、M[A, $]にA→αを記入する。
4. Mの未記入欄にerrorを記入する。

構文解析表

	id	+	*	(	)	$
E	E → T E'			E → T E'		
E'		E' → + T E'			E' →ε	E' →ε
T	T → F T'			T → F T'		
T'		T' →ε	T' → * F T'		T' →ε	T' →ε
F	F → id			F → (E)		

- 上記のアルゴリズムは任意の文法に対しても適用できるが、文法によってはMの欄に対して複数の規則が書き込まれることがある。

  【例】

  $P = \{S \rightarrow i\ C\ t\ S\ S' \mid a,\ S' \rightarrow e\ S \mid \varepsilon,$

  $C \rightarrow b\ \}$

  (M[S', ε] を求めてみよ。)
- LL(1)文法はこのようなことが起きない文法。

構文解析のアルゴリズム

1. X = a = $ ならば、”構文解析成功” を出力し停止。
2. X = a =!= $ ならば、スタックからXをpopし、入力ポインタを1つ進める。
3. a∈Vならば、M[X, a]を調べる。
   M[X,a]={X→ABC}ならば、C,B,Aの順にスタックにpushし、X→ABCを実行する。
   M[X,a]=errorならば、停止。

参考情報

1. 構文解析まで終われば、後は少し楽になります。
2. 構文解析は解析の中でも難関部分で、今日でも多くの研究がなされてます。
3. 構文解析の次は意味解析（解析の最終段階）。
4. それ以後は合成の段階になります。

と、いう具合に話が進むのですが、構文解析手法は次回に回します。

- その代わりに、最後にAntlrWorksの紹介をし、最後にレポート課題について説明をします。

AntlrとAntlrWorks

- Antlrはコンパイラーコンパイラーの1つ
- その発展版がAntlrWorks
  - Lex, Flex, Yacc, Bison などと同じ仲間。
  - 字句解析器生成機能と構文解析器生成機能が統合されているのが特徴。
  - 利点 :
    - Parserやコンパイラ作成に利用できる。
    - プログラミング言語の文法設計に利用できる。
    - コンパイラの動作学習にも役立てることが可能。
    - その他（楽しい？）

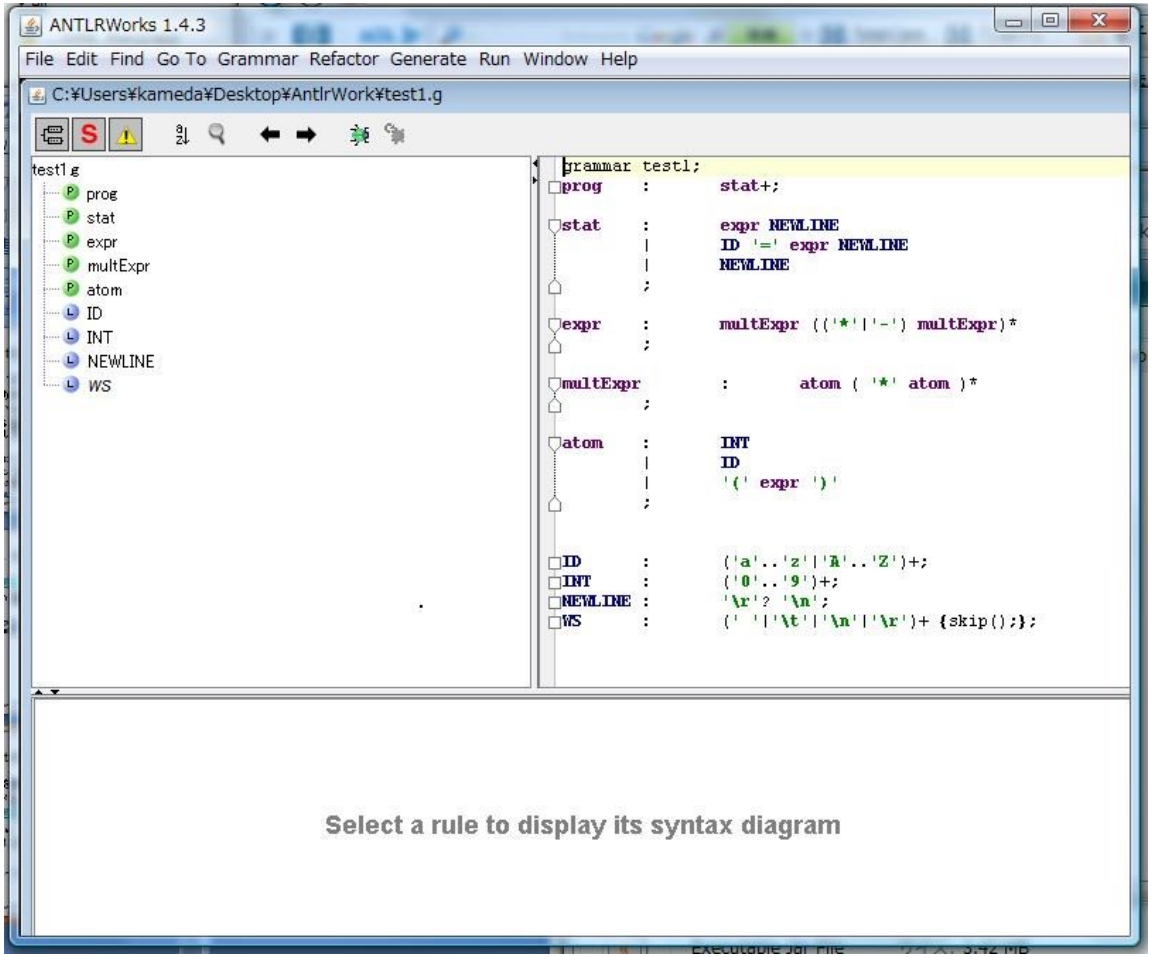
Antlr & AntlrWorksの情報

- http://www.antlr.org/
- このサイトから情報をふんだんにとりだすことができます。
  (This site is a spring, where much amounts of knowledge on parser generation.)

AntlrWorkについて

- AntlrWorksを使います。
- ダウンロードは、
  http://www.antlr.org/download.html
  からできます。
- antlrworks-1.4.3.jar をコピーします。
- 実行方法
  - Antlrworks-1.4.3.jar をダブルクリックする。

AntlrWorks起動後の画面



自主課題（antlrWorks）

- AntlrWorksを使って、
  - 統語規則の図表示の方法（表示手順）
  - デバッグ操作手順

を実際に実行しながら、その作業結果を簡単にまとめてみよう。（注）動作中にエラーが発生した場合には、その状況をメモしておくと後で役に立ちます。

（注）提出の必要はありません。

今後の予定

- 来週11月17日は休講です！
- 次回（１１月２４日）
  - First, Follow, 構文解析手順の仕上げ。
- 次々回
  - Flex と Bison を用いてのプログラミング。
    (字句解析器と構文解析器を作成する。)