文解析システム YAPXR の実現と評価†

林 達也††

本稿では，横型トップダウン方式の YAPXR 文解析システムの実現と評価について述べる．YAPXR は自然言語モデリングシステム SOLOMON の一つのコンポーネントとして位置付けられる．YAPXR が受け入れる文法機能は，従来の YAPX で許されていた拡張 CFG をさらに拡大し，文脈依存規則や文脈依存制約等を導入できるようにしている．われわれはこれを restricted context sensitive grammar (RCSG) と呼んでいる．RCSG は metamorphosis grammar, gapping grammar 等を除いて，記述能力の高い論理文法になっている．文脈依存規則は，ボトムアップ方式をとる他のシステムにおいても原理的に採り入れることができるものである．本方式を中規模英文法に対して適用した結果，記述能力と時間的効率の面で良好な prolog ベースの実用的文解析システムを構築できることを確認した．

1. まえがき

筆者は先に，論理型言語による横型トップダウン方式の構文解析法 YAP およびその拡張系 YAPX について述べた[1),2)]．また，核位置の概念を用いた効率的実現方法についても考察した[3)]．

本稿では，YAPXR (Revised version of YAPX) 文解析システムを実現し評価を行ったので，その具体的な機能と性能について述べることにする．

実現に際しては，文法機能を YAPX で許されていた拡張 CFG からさらに拡大し，文脈依存規則や文脈依存制約等を導入している．これを RCSG (restricted context sensitive grammar) と呼ぶことにする．RCSG は文献 11), 13), 14), 16), 17) を除いて，記述能力の高い論理文法になっている．

文脈依存規則は文法の記述力を強化し，文脈依存制約は解析効率の向上に有効である．これにより，記述性と性能面で実用的な文解析系を構築することが可能である．なお，文脈依存規則はボトムアップ方式をとる他のシステムでも原理的に導入することができる．

本システムは図 1 に示すように，われわれが目指している自然言語モデリングシステム SOLOMON (SOphisticated natural Language Oriented MOdeliNg system) の中で，一サブシステムとして位置付けられる．以下では，2 章で用語の定義，3 章で YAPXR で許される文法 RCSG について述べた後，4 章で実現方法，5 章で中規模英文法による適用評価について述べる．

† Implementation and Evaluation of the Sentence Analysis System YAPXR by TATSUYA HAYASHI (Fujitsu Laboratories Ltd.).

†† (株)富士通研究所

2. 用語の定義

本章では，形式言語理論で用いられる一般的用語・記法をベースとして，本稿で新たに用いる用語について定義する．

まず，文法の規則には，右辺の要素の直前にそれぞれ異なる自然数を付与しておくものとする．これを位置と呼ぶ．そして，ある要素の直前・直後の位置をそれぞれその要素の左位置・右位置と呼ぶ．ε 規則にも位置を付与するものとする．また，第 0 規則のみは特別に「b」，「e」を位置として用いて，$s_0 \rightarrow {}^{b}s^{e}\dashv$ とする．

2.1 最左導出位置

ある規則の右辺の要素を $A(\in V_N)$ とし，その左位置を n とする．そのとき，

$A \Rightarrow B\gamma$, $(B \in V,\ \gamma \in V^*)$

で，B の左位置が m ならば，m を n の最左導出位置と呼び，$m \in LD(n)$ で表す．

2.2 位置集合

$A(\in V_N)$ をある規則の右辺の左端でない要素とし，左位置を n とする．このとき，集合 $\{n\} \cup LD(n)$ を A の位置集合と呼ぶ．ただし，$A(\in V_T)$ に対しては，$\{n\}$ をもって A の位置集合とする．また例外として，第 0 規則の左端要素 s に対してのみ位置集合を認める．

2.3 核位置

位置集合においてある元を n とし他の任意の元を x としたとき，$x \in LD(n)$ ならば n をその集合の核位置と呼ぶ．

この定義により，規則右辺の左端要素の左位置は，前述した「b」を例外として，核位置にはなれないことが容易にわかる．なお，核位置は位置集合の id と

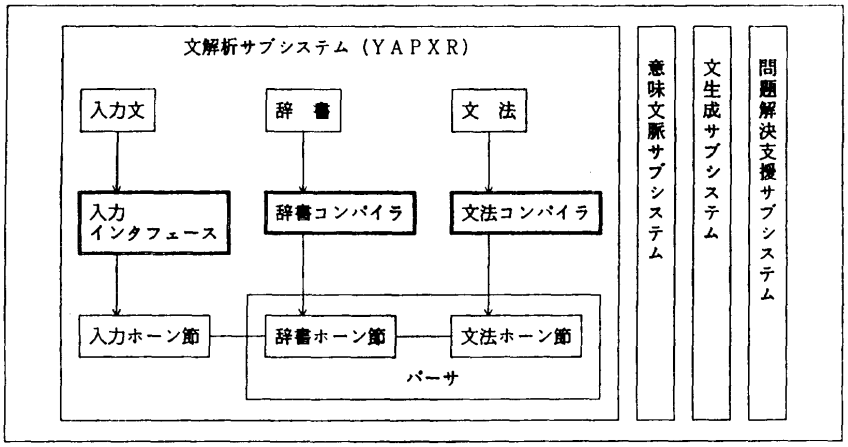

図 1 自然言語モデリングシステム SOLOMON
Fig. 1 Natural language modeling system SOLOMON.

みなすことができる．

2.4 解析パス

入力文を $a_1 \cdots a_n$ とする．入力部分列を $a_1 \cdots a_i (1 \leq i \leq n)$ とし，それを含む最左導出列の一つを

$$s_0 \Rightarrow a_1 \cdots a_i \alpha, (\alpha \in V^*)$$

とする．これに対してまず $s_0 \underset{pos}{\Rightarrow} \text{bse}\dashv$ とし，以下規則の適用を位置付きで行った導出

$$s_0 \underset{pos}{\Rightarrow} k_1 a_1 \cdots k_i a_i n_{i+1} \alpha'$$

を考える．$k_j (1 \leq j \leq i)$ は 1 個以上の位置の列である．ここでまず a_1 の直後の位置（k_2 の先頭，n_2 とする）に注目し，n_2 と同一規則上の位置を左方（この場合は k_1 だけ）へ走査しながら求め n_1 とする．n_1 と n_2 の間に他の位置が存在すれば対応する部分木を生成し，順序集合へ格納する．そして，n_1 を含めて n_1 から n_2 の間の位置および a_1 を（n_2 だけ残して），位置付き導出から削除する．これを順次 $a_2 \sim a_i$ について繰り返す*．

そして最後に，最左導出位置をすべて除去する．

この結果最終的に，位置付き最左導出は，

$$bn_{i1}n_{i2} \cdots n_{im}n_{i+1}\alpha'[\tau_1, \cdots, \tau_l],$$
$$0 \leq im \leq i-1,\ 1 \leq l \leq i$$

となる．$[\tau_1, \cdots, \tau_l]$ は部分木の順序集合である．この時，$[b, \cdots, n_{i+1}][\tau_1, \cdots, \tau_l]$ を入力部分列 $a_1 \cdots a_i$ に対する解析パスと呼ぶ．ただし通常は，$[b, \cdots, n_{i+1}]$ の部分だけを便宜上解析パスと呼ぶ場合もある．

定義から明らかなように，解析パスは核位置の列から成り，第 0 規則から始まりどの規則をどの順に適用して入力部分列 $a_1 \cdots a_i$ が得られたか，また，各規則のどの位置まで対応する（と思われる）部分列が検出されたかを示している．すなわち，解析パスはいわば最左導出の縮少表現であり，かつ異なる最左導出に対しては，得られる解析パスも異なる*．そして逆に，解析パスから対応する最左導出を求めることができる．

2.5 内部引数

規則の左辺および右辺の要素 A に対して，$A(IX)$ のようにかっこで括って引数を付与することができるが，これを内部引数と呼ぶ．

内部引数は，要素 A に対応する部分入力列を何らかの意味で特徴づける属性として用いられる．

引数の値は単語辞書から直接間接に得られる場合と，規則中で直接値を設定される場合の 2 通りある．

2.6 外部引数

要素 B の内部引数 EX が同一規則に出現する他の要素 A を定義する規則中で直接間接に参照される場合は，A の内部引数（もしあれば）の後に（/と/）で EX を括って $A(Y)(/EX/)$ のように示す．

これを外部引数と呼ぶ．外部引数は内部引数のように要素に固有ではなく，要素の出現に依存して数や種類が異なってよい．外部引数を用いることによって，外部情報を参照することができる**．

外部引数は後述するように，解析パスの早期絞り込みに有効である．また本稿では触れないが，トレース（痕跡）の文法情報，意味情報の伝達等にも利用される．

---

* 導出に ε 規則が用いられている場合には，ε を入力単語の一種とみなして，これに準ずる処理を行えばよい．

* 厳密には，複数の最左導出が同一の解析パスに対応する場合があるが，それらを互いに区別する必要はない．

** 別のやり方で外部情報の参照を可能にしているシステムが存在する[13),18)]．

3. 文法機能

本システムで許される文法を RCSG (Restricted CSG) と呼ぶ. RCSG は従来 YAPX で許されていた拡張 CFG, すなわち引数, 条件式 (ここでは制約と呼ぶ), スラッシュカテゴリ等を含む文法形式[2]を基本とすることに変りはないが, その他に実用上便利でかつ実現が容易なトレース対象外指定, 文脈依存規則および文脈依存制約が, 新たに追加機能として導入されている. 以下, RCSG について順を追って説明する.

3.1 基本形式

基本形式は,

左辺 → 右辺

ここで,

左辺: 非終端要素 [(内部引数, …)][(外部引数, …)]

右辺: ε or 右辺項…

右辺項: 構文要素 [(内部引数, …)][(外部引数, …)]
[スラッシュカテゴリ]
[{基本制約}…]

スラッシュカテゴリ[2]: /(σ) or //B/(σ),
$B \in V_N$, σ は「~」と $x(\in V)$ の列

基本制約: 引数に関する論理式

である. 呼応処理や意味処理を行う場合には, スラッシュ σ にも引数を与える必要があるが, 基本的構文処理主体の本稿では説明を省略する.

内部引数の値を直接設定したい場合は, 引数名の後に値をかっこで括って表す.

RCSG では, 右辺第1項にスラッシュカテゴリを記述できる (ex. ncomptx→ncompt/(~np)).

さらに, 右辺第1項がスラッシュカテゴリに対応して, トレースになることも差し支えない.

3.2 トレース対象外指定

これは他システム[9]ですでに採り入れられているが, RCSG においても導入している.

すなわち, 右辺の要素をトレースにはしないという場合には, 矢印→の代わりに⇒を用いて陽にその旨を示す (ex. sdecol⇒subj advx vp, pp⇒[p] np).

3.3 文脈依存規則

文脈依存規則は次の形式を持つ. ただし簡単のため, 引数や制約を省略して示す.

(a) {[¬] [( ] α, …[ )]} B [{[¬] [( ] α′, … [ )]}]→β,
$\exists p(A_p \to \mu\langle n\rangle A\delta)$, $A \Rightarrow B\eta$, $\mu, \beta, \delta, \eta \in V^*$,
n: 核位置ラベル

(b) {[¬]⟨n, …⟩} B→β,
$\exists p(A_p \to \mu\langle n\rangle A\delta)$, $A \Rightarrow B\eta$, $\mu, \beta, \delta, \eta \in V^*$,
n: 核位置ラベル

ここで,「¬」は否定を表す. また α, α′ は, 一つの規則の右辺に完全に含まれていなければならない. (a) と (b) は表現形式が異なるのみで機能的には同等である. B が n を左位置として持つ場合, あるいは n を左位置に持つ A から最左導出される場合, B→β を適用できる (否定のときはできない) ことを示す.

例を示すと,

np→detq ncompt/(~np) (1)

ncompt→srel (2)

{¬detq} srel→subj advx vp (3)

srel→[relpro] sdecol (4)

と指定することで, (1) の ncompt から最左導出される srel に対しては (4) は良いが (3) の適用を禁止することができる. これにより例えば, 部分入力列「many more books」に対して,「many」と「more books」あるいは「many more」と「books」をそれぞれ detq, ncompt から導出されたものと解釈する解析パスを排除することができる.

文脈依存規則は一般の文脈依存文法に比べて極めて制限の強いものであるが, この導入により実用上は, 上に見たように, 非終端要素や規則の不必要な増大を防止し, 文法を適正な規模に保つのに有効である. また, これはボトムアップ方式のパーサでも利用することができる.

3.4 文脈依存制約

一般に構文的条件は引数と基本制約を用いて指定することができる. 例えば,

sdecol⇒subj aux advx vp (FLEX)/(~obj)
{FLEX=en} ppx (5)

sdecol⇒subj advx vp (FLEX) {FLEX≠en} (6)

で, (5) (受動文の規則) の vp は FLEX (活用) が en (過去分詞) でなければならないことを示している. また, (b) は逆に, FLEX が en であってはならないことを示す. 次に,

vp(FLEX)→[vt](CATX, FLEX){CATX=vo}
obj (7)

は, vt (他動詞) の CATX (品詞小区分) が vo 型であるべきことを示す. そしてまた vt の FLEX 属性は vp の属性として継承される.

ところでここで,

sdecol⇒subj aux advx ⟨n 1⟩ vp/(～obj) ppx (8)

sdecol⇒subj advx ⟨n 2⟩ vp (9)

vp→[vt] (CATX, FLEX) {⟨n 1⟩ : FLEX=en, ⟨n 2⟩ : FLEX≠en} {CATX=vo} obj (10)

としても,(8)～(10)は(5)～(7)と機能的には同等の制約を表している.ここで n 1, n 2 は核位置ラベルである.違いは処理効率で,後者の場合に vt を読み込んだ時点で直ちに FLEX のチェックを行う.したがって,当該核位置の下で,解析パスの早期絞り込みを行うことができる.

(10)に現れるような核位置を伴った制約を,文脈依存制約と呼ぶ.文脈依存制約は $A(X)\Rightarrow[T](X)\alpha$ のように,非終端要素 A の引数が最左導出される終端要素の引数を継承している場合有効である*.

文脈依存制約の一般形式は次のとおりである.

{文脈制約,…},

ここで,文脈制約:⟨核位置ラベル,…⟩:基本制約

文脈依存制約は通常の制約部の前に記述するものとする.

次に,外部引数を参照する文脈依存制約の例を以下に示す.

sdecol⇒subj advx auxd (CATX 1) advx ⟨n 3⟩ vp (/CATX 1/) (11)

vp(/CATX 1/)→[vt] (CATX, FLEX) {⟨n 3⟩ : FLEX≠en∨CATX 1=have} {CATX=vo} obj (12)

この場合(11)の vp (/CATX 1/) に対して,要素 vp が CATX 1 を外部引数として宣言するといい,vp を定義する規則(12)に対して,外部引数 CATX 1 を参照している規則という.

さて,本システムは LR 手法をベースにしているが,state[8] そのものは用いていない.state の中味は一般に YAPXR における1個以上の位置集合の和集合と等価である.したがって,state は1個以上の核位置と対応する.ところで上述した基本制約,スラッシュカテゴリ,文脈依存規則および文脈依存制約は,個々の核位置と結び付いている.そこで,state そのままの形では,一般にこれらの機能を導入することはできないからである.また,state の id は恣意的なものなので,state の列から対応する導出を直観的に摑むことは一般に困難である.さらにまた,本システムでは規則当り1個の核位置のみしか解析スタックに格納しない方式を採っている.

* 文脈依存制約はユーザが直接指定しなくても,処理系(文法コンパイラ)の最適化処理によって基本制約から自動的に変換することが可能である.

4. 実現方法

YAPXR システムはすでに触れたように,文解析,意味文脈,文生成等を統合する自然言語モデリングシステム SOLOMON のコンポーネントの一つである.

YAPXR 実現の基本的メカニズムについては別に述べた通りで[1)~3)],左回帰要素や右辺第1項の要素の払い等は同じである.ここではしたがって,異なる部分について説明する.

説明に先立って,構文要素に対応する述語で用いられる引数を以下に示す.

n: 解析スタック(入力)の先頭要素
A_i: 解析スタック(入力)の残り
A_0, A_{01}: 解析スタック(出力,差分リスト)
O_i: 省略スタック(入力)
O_0, O_{01}: 省略スタック(出力,差分リスト)
L_i: 引数スタック(入力)
L_0, L_{01}: 引数スタック(出力,差分リスト)
T_i: 解析木スタック(入力)
T_0, T_{01}: 解析木スタック(出力,差分リスト)

ただし,煩雑を避けるため以下では,必要な引数のみを差分リスト,カット記号を省略して示す場合がある.

4.1 ε およびトレース処理の最適化

述語呼出しのオーバヘッドを減らすため,ε 規則やスラッシュ適用の際は当該述語を呼び出すホーン節を生成する代りに,たたみ込みを行い,解析パスを直接解析・解析木スタックに出力するようにホーン節を変更する.例えば,

sdecol→subj [10]bep [11]advx [12]pred

において,advx が ε 規則を持つ場合,bep のホーン節を,

bep (10, A, [[11|A]|[12|A]], T, [T|[advx(ep)|T]]).

とする.ε 規則を持つ要素が規則中に連続して出現する場合も,対応する複数個の解析パスを出力する.

スラッシュの適用に関しても,省略スタックをさらに用いる他は上記と同様にして行えばよい.ただし,トレースとなる要素が引数を持つ時は,省略スタック

の値を引数スタックにセットする.

4.2 文脈依存規則の処理

処理系は文脈依存規則が3.3節に示したものである限り，文法が与えられた時点であらかじめ，各核位置ごとに文脈依存規則の適用の可否を調べながら，位置集合を求めることができる．したがって，位置集合の要素に対応して必要なホーン節を生成できることはYAPX[2),3)]の場合と同様である．逆に言えば，YAPXRは文解析に対して何らの時間的空間的効率低下をもたらさない範囲で文脈依存規則を導入しているわけである.

4.3 引数と制約

引数（内部，外部）および制約の処理のために，解析スタックを兼用することは可能であるが，煩雑を避けるため本システムでは，引数スタックを用いることにしている.

まず内部引数は，述語が呼ばれた時点では引数スタックの先頭に存在するように処理される．次にその内部引数が規則の右方で参照されていれば，引数スタックにそのまま残し，そうでなければ引数スタックから削除する．引数の参照の有無を判断する際，規則左辺の要素は右辺の最後に位置するものとみなす.

次に外部引数であるが，これを宣言している（要素に対応する）述語が呼ばれた時点では，その内部引数，外部引数の順に引数スタック上部に格納されている.

これに対して，外部引数を参照している規則を適用する場合には，左辺の内部引数がもし指定されていたとしても，その内部引数はまだスタックにはセットされておらず，外部引数のみが右辺冒頭で引数スタックの上部に存在するものとして処理される.

処理系は引数スタック中のこれらの引数（内部，外部）を用いて，制約の置かれている直前の要素に対応するホーン節の中で，制約の評価を行うようにコードを生成する.

例えば，3.4 節の(12)で述語 vt が呼び出された時点では，CATX, FLEX, CATX 1 が引数スタックの第1～第3要素になっている．そこで(12)の制約は，

vt ($n3, A,$ [CATX, FLEX, CATX 1$|L$],
[[CATX 1$|L$]]) :-
FLEX$\neq$en$\vee$CATX 1=have, CATX=vo.

により正しく評価される．そして，CATX, FLEX はその後不要なので削除される.

なお，CATX 1 もやはり必要とされないが，便宜上スタックの先頭に残したままとし，(11)によるリダクションで述語 sdecol が呼ばれる時点で，スタックから削除される.

このようにして処理系は，外部・内部引数および制約を引数スタックを用いて統一的に管理している.

4.4 辞書および入力文の処理

YAPXR 文解析システムは図1に示すように，文法コンパイラ，辞書コンパイラ，入力インタフェイスから構成される．文法コンパイラは上述した処理を行い，RCSG を文法ホーン節に変換する.

辞書コンパイラは単語辞書を辞書ホーン節に変換する．現システムは構文処理が主目的なので，各単語には必要最少限の属性しか与えていない.

すなわち，エントリの構成は次のとおりである.

単語見出し，品詞，品詞小区分 (CATX)，活用 (FLEX) 1品詞1エントリなので，多品詞語の場合はマルチエントリとなる.

辞書コンパイラにより生成されるホーン節の形式を以下に示す.

$a00$ ($A_i, A_0, O_i, O_0, L_i, L_0, T_i, T_0, a$) :- !,
t_1_0 ($A_i, A_0, A_{01}, O_i, O_0, O_{01}, L_i, L_0, L_{01}, T_i,$
T_0, T_{01}, a, [catx 1, flex 1]),
t_2_0 ($A_i, A_{01}, A_{02}, O_i, O_{01}, O_{02}, L_i, L_{01}, L_{02},$
T_i, T_{01}, T_{02}, a, [catx 2, flex 2]),
⋮
A_{0n}=[ ], O_{0n}=[ ], L_{0n}=[ ], T_{0n}=[ ].
t_j_0 ([[$n|A_h$]$|A_i$], A_{0j-1}, A_{0j+1}, [$O_h|O_i$], $O_{0j-1},$
O_{0j+1}, [$L_h|L_i$], L_{0j-1}, L_{0j+1}, [$T_h|T_i$],
T_{0j-1}, T_{0j+1}, a, [catx j, flex j]) :- !,
t_j_01 ($n, Ah, A_{0j-1}, A_{0j}, O_h, O_{0j-1}, O_{0j},$
[catx j, flex $j|L_h$], L_{0j-1}, L_{0j}, [$t_j(a)|$
T_h], T_{0j-1}, T_{0j}), !,
t_j_0 ($A_i, A_{0j}, A_{0j+1}, O_i, O_{0j}, O_{0j+1}, L_i, L_{0j},$
$L_{0j+1}, T_i, T_{0j}, T_{0j+1}, a$, [catx j,
flex j]).
t_j_0 ([ ], A_{01}, A_{01}, −, O_{01}, O_{01}, −, L_{01}, L_{01},
−, T_{01}, T_{01}).

ここで，a～単語

t_j, catx j, flex j～ 品詞と属性値,

t_j_01～ 文法ホーン節を呼び出す述語,

である.

次に入力インタフェイスは，入力文 $a_1 \cdots a_n$ を以下のような入力ホーン節に変換する.

yapx :- yopen (A_i1, O_i1, L_i1, T_i1),

$a_1$00 (A_i1, A_i2, O_i1, O_i2, L_i1, L_i2, T_i1, T_i2, a_1),
$a_2$00 (A_i2, A_i3, O_i2, O_i3, L_i2, L_i3, T_i2, T_i3, a_2),
⋮
yclose (A_{in+1}, O_{in+1}, L_{in+1}, T_{in+1}).

ただし,

yopen : 各スタックに初期値（文法に依存）を設定するシステムに固有の述語,

yclose : 解析木を出力するシステムに固有の述語,

$a_1$00～a_n00 : 辞書ホーン節を呼び出す，入力文に対応した述語列,

である.

5. 適用評価

5.1 文法規模

ここでは，本システムを中規模英文法[9]に対して適用した結果について述べる[4].

上記文法はスラッシュカテゴリ（右辺第1項を除く）を導入した形式で記述されており，これを RCSG の特色を生かして書き直した場合の規則の数や種類は**表1**のようになる．文法規則の総数 299 は上記原文法[9]の 430 規則，それからスラッシュなしの場合の約 550 規則に相当する*.

表 1 RCSG の中規模英文法への適用例
Table 1 Characteristics of RCSG description of the medium scale English grammar.

項　　目	規則数
規則の総数	299
終端記号	28
非終端記号	75
ε 規則	18
左回帰則	23
スラッシュシンボル	7
スラッシュ則	21
右辺第1項のスラッシュ則	2
トレース対象外規則	241
文脈依存規則	24
基本制約規則	17
文脈依存制約規則	35
外部引数宣言	4
外部引数参照	10

表 2 解析時間と作業メモリ量
Table 2 Parsing time and working memory size.

番号	単語数	解析木（文献値）	最終パス数	解析時間 (sec) I モード	解析時間 (sec) C モード	作業メモリ量 (byte) I モード	作業メモリ量 (byte) C モード
1	7	2 (2)	32	0.416	0.067	151,800	21,168
2	8	1 (1)	22	0.216	0.083	85,904	11,828
3	6	1 (1)	35	0.183	0.033	62,508	10,912
4	4	2 (2)	60	0.200	0.033	67,616	11,296
5	8	2 (1)	144	0.933	0.133	310,396	43,776
6	8	1 (2)	128	1.066	0.200	411,800	51,452
7	6	1 (1)	36	0.366	0.050	140,220	16,820
8	11	1 (1)	37	0.516	0.100	173,522	24,136
9	7	1 (1)	23	0.134	0.033	59,492	9,128
10	8	4 (2)	159	0.900	0.117	312,248	46,064
11	8	2 (1)	44	0.584	0.083	195,844	26,712
12	5	1 (1)	23	0.200	0.017	67,280	9,584
13	8	2 (2)	97	0.518	0.100	201,880	31,736
14	7	2 (2)	30	0.267	0.034	104,444	16,268
15	11	1 (1)	33	0.300	0.083	107,712	12,892
16	5	1 (1)	65	0.184	0.066	65,824	10,592
17	11	1 (1)	37	0.567	0.100	205,844	26,224
18	10	1 (1)	34	0.317	0.100	147,796	21,156
19	7	1 (1)	23	0.250	0.050	87,852	14,096
20	9	4 (4)	191	1.184	0.200	423,916	60,868
21	4	1 (1)	28	0.200	0.033	65,484	7,964
22	9	2 (3)	47	0.333	0.050	123,408	15,984
23	8	1 (1)	68	0.617	0.117	246,064	31,100
24	8	1 (1)	167	0.650	0.134	252,684	40,724
25	23	14 (8)	630	2.650	0.483	1,010,624	165,260
26	6	2 (2)	71	0.617	0.034	81,196	14,624
27	7	1 (1)	30	0.283	0.083	96,524	15,604
28	9	1 (1)	18	0.250	0.050	110,276	15,900
29	10	1 (1)	53	0.450	0.083	157,344	21,968
30	11	2 (2)	80	0.350	0.067	145,640	23,632
31	18	5 (1)	256	1.100	0.300	358,276	67,372
32	16	5 (3)	278	1.717	0.317	652,844	99,076
33	22	10 (7)	997	19.916	1.217	1,222,456	325,036
34	19	4 (1)	183	1.100	0.200	398,892	64,592
35	20	1 (4)	56	2.417	0.433	881,960	111,768
36	25	21 (1)	1250	24.067	1.000	1,287,256	346,936
37	24	2 (12)	232	4.722	0.783	1,684,076	220,952
38	28	3 (4)	174	4.183	0.767	1,576,252	224,376
39	33	2 (1)	213	3.150	0.616	1,239,396	173,888
40	4	1 (1)	23	0.084	0.017	36,368	6,020
41	5	1 (1)	21	0.183	0.017	57,500	8,608

原文法ではスラッシュカテゴリの導入により規則数が減少しているが，RCSG においてはさらに ε 規則，右辺第1項のスラッシュカテゴリおよび文脈依存規則の導入が，規則数を一層減少させる上で効果を見せている.

5.2 解析時間および所要スペース

YAPXR によって生成された，上記の英文法に対するパーサに，例文を入力して解析時間および作業メモリ量を集計した結果を**表2**に示す.

評価環境は以下のとおりである.

- 計算機　SUN 3/60
- 言語　Quintus prolog R 2.4
- 文法規則　299
- 辞書　180 エントリ
- 例文　41 文（付録参照）
- 評価項目

（1） 最終パス数～入力文走査終了時の解析パス

* 文献 9) ではかっこを用いて複数の規則をまとめる略式表現が可能で，それによると規則数は 314 となる．ただし，略式表現はどのシステムにおいても導入することができるものである.

表 3 位置集合のサイズ別分布
Table 3 Size distribution of the position sets.

サイズ	1	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17	18	25	27	33
集合数	73	64	15	15	3	9	1	4	1	4	5	1	17	7	1	2	10	1	18	2
サイズ	35	36	37	38	39	41	43	48	51	52	53	56	60	62	74	84	103	113	115	171
集合数	25	25	8	6	1	3	1	3	2	3	6	10	2	11	1	4	1	25	3	4

集合の総数 377，集合の平均サイズ 26

数（含解析木）

（2） 解析時間～コンパイル／インタプリティブモードによる全解析木探索時間．ただし，ガーベジコレクション，スタックシフトの時間を除く．

（3） 作業メモリ量～グローバルスタックの使用量

なお，現在のバージョンではεやトレース処理の最適化を行っていない．また，制約を持つホーン節が本来不要なバックトラックを行うようになっている．

作業メモリ量に関する他システムの報告はないが，解析時間に関しては例文 1～25 についての報告がある[5]．また，例文 26～41 について一部を除き，3種の報告がある[6),7),10)]．前者は測定値に形態素解析時間を含みかつ，処理系は C prolog インタプリタという工合に測定条件の相違が大きいので比較は無理である．

後者も，(1) Quintus prolog のリリース番号，(2) カバーする言語の規模，(3)使用計算機が異なるので正確な比較はできない．しかし，リリース番号の相違は発売元によると特に性能改善を目的としていない．また文法規則数から見て，上記報告がカバーする言語の規模は本システムと同等かより小さいと思われる．つまり，条件は本システムの方がより厳しくなっている．

そこでハードの相違は性能を換算して補正することにして*，一応比較の価値はありそうである．

本システムの解析時間を1として各々のケースにつき比率を求めると，インタプリティブモードで 11～85，コンパイルモードで 1.2～23 という工合にいずれも1より大きい値が得られる．

しかし，前述したように測定条件が異なるので，これらの値は厳密な比較ではなくあくまでも参考として提示したものであることを断っておく．

* SUN 3/260, 3/60, 3/160, VAX 11/785 の性能をそれぞれ，4 MIPS, 3 MIPS, 2 MIPS, 1.5 MIPS として換算する．

表 4 中規模英文法に対するプログラムサイズ
Table 4 Size of the program generated from the medium scale English grammar.

	原形式	ホーン節	コンパイルコード
文法	299 規則	9135 個	1216 KB
辞書	180 エントリ	196 個	41 KB

本システムが良好な性能を持つ理由としては，

（1） 限定文脈依存文法（RCSG）の高い記述力により，規則の集合がコンパクトになる，

（2） 横型の解析方式を採っている，

（3） 文法が与えられた時点で，右辺第1項の対応する核位置をあらかじめ求めることができるので，ホーン節で陽にその核位置を用いれば良く，実行時にトップダウン予測を行う必要がない，

（4） 実行時にバックトラックを（本来）行わない，

（5） 文脈依存制約による解析パスの早期絞り込み，

（6） Quintus prolog のダブルハッシュによる高速実行に適合するべく，述語の第1引数をアトム（核位置）にしている，

などが考えられる．

一方スペース効率であるが，まず位置集合の要素数別分布を**表 3** に示す．要素数の平均は 26 なので，核位置方式は大まかにいって，作業メモリで約 1/25，解析速度で約 10 倍強の効果を持つわけである．

しかしながらプログラムサイズの方は，一般に左端要素に対応する核位置が複数個存在するため，増加傾向となる．上記英文法の場合には**表 4** のようになる．

位置集合方式と比べて，ホーン節の数で約 10 倍に増加している．

他のシステムと比較しても，スペース効率は下回る傾向にあるものと思われる．

その代り，文脈依存制約や外部引数の活用により，時間的効率の一層の向上が原理的に可能である．

また，大規模アドレス空間は現在容易に得られるので，スペースよりも時間の方が重要な因子であるとも

いえる.

6. あとがき

われわれは，制限付き文脈依存文法 RCSG を受け入れる横型トップダウンの文解析システム YAPXR を開発した．本論文では中規模英文法を対象にとり上げて適用評価を行い，機能および時間的効率の面で良好な結果が得られたことを示した．

現在，システムは基本部分ができ上った段階であり，最適化処理や文法開発支援のための良好なマンマシン・インタフェイスが整備されておらず，これが今後の課題である．また，英文法も例文に関係する部分しかチューニングされていないので，文法全体の整備が今後の課題として残されている．

さらに文解析だけに止まらずに，自然言語モデリングシステム SOLOMON の実現を目指して，意味文脈や文生成のサブシステムについても現在開発に取り組んでおり，別途報告する予定である．

謝辞 日頃いろいろと助言していただく田中穂積氏（東工大教授）に感謝する．また，システムの開発と評価に従事していただいた宮俊司，坂巻利哉，吉田健一（富士通 SSL）の諸氏に厚くお礼を申し上げる．

参考文献

1) 林 達也：論理型言語による構文解析法 YAP について，情報処理学会論文誌，Vol. 29, No. 9, pp. 835-842 (1988).
2) 林 達也：拡張 CFG とその構文解析法 YAPX について，情報処理学会論文誌，Vol. 29, No. 5, pp. 480-487 (1988).
3) 林 達也：YAPX の効果的実現法，情報処理学会論文誌，Vol. 30, No. 10, pp. 1354-1356 (1989).
4) 林 達也，宮 俊司，坂巻利哉，吉田健一：横型トップダウン文解析システムの実現と評価，情報処理学会自然言語処理研究会，No. 74-9, pp. 65-72 (1989).
5) 田中穂積，上脇 正，奥村 学，沼崎浩明：自然言語処理の為のソフトウェアシステム Lang LAB, *Proc. LPC 86*, pp. 5-12 (1986).
6) 杉村領一：ロジックプログラミングをベースにした自然言語解析システムの比較，情報処理学会自然言語処理研究会，Vol. 86, No. 60, p. 8 (1987).
7) Okunishi, T., Sugimura, R., Matsumoto, Y., Tamura, N., Kamiwaki, T. and Tanaka, H.: Comparison of Logic Programming Based Natural Language Parsing Systems, *Natural Language Understanding and Logic Programming*, Vol. 2, pp. 1-14, North-Holland, Amsterdam (1988).
8) Aho, A. V. and Ullman, J. D.: *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, Englewood Cliffs (1972).
9) 東工大田中研究室：中規模英文法（仮称）(1987).
10) 沼崎浩明，田村直良，田中穂積：並列論理型言語による一般化 LR 構文解析アルゴリズムの実現，情報処理学会自然言語処理研究会，No. 74-5, pp. 33-40 (1989).
11) Colmerauer, A.: *Metamorphosis Grammars, Natural Language Communication with Computers*, Springer, Berlin (1978).
12) Pereira, F. C. N. and Warren, D. H.: Definite Clause Grammars for Language Analysis, *Artif. Intell.*, Vol. 13, pp. 231-278 (1980).
13) Dahl, V. and Abramson, H.: On Gapping Grammars, *Proc. 2nd International Conference on Logic Programming*, pp. 77-88 (1984).
14) Dahl, V.: More on Gapping Grammars, *Proc. International Conference on Fifth Generation Computer Systems*, pp. 669-677 (1984).
15) 奥村 学，田中穂積：BUP 系解析システム上でのトップダウンな情報の制御について，情報処理学会論文誌，Vol. 29, No. 11, pp. 1043-1050 (1988).
16) 林 達也，宮 俊司，坂巻利哉，吉田健一：句構造文法に対する効率的文解析手法，情報処理学会自然言語処理研究会，Vol. 76-2, p. 8 (1990).
17) 林 達也：句構造文法に対する効率的解析手法，情報処理学会論文誌（投稿中）(1989).

付録 例文リスト
Appendix List of sample sentences.

1. There are three on the table now.
2. There are some men here from the city.
3. Those are the books she read.
4. Were they given her.
5. Could she have been given many of them.
6. I saw many more books than she did.
7. I saw that they were there.
8. The books that were on the table are difficult to read.
9. My uncle gave the girl several books.
10. She was given some books by her uncle.
11. The books were given her by her uncle.
12. He gave up her the books.
13. You could break this vase with that hammar.
14. That hammar could break this vase easily.
15. Did not those people want him to try to do it.
16. Are there any more left.
17. Is this any harder for him to do than that was.
18. This is a film that is developed in the research.
19. The system for interpreting dialogues is developed.
20. She was given more difficult books by her uncle.
21. Do not do that.
22. Be very careful not to do it too quickly.

23. Read these books, if you have time.
24. If you have time, read these books.
25. This paper presents an explanatory overview of a large and complex grammar that is used, in a computer system for interpreting English dialogues.
26. He explains the example and he illustrates the rule.
27. The structural relations are holding among constituents.
28. He explained the example and he illustrates the rule.
29. It is not tied to a particular domain of applications.
30. Diagram analyzes all of the basic kinds of phrases and sentences.
31. This paper presents an explanatory overview of a large and complex grammar that is used in a sentence.
32. Diagram analyzes all of the basic kinds of phrases and sentences and many quite complex ones.
33. The annotations provide important information for other parts of the system that interpret the expression, in the context of a dialogue.
34. For every expression it analyzes, diagram provides an annotated description of the structural relations holding among its constituents.
35. Procedures can also assign scores to an analysis, rating some applications of a rule as probable or as unlikely.
36. This paper presents an explanatory overview of a large and complex grammar, diagram that is used, in a computer for interpreting English dialogue.
37. Its procedures allow phrases to inherit attributes from their constituents and to acquire attributes from the larger phrases in which they themselves are constituents.
38. Consequently, when these attributes are used to set context sensitive constraints on the acceptance of an analysis, the contextual constrains can be imposed by conditions on constituents.
39. It is not tied to a particular domain of applications, and it can be extended to analyze additional constructions, using the formalism in which it is currently written in the rule.
40. I open the window.
41. Diagram is an augmented grammar.

(平成元年8月21日受付)
(平成2年3月 6 日採録)

林　　達也（正会員）

1937年生．1960年早稲田大学第一理工学部応用物理学科卒業．同年富士通(株)入社．以来，ソフトウェア工学（コンパイラ自動作成，設計支援，部品化），データベース（リレーショナル，分散，マルチメディア，オブジェクト指向），自然言語処理（機械翻訳，自然言語インタフェイス，情報検索），人工知能（エキスパートシェル，知識表現）等の研究開発に従事．富士通研究所情報処理研究部長代，ソフトウェア研究部長を経て現在川崎研究所所属．元本学会編集幹事．1973年度本学会論文賞受賞．1984年度日経最優秀製品賞（団体）受賞．著書「電子計算機のシステムプログラム」(産報)他．ACM，日本ソフトウェア科学会，日本モーツァルト協会各会員．AAI (Applied Artificial Intelligence) (Hemisphere) Editorial Board メンバ．