論　文

新しい LL(k) パーサとその最小化問題の計算量*

徳田雄洋** 井上謙蔵**

Abstract

The LL(k) parsers, constructed by either Lewis-Stearns' algorithm or Wood's algorithm, present a delay in detecting errors. In this paper, we first describe an algorithm for the construction of LL(k) parsers which always detect errors at the earliest stage of the analysis. Then an algorithm for minimizing the number of "states" of an LL(k) parser is described on condition that error detection may be delayed. The minimization problem itself, however, is NP-complete in a more general setting.

1. はじめに

トップダウン型構文解析手法は，その直接性の故に構文則向きコンパイリングの概念誕生とほぼ同時に出現した．例えば Irons による最初の構文則向きコンパイラ[10]，あるいは Brooker らによる最古のコンパイラ・コンパイラ[4]においては，それぞれトップダウン的手法が部分的，全面的に採用されている．しかしながら，トップダウン型構文解析が，recursive descentのような場合を除けば，ボトムアップ型構文解析に比べ，コンパイラの動作そのものに対する密着性に乏しいため，トップダウン型の構文解析可能な文法に関する研究は 60 年代後半になって開始された．Korenjak と Hopcroft による "S文法"[14] あるいは Knuth による特殊なトップダウン型文法の研究[13]が初期の例である．やがて，1968 年に Lewis と Stearns[18] により，左 (L) から右への走査で，左 (L) 解析列を，入力列に対する長さ k の先読みを行うのみで，後戻りなしに構成しうる文法クラスが考案され，LL(k) 文法と呼ばれた***．

LL(k) 文法は，標準的な意味では，トップダウン型で構文解析可能な最大の文法クラスであり，ボトムアップ型の LR(k) 文法に比べると，その表現能力はかなり劣る．しかしながら LL(k) 文法は，プログラム言語の構文則や意味処理の記述に際し，直感に訴える力が大きく，LL(1) 文法に基づく FORTRAN や ALGOL のコンパイラ[9]も実際に作成され，その有効性も確認されている．

決定問題に関連する研究を除くと，従来までの LL(k) 文法に関する研究は 2 つに大別できる．

第 1 は LL(k) 文法の縮少または拡大に関するもので，一般の LL(k) 文法に比べ小型の構文解析表の構成可能な "強 LL(k) 文法"[26] (Wood, 1969)，左回帰記号の出現も許す ALL(k) 文法[27] (吉村, 1975)，あるいは **if-then** と **if-then-else** がひきおこす "あいまい性" をも許す 2 種類の拡張 LL(k) 文法[2] (Aho 他, 1975) などがこの例である．

第 2 は LL(k) 条件の判定法に関するもので，文法のサイズと終端記号数の積の手数で LL(1) 条件を判定する方法[11] (Johnson 他, 1975)，一般の k に関して，文法のサイズ n に対し，$O(n^{k+2})$ で LL(k) 条件を判定する方法[9] (Hunt 他, 1975) などが報告されている (Johnson の方法では bit vector 操作が許されると，$O(n)$ で LL(1) 条件を判定する)．

本論文では LL(k) 文法のために従来から用いられてきた 2 つの LL(k) パーサ，Lewis-Stearns によるもの，Wood によるものが，いずれも誤りの早期検出に関し不完全であることを指摘し，常に最も早期に誤りを検出する LL(k) パーサの構成法を示す．次にこ

---

* New LL(k) Parsers and the Computational Complexity of their Minimization Problems by Takehiro TOKUDA and Kenzo INOUE (Department of Information Science, Tokyo Institute of Technology).

** 東京工業大学理学部情報科学科

*** 初期の稿では，Top-Down の頭文字を取って TD(k) 文法と呼ばれていた (文献3)．

のパーサは，誤り検出の遅延が許されるならば小型化を行うことが可能なこと，小型化により“強 LL(k) 文法”の場合は，“don't care”の項目を除き，Wood のパーサと一致することを示す．最後に，この小型化アルゴリズムは一種の組合せ問題を中に含んでしまうが，実際の LL(k) パーサの全体よりも少し広い対象の上で解釈すると，この組合せ問題は NP-完全であること，すなわちパーサのサイズの多項式時間内に問題を解くアルゴリズムが存在しないと予想されているクラスの問題と，計算量的には等価であることを示す．

なお，EI Djabri[6] (1973) は本研究のアプローチとは独立に，Lewis-Stearns パーサの“don't care”項目の同定，行の重ね合せ，列の重ね合せに，よるパーサの小型化法を提案しており，本論文のアルゴリズム2と本質的には一致する．

2. 従来の LL(k) パーサ

2.1 記法と定義

文脈自由文法 $G=(N, \Sigma, P, S)$ に対し，以下の関数，演算および定義を用いる[3]．ただし，N, Σ, P はそれぞれ非終端記号，終端記号，規則の集合で，S は開始記号である．ギリシア文字は終端・非終端記号の長さ0以上の列，英大文字は非終端記号，英小文字前半は終端記号，英小文字後半は終端記号の列を，それぞれ特に断わらない限り示すものとする．なお$\Rightarrow$は1回の導出，$\overset{*}{\Rightarrow}$は0回以上の導出を表わし，また特に lm を添えた場合は，その導出が最左導出であることを示すものとする．なおεは空列を示す．

定義 2.1 $\mathrm{FIRST}_k(\alpha)=\{x \mid \alpha \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} x\beta$ で $|x|=k$，あるいは $\alpha \overset{*}{\Rightarrow} x$ で $|x|<k\}$.
ただし，$|x|$ は列 x の長さを示す．

定義 2.2 $\Sigma^{*k}=\{x \mid x\in\Sigma^*, |x|\leq k\}$.

定義 2.3 $L_1 \oplus_k L_2=\{w \mid x\in L_1, y\in L_2$ に対し，$|xy|\leq k$ の時は $w=xy$, $|xy|>k$ の時は w は xy の長さ k の先頭部分列$\}$.
ただし，$L_1, L_2 \subset \Sigma^{*k}$ とする．

定義 2.4 $\mathrm{FOLLOW}_k(\alpha)=\{w \mid S \overset{*}{\Rightarrow} \gamma\alpha\delta, w\in \mathrm{FIRST}_k(\delta)\}$.

定義 2.5 文脈自由文法（以下，文法と呼ぶ）Gに対し，ある整数 k が存在して，次の条件(1)～(3)を満たす任意の2つの最左導出に関し，いつも $\beta=\gamma$ が成立する時，G を LL(k) であるという[18),3)].

(1) $S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} wA\delta \underset{\mathrm{lm}}{\Rightarrow} w\beta\delta \overset{*}{\Rightarrow} wx$,

(2) $S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} wA\delta \underset{\mathrm{lm}}{\Rightarrow} w\gamma\delta \overset{*}{\Rightarrow} wy$,

(3) $\mathrm{FIRST}_k(x)=\mathrm{FIRST}_k(y)$.

定義 2.6 文法 G に対し，ある整数 k が存在して，次の条件 (1)～(3) を満たす任意の2つの最左導出に関し，いつも $\beta=\gamma$ が成立する時，G を強 LL(k) であるという[24),6)].

(1) $S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} wA\delta \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} w\beta\delta \overset{*}{\Rightarrow} wx$,

(2) $S \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} w'A\delta' \overset{*}{\underset{\mathrm{lm}}{\Rightarrow}} w'\gamma\delta' \overset{*}{\Rightarrow} w'y$,

(3) $\mathrm{FIRST}_k(x)=\mathrm{FIRST}_k(y)$.

2.2 従来の構成アルゴリズム

一般の LL(k) 文法の構文解析表の構成法および強 LL(k) 文法の場合の構成法を簡単に述べる．

LL(k) 文法の構文解析法は，プッシュダウン・スタック1本と LL(k) 構文解析表1個を用いる k-predictive 法に基づいている (**Fig. 1** 参照)．パーサの動作は，スタック先頭1記号と走査入力列の長さ k の先読み列によって決定され，4種類の動作が可能である．動作指定が“**pop**”の時は，スタックの先頭1記号を捨て，入力列の走査位置を1つ進める．“**error**”の時は，入力列に誤りが発見されたことを告げ，解析を打ち切る．“**accept**”は入力列の構文解析が無事終了したことを示す．これら以外の時は，スタックの先頭1記号を，解析表の項目に記入してある記号列と置換し，併記されている規則番号を出力する．なお初期設定時に，スタックには特別な1記号（一般 LL(k) の時は $[S:\{\varepsilon\}]$，強 LL(k) の時は S）のみが置かれ，走査位置は入力列の先頭に設定されている．

k-predictive 法のポイントは，入力列を長さ k だけ先読みしながら，スタック内で開始記号 S からの最左導出をシミュレートする点であり，スタック内の記号列はいずれの場合も，本質的には非終端記号と終端

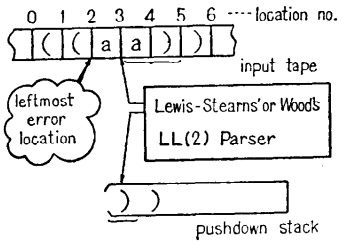

Fig. 1 Delay of error detection in the previous LL(2) parsers.

記号からなる左文形式の postfix である点に注意されたい.

以下, スタックの底を示す記号として¥を用いる.

[Lewis-Stearns の構成法]*

入力: LL(k) 文法 $G=(N, \Sigma, P, S)$.

出力: G のための LL(k) 構文解析表 M.

Ⅰ. 状態の枚挙

(1) 状態集合 $\mathcal{T}$ を, 状態 $[S:\{\varepsilon\}]$ のみを要素とする集合とする.

(2) $\mathcal{T}$ の中のそれぞれの状態 $[A:L]$ ($A\in N$, $L\subset\Sigma^{*k}$) について, 左辺が A である各々の規則 $A\to x_0B_1x_1B_2x_2\cdots B_mx_m$ ($B_i\in N$, $x_i\in\Sigma^*$, $1\leq i\leq m$, $0\leq l\leq m$) に対し, 次のような局所的後続先読み列集合 Y_i ($1\leq i\leq m$) を算出する.

$$Y_i=\mathrm{FIRST}_k(x_i\cdots B_mx_m)\oplus_k L$$

そして, 状態 $[B_1:Y_1]$, $[B_2:Y_2]$, …, $[B_m, Y_m]$ の中で $\mathcal{T}$ に所属していない状態のみを, $\mathcal{T}$ につけ加える.

(3) ステップ(2)を, 集合 $\mathcal{T}$ に新しい状態がつけ加えられなくなるまで繰り返す.

Ⅱ. 構文解析表 M の構成

$((N\times\Sigma^{*k})\cup\Sigma\cup\{¥\})\times\Sigma^{*k}$ の上で, 次のような構文解析表 M を定義する.

(1) $A\to x_0B_1x_1B_2x_2\cdots B_mx_m$ が j 番目の規則で, しかも $[A:L]\in\mathcal{T}$ の時, 先読み列 $u\in\mathrm{FIRST}_k(x_0\cdot B_1x_1B_2x_2\cdots B_mx_m)\oplus_k L$ に対し, 次のように定義する.

$$M([A:L], u)=\text{“}x_0[B_1:Y_1]x_1[B_2:Y_2]x_2\cdots[B_m:Y_m]x_m, j\text{”}.$$

なお, $m=0$ の場合は次のように定義する.

$$M([A:L], u)=\text{“}x_0, j\text{”}.$$

(2) $M(a, av)=$“**pop**”. ただし, すべての $v\in\Sigma^{*(k-1)}$.

(3) $M(¥, \varepsilon)=$“**accept**”.

(4) (1)～(3)以外の項目は“**error**”とする. ☒

[Wood の方法]

入力: 強 LL(k) 文法 $G=(N, \Sigma, P, S)$.

出力: G のための $(N\cup\Sigma\cup\{¥\})\times\Sigma^{*k}$ 上の構文解析表 M.

(1) $A\to\alpha$ が j 番目の規則ならば, 先読み列 $u\in\mathrm{FIRST}_k(\alpha)\oplus_k\mathrm{FOLLOW}_k(A)$ に対し, 次のように定義する.

$$M(A, u)=\text{“}\alpha, j\text{”}.$$

(2) $M(a, av)=$“**pop**”. ただし, すべての $v\in\Sigma^{*(k-1)}$.

(3) $M(¥, \varepsilon)=$“**accept**”.

(4) (1)～(3)以外の項目は“**error**”とする. ☒

2.3 従来のパーサの欠陥

Lewis-Stearns のパーサおよび Wood のパーサは, いずれも入力列中の誤りに対する早期検出能力が不十分であることを示す.

まず誤り検出位置を測る尺度として, LL(k) 文法のための入力列に関する“最左誤り位置”を定義する.

定義 2.7 LL(k) 文法 G のもとで, 与えられた入力列 xy($x, y\in\Sigma^*$) に対し, 次の関係を満たさない最短の x が存在する時, x と y の間の位置を (G と k に関する) xy の最左誤り位置であるという.

$$[S\underset{\mathrm{lm}}{\overset{*}{\Rightarrow}}x\alpha \text{ かつ } \mathrm{FIRST}_k(y)\in\mathrm{FIRST}_k(\alpha)].$$

すなわち最左誤り位置とは, 入力列の prefix である x は文法 G のある文の prefix となりうるが, 後続入力列はその x には後続しえない. 最短の x の, 右隣りの位置のことである. 定義から明らかなように, 最左誤り位置は与えられた入力列に対し, 高々1ヵ所しか存在しない. また, この最左誤り位置の定義は Levy の correct prefix property の概念[17]より強い点に注意されたい.

実例 1 次のような LL(2) 文法に対し,

$$S\to T \qquad T\to(T)$$
$$S\to aa \qquad T\to(a)$$

誤りを含む入力列“((aa))”を与える. Lewis-Stearns のパーサも Wood のパーサも, 位置3 (Fig. 1) で, すなわち先読み列が“a)”である時, 誤りの存在を検出する. しかしながら最左誤り位置は2であり, 両パーサとも誤りの検出が1位置右へずれてから行われている (この場合, 先読み列“aa”は, その左に左括弧が1つでも先行する時は存在しえず, スタックの中身が $[S:\varepsilon]$ (または S)の時のみ存在しうる.).

両パーサの誤り検出が遅延してしまう欠陥は次の理由による. すなわち両パーサとも“**pop**”動作を行う時, 先読み列の情報とは全く無関係に, スタック先頭1記号と先読み列の最左1記号との機械的マッチングに頼っているためである. したがって, 局所的に後続しうる先読み列の情報も終端記号に貯えてスタックにしまうことにすれば, “高級”なマッチングが実現できる. そしてこのマッチングを用いれば, 常に一番早い段階で誤りの検出を行うパーサが作れることになる.

---

* 後述するアルゴリズム1と形式を合わせるためオリジナルの形を変形してあるが, 本質的には同一である.

3. 誤り検出の完全な LL(k) パーサ

3.1 構成アルゴリズム

常に最左誤り位置で誤りの検出を行う構文解析表の構成法を示す.

[アルゴリズム 1]

入力: LL(k) 文法 $G=(N, \Sigma, P, S)$.

出力: G のための $((N\cup\Sigma\cup\{¥\})\times\Sigma^{*k})\times\Sigma^{*k}$ 上の構文解析表 M.

Ⅰ. 状態の枚挙

(1) 状態集合 $\mathcal{T}$ を, $\mathcal{T}=\{[S:\{\varepsilon\}]\}$ とする.

(2) $\mathcal{T}$ の中のそれぞれの状態 $[A:L]$ $(A\in N, L\subset\Sigma^{*k})$ について, 左辺が A である各々の規則 $A\to a_0B_1a_1B_2a_2\cdots B_ma_m$ $(B_i\in N\cup\{\varepsilon\}, a_i\in\Sigma\cup\{\varepsilon\}, 1\le i\le m, 0\le l\le m)$ に対し, 次のような局所的後続先読み列集合 Y_i $(1\le i\le m)$ を算出する.

$$Y_i=\mathrm{FIRST}_k(a_iB_{i+1}a_{i+1}\cdots B_ma_m)\oplus_k L.$$

そして, 状態 $[a_0:Y_0], [B_1:Y_1], [a_1:Y_1], \cdots, [B_m:Y_m], [a_m:Y_m]$ の中で, 左のフィールドが ε でなく, $\mathcal{T}$ 中にまだ含まれていない状態を, $\mathcal{T}$ につけ加える.

(3) ステップ(2)を, 集合 $\mathcal{T}$ に新しい状態が加えられなくなるまで繰り返す.

Ⅱ. 構文解析表 M の構成

(1) $A\to a_0B_1a_1B_2a_2\cdots B_ma_m$ が j 番目の規則で, しかも $[A:L]\in\mathcal{T}$ の時, 先読み列 $u\in\mathrm{FIRST}_k(a_0B_1a_1B_2a_2\cdots B_ma_m)\oplus_k L$ に対し, 次のように定義する.

$$M([A:L], u)=\text{“}[a_0:Y_0][B_1:Y_1]\cdots[B_m:Y_m][a_m:Y_m], j\text{”}.$$

ただし, $[\varepsilon:Y]=\varepsilon$ とする.

(2) 状態 $[a:L]\in\mathcal{T}(a\in\Sigma)$ の時, すべての $u\in L$ に対し次のように定義する.

$$M([a:L], u)=\text{“}\mathbf{pop}\text{”}$$

(3) $M([¥:\{\varepsilon\}], \varepsilon)=$“**accept**”

(4) (1)〜(3)以外の項目は“**error**”とする. ☒

3.2 基本的性質

3.1 節で示された LL(k) パーサは, 常に入力列の最左誤り位置で誤りの検出を行うこと, Lewis-Stearns のパーサはこのパーサに比べ, 誤りの検出位置が高々 $k-1$ 位置 $(k\ge1)$ 右へずれることを示す.

定理 3.1 アルゴリズム 1 によって得られるパーサは正しい LL(k) パーサであり, しかも常に最左誤り位置で誤りの検出を行う.

証明 以下の性質が構文解析過程の任意の時点(“**accept**”を除く)で成立することを示せば十分である.

スタック内の記号列 $(V_i\in N\cup\Sigma, 1\le i\le m)$ を,

$$[V_1, L_1][V_2, L_2]\cdots[V_m, L_m]$$

とすると, 以下の i)および ii)が成り立つ.

i) $V_1V_2\cdots V_m$ はある正しい(開始記号 S からの)左文形式の postfix である.

ii) $V_i\in\Sigma$ なら, $L_i=\mathrm{FIRST}_k(V_i\cdots V_m)$.
$V_i\in N$ なら, $i+1\le m$ の時,

$$L_i=\mathrm{FIRST}_k(V_{i+1}\cdots V_m),$$

$i=m$ の時, $L_i=\{\varepsilon\}$.

証明は最左誤りのステップ数に関する帰納法による. ☒

定理 3.2 Lewis-Stearns 法に基づく LL(k) パーサの誤り検出位置は, 入力列の最左誤り位置に比べ, 高々 $k-1$ 位置 $(k\ge1)$ 右へずれる.

証明 入力列 xy の x と y の間を最左誤り位置とする. すなわち, $k\ge1$ の時で次が成立している.

$$S\underset{\mathrm{lm}}{\overset{*}{\Rightarrow}}x\alpha \text{ で } \mathrm{FIRST}_k(y)\notin\mathrm{FIRST}_k(\alpha).$$

Lewis-Stearns のパーサでは, “**pop**”動作のみが誤りの検出を遅らせることができる. したがって“**pop**”が k 回以上起こったとすると, y と α の先頭 k 個の終端記号が一致してしまい. $\mathrm{FIRST}_k(y)\in\mathrm{FIRST}_k(\alpha)$ となる. これは x と y の間の位置が最左誤り位置であったことに反する. ☒

系 3.2 LL(1) 文法のための Lewis-Stearns パーサは, 常に入力列の最左誤り位置で誤りの検出を行う.

Wood のパーサに関しては, “**pop**”動作以外にも, 規則選択動作で誤りを“混入”させることができるため, 誤りの遅延度を $k-1$ 位置で抑えることはできない. 例えば, LL(1) 文法のための Wood パーサで, 誤り検出が最左誤り位置より右へ 1 位置ずれる例を作ることができる.

かくして, 常に最左誤り位置で誤り検出を行う LL(k) パーサが得られた (**Fig. 2**(次頁参照)参照. ただし空白項目は“**error**”を示す. なお状態の記号中で“{”と“}”は省略してある). この“完全”なパーサは, 従来の LL(k) パーサに比べ, 誤り検出が厳格であり, k-predictive アルゴリズムの下で, 最も早期に誤りの宣言を行うことができる. しかしながら, 従来のパーサに比べ“状態”数が多いことが欠点である. そこで次節で, この完全なパーサの小型化を試みる.

	aa	((	(*a*	*a*	*a*)	)	))	ε	otherwise
[*S* : ε] [*T* : ε] [*T* :)] [*T* :))]	[*a* : *aa*][*a* : *a*],1	[*T* : ε],2 [(: ((][*T* :)][) :)],3 [(: ((][*T* :))][) :))],3 [(: ((][*T* :))][) :))],3	[*T* : ε],2 [(: (*a*][*a* : *a*)][) :)],4 [(: (*a*][*a* : *a*)][) :))],4 [(: (*a*][*a* : *a*)][) :))],4						
[*a* : *aa*] [*a* : *a*] [*a* : *a*)] [(: ((] [(: (*a*] [) :)] [) :))] [¥ : ε]	pop	pop	pop	pop	pop	pop	pop	accept	

(1) $S \to aa$ (2) $S \to T$ (3) $T \to (T)$ (4) $T \to (a)$

Fig. 2 Complete LL(2) parser for example 1.

4. 完全な LL(k) パーサの小型化問題

4.1 小型化アルゴリズム

誤り検出の遅延が許されれば，完全な LL(k) パーサを小型化することが可能なことを示す．なお，アルゴリズム2は完全な LL(k) パーサから出発して，次々と正しい LL(k) パーサに適用可能な形式になっている点に注意されたい．

[アルゴリズム2]

入力：正しい LL(k) 構文解析表 M.

出力：正しい LL(k) 構文解析表 $\tilde{M}$.

(1) 次のような条件を満たす行 $[V : L_1]$, $[V : L_2], \cdots, [V : L_n]$ を表 M から見つける(ただし $V \in N \cup \Sigma$).

すべての i, j $(1 \leq i < j \leq n)$ とすべての $u \in \Sigma^{*k}$ に対し，$M([V : L_i], u) \neq$ **"error"** かつ $M([V : L_j], u) \neq$ **"error"** の時，記入項目内の同一位置にある2記号について次のⅰ)またはⅱ)が成立する．

ⅰ) 記号として，または規則番号として等しい．

ⅱ) 記号として等しくはないが，いずれも記号 $[V : L_1], [V : L_2], \cdots, [V : L_n]$ のどれかである．

なお，適当な行が見つからない時は，$\tilde{M} = M$ として本アルゴリズムは終了する．

(2) 表 M の行を，行 $[V : L_1], [V : L_2], \cdots, [V : L_n]$ を除き，表 $\tilde{M}$ に全部コピーする．表 $\tilde{M}$ に新しい行 $[V : L_1 \cup L_2 \cup \cdots \cup L_n]$ をつけ加える．ただし，$M([V : L_1], u) = \cdots = M([V : L_n], u) =$ **"error"** の時は，$\tilde{M}([V : L_1 \cup L_2 \cup \cdots \cup L_n], u) =$ **"error"**．これ以外の時は，$\tilde{M}([V : L_1 \cup L_2 \cup \cdots \cup L_n], u)$ の値は，$M([V : L_1], u), \cdots, M([V : L_n], u)$ のうちの非**"error"** 項目の値とする．次に，表 $\tilde{M}$ 中に出現する記号 $[V : L_1], [V : L_2], \cdots, [V : L_n]$ をすべて記号 $[V : L_1 \cup L_2 \cup \cdots \cup L_n]$ と置き換える． ☒

定理 4.1 アルゴリズム2は，正しい LL(k) パーサ M を正しい LL(k) パーサ $\tilde{M}$ に変形する．

証明 スタック内では，常に最左導出のシミュレーションが行われていることに着目すればよい．省略． ☒

定理4.1により，完全 LL(k) パーサ M の小型化問題を次のように整理することができる．

[LL(k) パーサの小型化問題]

[1] 行 $[a : L_1], [a : L_2], \cdots, [a : L_n]$ は1つの行 $[a, L_1 \cup L_2 \cup \cdots \cup L_n]$ に重ね合すことができる．

[2] 行 $[A : L_1], [A : L_2], \cdots, [A : L_n]$ は以下の条件を満たす時，1つの行 $[A : L_1 \cup L_2 \cup \cdots \cup L_n]$ に重ね合すことができる．

すべての i, j $(1 \leq i < j \leq n)$ とすべての $u \in \Sigma^{*}_{k}$ に対し，$M([A : L_i], u) \neq$ **"error"** かつ $M[A : L_j], u) \neq$ **"error"** ならば，次の(1)および(2)が成立する．

(1) $M([A : L_i], u], M([A : L_j], u)$ 内の同一位置にある記号 $[V : Y], [V' : Y']$ $(V, V' \in N \cup \Sigma, Y, Y' \in \Sigma^{*k})$ は，次の i)～iii) のいずれかの関係を満たす．

ⅰ) $V = V'$, $Y = Y'$.

ⅱ) 記号 $[V : Y], [V' : Y']$ は，記号 $[A : L_1], [A : L_2], \cdots, [A : L_n]$ のいずれかである．

ⅲ) 行 $[V : Y]$ と行 $[V' : Y']$ は一行に重ね合すことができる．

(2) 指定している規則の番号が等しい． ☒

Fig. 2 の構文解析表を小型化した例を **Fig. 3**, **Fig. 4** (次頁参照)に示す．Fig. 3 は完全な LL(k) パーサの **"pop"** 動作を重ね合せたものである．このように終端記号を左フィールドに持つ状態のみを重ね合せると，Lewis-Stearns パーサの類似版が得られるが，類

	aa	((	(*a*	*a*	*a)*	)	))	ε	otherwise
[S:ε] [T:ε] [T:)] [T:))]	[a:∗][a:∗],1	[T:ε],2 [(:∗][T:)][):∗],3 [(:∗][T:))][):∗],3 [(:∗][T:))][):∗],3	[T:ε],2 [(:∗][a:∗][):∗],4 [(:∗][a:∗][):∗],4 [(:∗][a:∗][):∗],4						
[a:aa,a,a)] [(:((,(a] [):),)),)] [¥:ε]	pop	pop	pop	pop	pop	pop	pop	accept	

Note: [a:∗]=[a:aa,a,a)], [(:∗]=[(:((,(a], [):∗]=[):),))]

Fig. 3 Minimized LL(2) parser for example 1.

	aa	((	(*a*	*a*	*a)*	)	))	ε	otherwise
[S:ε] [T:ε,),,))]	[a:∗][a:∗],1	[T:∗],2 [(:∗][T:∗][):∗],3	[T:∗],2 [(:∗][a:∗][):∗],4						
[a:aa,a,a)] [(:((,(a] [):),)),)] [¥:ε]	pop	pop	pop	pop	pop	pop	pop	accept	

Note: [T:∗]=[T:ε,),,))], [a:∗]=[a:aa,a,a)], [(:∗]=[(:((,(a], [):∗]=[):),))]

Fig. 4 Minimal LL(2) parser for example 1.

似版パーサの方が Lewis-Stearns パーサよりも "**pop**" 動作が忠実になっている点に注意されたい．すなわち Lewis-Stearns パーサでは，大域的に存在しない先読み列に対しても "**pop**" 動作が指定されているのに対し，我々の類似版では，大域的に存在しうる先読み列のみに "**pop**" 動作が指定されている．

さらに，非終端記号を左フィールドに持つ状態についてアルゴリズム2を適用すると，Fig. 4 のようなパーサが得られる．この Wood のパーサと類似した(例文法は実は強 LL(2) 文法である) パーサに関しても，Lewis-Stearns パーサの場合と同一の意味で，本来の Wood パーサより "**pop**" 動作が忠実になっている．

4.2 最小状態数の評価

完全な LL(k) パーサをアルゴリズム2により，最小状態数 (行数) のパーサに小型化することは，一般には困難であるが(5節参照)，容易に最小状態数が評価できる場合も存在する．

定理 4.2 LL(1) 文法および強 LL(k) 文法のための最小状態数パーサは $|\Sigma|+|N|+1$ 状態である．

証明 省略. ☒

なお，他の場合の最小状態数評価については文献 25 を参照されたい．

5. 最小化問題の手数

5.1 NP-完全性

4節で与えられた小型化問題に対し，最小状態数パーサを求めるアルゴリズムの手数を，間接的に評価する．そのため，Cook に負う NP-完全性[5]について簡単に説明する．

定義 5.1 言語 $L\subset\{0,1\}^*$ に対し，ある多項式 $P(\cdot)$と整数 n_0 が存在して，任意の長さ $n(n\geq n_0)$ の L の文が与えられても単テープチューリング機械 M により $P(n)$ ステップ以内で認識できる時，言語 L は機械 M により多項式時間で認識できるという．

定義 5.2 決定性単テープチューリング機械，非決定性単テープチューリング機械によって，多項式時間で認識できる $\{0,1\}$ 上の言語のクラスをそれぞれ $\boldsymbol{P}$, $\boldsymbol{NP}$ と呼ぶ．

定義 5.3 言語 L_0 は次の条件 i), ii) を満たす時，NP-完全であるという．

i) $L_0\in\boldsymbol{NP}$,

ii) L_0 を多項式時間で認識する決定性単テープチューリング機械 M_0 が与えられれば，$\boldsymbol{NP}$に属する任意の言語 L に対し，L を多項式時間で認識する決定性単テープチューリング機械 M_L を効果的に構成することができる．

定義 5.4 問題 Q を，"標準的" な符号化により，$\{0,1\}$ 上の言語 L_Q の認識問題に直した時，L_Q が NP-完全ならば，問題 Q を NP-完全な問題と呼ぶ．

現在までのところ，NP-完全な問題に対して，問題を解く決定性多項式時間アルゴリズムの例も，また本質的に指数関数時間を要する例も見つかっていないた

め，$NP \stackrel{?}{=} P$ という問題は未解決である[1),8)]．

Cook[5)] は初めて“連言標準形の充足可能性問題”が NP-完全であることを示したが，その後 Karp[12)] は Cook の定義の第2の条件を強めて，任意の NP の問題が，適当な決定性多項式時間アルゴリズムにより，その問題に変形可能である，としても，非常に広範囲の組み合せ問題，例えば“巡回セールスマン問題”，“ナップザック問題”，“グラフ彩色問題”，“整数集合の等和分割問題”などが完全であることを示した．Karp の意味*で NP-完全であることが知られている他の例としては，例えば次のようなものがある**．タスクの最適スケジューリング問題 (Karp,Ullman)，順編成ファイルの最適レコード配置問題 (Karp, Pratt)，デッドロックの回避問題（奥井他），LR(k) 状態遷移表型テーブルの最小重ね合せ問題（徳田），GOTO 文最小プログラム問題（西関他）．

なお以上の議論は，アルゴリズムを実現する装置を他の機械，例えば多テープチューリング機械，多ヘッドチューリング機械，かけ算命令のないランダムアクセス機械に移しても通用するが，かけ算命令のあるランダムアクセス機械の上では，$NP=P$ となることが，Simon[23)] により報告されている（かけ算命令のコストも，logarithmic ではなく単位コストである点に注意されたい）．

また，Sahni[21)] により，精度が繰り返しごとに上昇するナップザック問題の近似解法も報告されている．

5.2 広い意味の最小化問題の NP-完全性

4節で与えられた LL(k) 構文解析表の最小化問題を，$((\{¥\} \cup \Sigma \cup N) \times \Sigma^{**}) \times \Sigma^{**}$ 上の一般の表に関する組み合せ問題として考えるなら，NP-完全であることを示す．したがって“正しい LL(k) 構文解析表”が持つ固有な性質を反映させない限り，現在のどんなアルゴリズムで解いても，最悪の場合は指数関数的処理時間を要することがわかる．

定理 5.1 次の組み合せ問題は，一般の表に関する問題としてみるならば，NP-完全である．

イ) 不完全順序機械の最小化問題[7),20),22)]．

ロ) LL(k) 構文解析表の最小化問題．

証明 これらの2つの問題が NP に属することは，いわゆる“guessing and verifying”法[8)]により明らかである．したがって，NP-完全である“グラフ彩色問題”が，問題のサイズの多項式時間以内に，これらの問題に変形可能であることを示せば十分である．

イ) 彩色問題∝不完全順序機械最小化

節点数 n の無向グラフ G に対して，次のような $n+2$ 行 $n(n-1)/2$ 列の出力関数表 F と状態遷移関数表 H を作る．

G の2節点 N_i と N_j において，N_i と N_j が枝で結ばれている時，状態 T_i, T_j と入力記号 $X_{\{i,j\}}$ について，表 H に関しては，

$$H(T_i, X_{\{i,j\}}) = T_{n+1},\ H(T_j, X_{\{i,j\}}) = T_{n+2}$$

とし，他の表 H の項目は“undefined”とする．表 F については，

$$F(T_{n+1}, X_{\{1,2\}}) = 1,\ F(T_{n+2}, X_{\{1,2\}}) = 2$$

とし，他の表 F の項目は“undefined”とする．

ロ) 不完全順序機械最小化∝LL(k) 解析表最小化

n 行 m 列の出力関数表 F（出力記号は数字とする）と状態遷移関数表 H を持つ不完全順序機械に対し，次のような LL(k) 構文解析表の一般形となる n 行 $m+1$ 列の表 I を作る．ただし，左フィールドが終端記号の状態に関しては，多項式時間で“重ね合せ”可能であるため，表 I には構成しないことにする．

状態 T_i, T_j と入力記号 X に関し，

$$H(T_i, X) = T_{i'},\ H(T_j, X) = T_{j'}$$

かつ $T_{i'} \neq T_{j'}$

ならば，表 I の項目 $I(T_i, u_0), I(T_j, u_0)$ 中の同じ位置にそれぞれ記号 $[A : T_{i'}]$, $[A : T_{j'}]$ を記入する．記入の全く行われなかった表 I の列 u_0 の項目には“**error**”を記入し，“**error**”以外の列 u_0 の項目には，すべて規則番号1を与える．

列 u_1 から列 u_m に関しては，表 F の列 l の記入 $F(T_i, X) = j$ に対し，$I(T_i, u_l)$ の指定する規則番号を $j+1$ とし，$F(T_i, X) =$“**undefined**”に対し，$I(T_i, u_l) =$“**error**”とする．表 I の列 u_1 から列 u_m に関し，規則番号が記入されている箇所には，それぞれ右辺が終端記号列となっている規則を記入する． ☒

6. むすび

早期誤り検出の完全な LL(k) パーサの構成法を示し，その最小化問題の計算量を，広い解釈の下で評価した．文献6を教示された J. D. Ullman 教授に感謝する．

---

* 文献16では Cook と Karp の定義がそれぞれ Post(1944) の Turing reducibility と many-one reducibility の多項式時間版である点を注意している．Cook の意味で NP-完全であるが，Karp の意味で NP-完全でない例は知られていない（文献1）．

** 邦文献の出典は電子通信学会技術研究報告 (1976) AL 75-73, 81, 82. その他の出典は文献 1,12.

参考文献

1) A. V. Aho et al.: **The Design and Analysis of Computer Algorithms**, Addison-Wesley (1974).
2) A. V. Aho et al.: Deterministic Parsing of Ambiguous Grammars, C. ACM 18-8, 441～452 (1975).
3) A. V. Aho et al.: **The Theory of Parsing, Translation, and Compiling**, Vol.1 Prentice Hall (1972).
4) R. A. Brooker et al.: The Compiler-Compiler, Ann. Rev. Auto. Programming, Vol. 3, Pergamon, 229～275 (1963).
5) S. A. Cook: The Complexity of Theorem Proving Procedures, 3rd ACM Conf. STOC, 151～158 (1971).
6) N. El Djabri: Reducing the Size of LL(1) Parsing Tables, TR-119, Princeton Univ. (1973).
7) S. Ginsburg: A Technique for the Reduction of a given Machine to a Minimal State Machine, IRE T-EC 8, 346～355 (1959).
8) J. Hartmanis et al.: On the Structure of Feasible Computations, **GI-4 Jahrestagung**, Springer-Verlag (1975).
9) H. B. Hunt et al.: On the Complexity of LR(k) Testing, C. ACM 18-12, 707～716 (1975).
10) E. T. Irons: A Syntax Directed Compiler for ALGOL 60, C. ACM 4-1, 51～55 (1961).
11) D. B. Johnson et al.: Efficient Construction of LL(1) Parsers, TR-164, Penn. State Univ. (1975).
12) R. M. Karp: Reducibility among Combinatorial Problems, **Complexity of Comp. Computations**, Plenum Press (1972).
13) D. E. Knuth: Top-Down Syntax Analysis, Acta Informatica 1-2, 79～110 (1971).
14) A. J. Korenjak et al.: Simple Deterministic Languages, 7th IEEE Conf. SWAT, 36～46 (1966).
15) R. Kurki-Suonio: Note on Top Down Languages, BIT, 9-3, 225～238 (1969).
16) R. E. Lander: On the Structure of Polynomial Time Reducibility, J. ACM 22-1, 155～171 (1975).
17) J. P. Levy: Automatic Correction of Errors in Programming Languages, TR 71-116, Cornell Univ. (1971).
18) P. M. Lewis et al.: Syntax Directed Transduction, J. ACM 15-3, 464～488 (1968).
19) P. M. Lewis et al.: An ALGOL Compiler Designed Using Automata Theory, Symp. Comp. Auto., Poly. Inst. Brooklyn (1971).
20) M. C. Paull et al.: Minimizing the Number of States in Incompletely Specified Sequential Switching Functions, IRE T-EC 8 (1959).
21) S. Sahni: Approximate Algorithms for the 0/1 Knapsack Problem, J. ACM 22-1, 115～124 (1975).
22) 佐藤: 線形グラフを用いた不完全順序機械の最小化の一手法, 信学論D 58-5, 225～232(1975).
23) J. Simon: On the Power of Multiplication in Random Access Machines, TR 74-205, Cornell Univ. (1974).
24) 徳田, 井上: Lewis-Stearns アルゴリズムに基づく LL(k) パーサの誤り検出の遅れについて, 信学論D, 59-3, 215～216 (1976).
25) T. Tokuda: A Combinatorial Method for Minimizing LL(k) and LR(k) Parsers, M. S. Th., Tokyo Inst. Tech. (1976).
26) D. Wood: A Note on Top-Down Deterministic Languages, BIT 9-4, 387～399 (1969).
27) 吉村: 能率のよいトップダウン型構文解析プログラムの自動作成について, 情報処理 16-3, 195～204 (1975).

(昭和51年5月31日受付)
(昭和52年11月2日再受付)