情報処理学会論文誌　データベース　Vol. 3　No. 2　80–90 (June 2010)

ダブル配列の遷移拡張による LR 構文解析表の実現と解析速度の高速化

蔵満琢麻[†1]　重越秀美[†1]　望月久稔[†1]

LR 構文解析は，C 言語や Pascal などのプログラミング言語，SQL などの問合せ言語，XML パーサ，プロトコルパーサなどの様々な場面で利用されており，時間的，領域的に効率的な LR 構文解析表の実現が求められる．本論文では，ダブル配列の遷移を拡張して LR 構文解析表を実現し，解析時の状態遷移に要する計算量を抑制することで，LR 構文解析に要する時間を短縮する手法を提案する．実験の結果，提案手法は構文解析器生成系の一種である Bison と比較して LR 構文解析に要する時間を 10〜23%削減した．

An Implementation of LR Parsing Table by Extension of the Double-array Transition for Speeding Up LR Parsing

Takuma Kuramitsu,[†1] Hidemi Shigekoshi[†1] and Hisatoshi Mochizuki[†1]

LR parsers are used widely, such as programing language compilers, SQL parsers, XML parsers, and protocol parsers. So it is important to implement efficient LR parsing tables. In this paper, we present an efficient implementation of LR parsing table with Double-Array that is extended transition function. Our experiment show that proposal method can decrease the parsing time about 10〜23% in comparison with parser generator Bison.

†1 大阪教育大学
Osaka Kyoiku University

1. はじめに

コンピュータは定められた処理を高速かつ正確に実行できるが，CPU が直接実行できる機械語は人間にとって理解しがたい形式であり，コンピュータに複雑な処理を実行させるために人手で大量の機械語を入力することは現実的ではない．そのため，人間が理解しやすい形式の入力を解析する構文解析器の作成が必要になるが，様々なアプリケーションごとに，複雑な文法を正しく解析する効率的な構文解析器を作成するには大変な労力が必要である[1)]．

そこで，開発者の負荷を軽減するために構文解析器生成ツールが使用される．Yacc[2)] や Bison[3)] は，BNF に基づく形式で記述した規則の集合を入力として与えることで，生成規則に沿った LR 構文解析器を自動的に生成する．これらのツールは，C 言語や Pascal などのプログラミング言語，SQL などの問合せ言語，XML パーサ[4)]，プロトコルパーサの生成[5)] など様々な場面で利用される．

Yacc や Bison が採用している LR(1) 構文解析は，シフト還元構文解析の一種で，LR 構文解析表と呼ばれる一種の状態遷移表を生成規則の集合から構築することで解析時間を短縮する手法[1)] であり，入力の長さに比例して解析時間が増加する．近年，コンピュータの高性能化やインターネットの普及により，取り扱うデータの量は増加する一方であり，構文解析器には解析速度の高速性が求められる．

ここで，トライ構造の状態遷移表を効率的に実現する手法にダブル配列[6),7)] がある．ダブル配列は，隣接行列を 2 本の 1 次元配列に圧縮したデータ構造で，ある状態における遷移先を遷移種の内部表現値から算出する手法であり，高い空間効率と状態遷移の高速性をあわせ持つ．

本論文では，ダブル配列上に LR 解析表の還元アクションを管理するための状態を新たに定義することで，ダブル配列による LR 構文解析表を実現し，解析時の状態遷移に要する計算量を抑制することで，LR 構文解析に要する時間を短縮する手法を提案する．また，ダブル配列による LR 構文解析表を圧縮し，より効率的な解析表を実現する方法を提案する．

以下，2 章で，LR 構文解析表とダブル配列について説明し，3 章で，提案手法について述べ，4 章で提案手法に理論的，実験的評価を与え，5 章で，本論文のまとめと今後の課題について述べる．

2. LR 構文解析表とダブル配列

本章では，LR 構文解析表[1)] について述べた後，提案手法で用いるダブル配列について説

明する．以下，LR 構文解析表作成時に登録する文法 G に含まれる規則 g の数を $|G|$，規則 g の規則番号を I_g と表記し，文法中に存在する終端記号の集合を T，非終端記号の集合を N，それぞれの要素数を $|T|$，$|N|$ と表記する．

2.1　LR 構文解析表

本節では LR 構文解析表の構成について述べた後，解析表の実装例として 2 次元の配列構造と Bison[8]（version2.3）が生成する解析表について説明する．

まず，LR 構文解析表の構成について説明する．LR 構文解析表は，Action 関数と Goto 関数からなる[1]．Action 関数は，状態 x と終端記号 a を引数とし，shift(t)，reduce(I_g)，accept，error のいずれかのアクションを返す関数である．ここで，shift(t) は，状態 t へ遷移し，新しい終端記号をトークン列から取得するアクションで，reduce(I_g) は規則 g を還元するアクションを表す．Goto 関数は，reduce(I_g) 実行時に呼び出される関数で，規則 g の還元で生じる非終端記号 A による遷移先 t を返す．以下，状態 x における終端記号 a によるアクションを，Action(x, a) と記述し，非終端記号 A による遷移先を Goto(x, A) と記述する．

LR 構文解析表を実現する単純な方法は，状態と終端・非終端記号を添字とする 2 次元の配列構造を用いることである．**図 1** (a) に示す文法 Q に対して実現した 2 次元配列による LR 構文解析表を図 1 (c) に示す．また，図 1 (c) における goto グラフを図 1 (b) に示す．図 1 (c) 中，Action 表においては shift(t) を s t，reduce(I_g) を r I_g，accept を acc と表記し，error を空のエントリとして表し，Goto 表においては遷移先の状態番号を格納する．たとえば，Action(1, ‘q’)=shift(2)，Action(2, ‘p’)=reduce(4)，Goto(4, E)=6 である．図 1 (b) において丸枠が状態，矢印が遷移先を表し，矢印付近の文字は遷移のラベルを表す．

2 次元配列を用いて LR 構文解析表を実現すると，解析時に次のアクションを $O(1)$ 時間で決定できるが，図 1 (c) のように配列がスパースな場合，記憶領域の面で効率的ではない[1]．

次に，Bison が生成する LR 構文解析表について述べる．Bison は 1 次元配列 Base，DefAct，Check，Table，NBase，DefGoto を用いて[⋆1]2 次元配列を圧縮し，記憶効率が良い LR 構文解析表を実現する．終端記号 ‘$’，‘p’，‘q’ の内部表現値をそれぞれ 0，1，2，非終端記号 S′，S，E の内部表現値をそれぞれ 3，4，5 とし，図 1 (c) に対して Bison の表圧縮法を適用した LR 構文解析表を図 1 (d) に示す．

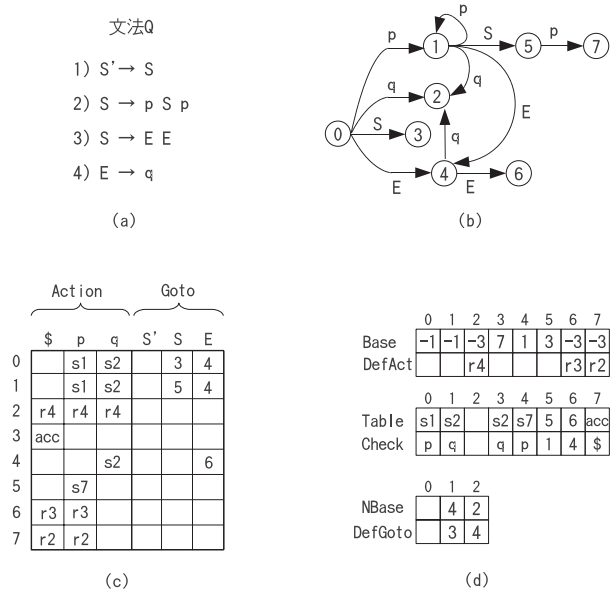

図 1　文法 Q に対する LR 構文解析表と goto グラフ
Fig. 1　The LR parse table and the goto graph of the grammar Q.

Bison は配列 Base，Check，Table を用いて，状態 x における終端記号 a によるアクションを式 (1) により配列 Table 上に定義する．配列 Base の添字 x は LR 構文解析表の状態番号に 1 対 1 対応し，各状態の Base 値は配列 Table のエントリへのオフセットとなり，Action(x, a) は，状態 x における Base 値と終端記号 a の内部表現値を足し合わせた位置に格納される[8]．たとえば，図 1 (d) 中において Action(1, ‘q’)=shift(2) は，Base[1]=−1 に ‘q’=2 を足し合わせた位置である Table[1] に格納される．解析時において Bison は，式 (1) により配列 Table に格納したアクションを参照して状態遷移を行う．

$$\begin{cases} entry = \mathrm{Base}[x] + a \\ \mathrm{Check}[entry] = a \\ \mathrm{Table}[entry] = \mathrm{Action}(x,\ a) \end{cases} \tag{1}$$

⋆1 Bison2.3 におけるプログラムソース中の配列名はそれぞれ，yypact，yydefact，yycheck，yytable，yypgoto，yydefgoto である．

$$\begin{cases} entry = \mathrm{NBase}[A - |\mathrm{T}|] + x \\ \mathrm{Check}[entry] = x \\ \mathrm{Table}[entry] = \mathrm{Goto}(x,\ A) \end{cases} \tag{2}$$

このように，各状態におけるアクションの格納位置が Base 値により定まるため，アクションがすべて等しい状態においては，同じ Base 値を格納することで配列 Table 上のエントリを共有でき，記憶領域の抑制が図れる．たとえば，図 1 (d) において，状態 0 と状態 1 は同じ Base 値を持ち，Table[0]，Table[1] に格納したアクションを共有する．

また，Bison は還元アクションが存在する状態 x において，最頻出の還元規則をデフォルトの還元規則（以下，DR 規則）として配列 DefAct に格納し，配列 Table に定義するエントリ数を削減する．たとえば，図 1 (c) において状態 2 における DR 規則は規則 4 であり，図 1 (d) の DefAct[2] には規則番号 4 を格納する．Bison は解析時に状態 x におけるアクションを取得する際，式 (1) を満たすアクションが存在しなければ，DefAct[x] を参照し，DR 規則 I_g が存在すれば reduce(I_g) を実行し，存在しなければ error として解析を終了する[8]．

Bison が生成する解析表では Goto 関数も同様の表圧縮法を用いて定義される．配列 NBase における添字は非終端記号の内部表現値と 1 対 1 対応し，非終端記号 A による Goto 関数による遷移先は式 (2) により配列 Table 上に定義され，各非終端記号において最頻出の遷移先はデフォルトの遷移先として配列 DefGoto 上に定義される[8]．

これらの表圧縮法は，Aho ら[1] により詳しく紹介されており，様々な LR 構文解析器生成系[9)–11)] に利用されている．上記の表圧縮法を適用した LR 構文解析表では，解析表に定義するエントリ数の削減により 2 次元配列構造よりも記憶領域を抑制できるが，Action 関数や Goto 関数による遷移先を取得する際に，式 (1)，式 (2) により 3 つの配列 Base（または NBase），Check，Table の参照を要する．

2.2　ダブル配列

本節では，提案手法で用いるダブル配列について説明する．Aoe らは，初期状態を除く各状態の入次数が 1 である木構造の状態遷移表を効率的に実現する手法としてダブル配列[6),7)] を提案した．ダブル配列は 2 本の配列 Base と Check を用いて，状態 x から遷移種 a による遷移先 t を式 (3) により定義する手法で，配列の添字が状態番号に対応する．

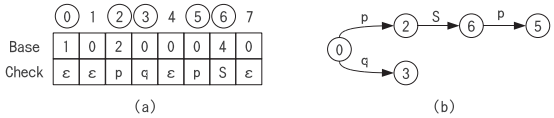

図 2　ダブル配列と状態遷移図
Fig. 2　Double-Array and the state transition diagram.

$$\begin{cases} t = \mathrm{Base}[x] + a \\ \mathrm{Check}[t] = a \end{cases} \tag{3}$$

図 2 (a) は図 1 (b) の goto グラフをダブル配列により部分的に実現した状態遷移表で，状態 0，2，3，6，5 は，図 1 における状態 0，1，2，5，7 とそれぞれ対応する．図 2 (b) は図 2 (a) に対応する状態遷移図を表す．配列の添字を丸枠で囲んでいない要素はダブル配列における未使用の要素を表し，配列 Check 内の ε は遷移種として使用しない値を表す．

ここで，2.1 節で述べた Bison の遷移式 (1) とダブル配列の遷移式 (3) を比較するため，状態遷移に要する時間計算量として，1 回の状態遷移において参照する配列の要素数を遷移計算量と定義する．Bison の遷移式 (1) が配列 Base，Check，Table の 3 要素を参照する必要があるのに対し，ダブル配列の遷移式 (3) で参照する配列の要素数は，配列 Base，Check の 2 要素のみであるため，ダブル配列は配列 Table を参照しない分，Bison よりも遷移計算量を抑制できる．また，ダブル配列は配列 Table が不要となる分，記憶効率の面でも優れている．

しかし，ダブル配列は遷移元の Base 値と遷移種により遷移先の状態番号が一意に定まり，遷移の数だけ異なる遷移先が定義される[12)] ため，複数の遷移を特定の状態へ直接定義できず，一般の有限オートマトンにおける遷移表を作成できない．たとえば，図 2 の状態 2 から遷移種 ‘q’ により既存の状態 3 への遷移を定義する場合を考える．遷移式 (3) により，状態 2 から遷移種 ‘q’ による遷移先の状態番号は 4（Base[2]+‘q’）に定まるため，状態 2 から既存の状態 3 へ直接に遷移を定義できない．

そこで青江は，複数定義される遷移先の集合 $\{t_1 \ldots t_m\}$ を，1 つの状態に統合して取り扱い，擬似的に入次数が 2 以上となる状態をダブル配列上に定義する手法[12)] を提案した．この手法は，遷移先集合 $\{t_1 \ldots t_m\}$ から任意の状態 t_i を状態集合の代表として取り扱い，t_i 以外の状態に t_i へのポインタを定義することで状態を統合する．以下，状態集合の代表である状態を統一状態，統一状態へのポインタを管理する状態を間接状態と呼び，統一状態と

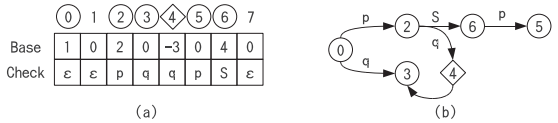

図 3　拡張ダブル配列と状態遷移図
Fig. 3　Extended Double-Array and the state transition diagram.

間接状態を導入したダブル配列を拡張ダブル配列と呼ぶ.

図 3 (a) に，図 2 における状態 2 から遷移種 'q' による状態 3 への遷移を定義した拡張ダブル配列を示し，対応する状態遷移図を図 3 (b) に示す．図 3 中，丸枠の状態が統一状態，菱形枠の状態が間接状態を表し，間接状態 4 の Base 値には統一状態 3 へのポインタとして状態番号を負値で格納する．

間接状態の Base 値を参照して遷移先を求める状態遷移の遷移計算量は，式 (1) により配列 Table を参照して遷移先を求める状態遷移と同程度必要といえる．ここで，拡張ダブル配列は，統一状態から統一状態への式 (3) による遷移において遷移計算量を抑制できるため，式 (1) による状態遷移表よりも総遷移計算量を抑制できるといえる．そこで，拡張ダブル配列による状態遷移表を利用し，状態遷移の遷移計算量を抑制した LR 解析表を実現することが考えられる．しかし，拡張ダブル配列は，状態遷移ではない還元アクションを定義できず，Action 関数を実現できないため，直接 LR 解析表に適用できない.

3. ダブル配列により状態遷移を高速化した LR 構文解析表

本章では，ダブル配列の遷移をさらに拡張して LR 構文解析表を実現することで，状態遷移に要する遷移計算量を抑制し，LR 構文解析に要する時間を短縮する方法を提案する．以下，ダブル配列による LR 構文解析表の実現法について述べた後，アクション共有などの表圧縮法をダブル配列上で適用する方法について述べ，提案手法の解析アルゴリズムを解説する.

3.1　ダブル配列による LR 構文解析表の実現

LR 構文解析表における Goto 関数は統一状態と間接状態を導入した拡張ダブル配列を用いることで容易に実現できる．本節では還元状態を新たに定義し，ダブル配列上で Action 関数を実現することで，解析時の状態遷移を高速に行える LR 構文解析表を実現する手法を提案する.

まず，LR 構文解析表の Action 関数をダブル配列上で実現するために，還元状態の定義

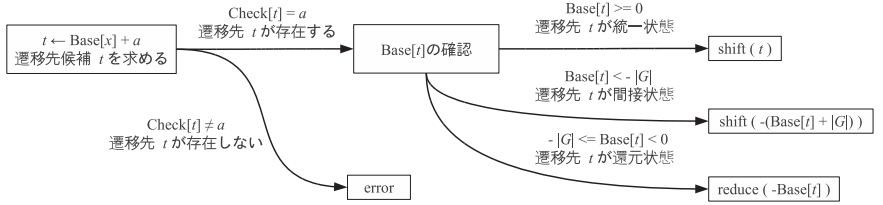

図 4　アクションの決定方法
Fig. 4　The method of selecting action.

について述べる．提案手法では，状態 x が終端記号 a により規則 g を還元するとき，状態 x から終端記号 a による遷移先 t を作成し，Base[t] に規則 g を表す値を格納する．以下，この状態 t を還元状態と呼ぶ.

還元状態の定義により，ダブル配列上には，統一状態，間接状態を含む 3 種類の状態が存在する．そこで，統一状態 α，間接状態 β，還元状態 γ の Base 値を，それぞれ以下の式 (4)，式 (5)，式 (6) で定義する．式中，統一状態 α における遷移先の集合を $S(\alpha)$，遷移ラベルの集合を $L(\alpha)$ とする．ここで，$|S(\alpha)| = |L(\alpha)|$，$0 < I_g \leq |G|$ である.

$$\mathrm{Base}[\alpha] = base, \qquad (base \geq 0,\ S(\alpha) = \{base + a;\ a \in L(\alpha)\}) \tag{4}$$

$$\mathrm{Base}[\beta] = -(|G| + t), \qquad (\beta \text{ が統一状態 } t \text{ へのポインタを持つ}) \tag{5}$$

$$\mathrm{Base}[\gamma] = -I_g, \qquad (\gamma \text{ が規則 } g \text{ の規則番号 } I_g \text{ を持つ}) \tag{6}$$

式 (4)〜式 (6) により，Base[β]$< -|G| \leq$Base[γ]$< 0 \leq$Base[α] がつねに成り立つため，Base 値を参照することで，状態の種類を判別し，実行するアクションを決定できる．**図 4** に，統一状態 x における終端記号 a によるアクションを決定する際の流れ図を示す．まず，統一状態 x において，終端記号 a による遷移先 t を求め，状態 t が統一状態であれば shift(t)，間接状態であれば shift($-$(Base[t]+|G|))，還元状態であれば reduce($-$Base[t]) を行う．ただし，提案手法では，開始規則の還元を構文受理（accept）とし，遷移先が存在しない場合を構文エラー（error）と定義する.

終端記号 '$', 'p', 'q' の内部表現値をそれぞれ 0，1，2，非終端記号 S′，S，E の内部表現値をそれぞれ 3，4，5 とし，文法 Q を登録したダブル配列による LR 構文解析表を**図 5** (a) に示し，対応する状態遷移図を図 5 (b) に示す．図 5 において，添字を丸で囲んでいる要素が統一状態，菱形で囲んでいる要素が間接状態，四角で囲んでいる要素が還元状態である．また，添字を囲んでいない要素は未使用要素であり，Base 値に 0，Check 値に遷移のラベルとして使用しない値 ε（$= |N| + |T|$）を格納して他の状態と区別する．図 1 の LR 構文

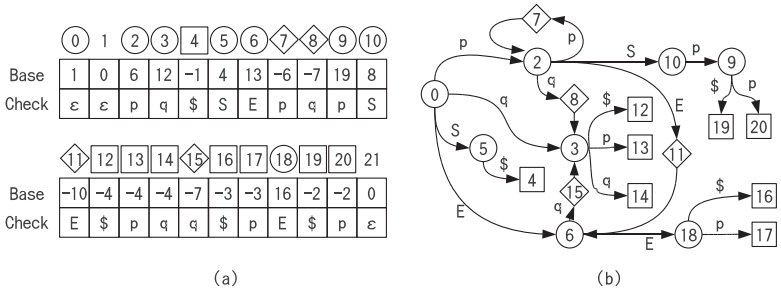

図 5　ダブル配列による LR 構文解析表と状態遷移図
Fig. 5　The LR parse table with Double-Array and the state transition diagram.

表 1　図 1 の解析表と図 5 との状態対応表
Table 1　The state correspondence table between figure 1 and figure 5.

図 1 の解析表における状態番号	0	1	2	3	4	5	6	7
図 5 の解析表における状態番号	0	2	3	5	6	10	18	9

解析表における状態と図 5 の解析表における統一状態との対応表を**表 1** に示す．図 1 の解析表における状態 1，2 は統一状態 2，3 に対応する．

図 1 の Action(0, ‘q’) = shift(2) に該当するアクションは，図 5 の統一状態 0 から終端記号 ‘q’ による状態遷移であり，ダブル配列の遷移式 (3) により統一状態 0 から統一状態 3（Base[0]+‘q’）への遷移として定義する．また，図 1 の Action(1, ‘q’) = shift(2) に該当するアクションは，図 5 の統一状態 2 から統一状態 3 への状態遷移である．提案手法は，統一状態 2 から終端記号 ‘q’ による遷移先として間接状態 8（Base[2]+‘q’）を定義し，状態 3 へのポインタとして，文法 Q の規則数 4 を足し合わせた値 7 を間接状態 8 における Base 値に負値で格納することで，統一状態 2 から統一状態 3 への状態遷移を間接的に定義する．

また，図 1 の Action(2, ‘$’) = Action(2, ‘p’) = Action(2, ‘q’) = reduce(4) に該当するアクションは，統一状態 3 からの遷移先として還元状態 12（Base[3]+‘$’），13（Base[3]+‘p’），14（Base[3]+‘q’）を作成し，それぞれの Base 値に還元規則の規則番号 4 を負値で格納することで定義する．

ダブル配列上で LR 構文解析表を実現することで，統一状態 0 から統一状態 3 への遷移のように，間接状態を経由しない状態遷移において遷移計算量を抑制でき，解析時間の短縮が図れる．

3.2　表圧縮法の適用

本節では，統一状態におけるデフォルトの還元規則（DR 規則），Action 関数の共有，各非終端記号によるデフォルトの遷移先をダブル配列上に定義し，より効率的な LR 構文解析表を実現する方法を提案する．

まず，統一状態における DR 規則を定義する方法について説明する．Bison は状態数分の要素を持つ配列 DefAct を設けて，各状態における DR 規則を管理する[8]．同様に提案手法においても新たにダブル配列と等しいサイズの配列を設けて統一状態における DR 規則を管理することが考えられるが，統一状態に該当する要素以外が未使用になり，記憶効率の面で望ましくない．そこで提案手法では，統一状態 x における DR 規則を，ダブル配列上の隣接要素 $x+1$ の Check 値を用いて管理する．以下，この隣接要素 $x+1$ を統一状態 x の付属要素と呼ぶ．提案手法は統一状態に対してのみ付属要素を作成し，付属要素の Check 値に規則番号 I_g を負値で格納することで DR 規則をダブル配列上に定義する．

しかし，ある状態 x から，内部表現値が連続する遷移のラベル a と $a+1$ による遷移が存在する場合，遷移定義式 (3) より，遷移先の状態 t と $t+1$ がダブル配列上で隣接し，状態 $t+1$ の存在により状態 t の付属要素を作成できない．そこで，遷移先の状態がダブル配列上で隣接しないように，提案手法の遷移を式 (7) により定義する．式 (7) と付属要素を用いることで，提案手法は任意の状態に情報を追加できる．また，式 (7) による遷移定義は，式 (3) による遷移定義と同様に，式 (1) と比較して配列 Table を参照しない分，状態遷移に要する遷移計算量を抑制できる．

$$\begin{cases} t = \mathrm{Base}[x] + 2a \\ \mathrm{Check}[t] = a \end{cases} \tag{7}$$

次に，Action 関数を共有する方法について説明する．式 (3) により遷移を定義するダブル配列において，同じ Base 値を持つ状態は遷移を共有する[13]．提案手法においても，遷移を共有して解析表の記憶領域をさらに抑制することが考えられる．しかし，前節で解説したダブル配列による LR 構文解析表は，Action 関数を定義する遷移と，Goto 関数を定義する遷移を同じ Base 値により各状態に定義するため，異なる状態間で Action 関数のみを共有できない．そこで，節点 x における Goto 関数を定義する遷移を，節点 x の付属要素 $x+1$ の Base 値を用いて定義する．このように，各状態が Action 関数と Goto 関数を異なる Base 値により定義することで，Goto 関数を定義する遷移が異なる状態においても，Action 関数を定義するための遷移がすべて一致する状態の遷移を共有できる．

最後に，各非終端記号によるデフォルトの遷移先を定義する方法について説明する．提案手法では，非終端記号 A によるデフォルトの遷移先 t を，非終端記号 A の内部表現値から算出できるように，式 (8) を用いてダブル配列上に定義する．ここで，式 (8) において非終端記号 A の内部表現値を 2 倍するのは，デフォルトの遷移先 t における付属要素を定義するためである．ただし，開始規則の左辺 S' による遷移先は存在しないため，式 (8) において，$A \neq$ S'，$|T| < A < |T| + |N|$ である．式 (8) により，提案手法は間接参照せずにデフォルトの遷移先へ直接に遷移できる．

$$t = A \times 2 \tag{8}$$

文法 Q を登録した提案手法の解析表を**図 6** (a) に示し，対応する状態遷移図を図 6 (b) に示す．また，図 1 の解析表における状態と図 6 の解析表における統一状態との対応表を**表 2** に示す．図 6 において，統一状態を楕円形で表し，状態番号を破線左の数値で表す．破線右の数値は付属要素を表す．また，状態遷移図において統一状態右上の括弧内における数値は DR 規則の規則番号を表す．

提案手法は，統一状態 5 の DR 規則 4 を，付属要素 6 の Check 値に規則番号 4 を負値で格納することで定義する．これにより，図 5 の解析表が統一状態 3 において，還元アクションを定義するために 3 つの還元状態を遷移先として作成する必要があるのに対し，図 6 の解析表では付属要素 1 つ分の領域で統一状態 5 における還元アクションを定義できる．

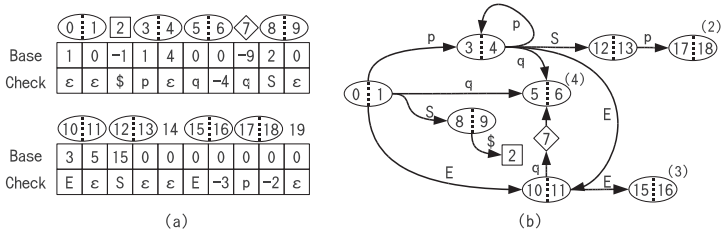

図 6　提案手法の LR 構文解析表と状態遷移図
Fig. 6　The proposal LR parse table and the state transition diagram.

表 2　図 1 の解析表と図 6 の解析表との状態対応表
Table 2　The state correspondence table between figure 1 and figure 6.

図 1 の解析表における状態番号	0	1	2	3	4	5	6	7
図 6 の解析表における状態番号	0	3	5	8	10	12	15	17

図 6 の状態 3 では，Goto 関数を定義する遷移を状態 3 の付属要素 4 の Base 値を用いて定義し，状態 3 の Base 値を状態 0 の Base 値と等しくすることで，状態 0 と Action 関数を定義するための遷移を共有する．これにより図 5 の解析表が状態 2 において，状態 2，3 への遷移を定義するために間接状態 7，8 を遷移先として作成する必要があるのに対し，図 6 の解析表では間接状態を定義することなく，状態 3 から状態 3，5 へ直接に遷移できる．

このように，表圧縮法をダブル配列上で実現することで，提案手法はダブル配列上に定義する状態数を抑制でき，より効率的な LR 構文解析表を実現できる．

以上で定義した提案手法における Action 関数と Goto 関数を**表 3** にまとめる．提案手法は，状態 x における終端記号 a によるアクションや，状態 x における非終端記号 A による遷移先を，遷移先 t の有無や，遷移先 t の種類により決定する．次節では，表 3 に基づく解析アルゴリズムを解説する．

表 3　提案手法における Action 関数と Goto 関数
Table 3　The Action function and the Goto function of proposal method.

Action関数

遷移先 t = Base[x] + 2a の有無	アクションの選択	Action(x, a)
遷移先 t が存在しない Check[t] ≠ a	DR規則が存在する - Check[x + 1] >0	DR規則を還元 reduce (- Check[x + 1])
	DR規則が存在しない - Check[x + 1] <= 0	error
遷移先 t が存在する Check[t] = a	遷移先 t が統一状態 Base[t] >= 0	状態 t へシフト shift (t)
	遷移先 t が間接状態 Base[t] < -|G|	状態(-(Base[t] + |G|))へシフト shift (-(Base[t] + |G|))
	遷移先 t が還元状態 - |G| <= Base[t] < 0	規則(-Base[t])を還元 reduce (-Base[t]) (-Base[t] = 1のときaccept)

Goto関数

遷移先 t = Base[x+1] + 2A の有無	遷移先 t の種類	Goto(x, A)
遷移先 t が存在しない Check[t] ≠ A		デフォルトの遷移先 2A
遷移先 t が存在する Check[t] = A	遷移先 t が統一状態 Base[t] >= 0	遷移先 t
	遷移先 t が間接状態 Base[t] < 0	間接参照による遷移先 - (Base[t] + |G|)

入力：解析表を構成する配列 Base と Check，入力トークン列 $X = a_1 a_2 \cdots a_i \cdots a_n\$$

```
(AN1)   Push(0); i = 1
(AN2)   while (1)
(AN3)     t ←Base[Top()] + 2a_i;
(AN4)     if (Check[t] ≠ a_i) then                  // 遷移先 t が存在しない場合
(AN5)       rule ← −Check[Top() + 1];               //
(AN6)       if (rule > 0) then                      //     DR 規則が存在する場合
(AN7)         Reduce(rule);                         //           reduce(rule)
(AN8)       else                                    //     DR 規則が存在しない場合
(AN9)         return error;                         //           error
(AN10)    else                                      // 遷移先 t が存在する場合
(AN11)      if (Base[t] ≥ 0) then                   //     遷移先 t が統一状態
(AN12)        Push(t); i ← i + 1;                   //           shift(t)
(AN13)      else if (Base[t] < −|G|) then           //     遷移先 t が間接状態
(AN14)        Push(−(Base[t]+|G|)); i ← i + 1;      //           shift(−(Base[t]+|G|))
(AN15)      else                                    //     遷移先 t が還元状態
(AN16)        rule ← −Base[t];                      //
(AN17)        if (rule = 1) then                    //           還元規則 rule が開始規則
(AN18)          return accept;                      //                 accept
(AN19)        Reduce(rule);                         //           reduce(rule)
(AN20)  end
```

図 7　解析アルゴリズム
Fig. 7　Algorithm: Analize.

3.3　解析アルゴリズム

本節では，**図 7** に示す提案手法の解析アルゴリズムについて説明した後，解析例を示す．アルゴリズム中で用いる関数 Push(*state*)，Pop(*num*)，Top()，Left(*rule*)，Length(*rule*) を以下に示し，関数 Reduce(*rule*) を**図 8** に示す．

- **Push(***state***)**： スタックに状態 *state* をプッシュする．
- **Pop(***num***)**： スタックから *num* 個の要素をポップする．
- **Top()**： スタックの先頭要素を返す．
- **Left(***rule***)**： 規則 *rule* における左辺の非終端記号を返す．
- **Length(***rule***)**： 規則 *rule* における右辺の長さを返す．

図 7 の解析アルゴリズムは (AN3) において，スタックの先頭に存在する状態から終端記号 a_i による遷移先の候補 t を計算し，(AN4)〜(AN19) で遷移先 t の有無や，状態 t の種類

Function：Reduce(*rule***)**

```
(RE1)   A ← Left(rule); l ← Length(rule);
(RE2)   Pop(l);
(RE3)   t ← Base[Top() + 1] + 2A;
(RE4)   if (Check[t] ≠ A) then          // 遷移先 t が存在しない場合
(RE5)     Push(2A);                     //       状態 2A へデフォルト遷移
(RE6)   else if (Base[t] >= 0) then     // 遷移先 t が統一状態
(RE7)     Push(t);                      //       状態 t へ直接的に遷移
(RE8)   else                            // 遷移先 t が間接状態
(RE9)     Push(−(Base[t] + |G|));       //       状態 (−(Base[t] + |G|)) へ間接的に遷移
(RE10)  return;
```

図 8　関数：Reduce
Fig. 8　Function: Reduce.

によって実行するアクションを決定する．

まず，遷移先 t が存在しない場合，(AN5)〜(AN9) で，DR 規則の還元，または error を実行する．(AN5)，(AN6) において DR 規則の有無を確認し，DR 規則が存在すれば (AN7) で還元を，存在しなければ (AN9) において error を返して終了する．

また，遷移先 t が存在する場合，(AN11)〜(AN19) で，シフト，還元，accept のいずれかのアクションを実行する．状態 t が統一状態であれば (AN12) において状態 t へのシフトを実行する．間接状態であれば (AN14) において状態 t の Base 値から遷移先の状態番号を得てシフトする．状態 t が還元状態であれば，(AN16) において還元状態 t の Base 値から規則番号 *rule* を取得し，規則番号 *rule* が開始規則 1 であれば (AN18) において accept を返して終了し，そうでなければ (AN19) において関数 Reduce(*rule*) を実行する．

図 8 に示す関数 Reduce は，(RE1)〜(RE3) において，規則 *rule* の還元により生じる非終端記号 A による遷移先の候補 t を求め，(RE4)〜(RE9) で遷移先 t の有無や，状態 t の種類によって遷移先を決定する．

まず，遷移先 t が存在しない場合，(RE5) において非終端記号 A におけるデフォルトの遷移先 $2A$ へ状態遷移する．

また，遷移先 t が存在する場合，(RE6) で遷移先 t が統一状態であれば (RE7) で状態 t へ直接的に遷移し，間接状態であれば (RE9) で状態 (−(Base[t]+|G|)) へ間接的に遷移する．

以下，図 6 の解析表を用いて，入力トークン列 ‘p’‘q’‘q’‘p’‘$’ の解析例を示す．まず，(AN1) で初期状態 0 をスタックにプッシュする．次に，(AN3) でスタックの最上段の状態 0 と終端記

号 'p' により遷移先の候補 3（Base[0]+2×'p' = 3）を求める．(AN4) において Check[3]='p' で遷移先 3 が存在するため，(AN11) 以降において状態 3 の種類によりアクションを決定する．ここで，(AN11) において Base[3] ≥ 0 で遷移先が統一状態であるため，(AN12) で shift(3) を実行する．このように，提案手法では遷移先の状態が統一状態である場合，間接参照せずに状態遷移できる．同様に，状態 3 と終端記号 'q' により shift(5) を実行する．

次に，(AN3) で状態 5 における終端記号 'q' により，遷移先の候補 4（Base[5]+2×'q' = 4）を求め，(AN4) で遷移先の有無を確認する．Check[4]≠'q' で遷移先 4 が存在しないため，(AN5) で状態 5 の付属要素 6（5+1）の Check 値を参照して DR 規則 4 を取得し，(AN7) で Reduce(4) を呼び出す．このように，提案手法では遷移先の状態が存在しない場合，隣接する付属要素を参照することで DR 規則を取得できる．

Reduce(4) において，(RE1) で，規則 4 における左辺の非終端記号 E，右辺の長さ 1 を取得し，(RE2) で，スタックから 1 つの状態をポップする．ここで，(RE3) でスタックの先頭状態 3 から非終端記号 E による Goto 遷移先の候補 14（Base[3+1]+2×E）を求めるが，(RE4) で Check[14] ≠ E より遷移先の状態が存在しないため，(RE5) で非終端記号 E におけるデフォルトの遷移先 10（2×E）をスタックにプッシュする．

その後，shift(10)，reduce(4)，reduce(3)，shift(17)，reduce(2) を実行し，状態 8 において開始規則 1 を還元して解析を終了する．

4. 評　　価

本章では Bison [8]（version2.3）が生成する解析表を比較手法とし，解析時における解析時間，記憶領域について理論的，実験的評価を与える．

4.1 理論的評価

まず，解析時間について評価する．ただし，解析時間はトークン列を構文解析する時間とし，解析表の構築や入力トークン列の読み込みに要する時間を含めない．以下，提案手法における状態遷移の内訳を説明した後，状態遷移に要する計算量について，1 遷移あたりに参照する配列の要素数を遷移計算量として Bison が生成する解析表と比較する．

提案手法の解析表には，従来の LR 構文解析表における状態と 1 対 1 対応する統一状態と，統一状態へのポインタを保持する間接状態が存在する．提案手法における状態遷移は統一状態から次の統一状態までの状態遷移である．以下，統一状態から直接に統一状態へ遷移することを直接遷移，統一状態から間接状態のポインタを参照して統一状態へ遷移することを間接遷移と呼ぶ．提案手法の遷移式 (7) より，直接遷移で参照する配列の要素数は，配列 Base，Check の 2 要素である．また，間接遷移で参照する配列の要素数は，間接状態の Base 値を含めた 3 要素である．よって，直接遷移は間接遷移と比較して，間接状態の Base 値を参照する必要がない分，遷移計算量を抑制できる．

表 4　Bison の解析表を構成する配列
Table 4　Arrays of Bison's parse table.

配列名	機能説明	要素数
Base	各状態におけるアクションを定義するための基底値を管理する．	L
DefAct	各状態における DR 規則を管理する．	L
Check	解析表におけるエントリの有無を判定する値を管理する．	U（$\leq K$）
Table	解析表のエントリ管理する．	U（$\leq K$）
DefGoto	各非終端記号によるデフォルトの遷移先を管理する．	$\|N\|$
NBase	各非終端記号による Goto 関数を定義するための基底値を管理する．	$\|N\|$

一方，Bison の遷移式 (1) が参照する配列の要素数は，配列 Base，Check，Table の 3 要素であり，Bison の 1 遷移あたりに要する遷移計算量は，提案手法の間接遷移に要する遷移計算量とほぼ等しい．よって，直接遷移により状態遷移の総計算量を抑制できる提案手法は，Bison より解析時間を短縮できるといえる．

次に，解析表の記憶領域を，各配列に必要な要素数の和として評価する．評価の指標として，解析表に登録する文法における終端記号の数を $|T|$，非終端記号の数を $|N|$ とし，2 次元配列を用いた LR 構文解析表における状態数を L，2 次元配列の要素数を K（$K = L \times (|T|+|N|)$）とする．**表 4** に，Bison の解析表を構成する配列の機能説明と各配列に必要な要素数を示す．

Bison は，アクションの共有や DR 規則を定義することで，配列 Table に定義するアクション数を抑制するため，配列 Check，Table で使用する要素 U は K 以下となる．表 4 より，Bison の解析表を実現するために必要な記憶領域は $2\times(L+U+|N|)$ である．

一方，提案手法は，2 つの配列 Base，Check により LR 構文解析表を実現する．提案手法で定義した間接状態と還元状態は，LR 構文解析表のエントリを定義するための状態であり，ダブル配列上に定義する要素数は，付属要素とデフォルトの遷移先となる状態を除けば，Bison が配列 Table に定義する要素の数 U と一致する．ここで，付属要素の数は統一状態の数と一致するため L であり，デフォルトの遷移先となる状態の数は，開始規則の左辺を除く非終端記号の数と一致するため，$|N|-1$ である．よって，提案手法の解析表を実現するために各配列に必要な要素数の合計は $2\times(L+U+|N|-1)$ で Bison よりもつねに 2 つ少なく，各配列の 1 要素あたりの記憶領域が等しいと仮定すると，提案手法の記憶領域は Bison

と同等であるといえる．

ただし，提案手法の配列 Base，Check や Bison の配列 Check，Table 上には未使用である要素が含まれるため，いずれの手法も記憶領域が未使用要素の数だけ増加する．未使用要素を含めた記憶領域は，次節で実験的に評価する．

4.2　実験的評価

Bison との比較実験を Intel Core2 Duo 2.93 GHz，Fedora8 上で行った．実験では，ANSI C，Pascal，Ruby（version1.9）の文法，MySQL（version5.1）の文法，プロトコルパーサを生成する BinPAC[5] の文法から解析表をそれぞれ構築し，各文法に対応するトークン列 6.56 M 個，6.32 M 個，6.02 M 個，6.00 M 個，5.11 M 個をそれぞれ解析した．本実験において，提案手法の解析表は Bison が出力した解析表を提案手法の定義式に従って変換することで作成した．また，参考として 2 次元配列構造の LR 構文解析表をあわせて実験した．

表 5 に，実験に用いた各文法における規則数，文法中の終端記号の数 $|T|$，非終端記号の数 $|N|$，2 次元配列による LR 構文解析表における状態数 L，Bison が解析表を構築する時間（表中，構築時間），提案手法の解析表への変換時間（表中，変換時間）を示す．また，**表 6** に，解析時間，解析表に必要な記憶領域を示し，**表 7** に，提案手法と Bison における解析表に必要な各配列の使用要素数を示し，**表 8** に，提案手法と Bison における配列 Check の配列サイズ，未使用要素数，解析時における直接遷移回数，間接遷移回数を示す．以下，解析表に必要な記憶領域と解析時間について順に評価する．

まず，記憶領域について評価する．提案手法，Bison は表圧縮法の適用により，表 6 に示すように，2 次元の配列構造と比較して解析表に必要な記憶領域を大幅に抑制した．

前節でも述べたように，提案手法は各配列における使用要素数の合計を表 7 に示すように Bison よりもつねに 2 つ抑制できる．Ruby の文法による実験において，提案手法は，表 8 に示すように配列上の未使用要素数が Bison よりも減少したことにより，表 6 に示すように Bison が解析表に必要な記憶領域 46,880 Byte の約 85.3%にあたる記憶領域 39,968 Byte で解析表を実現した．

しかし，他の文法による実験では Bison よりも解析表に必要な記憶領域が増加した．これは，複数の配列を定義する Bison が，各配列における 1 要素あたりの記憶領域を表 7 に示すように提案手法よりも抑制できることと，提案手法における配列上の未使用要素数が表 8 に示すように Bison よりも増加したためである．以下，配列 1 要素あたりの記憶領域と未使用要素数について順に述べる．

ANSI C，Pascal，BinPAC の文法においては，規則数を表現するために必要なバイト幅

表 5　実験で用いた文法の詳細と解析表の構築時間
Table 5　Details of grammars and times of building parse tables.

	規則数	$\|T\|$	$\|N\|$	L	構築時間 (s)	変換時間 (s)
ANSI C	239	90	69	400	0.033	0.002
Pascal	255	67	135	410	0.040	0.002
Ruby	561	149	169	970	0.175	0.014
MySQL	2,364	589	826	4,035	1.574	0.094
BinPAC	219	121	58	496	0.080	0.002

表 6　解析時間と解析表に必要な記憶領域
Table 6　Parsing time and memory required.

	ANSI C	Pascal	Ruby	MySQL	BinPAC
解析時間（s）					
提案手法	1.33 (0.77)	0.67 (0.78)	0.70 (0.81)	0.84 (0.80)	0.26 (0.90)
Bison	1.73 (1.00)	0.86 (1.00)	0.86 (1.00)	1.05 (1.00)	0.29 (1.00)
2 次元配列	1.50 (0.87)	0.75 (0.87)	0.81 (0.94)	1.01 (0.96)	0.31 (1.07)
記憶領域（Byte）					
提案手法	8,660	5,940	39,968	309,660	10,616
Bison	7,724	3,778	46,880	213,982	9,252
2 次元配列	63,600	82,820	308,460	5,709,525	88,784

＊括弧内の数値は Bison を 1 としたときの割合

表 7　解析表に必要な各配列の使用要素数
Table 7　The number of used array elements.

	ANSI C	Pascal	Ruby	MySQL	BinPAC
提案手法					
Base	1,598 (2)	1,029 (2)	6,596 (2)	32,041 (4)	1,870 (2)
Check	1,598 (2)	1,029 (2)	6,596 (2)	32,041 (2)	1,870 (2)
合計	3,196	2,058	13,192	64,082	3,740
Bison					
Base	400 (2)	410 (2)	970 (2)	4,035 (4)	496 (2)
DefAct	400 (1)	410 (1)	970 (2)	4,035 (2)	496 (1)
Check	1,130 (2)	485 (2)	5,458 (2)	27,181 (2)	1,317 (2)
Table	1,130 (2)	485 (2)	5,458 (2)	27,181 (2)	1,317 (2)
NBase	69 (2)	135 (2)	169 (2)	826 (2)	58 (2)
DefGoto	69 (2)	135 (2)	169 (2)	826 (2)	58 (2)
合計	3,198	2,060	13,194	64,084	3,742

＊括弧内の数値は配列の 1 要素あたりに必要な記憶領域（Byte）

表 8　配列 Check の要素数と解析時の遷移回数（M 回）
Table 8　The number of Check array elements and the number of transition frequencies.

	ANSI C		Pascal		Ruby		MySQL		BinPAC	
	提案	Bison	提案	Bison	提案	Bison	提案	Bison	提案	Bison
配列サイズ	2,165	1,562	1,485	502	9,992	10,581	51,610	46,617	2,654	1,883
未使用要素数	567	432	456	17	3,396	5,123	19,569	19,436	784	566
直接遷移回数	53.24	-	23.36	-	22.13	-	29.88	-	10.09	-
間接遷移回数	6.59	59.83	4.03	27.39	5.36	27.49	3.58	33.45	1.12	11.21

は 1 バイトであり，Bison の DR 規則を管理する配列 DefAct の 1 要素あたりに必要な記憶領域は表 7 に示すように 1 バイトである．これに対し，提案手法は付属要素の Check 値を用いて DR 規則を定義するため，DR 規則の定義に必要な記憶領域が配列 Check の 1 要素あたりの記憶領域である 2 バイトとなり，Bison よりも解析表に必要な記憶領域が増加する．

未使用要素数が増加した原因としては，付属要素をダブル配列上に追加したことで，各状態における Base 値の影響範囲[13] が拡大し，各状態のダブル配列上における位置を決定する問題[14] が複雑化したことが考えられる．

最大で Pascal の文法において提案手法の記憶領域は Bison の約 1.57 倍に増加したが，提案手法は実験で用いた文法の解析表を数 kByte〜数 100 kByte 程度の記憶領域で実現した．

次に，解析時間について評価する．表 8 に示すように，Bison が状態遷移を行う際に必ず間接遷移するのに対し，提案手法は，大半の状態遷移を遷移計算量が比較的少ない直接遷移で行った．ANSI C の文法における実験では，総遷移回数 59.83 M 回のうち，約 89%にあたる 53.24 M 回の状態遷移を直接遷移により行い，表 6 より Bison に対して解析時間を約 23%短縮した．同様に，表 6 より，Pascal，Ruby，MySQL，BinPAC の文法においては，それぞれ 22%，19%，20%，10%解析時間を短縮した．

以上，提案手法は実用的な記憶領域で解析時間を Bison よりも短縮できることを示した．

5. おわりに

本論文では，ダブル配列の遷移を拡張することにより，状態遷移を高速に行える LR 構文解析表を実現する方法を提案した．Bison が LR 構文解析時において，すべての状態遷移を間接参照により実行するのに対し，提案手法は，状態遷移の大半において遷移式により求めた遷移先の状態へ直接に遷移でき，状態遷移に要する総計算量を抑制できる．

実験の結果，提案手法は Bison と比較して LR 構文解析に要する時間を 10〜23%削減した．提案手法を用いることで，従来手法よりも高速な LR 構文解析器を実現でき，LR 構文解析を利用するシステムの実行時間を短縮できるといえる．

今後の課題として，提案手法を一般化 LR 法[15),16)] に対応させ，提案手法の応用範囲を広げることがあげられる．

参 考 文 献

1) Aho, A.V., Sethi, R. and Ullman, J.D.: コンパイラ—原理・技法・ツール，サイエンス社 (2009).
2) Johnson, S.C.: YACC: Yet Another Compiler-Compiler, Computing Sience Technical Report 32, AT&T Bell Laboratories, Murray Hill, N.J. (1975).
3) Levine, J.R.: *Flex & Bison*, Oreilly & Associates Inc. (2009).
4) 蜂巣吉成，野呂昌満，張　漢明：データ型を考慮した軽量な XML 文書処理系の自動生成，コンピュータソフトウェア，Vol.19, No.5, pp.2–12 (2002).
5) Pang, R., Paxson, V., Sommer, R. and Peterson, L.: binpac: A yacc for Writing Application Protocol Parsers, *Proc.ACM SIGCOMM Internet Measurement Conference, IMC*, pp.289–300 (2006).
6) Aoe, J.: An Efficient Digital Search Algorithm by Using a Double-Array Structure, *IEEE Trans.Softw.Eng.*, Vol.15, No.9, pp.1066–1077 (1989).
7) Yata, S., Oono, M., Morita, K., Fuketa, M., Sumitomo, T. and Aoe, J.: A compact static double-array keeping character codes, *Information Processing and Management*, Vol.43, No.1, pp.237–247 (2007).
8) Free Software Fundaction: Bison—GNU parser generator. http//www.gnu.org/software/bison/
9) Corbett, R.: BYACC—BERKELEY YACC. http://invisible-island.net/byacc/byacc.html
10) 森公一郎：kmyacc—多言語対応 LALR パーサー生成系. http://www005.upp.so-net.ne.jp/kmori/kmyacc/
11) Dodd, C. and Maslov, V.: BtYacc: BackTracking Yacc. http://www.siber.com/btyacc/
12) 青江順一：ダブル配列による有限状態機械の効率的インプリメンテーション，電子情報通信学会論文誌 D，Vol.J70-D, No.4, pp.653–662 (1987).
13) 蔵満琢麻，松浦寛生，望月久稔：遷移先関数を拡張した効率的なパターン照合機械の設計と実装，電子情報通信学会論文誌 D，Vol.J92-D, No.3, pp.321–337 (2009).
14) 矢田　普，大野将樹，森田和宏，泓田正雄，吉成友子，青江順一：接頭辞ダブル配列における空間効率を低下させないキー削除法，情報処理学会論文誌，Vol.47, No.6, pp.1894–1902 (2006).
15) McPeak, S. and Necula, G.C.: Elkhound: A Fast, Practical GLR Parser Generator, *13th International Conference on Compiler Construction*, Vol.2985 of LNCS,

pp.73–88 (2004).
16) Johnstone, A., Scott, E. and Economopoulos, G.: The Grammar Tool Box: A Case Study Comparing GLR Parsing Algorithms, *Electronic Notes in Theoretical Computer Science*, Vol.110, No.31, pp.97–113 (2004).

(平成 21 年 12 月 19 日受付)
(平成 22 年 4 月 7 日採録)

（担当編集委員　喜田 拓也）

蔵満　琢麻（学生会員）

昭和 61 年生．平成 20 年大阪教育大学教育学部教養学科情報科学専攻卒業．平成 22 年同大学大学院修士課程修了．同年キヤノン IT ソリューションズ株式会社に入社，現在に至る．情報検索，パターン照合，自然言語処理に関する研究に従事．電子情報通信学会会員．

重越　秀美（学生会員）

昭和 62 年生．平成 21 年大阪教育大学教育学部教養学科情報科学専攻卒業．現在，同大学大学院修士課程在学中．情報検索に関する研究に従事．日本データベース学会学生会員．

望月　久稔（正会員）

昭和 44 年生．平成 5 年徳島大学工学部知能情報工学科卒業．平成 7 年同大学大学院博士前期課程修了．平成 10 年同大学院博士後期課程修了．博士（工学）．同年大阪府立工業高等専門学校電子情報工学科講師．平成 15 年大阪教育大学教育学部教養学科情報科学講座講師．平成 19 年大阪教育大学教育学部教養学科情報科学講座准教授．現在に至る．情報検索，自然言語処理，知識表現の研究に従事．電子情報通信学会，人工知能学会，自然言語処理学会各会員．