

ダブル配列による GLR 解析表の実現と SQL 構文解析の高速化

蔵満 琢麻[†] 重越 秀美[†] 望月久稔^{††}

[†], ^{††} 大阪教育大学 〒 582-8582 大阪府柏原市旭ヶ丘 4-698-1

E-mail: [†]{takuma,shige}@mugen.cc.osaka-kyoiku.ac.jp, ^{††}motizuki@cc.osaka-kyoiku.ac.jp

あらまし LR 構文解析は、C 言語などのプログラミング言語、SQL などの問い合わせ言語、情報抽出、テキストの誤り訂正、音声認識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 解析表の実現が求められる。本論文では、一般の文脈自由文法が取り扱える GLR 解析表をダブル配列の拡張により実現し、GLR 構文解析に要する時間を短縮する手法を提案する。SQL 構文解析実験の結果、提案手法は構文解析器生成系の一つである Bison と比較して GLR 構文解析に要する時間を約 22%抑制した。

キーワード LR パーサ, ダブル配列, GLR

An Imprementation of GLR Parsing Table Using Double-Array and Speeding Up SQL Parsing

Takuma KURAMITSU[†], Hidemi SHIGEKOSHI[†], and Hisatoshi MOCHIZUKI[†]

[†] Osaka Kyoiku University Asahigaoka 4-698-1, Kashiwara-shi, Osaka, 582-8582, Japan

E-mail: [†]{takuma,shige}@mugen.cc.osaka-kyoiku.ac.jp, ^{††}motizuki@cc.osaka-kyoiku.ac.jp

Abstract LR parsers are used widely, such as programing language compilers, SQL parsers, information extraction, grammatical error correction, and speech recognition. So it is important to implement efficient LR parsing tables. In this paper, we present an efficient implementation of GLR parsing table that can handle all context-free grammar with Double-Array that is extended transition function. Our experiment show that proposal method can decrease the SQL parsing time about 22% in comparison with parser generator Bison.

Key words LR parser, Double-Array, GLR

1. はじめに

LR 解析¹⁾は、シフト還元構文解析の一種で、あらかじめ文法に対して前処理を行って解析表を作成することで高速な構文解析を実現する。LR 解析は、C 言語などのプログラミング言語のコンパイラ、SQL パーサ²⁾、情報抽出³⁾、テキストの誤り訂正⁴⁾、音声認識⁵⁾など、様々な分野に用いられており、時間的、領域的に効率的な LR 解析表の実現法が求められる。現在、構文解析器生成系として一般的に用いられる Bison⁶⁾は、行置換法¹⁾を利用して、記憶領域のコンパクト性と解析速度の高速性をあわせもつ LR 解析表を生成する。

本論文では、トライ構造を効率的に実現できるダブル配列⁷⁾⁸⁾による LR 解析表⁹⁾を拡張して、一般の文脈自由文法が取り扱える GLR 解析表を実現する方法を提案し、GLR 構文解析時間の短縮を図る。以下、2 章で LR 解析表について説明し、3 章で提案手法について述べる。その後、4 章で提案手法を評価し、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 解析表

本章では、LR 解析表と GLR 解析表の構成について述べた後、Bison が生成する解析表とダブル配列による LR 解析表について述べる。以下、LR 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 解析表と GLR 解析表の構成

本節では、LR 解析表の構成について説明した後、GLR 解析表と 2 次元配列構造による解析表について述べる。

まず、LR 解析表の構成¹⁾について述べる。LR 解析表は、Action 関数と Goto 関数からなる。Action 関数は、状態 x と終端記号 a を引数とし、 $\text{shift}(t)$ 、 $\text{reduce}(I_g)$ 、 accept 、 error のいずれかのアクションを返す関数である。ここで、 $\text{shift}(t)$ 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、 $\text{reduce}(I_g)$ は規則 g を還元するアクションを表す。Goto 関数は、 $\text{reduce}(I_g)$ 実行時に呼び出される関数で、規則 g

- 1) $S' \rightarrow S \$$ 2) $S \rightarrow p S p$ 3) $S \rightarrow E q$
 4) $S \rightarrow q$ 5) $E \rightarrow q$ 6) $E \rightarrow p$

図1 文法 Q

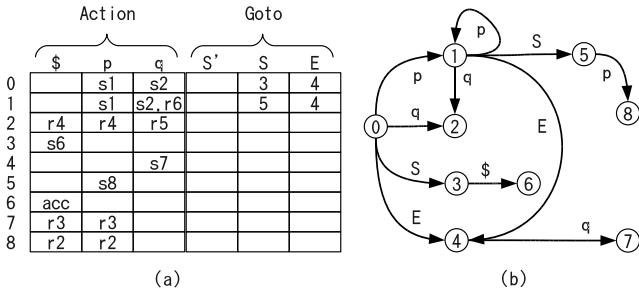


図2 GLR 解析表と状態遷移図

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。

次に、GLR 解析表¹⁰⁾ について述べる。GLR 法は、すべての文脈自由文法を取り扱えるように LR 法を拡張した手法で、Action 関数における 1 つのエントリに複数のアクションを定義する。以下、GLR 解析表の Action 関数において、複数の要素をもつアクション集合を競合アクションと呼ぶ。

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 に示す文法 Q を登録した 2 次元配列構造による GLR 解析表を図 2(a) に示す。図 2(a) 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。例えば、 $Action(0, 'p')=shift(1)$, $Action(2, 'q')=reduce(5)$, $Goto(1, S)=5$ である。 $Action(1, 'q')$ は競合アクションで、複数のアクション $shift(2)$, $reduce(6)$ を含む。

また、図 2(a) に対応する状態遷移図を図 2(b) に示す。図 2(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 次元の配列構造を用いて解析表を実現すると、解析時に次のアクションを $O(1)$ で決定できるが、図 2(a) 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⁾。

2.2 Bison の解析表

行置換法¹⁾ は、2 次元の配列構造を 1 次元の配列構造へ写像し、エントリ数に応じた記憶領域に圧縮する手法である。Bison では、1 次元配列 Base, DefAct, NBase, DefGoto, Check, Table, ConfP, ConfList を用いて^(注1)、2 次元配列を圧縮し、記憶効率がよい GLR 構文解析表を実現する。

Bison は、配列 Base, Check, Table を用いて、 $Action(x, a)$ を式 (1) により配列 Table 上に写像し、配列 NBase, Check, Table を用いて、 $Goto(x, A)$ を式 (2) により配列 Table 上に写

	0	1	2	3	4	5	6	7	8
Base	-1	1	4	5	6	8	-3	-3	-3
DefAct	0	0	r4	0	0	0	r1	r3	r2

	0	1	2	3	4	5	6	7	8	9
Check	p	q	p	q	1	\$	q		q	p
Table	s1	s2	s1	s2	5	s6	r5		s7	s8
ConfP	0	0	0	1	0	0	0		0	0

	0	1	2
NBase	-3	3	-3
DefGoto	-1	3	4

	0	1	2
ConfList	0	r6	0

図3 Bison の GLR 解析表

像する⁶⁾。GLR 解析表を実現するには、配列 Table と同じサイズの配列 ConfP を用いて、アクション集合のリストを格納する配列 ConfList へのリンクを定義することで、競合アクションを定義する⁶⁾。

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

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

また、Bison は、Action 関数において各状態における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、状態数分の要素をもつ配列 DefAct に格納し、Goto 関数において各非終端記号における最頻出の遷移先をデフォルトの遷移先として、非終端記号数分の要素をもつ配列 DefGoto に格納することで、配列 Table に定義するエントリ数を削減する⁶⁾。

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 2(a) に対して Bison の表圧縮法を適用した GLR 解析表を図 3 に示す。図 3 中、配列 Check, Table, ConfP において空白の要素は未使用の要素を表す。

例えば、 $Action(0, 'p')=shift(1)$ は、式 (1) により、 $Table[Base[0]+'p']=Table[0]$ に格納され、 $Goto(1, S)=5$ は、式 (2) により、 $Table[NBase[S-|T|]+1]=Table[4]$ に遷移先 5 へのポインタとして格納される。また、競合アクションである $Action(1, 'q')=\{shift(2), reduce(6)\}$ において、 $shift(2)$ は式 (1) により、 $Table[Base[1]+'q']=Table[3]$ に格納され、 $reduce(6)$ はアクション集合の要素として $ConfList[1]$ に格納される。このとき、配列 ConfList へのリンクとして、 $ConfP[3]$ には配列 ConfList の添字 1 が格納される。また、状態 7, 8 における DR 規則はそれぞれ規則 3, 規則 2 で、 $DefAct[7]$, $DefAct[8]$ にそれぞれ格納され、非終端記号 S, E におけるデフォルトの遷移先はそれぞれ状態 3, 状態 4 で、 $DefGoto[1]$, $DefGoto[2]$ にそれぞれ格納される。

Bison が生成する解析表は、表のエントリ数に応じた記憶領域で実現できるが、Action 関数や Goto 関数による遷移先を取得する際に、式 (1) や式 (2) による演算を要する。

2.3 ダブル配列による LR 解析表

本節では、提案手法で用いるダブル配列⁸⁾ について説明した後、ダブル配列による LR 解析表⁹⁾ について説明する。

まず、ダブル配列について説明する。ダブル配列は 2 本の配列 Base と Check を用いて、木構造の状態遷移表を効率的に実

(注1): Bison2.4 におけるプログラムソース中の配列名はそれぞれ、yycompact, yydefact, yygoto, yydefgoto, yycheck, yytable, yyconfp, yyconfll である。

現する手法で、配列の添字が状態番号に対応し、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。

$$\begin{cases} t = \text{Base}[x] + a \\ \text{Check}[t] = a \end{cases} \quad (3)$$

ここで、2.2 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考える。以下、1 回の状態遷移で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。

青江らは、ダブル配列を一般の有限オートマトンにおける状態遷移表に拡張するため、状態集合 $\{t_1 \dots t_m\}$ を、1 つの状態に統合して取り扱い、入次数が 2 以上となる状態を擬似的に定義する手法¹¹⁾ を提案した。この手法は、状態集合 $\{t_1 \dots t_m\}$ 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。

拡張ダブル配列を用いることで、LR 解析表におけるシフトアクションと Goto 関数における遷移先を定義できる。筆者らはダブル配列の遷移をさらに拡張し、ダブル配列上に還元アクションを定義することで LR 解析表を実現する手法を提案した⁹⁾。以下、ダブル配列による LR 解析表 (以下、DALR) について説明する。

DALR では、状態 x において終端記号 a によるアクションが還元である場合、状態 x から終端記号 a による遷移先 t を作成し、遷移先 t の Base 値に還元規則の規則番号を負値で格納することで還元を定義する。以下、還元規則の規則番号を管理する状態を還元状態と呼ぶ。DALR は、還元状態、統一状態、間接状態を区別するため、統一状態の Base 値には正値を格納し、間接状態の Base 値には統一状態へのポインタとして、状態番号に規則数を足し合わせた値を負値で格納する。

また、ダブル配列上に定義する状態数を抑制するため、開始規則における左辺の非終端記号を除く非終端記号 A によるデフォルトの遷移先を $2A$ と定義し、状態 x における DR 規則を、隣接要素 $x+1$ の Check 値に格納する。以下、状態 x の隣接要素 $x+1$ を、状態 x の付属要素と呼ぶ。

DALR は、状態 x から終端記号 a による遷移先 t を、状態 x の Base 値を用いて式 (4) により定義し、非終端記号 A による遷移先 t を、状態 x の付属要素の Base 値を用いて式 (5) により定義する。

$$\begin{cases} t = \text{Base}[x] + 2a \\ \text{Check}[t] = a \end{cases} \quad (4) \quad \begin{cases} t = \text{Base}[x+1] + 2A \\ \text{Check}[t] = A \end{cases} \quad (5)$$

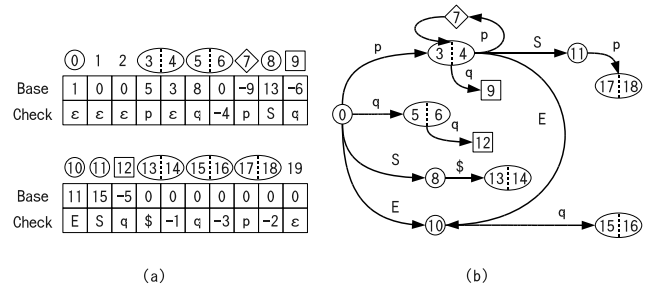


図 4 DALR と状態遷移図

文法 Q を登録した DALR を図 4 (a) に示し、対応する状態遷移図を図 4 (b) に示す。ただし、アクションが競合する場合は還元を優先して定義する。図 4 において、付属要素が存在しない統一状態を丸枠で表し、付属要素が存在する統一状態を楕円形で表す。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、図 4 (a) において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。

例えば、図 2 の Action(0, 'p') = shift(1) に該当するアクションは、図 4 の統一状態 0 から終端記号 'p' による状態遷移であり、式 (4) により統一状態 0 から統一状態 3 ($\text{Base}[0] + 2 \times 'p'$) への遷移として定義する。また、図 2 の状態 7, 8 に該当する図 4 の統一状態 15, 17 は、DR 規則の規則番号 3, 2 をそれぞれの付属要素 16, 18 の Check 値に負値でもつ。

図 2 の Action(1, 'q') における競合アクションのうち、reduce(6) 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 9 ($\text{Base}[3] + 2 \times 'q'$) を作成し、Base 値に還元規則の規則番号 6 を負値で格納することで定義する。ここで、統一状態 3 から終端記号 'q' による遷移先として還元状態 9 が存在するため、図 2 の Action(1, 'q') における競合アクションのうち、shift(2) に該当するアクションを定義できない。DALR は、アクションを集合として管理できず、競合アクションにおけるすべての要素を定義できないため、GLR 法に適用できない。

3. ダブル配列による GLR 解析表

本章では、ダブル配列による LR 解析表を拡張し、GLR 解析表を実現することで、GLR 構文解析に要する時間を短縮する手法を提案する。以下、競合アクションを定義するため、ダブル配列上に競合アクションを管理する競合状態を新たに定義する方法について述べる。その後、ダブル配列上の遷移計算量をさらに抑制する方法と付属要素の削減方法について述べ、状態の判別方法について述べた後、提案手法における Action 関数と Goto 関数の定義について述べる。

3.1 競合アクションの定義

本節では、競合アクションをダブル配列上に定義するため、アクション集合のリストを管理する配列 ConfList と、競合状態の定義について述べる。

提案手法は、状態 x が終端記号 a により n 個の要素を含む競合アクション $\{act_1, act_2, \dots, act_n\}$ をもつ場合、図 5 に示すように、 $act_1, act_2, \dots, act_n$ を配列 ConfList 上の連続する領域、

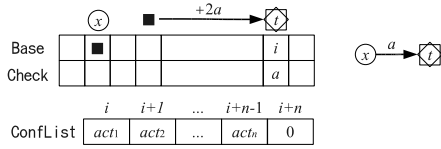


図 5 提案手法における競合アクションの定義

ConfList[i], ConfList[i + 1], ..., ConfList[i + n - 1] にそれぞれ格納し, ConfList[i + n] に集合の終点を表す 0 を格納する. このとき, シフトアクションにおいては, 遷移先の状態番号を負値で格納し, 還元アクションにおいては還元規則の規則番号を正值で格納する. また, 状態 x から終端記号 a による遷移先 t を作成し, 配列 ConfList へのリンクとしてアクション集合の始点となる添字 i を状態 t の Base 値に格納することで, 競合アクションを定義する. 以下, 配列 ConfList へのリンクを管理する状態 t を競合状態と呼ぶ.

3.2 遷移計算量の抑制と付属要素の削減

本節では, 統一状態と間接状態に等価な情報をもたせることで, ダブル配列上の遷移計算量を抑制する方法について述べた後, DR 規則を Base 値で管理する状態を新たに定義することで, 付属要素を削減する方法について述べる.

まず, ダブル配列上の遷移計算量を抑制する方法について述べる. 式 (4), 式 (5) による遷移表では, 同じ Base 値を持つ状態は等しい遷移を持つ. 例えば, 図 6 に示すように, 状態 t が終端記号 a により状態 u への遷移をもち, 状態 t と状態 t' の Base 値が等しい場合, $u = \text{Base}[t] + 2a$ かつ $u = \text{Base}[t'] + 2a$ が成り立ち, 状態 t と状態 t' は遷移先として状態 u を共有する.

そこで, 間接状態 t' の Base 値に, 統一状態 t へのリンクではなく, 統一状態 t の Base 値を格納する. また, 状態 t が付属要素をもつ場合, 状態 t' にも同じ付属要素を付加することで状態 t と状態 t' を等価な状態として定義する.

これにより, 解析時において, 統一状態, 間接状態を区別せずに状態遷移できるため, 遷移計算量をさらに抑制できる. 以下, 統一状態, 間接状態をシフト状態と呼ぶ.

次に, 付属要素を削減し, 解析表で使用する要素数を抑制する方法を述べる. DR 規則以外のアクションが存在しない状態は, Action 関数を定義するための Base 値を使用しない. そこで, DR 規則以外のアクションが存在しない状態を DR 状態として定義し, Base 値に DR 規則の規則番号を格納する. DR 状態においては, 非終端記号による遷移先が存在しない場合, 付属要素を付加する必要がなくなるため, ダブル配列上に定義する付属要素の数を削減できる.

3.3 状態の判別

本節では, ダブル配列上に定義した各状態を判別する方法について述べた後, 提案手法の例を示す.

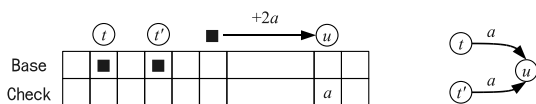


図 6 遷移先の共有

表 1 各状態のまとめ

SFlag	状態の種類	Base 値	付属要素付加条件
0	シフト状態	Action 関数の基底値	(1) (2)
1	DR 状態	DR 規則の規則番号	(1)
2	還元状態	還元規則の規則番号	-
3	競合状態	ConfList へのリンク	-

提案手法のダブル配列上にはシフト状態, DR 状態, 還元状態, 競合状態を含む 4 種類の状態が存在する. そこで, 各状態を区別するため, 状態判別用のフラグを管理する配列 SFlag を設け, シフト状態, DR 状態, 還元状態, 競合状態における SFlag の要素に, 0, 1, 2, 3 をそれぞれ格納することで各状態を区別する. また, これらの状態と付属要素を区別するため, 付属要素であることを示すフラグを管理する配列 AFlag を設け, 付属要素における AFlag の要素に 1 を格納し, それ以外の要素には 0 を格納する.

ここで, 状態 x における付属要素の付加条件 (1), (2) を以下に示し, 提案手法におけるダブル配列上の状態を表 1 にまとめる. また, 状態別のアクション決定方法を表 2 に示し, Goto 関数による遷移先の決定方法を表 3 に示す.

条件 (1): 状態 x に非終端記号による遷移先が存在する.

条件 (2): 状態 x に DR 規則が存在する.

図 7(a) に文法 Q を登録した提案手法の GLR 解析表を示し, 図 7(b) に状態遷移表を示す. 図 7 において, 付属要素が存在しないシフト状態を丸枠で表し, 付属要素が存在するシフ

表 2 状態別のアクション決定方法

状態 x の種類	遷移先 $t = \text{Base}[x] + 2a$ の有無	アクションの選択	Action(x, a)
状態 x がシフト状態 SFlag[x] = 0	遷移先 t が存在する Check[t] = a	遷移先 t がシフト状態, または DR 状態 SFlag[t] = 0 SFlag[t] = 1	状態 t へシフト shift(t)
		遷移先 t が還元状態 SFlag[t] = 2	規則 Base[t] を還元 reduce (Base[t])
		遷移先 t が競合状態 SFlag[t] = 3	配列 ConfList に Base[t] を始点として格納したアクション集合
状態 x が DR 状態 SFlag[x] = 1	遷移先 t が存在しない Check[t] $\neq a$	DR 規則が存在する AFlag[x] = 1 & Check[$x+1$] > 0	DR 規則を還元 reduce (Check[$x+1$])
		DR 規則が存在しない AFlag[x] = 0 Check[$x+1$] = 0	error
状態 x が DR 状態 SFlag[x] = 1	遷移先 t が存在しない Check[t] $\neq a$	DR 規則が開始規則 Base[x] = 1	accept
		DR 規則が開始規則ではない Base[x] $\neq 1$	DR 規則を還元 reduce (Base[x])

表 3 Goto 関数による遷移先の決定方法

付属要素の有無	遷移先 $t = \text{Base}[x+1] + 2a$ の有無	Goto(x, A)
状態 x に付属要素が存在する AFlag[$x+1$] = 1	遷移先 t が存在する Check[t] = a	遷移先 t
	遷移先 t が存在しない Check[t] $\neq a$	デフォルトの遷移先 $2a$
状態 x に付属要素が存在しない AFlag[$x+1$] = 0		デフォルトの遷移先 $2a$

ト状態を楕円形で表す．四角枠と菱形枠により囲んだ状態 13 は競合状態，丸枠と四角枠により囲んだ状態 2, 14, 15 は DR 状態を表す．

図 2 の Action(1, 'q') における競合アクション {shift(2), reduce(6)} のうち shift(2) に該当するアクションは，シフト状態 3 からシフト状態 5 への状態遷移である．提案手法は，shift(5) を定義するため，ConfList[1] に遷移先の状態番号 5 を負値で格納し，reduce(6) を定義するため，ConfList[2] に規則番号 6 を格納する．また，シフト状態 3 からの遷移先として作成した競合状態 13 (Base[3]+2×'q') の Base 値に，アクション集合の始点となる添字 1 を格納することで状態 3 における競合アクションを定義する．

図 7 の DR 状態 14, 15 は，図 2 における状態 7, 8 に該当する状態で，Base 値に DR 規則の規則番号 3, 2 をそれぞれ格納する．図 4 の DALR における統一状態 15, 17 が付属要素を用いて DR 規則を定義する必要があるのに対し，提案手法の DR 状態においては，付属要素を定義することなく DR 規則を定義でき，解析表で使用する要素数を抑制できる．

3.4 解析表の参照方法

本節では，提案手法における解析表の参照方法として，Action 関数と Goto 関数を定義する．

まず，図 8 で定義する Action 関数について説明する．提案手法は，状態 x における終端記号 a によるアクションを，状態 x の種類，遷移先 t の有無，遷移先 t の種類によって決定する．図 8 に示す Action 関数は，手順 1 で状態 x における SFlag の値から状態 x の種類を判定し，DR 状態であれば，還元，または accept を返し，シフト状態であれば手順 2 で遷移先 t の有無を判定する．手順 2 において遷移先 t が存在する場合は，手順 2-1 で SFlag の値から遷移先 t の種類を判定して実行するアクションを表 2 に基づいて決定し，遷移先 t が存在しない場合は，手順 2-2 でデフォルトの還元アクション，または error を返す．

Action 関数の例として，図 7 における Action(0, 'p'), Action(3, 'q'), Action(14, '\$') について説明する．

まず，式 (4) によるシフトアクションの例として Action(0, 'p') について述べる．手順 1 により SFlag[0]=0 で状態 0 がシフト状態であるため，手順 2 により遷移先の有無を確認する．Base[0]+2×'p'=3, Check[3]='p' で，遷移先 3 が存在するた

Action(x, a)

引数：状態 x と終端記号 a 戻り値：アクション

手順 1：状態 x の種類の判定

SFlag[x]=1 で状態 x が DR 状態である場合，DR 規則の規則番号 Base[x] が開始規則 1 であれば accept を返し，そうでなければ reduce(Base[x]) を返す．SFlag[x]=0 で状態 x がシフト状態である場合，手順 2 を実行する．

手順 2：遷移先 t の有無の確認

遷移先 t (Base[x]+2 a) を計算し，Check[t]= a で遷移先 t が存在するときは手順 2-1, Check[t]≠ a で遷移先 t が存在しないときは手順 2-2 を実行する．

手順 2-1：遷移先 t の種類によるアクションの決定

SFlag[t]=0, または SFlag[t]=1 で遷移先 t がシフト状態か DR 状態であれば shift(t) を返す．SFlag[t]=2 で遷移先 t が還元状態であれば reduce(BASE(t)) を返す．SFlag[t]=3 で遷移先 t が競合状態であれば，競合アクションとして，配列 ConfList に Base[t] を始点として格納したアクション集合を返す．

手順 2-2：付属要素 $x+1$ によるデフォルトの還元アクション

AFlag[$x+1$]=1 かつ Check[$x+1$] > 0 で状態 x に DR 規則 Check[$x+1$] が存在する場合，reduce(Check[$x+1$]) を返す．そうでなければ error を返す．

図 8 関数 Action

め，手順 2-1 で遷移先 3 の種類によりアクションを決定する．SFlag[3]=0 で，状態 3 がシフト状態であるため，shift(3) を返す．このように，提案手法では，遷移先へのポインタを参照することなく，シフトアクションによる遷移先を Base 値と遷移種の内部表現値の和として直接設定できる．

次に，競合アクションの例として Action(3, 'q') について述べる．手順 1 により SFlag[3]=0 で状態 3 がシフト状態であり，手順 2 により Base[3]+2×'q'=13, Check[13]='q' で，遷移先 13 が存在するため，手順 2-1 で遷移先 13 の種類によりアクションを決定する．SFlag[13]=3 で，状態 13 が競合状態であるため，Base[13]=1 を始点として配列 ConfList に格納したアクション集合 {shift(5), reduce(6)} を返す．

最後に，DR 状態における還元アクションの例として Action(14, '\$') について述べる．手順 1 により SFlag[14]=1 で状態 14 が DR 状態であるため，DR 規則の規則番号 Base[14]=3 を取得し，reduce(3) を返す．

次に，図 9 で定義する Goto 関数について説明する．提案手法は，状態 x における非終端記号 A による遷移先を，状態 x における付属要素の有無や，遷移先 t の有無によって決定する．図 9 に示す Goto 関数は，状態 x に付属要素が存在し，式 (5) による遷移先 t が存在する場合は t を返し，そうでない場合はデフォルトの遷移先 2 A を返す．

Goto 関数の例として，図 7 における Goto(3, S), Goto(3, E) について説明する．

まず，式 (5) による遷移例として Goto(3, S) について述べる．AFlag[3+1]=1 で状態 3 に付属要素 4 が存在し，Base[4]+2×S=9, Check[9]=S で，式 (5) による遷移先 9 が

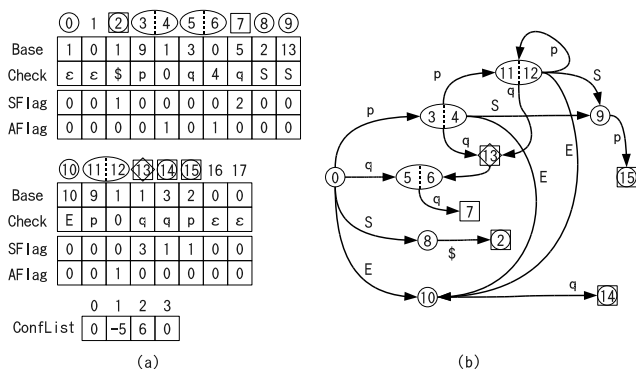


図 7 提案手法の GLR 構文解析表と状態遷移図

Goto(x, A)

引数：状態 x と非終端記号 A 戻り値：遷移先

AFlag[$x+1$]=1 で状態 x に付属要素が存在し、 $t=Base[x+1]+2A$ 、Check[t]= A で遷移先 t が存在する場合は t を返す。そうでなければデフォルトの遷移先 $2A$ を返す。

図 9 関数 Goto

表 4 Bison の GLR 解析表を構成する配列

配列	機能説明
Base	Action 関数を定義するための Base 値を管理する。
DefAct	DR 規則を管理する。
NBase	Goto 関数を定義するための Base 値を管理する。
DefGoto	デフォルトの遷移先を管理する。
Check	エントリの有無を判定する値を管理する。
Table	解析表のエントリ管理する。
ConfP	配列 ConfList へのリンクを管理する。
ConfList	競合アクションの集合を管理する。

存在するため、遷移先として状態 9 を返す。このように、提案手法では、遷移先へのポインタを参照することなく、Goto 関数による遷移先を Base 値と遷移種の内部表現値の和として直接設定できる。

次に、デフォルトの遷移例として Goto(3, E) について述べる。AFlag[3+1]=1 で状態 3 に付属要素 4 が存在し、Base[4]+2×E=11、Check[11]≠E で、式 (5) による遷移先が存在しないため、デフォルトの遷移先として状態 10(2×E) を返す。

ダブル配列上で GLR 解析表を実現することで、状態 0 から状態 3 への式 (4) による遷移や、状態 3 から状態 9 への式 (5) による遷移において、遷移先へのポインタを参照することなく状態遷移できるため、式 (1) や式 (2) による状態遷移よりも遷移計算量を抑制でき、GLR 解析の高速化が図れる。

4. 評価実験

提案手法を評価するため、Bison2.4⁶⁾ との比較実験を Intel Core2 Duo 2.93GHz、Fedora8 上で行った。実験では、SQL 文法のサブセット¹²⁾ から GLR 解析表を構築し、約 26MB の SQL クエリを解析した。以下、各手法の実装について述べた後、解析時間と解析表に要する記憶領域について、それぞれ評価する。

まず、Bison の実装⁶⁾ について述べる。Bison は複数の 1 次元配列を用いて GLR 解析表を実現する。表 4 に Bison の GLR 解析表を構成する配列の機能説明を示す。

Bison は、登録する文法の規則数、解析表の状態数、配列 Table の配列サイズから、各配列における 1 要素あたりの記憶領域を定める。本実験において、Bison の配列 Base、DefAct、NBase、DefGoto、Check、Table、ConfList における 1 要素あたりの記憶領域は 2Byte であり、配列 ConfP における 1 要素あたりの記憶領域は 1Byte であった。

表 5 SQL 文法詳細

文法数	終端記号数	非終端記号数	競合アクション数
304	255	73	61

表 6 解析実験結果

	提案手法	NDR	Bison
解析時間	1.03 (0.78)	1.04 (0.79)	1.32 (1.00)
記憶領域	15,516 (1.30)	17,424 (1.46)	11,933 (1.00)

* 括弧内の数値は Bison を 1 としたときの割合

表 7 提案手法における遷移回数の内訳

総遷移回数	DA 遷移	競合時遷移	デフォルト遷移
13,754,369	10,641,408	16,384	3,096,577

次に、提案手法の実装について述べる。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。本実験において、提案手法では、1 要素あたり 4Byte の配列 DA を用いて、各状態の Base 値、SFlag 値、AFlag 値、Check 値を 1 つの配列上で定義した。

次に、実験結果から、解析時間、解析表に必要な記憶領域について順に評価する。ここで、提案手法における DR 状態の有効性を示すため、DR 状態を定義しない手法(以下、NDR)をあわせて実験した。

実験に用いた文法の詳細を表 5 に示し、解析時間(秒)、解析表に必要な記憶領域(Byte)を表 6 に示す。また、表 7 に、解析時における提案手法の遷移回数の内訳として、遷移式(4)、(5)によりダブル配列上で状態遷移した回数(表中、DA 遷移)、アクション競合時の遷移回数、デフォルトの遷移回数を示し、表 8 に、解析表を構成する各配列に必要な要素数を示し、表 9 に、提案手法と NDR におけるシフト状態、DR 状態、還元状態、競合状態、付属要素の要素数をそれぞれ示す。

まず、解析時間について評価する。提案手法における遷移式(4)、(5)によるダブル配列上の状態遷移は、Bison における状態遷移よりも配列 Table を参照しない分遷移計算量を抑制できる。表 7 に示すように、提案手法は大半の状態遷移を遷移計算量の比較的少ない DA 遷移により行った。これにより提案手法は、表 6 に示すように Bison の解析時間 1.32 秒に対して、約 78%の時間にあたる 1.03 秒で SQL クエリを構文解析した。また、表 6 に示すように、提案手法は DR 状態を作成しない手法 NDR とほぼ同等の解析時間であった。これにより、DR 状態を付加することによる解析時間に対するオーバーヘッドは発生しないことが分かる。

次に、記憶領域について評価する。以下、各手法における配列上の使用要素数について述べた後、未使用要素を含めた記憶領域について評価する。

提案手法が定義した各状態は、GLR 構文解析表のエントリを定義するための状態であり、ダブル配列上で使用する要素数は、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 上で使用する要素の数と一致する。表 9 より、シフト状態、DR 状態、還元状態、競合状態の合計は 1,470 個であり、デフォルトの遷移先の状態数 72 個(開始規則における

表 8 解析表を構成する各配列に必要な要素数

	配列サイズ	使用要素数	未使用要素数
提案手法			
DA	3,787 (15,148)	2,260	1,527
ConfList	184 (368)	184	-
領域合計	(15,516)	(9,408)	(6,108)
NDR			
DA	4,264 (17,056)	2,558	1,706
ConfList	184 (368)	184	-
領域合計	(17,424)	(10,600)	(6,824)
Bison			
Base	630 (1,260)	630	-
DefAct	630 (1,260)	630	-
NBase	73 (146)	73	-
DefGoto	73 (146)	73	-
Check	1,775 (3,550)	1,398	377
Table	1,775 (3,550)	1,398	377
ConfP	1,775 (1,775)	1,398	377
ConfList	123 (246)	123	-
領域合計	(11,933)	(10,048)	(1,885)

* 括弧内の数値は必要な記憶領域 (Byte)

表 9 提案手法における状態の内訳

	シフト状態	DR 状態	還元状態	競合状態	付属要素
提案手法	1,078	316	15	61	790
NDR	1,394	0	15	61	1,088

左辺の非終端記号を除く非終端記号の数)を差し引いた 1,398 個は,表 8 より, Bison における配列 Table の使用要素数と一致する.

ここで, Bison が配列 Table の全使用要素に対して,競合アクションを管理する配列 ConfList へのリンクを配列 ConfP により定義するのに対し,提案手法では,競合状態により競合アクション数に応じた要素数で配列 ConfList へのリンクを定義できるため,提案手法は競合アクションを定義するために必要な使用要素数を Bison よりも抑制できる.表 8 に示すように, Bison の配列 ConfP における使用要素は 1,398 個で,表 9 に示すように,提案手法の競合状態は 61 個であった.

しかし,提案手法におけるデフォルトの還元規則や Goto 関数の Base 値を定義するための付属要素は,表 9 より 790 個で,表 8 に示す Bison の配列 DefAct や NBase における使用要素数の合計 703 個と比較すると 87 個増加した.これは,提案手法が等価なシフト状態を作成する際に付属要素も合わせて複製するためである.これにより,表 8 に示すように, NDR が各配列で使用した領域の合計 10,600Byte は, Bison が使用した領域の合計 10,048Byte よりも増加したが,提案手法は使用した領域の合計を 9,408Byte に抑制した.これは,提案手法が DR 状態の定義により,付属要素数を NDR よりも抑制できるためである.表 9 に示すように,提案手法の付属要素数は 790 個であり,提案手法は NDR における付属要素数 1,088 個の約 27%にあたる 298 個の付属要素を削減した.

次に,配列上の未使用要素を含めた記憶領域について述べる.

表 8 より,ダブル配列上の未使用要素が Bison における解析表の未使用要素よりも増加したことにより,提案手法の記憶領域は表 6 に示すように Bison の約 1.30 倍に増加したが, GLR 解析表を実現するために必要な記憶領域は,未使用要素を含めても約 16KByte 程度であった.

5. おわりに

本論文では,ダブル配列を拡張して GLR 解析表を実現することで, GLR 解析の状態遷移に要する計算量を抑制し, GLR 法による SQL の構文解析時間を Bison と比較して約 22%抑制した.

今後の課題として,ダブル配列上のガベージコレクション¹³⁾を提案手法に実装し,未使用要素の増加を抑制することが挙げられる.

文 献

- 1) A. V. Aho, R. Sethi and J. D. Ullman: “コンパイラ I 原理・技法・ツール”, サイエンス社 (1990).
- 2) K.-G. Doh, H. Kim and D. A. Schmidt: “Abstract Parsing: Static Analysis of Dynamically Generated String Output Using LR-Parsing Technology”, Vol. 5673 of LNCS, pp. 256–272, Springer Berlin, Heidelberg (2009).
- 3) 江里口善生, 木谷強: “富田一般化 LR パーザを用いた情報抽出”, 情報処理学会論文誌, **38**, 1, pp. 44–54 (1997).
- 4) M. Shishibori, S. S. Lee, M. Oono and J. Aoe: “Improvement of the LR parsing table and its application to grammatical error correction”, Information Sciences, **148**, pp. 11–26 (2002).
- 5) 秋葉友良, 伊藤克巨: “LR 表縮退法の提案と自然言語処理および音声認識への応用”, 信学技報, NLC101-40, pp. 21–28 (2001).
- 6) Free Software Fundaction: “Bison - gnu parser generator”, <http://www.gnu.org/software/bison/>.
- 7) J. Aoe: “An efficient digital search algorithm by using a double-array structure”, IEEE Transactions on Software Engineering, **15**, 9, pp. 1066–1077 (1989).
- 8) S. Yata, M. Oono, K. Morita, M. Fuketa, T. Sumitomo and J. Aoe: “A compact static double-array keeping character codes”, Information Processing and Management, **43**, 1, pp. 237–247 (2007).
- 9) 蔵満琢麻, 重越秀美, 望月久稔: “ダブル配列による遷移表を用いた LR 解析における shift・goto 操作の高速化”, 第 8 回情報科学技術フォーラム (FIT2009), pp. 117–118 (2009).
- 10) 田中穂積: “自然言語解析の基礎”, 産業図書 (1989).
- 11) 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, **J70-D**, 4, pp. 653–662 (1987).
- 12) J. R. Levine: “Flex & Bison”, Orelly & Associates Inc. (2009).
- 13) 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, **47**, 6, pp. 1894–1902 (2006).