ダブル配列によるGLR解析表の設計

蔵満 琢麻† 重越 秀美† 望月 久稔†
†大阪教育大学

1 はじめに

GLR解析[1]はコンピュータの様々な場面に利用されており，時間的，領域的に効率的なGLR解析表の実現法が求められる．

解析表の効率的な実現方法として，デフォルトのアクションや遷移先を定義することで解析表に記述するエントリの数を抑制する手法や，複数の1次元配列を用いて2次元の解析表を圧縮する行置換法がAhoら[2]により紹介されている．構文解析器生成系の一種であるBison[3]は，これらの表圧縮手法[2]を利用して解析表を生成する．

ここで，行置換法をトライ構造の状態遷移表に特化して改善した手法にダブル配列[4]がある．ダブル配列は，ある状態からの遷移先を，遷移種の内部表現値から算出する手法で，通常の行置換法と比較して，遷移先へのポインタを参照する必要がない分，高速に状態遷移できる．しかし，ダブル配列は，解析表を実現するために必要な還元アクションや競合アクションなど，状態遷移以外のアクションを定義できず，GLR解析表に直接適用できない．

本論文では，ダブル配列を拡張してGLR解析表を実現し，状態遷移に要する計算量を抑制することでGLR解析の高速化を図る．

2 ダブル配列によるGLR解析表

本章では，GLR解析表の構成について述べた後，ダブル配列を拡張してGLR解析表を実現する方法について述べる．以下，解析表の作成時に登録する文法Gに含まれる規則gの規則番号をI_gと表記する．

まず，GLR解析表の構成[1]について述べる．GLR解析表はAction関数とGoto関数から構成される．Action関数は，状態xと終端記号aを引数とし，shift, reduce, 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である．GLR解析表はAction関数の戻り値として複数のアクション（以下，競合アクション）を許す[1]．Goto関数は，還元した非終端記号Aと状態xを引数とし，状態xから非終端記号Aによる遷移先tを返す．以下，状態xにおいて，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．

表1: ダブル配列上に定義する状態

状態	役割
シフト状態	GLR解析表における通常の状態
還元状態	還元規則番号の管理
DR状態	デフォルトの還元規則番号の管理
競合状態	アクション集合へのリンクの管理

次に，ダブル配列を拡張して，GLR解析表を実現する方法を説明する．ダブル配列[4]は2本の1次元配列Base, Checkにより状態遷移表を構成し，配列の添字が状態番号に対応する．通常，ダブル配列は状態xから遷移種aによる遷移先tを，状態xのBase値と遷移種aの内部表現値との和により決定する[4]が，提案手法では後述の付属要素を定義するため，内部表現値の2倍との和により決定し，式(1)で遷移を定義する．

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

2.1 Action関数の定義

提案手法では，GLR解析表における全てのアクションを定義するため，表1に示す4種類の状態と付属要素をダブル配列上に定義する．以下，シフト，還元，デフォルトアクション，競合アクションの定義を順に説明する．

まず，シフトの定義について述べる．Action(x, a)=shift(t)のとき，式(1)により状態tをダブル配列上に作成することでシフトを定義する．以下，このシフトを定義する状態をシフト状態と呼ぶ．

次に，還元の定義について述べる．Action(x, a)=reduce(I_g)のとき，式(1)により状態tをダブル配列上に作成し，状態tのBase値に還元規則番号I_gを格納することで還元を定義する．以下，この還元を定義する状態を還元状態と呼ぶ．

Design of the GLR parse table with Double-Array
Takuma KURAMITSU†, Hidemi SHIGEKOSHI† and Hisatoshi MOCHIZUKI†
†Osaka Kyoiku University

次に，デフォルトアクションの定義について説明する．状態 x における最頻出の還元規則をデフォルトのアクションとして定義することで，解析表に定義するアクション数を抑制でき，記憶領域を抑制できる [2]．提案手法は，状態 x における最頻出の還元規則 g の規則番号 I_g を，状態 x の隣接要素 $x+1$ の Check 値に格納することでデフォルトアクションを定義する．以下，隣接要素 $x+1$ を状態 x の付属要素と呼ぶ．

ここで，デフォルトアクション以外のアクションが存在しない状態 x においては，解析表に必要な記憶領域を抑制するため，付属要素を作成せずに，状態 x の Base 値に規則番号 I_g を格納する．以下，デフォルトアクションを管理する状態を DR 状態と呼ぶ．

次に，競合アクションの定義について述べる．提案手法は，競合アクションを管理するためにアクション集合配列を新たに設ける．状態 x において，終端記号 a により競合アクションが存在する場合，まず，各アクションをアクション集合配列に格納する．次に，状態 x からの遷移先として式 (1) により状態 t をダブル配列上に作成し，アクション集合の格納位置へのリンクを状態 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，この競合アクションを定義する状態を競合状態と呼ぶ．

提案手法は Action(x, a) 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．ただし，開始規則 1 の還元を accept とし，遷移先，及びデフォルトのアクションが存在しない場合を error として定義する．

2.2 Goto 関数の定義

提案手法は，状態 x から非終端記号 A による遷移先 t を，状態 x における付属要素 $x+1$ の Base 値と非終端記号 A の内部表現値を用いて式 (2) で定義する．ただし，非終端記号 A におけるデフォルトの遷移先を，非終端記号 A の内部表現値の 2 倍として定義し，ダブル配列上に定義する状態数を抑制する．

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

提案手法は，解析表上の状態遷移を式 (1)，式 (2) により遷移先へのポインタを参照せずに行えるため，Aho らにより紹介される行置換法 [2] を用いる手法よりも高速に状態遷移できる．

3 実験

提案手法を評価するため，Bison2.4[3] との比較実験を Intel Core2 Duo 2.93GHz，Fedora8 上で行った．実験では，SQL 文法のサブセットと ANSI C の文法を使用し，SQL のトークン列 6.5M と C 言語のトークン列 6.6M をそれぞれ解析した．表 2 に，総遷移回数（表中，総遷移），式 (1)，または式 (2) によりダブル配列上を遷移した回数（表中，DA 遷移），解析時間，記憶領域を示す．ここで，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，解析時間，記憶領域について順に述べる．

表 2: 実験結果

	SQL		ANSI C	
	提案	Bison	提案	Bison
総遷移 (M 回)	13.75	13.75	59.86	59.86
DA 遷移 (M 回)	10.64	-	13.26	-
解析時間 (秒)	1.03	1.34	5.17	5.64
記憶領域 (Byte)	15,516	11,933	8,524	9,242

表 2 に示すように，提案手法は各文法の実験で解析時間を Bison よりも短縮した．これは，Bison が状態遷移の際に，必ずポインタを参照して遷移先を決定するのに対し，提案手法は，状態遷移の一部を比較的遷移計算量の少ない DA 遷移により行うためである．SQL においては，総遷移回数 13.75M 回の約 77%にあたる 10.64M 回が DA 遷移であり，提案手法は Bison に対して解析時間を約 22%削減した．また，ANSI C においては，総遷移回数 53.86M 回の約 22%にあたる 13.26M 回が DA 遷移であり，解析時間を約 8%削減した．

提案手法の解析表に必要な記憶領域は，SQL で Bison の約 1.3 倍に増加したが，提案手法，Bison とも，各解析表を 8〜16KByte 程度の記憶領域で実現した．

4 おわりに

本論文では，ダブル配列を拡張して，状態遷移が高速な GLR 解析表の実現法を提案した．今後の課題として，他の文法により提案手法を検証することや，提案手法による構文解析器生成系を作成することが挙げられる．

参考文献

1) 田中穂積：自然言語解析の基礎，産業図書 (1989).
2) 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).
3) Free Software Fundaction: Bison - GNU parser generator, http//www.gnu.org/software/bison/.
4) Yata, S., et al.: A compact static double-array keeping character codes, *IPM*, Vol. 43, No. 1, pp. 237–247 (2007).