D-002

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
Speeding Up Transition Operations of LR Parsing by Using the Transition Table with Double-Array

蔵満 琢麻† 重越 秀美† 望月 久稔†
Takuma Kuramitsu Hidemi Shigekoshi Hisatoshi Mochizuki

1．はじめに

LR 解析表 [1] は文脈自由文法の一種である LR 文法の構文解析に用いられ，コンパイラ，情報抽出，テキストの誤り訂正，自然言語処理，音声認識など幅広く利用されており，時間的，領域的に効率的な実現法が求められる．

Aho らは，LR 解析表の Action 関数や Goto 関数に，デフォルトのアクションを定義することで LR 解析表に記述するアクションの数を抑制し，複数の 1 次元配列を用いて 2 次元の LR 解析表を圧縮する手法を提案した [1]．構文解析器生成系の Yacc や Bison[2] は，Aho らの提案した表圧縮手法 [1] を利用して LR 解析表を生成する．

ここで，Aho らの表圧縮手法を状態遷移表に特化して改善した手法にダブル配列法 [3, 4] がある．ダブル配列法は，ある状態からの遷移先を単純な演算により一意に決定できる手法で，Aho らの手法と比較して，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を行える．本論文ではダブル配列を拡張して LR 解析表を実現し，LR 解析において状態遷移が発生する shift・goto 操作の高速化を図る．

2．ダブル配列を拡張した LR 解析表

ダブル配列 [3] 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．以下，状態 x における配列 Base，Check の要素をそれぞれ $B[x]$，$C[x]$ と表記する．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，$B[x]$ と遷移種 a の内部表現値との和により決定する [3, 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．

$$\begin{cases} t = B[x] + 2a \\ C[t] = a \end{cases} \quad (1)$$

以下，LR 解析表の作成時に登録する文法 G に含まれる規則の数を $|G|$，規則 g の規則番号を I_g と表記し，ダブル配列を拡張した LR 解析表の Action 関数，Goto 関数を順に説明する．

2.1 Action 関数の定義

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 をスタックにプッシュ（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である．以下，reduce(I_g)，shift(t)，デフォルトのアクションをダブル配列上に定義する方法を順に述べた後，提案手法の Action 関数を定義する．

まず，reduce(I_g) の定義法を述べる．提案手法は，Action(x, a) = reduce(I_g) を，状態 x から終端記号 a による遷移先 t を作成し，$B[t]$ に規則番号 I_g を負値で格納して定義する．以下，この状態 t を還元状態と呼ぶ．

次に，shift(t) の定義法を述べる．提案手法は，Action(x, a) = shift(t) を，式 (1) により状態 x から遷移種 a による遷移先 t を作成することで定義する．しかし，式 (1) では Base 値が異なる状態から同一状態への遷移を直接定義できない．そこで，状態 x から遷移種 a により既存の状態 u への遷移を作成する必要が生じた場合，式 (1) による遷移先 t の Base 値に既存状態 u へのポインタを格納し，状態 x から既存状態 u への遷移を間接的に定義する．以下，既存の状態 u を統一状態，統一状態へのポインタを管理する状態 t を間接状態と呼ぶ．間接状態 t の Base 値には，統一状態 u の状態番号と規則数 $|G|$ の和を負値で格納する．このように規則数を加算することで，間接状態と還元状態を Base 値により区別できる．

次に，各統一状態にデフォルトのアクションを定義し，ダブル配列上に定義するアクションの数を抑制する方法を述べる．提案手法は，統一状態 x におけるアクションの中で最も多く現れる還元規則 g（$\neq$ 開始規則 1）をデフォルトの還元規則（以下，DR 規則）として取り扱い，配列上の隣接要素 $x+1$ の Check 値に規則番号 I_g を負値で格納することで DR 規則を定義する．以下，この隣接要素 $x+1$ を状態 x の付属要素と呼ぶ．DR 規則を定義することで，配列上に定義する還元状態の数を抑制できる．

提案手法は状態 x において，終端記号 a による遷移先 t の有無を確認し，遷移先の種類によってアクションを決定する．提案手法における Action 関数を以下に定義する．

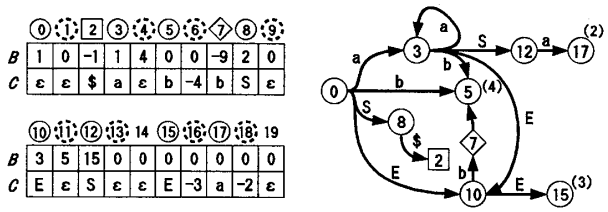
Action(x, a)

遷移変数 t に $B[x] + 2a$ を格納する．$C[t] = a$ で遷移先 t が存在する場合，$B[t] \geq 0$ ならば shift(t)，$B[t] < -|G|$ ならば shift($-(B[t] + |G|)$)，$-|G| \leq B[t] < -1$ ならば reduce($-B[t]$)，$-B[t]$ が開始規則 1 ならば accept（reduce(1)）を返す．$C[t] \neq a$ で遷移先 t が存在しない場合，$C[t+1] < 0$ ならば reduce($-C[t+1]$)，そうでなければ error を返す．

2.2 Goto 関数の定義

Goto 関数は，reduce(I_g) 実行時に呼び出され，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．

† 大阪教育大学, Osaka Kyoiku University



	⓪	①	②	③	④	⑤	⑥	⑦	⑧	⑨
B	1	0	-1	1	4	0	0	-9	2	0
C	ε	ε	$	a	ε	b	-4	b	S	ε

	⑩	⑪	⑫	⑬	14	⑮	⑯	⑰	⑱	19
B	3	5	15	0	0	0	0	0	0	0
C	E	ε	S	ε	ε	E	-3	a	-2	ε

図 1　提案手法の LR 解析表と状態遷移図

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，既存の状態 u への遷移を作成する場合は，shift(t) の定義と同様に間接状態を用いて状態 u への遷移を定義する．

$$\begin{cases} t = B[x+1] + 2A \\ C[t] = A \end{cases} \tag{2}$$

ここで，各非終端記号によるデフォルトの遷移先を定義することで，ダブル配列上に定義する状態数を抑制することが考えられる．提案手法は，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，Goto 関数に必要となる状態数を抑制する．提案手法における Goto 関数を以下に定義する．

Goto(x, A)

遷移変数 t に $B[x+1]+2A$ を格納する．$C[t] = A$ で遷移先 t が存在する場合，$B[t] \geq 0$ ならば t を返し，そうでなければ $-(B[t]+|G|)$ を返す．$C[t] \neq A$ で遷移先が存在しない場合はデフォルトの遷移先 $2A$ を返す．

例：終端記号$, a, b の内部表現値をそれぞれ 0, 1, 2, 非終端記号 S', S, E の内部表現値をそれぞれ 3, 4, 5 とし，文法 P={(1) S' → S, (2) S → a S a, (3) S → E E, (4) E → b} を登録した提案手法の LR 解析表と状態遷移図を図 1 に示す．ここで，'$' は入力トークン列の末尾を表す終端記号である．図 1 中，統一状態を丸枠，間接状態を菱形枠，還元状態を四角枠，付属要素を破線の丸枠で示す．ここで，配列の添字を囲んでいない要素は未使用の要素であり，Check 値に遷移種として用いない値 ϵ $(= |N| + |T|)$ を格納して他の状態と区別する．状態遷移図における括弧内の数値は統一状態における DR 規則を表す．

図 1 の状態 2 は，Action(8, $)=reduce(1)=accept を定義する還元状態で，Base 値に規則番号 1 を負値でもつ．状態 7 は Action(10, b)=shift(5) を定義するための間接状態で，統一状態 5 へのポインタ（$-(5+|P|)$）を Base 値にもつ．また，統一状態 5 における付属要素 6 は，統一状態 5 の DR 規則 4 を Check 値に負値でもつ．（例終）

提案手法は，統一状態から統一状態への遷移や，各非終端記号によるデフォルトの遷移を，遷移先へのポインタを参照せずに行えるため，Aho らの解析表 [1] を用いる手法よりも高速に shift・goto 操作を行える．

表 1　実験結果

	ANSI C		Pascal	
	提案	Bison	提案	Bison
直接遷移 (M 回)	53.24	-	23.36	-
間接遷移 (M 回)	6.59	59.83	4.03	27.39
解析時間 (秒)	1.48	1.98	0.74	0.99
記憶領域 (Byte)	8,660	8,124	5,940	4,188

3．実験

提案手法を評価するため，Bison2.3[2] との比較実験を Intel Core2 Duo 2.93GHz，Fedora8 上で行った．実験では，ANSI C と Pascal の文法を使用し，C 言語のトークン列 6.6M と Pascal のトークン列 6.3M をそれぞれ解析した．表 1 に，遷移先へのポインタを参照せずに状態遷移した回数（表中，直接遷移），遷移先へのポインタを介して間接的に状態遷移した回数（表中，間接遷移），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 1 より，提案手法は Bison と比較して，解析時間を ANSI C で約 25%，Pascal で約 26% 削減した．これは，Bison が状態遷移の際に，必ず遷移先への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，大半の状態遷移において遷移先を単純な演算により決定することが可能で，shift・goto 操作をより高速に行えるためである．

提案手法の解析表に必要な記憶領域は，ANSI C と Pascal の文法でそれぞれ Bison の約 1.1 倍，約 1.4 倍に増加したが，それぞれ 10KByte 未満の記憶領域で実現した．

4．おわりに

本論文では，ダブル配列による遷移表を拡張して，従来手法よりも高速に shift・goto 操作を行える LR 解析表の実現法を提案した．実験の結果，提案手法は Bison と比較して構文解析に要する時間を約 25% 削減した．

今後の課題として，より大規模な文法を登録して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) Aho, A. V., Sethi, R. and Ullman, J. D.: コンパイラ I 原理・技法・ツール，サイエンス社 (1990).
2) Free Software Fundaction: Bison - GNU parser generator, http://www.gnu.org/software/bison/.
3) 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).
4) Yata, S., et al.: A compact static double-array keeping character codes, *Information Processing and Management*, Vol. 43, No. 1, pp. 237–247 (2007).