1K-03

flexにおける遷移関数と正規表現マッチングの高速化

増田　優香†　　篠原　哲平†　　望月　久稔†
†大阪教育大学

1　はじめに

コンピュータにおいて，形式が異なる多種多様なデータを扱うためには，入力データを処理できる形に変換する必要がある．字句解析ツールである flex[1] は，変換する規則として拡張正規表現 [2] を扱い，状態遷移表を利用した遷移関数でマッチングするプログラムソースを自動生成する．プログラムは，遷移が一意に定まる決定性有限オートマトンにより照合する．

入力データの照合はコンピュータ処理において前処理であるため高速であるほど良い．そこで本研究は，flex における遷移関数での 1 遷移を確定する配列の参照回数を削減し，より高速なマッチングを目的とする．

2　flexにおける遷移関数のデータ構造

入力データを照合する flex の遷移関数は，遷移先節点番号を決める基底値を格納する配列 $base$，遷移先節点番号を格納する配列 nxt，遷移元節点番号を格納する配列 chk からなる．節点 s から遷移種 e による節点 t への遷移情報を nxt と chk 上の位置 $base[s]+e$ に式 (1)，式 (2) で定義する [3].

$$nxt[base[s]+e] = t \quad (1)$$

$$chk[base[s]+e] = s \quad (2)$$

遷移関数と状態遷移図を図 1 に示す．図中の記号は，丸印で節点番号，矢印で遷移種，＜＞で基底値，ひし形で配列 chk に格納する値を示す．flex は，1 遷移あたりにおける遷移情報の位置で 2 回，配列の参照で 4 回計算する．

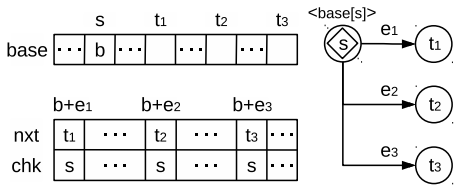

図 1: flex の遷移関数と状態遷移図

Improving Transition Function and Regular Expression Matching in flex
†Yuka MASUDA　†Teppei SHINOHARA　†Hisatoshi MOCHIZUKI
†Osaka Kyoiku University

3　遷移関数の設計とマッチング法

正規表現を高速にマッチングするために，2 つの配列 $BASE$ と $CHECK$ からなる遷移関数を定義し，1 遷移当たりにおける配列の参照回数削減を図る．

2 つの配列で高速にトライ木を探索する手法としてダブル配列があり，矢田らの手法 [4] では配列 $CHECK$ に遷移種を格納する．しかし，遷移情報の位置が遷移先節点番号であるため，複数の異なる遷移元節点の基底値が固定される．遷移元節点が互いに異なる遷移パターンの場合に，それぞれの遷移先節点を区別できない．つまり，一つの節点において複数の遷移元節点が定義できず，決定性有限オートマトンを定義できない．そこで本論では，配列 $BASE$ を遷移元節点の基底値ではなく，遷移先節点の基底値と定義する．基底値の設定はダブル配列と同様であるが，節点毎に一意と定める．遷移する際には節点の基底値を状態番号として遷移関数を辿る．

以下に，遷移関数を式 (3)，(4) で定義し，マッチングの手順を示す．両式は節点 s と t の基底値を s' と t' とし，遷移種 e により節点 s から節点 t へ遷移する．

$$BASE[\,s'+e\,] = t' \quad (3)$$

$$CHECK[\,s'+e\,] = e \quad (4)$$

手順 1 初期状態番号として初期基底値を設定する．

手順 2 遷移先節点の存在を確定するために，式 (4) で遷移種と配列 $CHECK$ の値を比較する．同値の場合は手順 3 へ．

手順 3 現在の状態番号を式 (3) を用いて遷移先節点の基底値に更新する．

手順 4 入力を次へと進め，残りがあれば手順 2 へ．無ければ終了する．

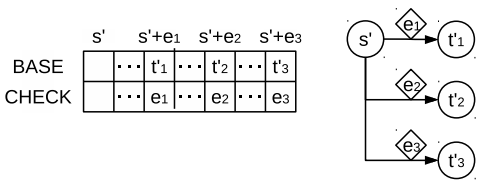

図 2: 提案手法の遷移関数と状態遷移図

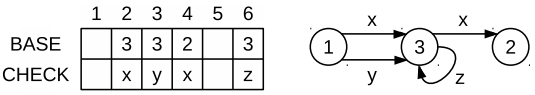

図 3: $(x|y)z*x$ に対する遷移関数と状態遷移図

式 (3)，(4) における遷移関数と状態遷移図を図 2 に示す．図中の記号において，丸印は状態番号，ひし形は配列 $CHECK$ に格納する値へ変更する．提案手法は，遷移情報の位置について配列の参照を必要とせず，式 (3)，(4) は手順 3 と 2 で 1 度ずつ用いる．したがって，1 遷移あたりにおける配列の参照は 2 回であり，flex と比べて少ない．

正規表現 $(x|y)z*x$ に入力 yzx でマッチングする例として，遷移関数と状態遷移図を図 3 に示す．ここで，遷移の開始は右図の最左にある節点とし，遷移種は内部表現値を $x=1$，$y=2$，$z=3$ とする．

手順 1 で初期状態番号は遷移を開始する節点の基底値 1，入力は y となる．手順 2 で式 (4) と図 3 により遷移種 y と $CHECK[1+2]$ は同値である．手順 3 で状態番号を式 (3) より $BASE[1+2]=3$ に更新する．手順 4 で次の入力 z より手順 2 に戻る．残りの入力も同様に遷移し，最後の入力 x で状態番号 2 に遷移する．

仮に状態番号が 3 で入力が z であれば，手順 2 において遷移種 z と $CHECK[3+3]$ は同値であり，手順 3 で状態番号を $BASE[3+3]=3$ と更新する．これと，状態番号 1 から遷移種 y による状態番号 3 への遷移は，節点番号 3 が複数の遷移元節点から遷移できることを示す．よって，式 (3)，(4) による遷移関数は一つの節点において複数の遷移元節点が定義できる．

4 参照回数とマッチングの評価

flex[1] が生成したプログラムに対して，配列の参照回数削減を図ったことによるマッチングの時間と記憶領域を評価する．実験は，Intel(R) Core(TM) i7 920 @ 2.67GHz，CentOS6.6 32bit 上で，長さ 10〜100 までの浮動小数点数をそれぞれの長さ毎に 10 万件をランダムに作成したデータのマッチングで比較する．

提案手法は flex に対して，配列の参照回数を示す図 4 より 50 %削減し，マッチング時間を示す図 5 より 17〜23 %短縮した．参照回数とマッチング時間について変化率の相関係数は 0．99 で正の相関を示し，配列の参照回数を削減して，マッチング時間を短縮した．しかし，浮動小数点数を照合する拡張正規表現の遷移関数に要する記憶領域は，flex が生成したプログラムの 1303byte に対し，提案手法は 1428byte と 10 %増加した．

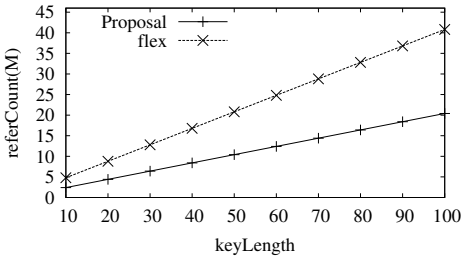

図 4: 参照回数

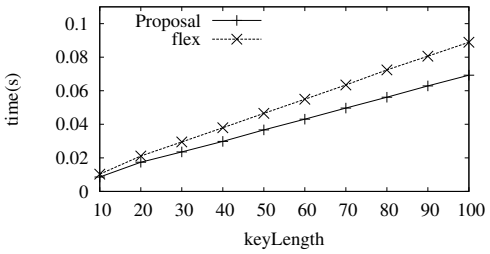

図 5: マッチング時間

5 おわりに

flex における正規表現の高速なマッチングを実装するために，配列の参照回数を削減する遷移関数を提案した．実験により，最適化した提案手法は参照回数を削減し，マッチングを高速化した．

今後の課題として，増加した記憶領域の削減，flex のオプションによるデータ構造とマッチング法の分析などが挙げられる．

参考文献

[1] flex 2.5.3：https://github.com/westes/flex/releases/tag/flex-2_5_35-10_fc13/，2017/1/3.

[2] John R. Levine：flex & bison, pp.19-22, O'REILLY, 2009.

[3] Alfred V.Aho，Monica S.Lam，Ravi Sethi，Jeffrey D.Ullman：Compilers Principles, Techniques, & Tools Second Edition, pp.185-186, PEARSON Addison Wesley, 2007.

[4] Susumu Yata，Masaki Oono，Kazuhiro Morita，Masao Fuketa，Toru Sumitomo，Jun-ichi Aoe: A compact static double-array keeping character codes，Information Processing and Management，Volume 43，No.1，pp.237-247，2007.