4L-05

拡張文字列パターンによる CG モデルデータの高速構文解析手法の提案

渡邉 優太† 坂下 善彦‡

湘南工科大学大学院 工学研究科 電気情報工学専攻†

湘南工科大学 工学部 情報工学科‡

1 はじめに

我々は CG(Computer Graphics)の描画処理中に CG モデルの追加や削除を利用者が行うインタラクティブな可視化システムの実現を目指している。そのため,形状定義データである CG モデルから描画処理機構へのデータ変換処理の高速化が必須であり,それを実現する為の構文解析器が必要になる.そこで本研究では拡張文字列パターンの生成と検索を可能にする Extended Shift-And 法(以下, ESA)[1]を用いる.

ESA ではビット並列手法を用いるため,高速に拡張文字列パターンを検索することが可能であるが, 構文解析器としての機能は備わっていない. 本研究では ESA を拡張し,構造化データの構文解析を可能とするアルゴリズムを提案する.

2 拡張文字列

拡張文字列とは正規表現よりも制限された表現手法である. ここでは$\sum$ を出現する文字集合としており,その場合, 照合を行うテキスト$T = t_0\ t_1 \ldots t_n \in \sum^*$である. 次に拡張文字列パターン$P = p_0\ p_1 \ldots p_m (m \geq 0)$の定義を示す.この定義に出てくる$L$ = 拡張文字列パターンの言語, ε = 空文字を表している.

1) 定数文字：$a \in \sum, L(a) = a$
2) ワイルドカード：$".", L(.) = \sum$
3) 文字クラス：
   $\beta = [abc \ldots] \subseteq \sum, L(\beta) = a \cup b \cup c \cup \ldots$
4) オプション文字：
   $\beta? = (\beta|\varepsilon), L(\beta?) = \beta \cup \varepsilon$
5) 0 個以上の非限定繰り返し：
   $\beta^*, L(\beta^*) = L(\beta)^*$
6) 1 文字以上の非限定繰返し：
   $\beta^+, L(\beta^+) = L(\beta)L(\beta)^*$

ESA では上記で示した定義の拡張文字列パターンの生成及び,照合を行うアルゴリズムになる. まず, ESA は非決定性有限オートマトン(以下, NFA)の生成を行う.後の照合時にビット演算と算術減算を用いて照合を行う. このアルゴリズムはテキスト長をn, パターン長をm, NFA の状態数をl, レジスタ長をwとする. ESA は生成処理に$O(m + |\Sigma|)$時間, 照合処理に$O(n\lceil l/w \rceil)$時間を要する.

このアルゴリズムはパターンマッチングを行うアルゴリズムであるため, 特定の文字列の抽出を行うことが不可能である. また, 1) ~ 6)までの定義では我々が扱うパターンを表現することは不可能である. そのため, 現状のアルゴリズムでは構文解析器として用いることはできない.

3 提案手法

ESA では NFA の生成と照合を行うアルゴリズムであるが構文解析器としての機能が備わっていない.そこで2章で示した定義に次の定義を提案手法に加える.

7) メタ文字：$M \in$ ". ","∗","+","?"…
8) エスケープ：" \ ", $L(\backslash M) =$ "M"
9) シーケンス：
   $\beta = [a - z] \subseteq \sum, L(\beta) = a \cup b \ldots \cup z$
10) ワイルドカードの 0 個以上の繰り返し：
   ".* " , $L(.^*)L(\beta) = L(.)^*L(\beta)$
11) ワイルドカードの 1 個以上の繰り返し：
   ".+ " , $L(.^+)L(\beta) = L(.)L(.)^*L(\beta)$
12) 抽出文字列
   $\{string\} \in \sum, Buf \leftarrow string$

7)は定義されたメタ文字の集合を表す. 8)は " \ " の直後に存在するメタ文字を単一の 1 文字として扱う. 9)は連続する文字を選択する為の文字クラス. 10), 11)はワイルドカードの繰り返しの直後にある文字間引いた集合の繰り返し. 12)は{ }で閉じられた文字列の抽出箇所の指定になる.

これらの定義を追加し, 拡張した ESA によって生成された NFA を図 1 に示す. この NFA では文字集合$\sum = ASCII\ Code$, パターンを$P =.*(\{-?[0-9]+\})$とした場合の NFA になる. この NFA では状態 2 から 4 の状態にいる文字列の抽出を行う.

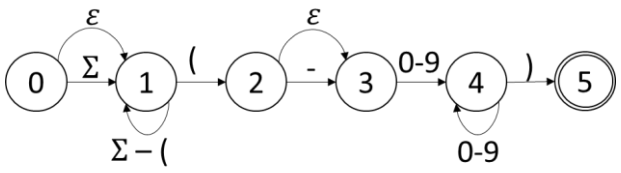

図 1. $P =.*(\{-?[0-9]+\})$の NFA

準備として記号の説明を行う. 演算子は" & " = 論理積, " | " = 論理和, " - " = 算術減算, " << " = 左シ

Proposal of fast syntactic analysis method using the extended pattern matching for the CG model data

† Shonan Institute of Technology, Graduate School

‡ Shonan Institute of Technology, Department of Information Sience

フト, “~“ = 否定, “^” = 排他的論理和で表す. 次にアルゴリズムで用いる *D, Df, A, I, F, E, Eg, B, S* は次の意味を持つ.

- *D* = NFA の状態
- *Df* = 次の状態への準備
- *A* = 任意の文字 (* , ? など)
- *I* = *A* に対しての始まり
- *F* = *A* に対しての終わり
- *E* = 抽出文字列
- *Eg* = 抽出文字列の区切れ
- *B* = 文字ごとの出現状態
- *S* =文字ごとの繰り返し

これらを踏まえ, 拡張した ESA アルゴリズムを図 2 に示す. このアルゴリズムは 3 行目の処理で照合を行っている.そして, 8 行目の処理で受理されるか否かの判定を行っている. また, 6, 7 行目の処理は次の照合処理の準備になる. 6, 7行目の処理を行うことにより, “?”や”*”のような ε に対しての状態遷移が可能となる.我々が拡張したのは 4, 5 行目である.4 行目で抽出する必要がある文字の判定を行い, 5 行目で抽出文字の終わりの判定を行う.これらにより文字列の抽出を可能とする.

Searching(B,S, A, I, F, E, Eg, T , l)
1. *D←0, Df←0*
2. *For i=0 to n do*
3. *D←((D≪1) | 1) & B[T[i]] | (D & S[T[i]])*
4. *If D & E≠0 Then Extract Character*
5. *If D & Eg≠0 Then Gap of Extract Character*
6. *Df←D | F*
7. *D←D | { A & ((~(Df−I)) ^ Df) }*
8. *If D & 2^{l-1} ≠0 Then End*

図 2. 拡張した ESA アルゴリズム

図 2 で示したアルゴリズムを用いた実行例を示す.ここでは図 1 の NFA を例にして行う.まず, 生成した *A, I, F, E, Eg, B, S* を表 1 に示す.表 1 のパターンを用いて図 2 のアルゴリズムを実行させた結果を表 2 に示す.表 2 ではテキスト$T = (ab(-12)$ とした場合で照合を行ったものとなる.表2にある状態 *D* は図 2 の 3 行目の処理後の *D* であり, 更新後の *D* とあるのは 7 行目の処理後の *D* となる. また, *E* よって判定を行いこの実行例では”-12”という文字列の抽出を行う.

表 1.P =.* ({−?[0 − 9]+})の拡張文字列パターン

文字	B	S
(	$(00010)_2$	$(00000)_2$
-	$(00101)_2$	$(00001)_2$
0 − 9	$(01001)_2$	$(01001)_2$
)	$(10001)_2$	$(00001)_2$
上記以外の文字	$(00001)_2$	$(00001)_2$

A	$(00101)_2$
I	$(00010)_2$
F	$(00101)_2$
E	$(01100)_2$
Eg	$(10000)_2$

表 2. $T = (ab(-12)$に対する照合

読み込み文字	状態D	Df	更新後のD	E
(	$(00001)_2$	$(00101)_2$	$(00001)_2$	×
a	$(00001)_2$	$(00101)_2$	$(00001)_2$	×
b	$(00001)_2$	$(00101)_2$	$(00001)_2$	×
(	$(00011)_2$	$(00111)_2$	$(00111)_2$	×
-	$(00101)_2$	$(00101)_2$	$(01001)_2$	○
1	$(01001)_2$	$(01101)_2$	$(01001)_2$	○
2	$(01001)_2$	$(01101)_2$	$(01001)_2$	○
)	$(10001)_2$	$(10101)_2$	$(10001)_2$	×

表 3. 各構文解析器による実行時間(sec)

	実行時間	最大	最小
既存の構文解析器	0.0124	0.0153	0.0114
提案手法	0.0476	0.0589	0.0388

4 実験

我々が扱うデータは木構造もつデータとなる. ここでは深さ優先探索によって, 葉ノードに位置するパラメータの抽出を行っている.ここで既存の構文解析器と提案手法を用いた構文解析器の実行速度の比較を PC(openSUSE 13.2, intel Core i7-4790 3.6GHz, 8GB, gcc 4.8.3)によって行った.この実験ではファイル(36.6KB)を読込み, 構文解析が完了するまでの実行時間を表 3 に示す. この実験ではそれぞれ 10 回実験を行い, その平均値としている.

5 まとめ

提案手法を用いて, 構造を持つ CG モデルデータの解析を行うことが可能になった.しかし, 従来の構文解析器比べ, 4 倍近くの時間がかかっており高速には行えていない. これは現状のアルゴリズムではレジスタ長を超える状態の生成が不可能である. この制約があるため,今回行った実験では NFA を各ノードに分割し, テキストに対して照合及び, 抽出を行っている. 今回の実験で使用したデータは木の総ノード数が 108 個あり, 全文検索 2 回行い見つかった根ノードの子が 16 個ある. その後, 見つけたノードの部分木で計 92 回の照合を行う.この様に複数回の照合を行っていることによって実行時間が低速になる主な要因と考える.

今後はレジスタ長を越える状態に対応した RunNFA[2]を, 今回提案した手法に適用し, 構文解析器としての高速化を図る.

参考文献

[1] Gonzalo Navarro and Mathieu, Flexible Pattern Matting in Strings Cambridge University Press, 2002, 221 p.

[2] 笹川裕人, 金田悠作, 有村博紀,長大な拡張文字列パターンに対する大規模文字列照合の高速化, 日本データベース学会論文誌, 2012, vol. 11, no. 1, p. 55-60.