字句解析器生成系での最短一致法の提案

中田育男† 田村祐子† 中井央†

字句解析器生成系が生成する字句解析器では，複数の字句に一致する場合は，通常は最も長いものをとる，いわゆる最長一致の方式がとられる．しかし，時には最短一致が望ましい場合もある．たとえば，最短一致が指定できれば，C言語のコメントを簡潔な正規表現で定義することができる．本論文では，字句解析器生成系で最短一致を指定する方法と，その指定があった場合の字句解析器生成の方法を提案する．

Shortest-match Method in Lexer Generators

IKUO NAKATA,† YUUKO TAMURA† and HISASHI NAKAI†

A lexer, or a lexical analyzer, generated by a lexer generator usually recognizes tokens by the longest-match, i.e., the longest initial substring of the input that can match any regular expression is taken as the next token. In some cases, however, the shortest-match is preferable. For example, the regular expression for the comment of C language can be expressed by a simple regular expression if the shortest-match can be specified in the expression. In this paper, we propose a method that specifies the shortest-match and a lexer generation method that implements such a specification.

1. はじめに

字句解析器生成系が生成する字句解析器では，複数の字句に一致する場合は，通常は最も長いものをとる，いわゆる最長一致の方式がとられる[1),2)]．その理由は，たとえば字句として

```
less_than_or_equal = "<="
less_than          = "<"
assign             = "="
```

が定義されているときに，入力「<=」を less_than と assign の2つの字句として認識するのでなく，less_than_or_equal として認識するようにするためである．

字句の定義は通常正規表現の形で与えられ，字句解析器生成系は正規表現から NFA（Nondeterministic Finite Automaton：非決定性有限オートマトン）を作成し，その NFA から得られる DFA（Deterministic Finite Automaton：決定性有限オートマトン）の形で字句解析器を生成する[1),6)]．一般に，その DFA には複数個の最終状態があるが，字句解析器は，どれかの最終状態に達しても，さらに遷移が可能であるかぎり解析を続行し，遷移が不可能になった時点で，通ってきた最終状態のうちの最後の最終状態まで戻ってそこで認識したことにする．それが最長一致の方式である．

ところで，C言語のコメントが簡単な正規表現で表せないことはよく知られている．たとえばある文献[7)]では，lex[4)] の表記法でそれを

```
"/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
```

という複雑な正規表現で表している．ここで，「*」は0回以上の繰返し，「[^x]」は x 以外の文字を表す．これは，コメントの形になる場合とならない場合をいろいろ考えて得られた表現であるが，そのように考えることをせず，後に述べる方法によって DFA を作って，その DFA から求めればもう少しは簡単になる[6)]．

多くの字句解析器生成系では，そのような複雑な記述をしなくてもよいようにするために，通常の状態と「/*」を読んだ後の状態とを分けることができるようにしている．その方式では，たとえば次のように記述することができる[2)]．

```
<YYINITIAL>[a-z]+    {return IDENTIFIER;}
<YYINITIAL>"/*"      {yybegin(COMMENT);}
<COMMENT>"*/"        {yybegin(YYINITIAL);}
<COMMENT>.           { }
```

ここで，最初は <YYINITIAL> という状態にあり，<YYINITIAL> がついている正規表現から得られる

† 図書館情報大学
University of Library and Information Science

DFAが働く．「/*」を読んだところで <COMMENT> の状態に入ってDFAが切り替わる．<COMMENT> のDFAでは「*/」か任意の文字「.」を字句として読むことができるが，"*/" が先に書かれているのでそれが優先される．したがって，最初の「*/」を読んだところで <YYINITIAL> に切り替わるのである．このようにしてコメントを読むことができる．しかし，これも分かりやすい表現とはいえない．C言語のコメントを表現する最も簡潔な正規表現としては，

```
"/*".*"*/"
```

の形で表現できるのが望ましい．ここで，「.*」は任意文字の0回以上の繰返しを表す．しかし，最長一致の方式のもとでは，この正規表現に一致するのは最初の「/*」から最後の「*/」までとなってしまう．もし，この形に一致する一番短いものをとるという最短一致を指定することができれば，その問題は解決する．以下ではその方式，すなわち最短一致の指定方法と，その指定があった場合の字句解析器生成の方法を提案する．

2. 最短一致の記法

最短一致を指定する方法としては，最短一致の対象となる正規表現の直後に特別な記号を付けて指定することとする．たとえば，特別な記号として「@」を使うことにすると，C言語のコメントは，

```
"/*".*"*/"@
```

と表現できる．この特別な記号のついていない正規表現については，従来どおり最長一致の方式をとるものとする．なお，最短一致の記号は正規表現の最後に付けるものとし，R@S のように途中に付けることはできないものとする．

一般に複数個の字句定義があった場合には，ある文字列が2つ以上の字句定義に一致する場合はありうる．通常の字句解析器生成系では，そのような場合には，先に定義されていた字句定義の方を優先する[1),2)]．それにならって，ここでは，2つ以上の字句定義に一致した場合に，次の優先順位に従う（番号の小さな方が高い）こととする．

(1) 先に定義された最短一致の定義
(2) 最短一致の定義
(3) 先に定義された通常の定義

たとえば，次のような定義があったとき

```
id1 = "ident"
id2 = "id"[a-z]*"t"
cm1 = "/*".*"*/"@
cm2 = "/*".*"/"@
```

文字列

```
idt ident /* dsfg */ /* fd/
```

が与えられると，それは次のような字句の並びとして認識されることになる．

```
id2 id1 cm1 cm2
```

3. 最短一致の実現法

最短一致を実現するためには，通常の

正規表現 → NFA → DFA

という変換操作に際して，次のような追加を行えばよい．

(1) NFA 作成時

通常NFAは各字句の正規表現に対してまずは別々に作成される．たとえば前記の id1～cm2 の正規表現に対して4つのNFAが作られる．このときR@の形の正規表現に対してはRのNFAの最終状態（NFAの最終状態はただ1つとしてよい）に「最短一致」という印をつける．NFAの全体は，1つの初期状態から各NFAの初期状態への ε 遷移をつけることによって完成する．

(2) DFA 作成時

DFAの状態はNFAの状態の集合として作られる．DFAが作られていく過程では，作成中のDFAのある状態からある1つのNFAの状態を選び，そのNFA状態からの遷移をDFAに追加していくことを繰り返し，新たに何も追加されなくなったら，完成である．その際，NFAはもとの正規表現が定義された順に選ぶ．今DFAのある状態D1からNFAの状態N1を選んだとする．NFAでN1からの記号aによる遷移（a遷移）があってその遷移先がN2であるとき，DFAでのD1からのa遷移の遷移先D2にN2を加えるのが一般的な方法である（D2がなければそれを新たに空集合として作ってからN2を加える）．その際，最終状態に関しては以下のようにする．

(a) D2に「最短一致」の印のついた最終状態があれば，N2は付け加えない（先に定義された最短一致が優先される）．
(b) そうでない場合，
- N2が「最短一致」の印のついた最終状態であれば，D2に入っていたすべての状態を取り除き，N2をD2に入れる（D2からの遷移がなくなり最短一致が実現できる．それはまた通常の定義に優先する）．
- N2が通常の最終状態であり，D2に通常の最終状態があれば，N2は付け加えな

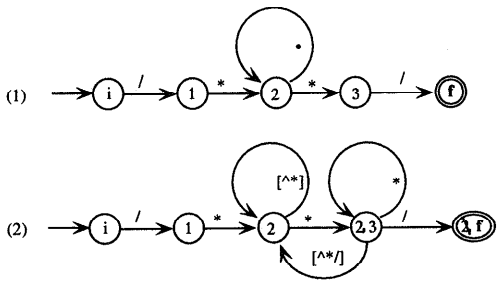


図 1 "/*".*"*/"@の (1) NFA と (2) DFA
Fig. 1 (1) NFA and (2) DFA for "/*".*"*/"@.

い（先に定義された通常の定義が優先される）．

(c) それ以外の場合は D2 に N2 を付け加える．

たとえば，

```
"/*".*"*/"@
```

に対しては，**図 1** の (1) の NFA から (2) の DFA が得られる．この DFA から逆に正規表現を求めてみると，たとえば

```
"/*"([^*]|"*"+[^*/])*"*"+"/"
```

が得られる（「+」は 1 回以上の繰返しを表す）．

以上の方式を実験で確認するために，Java で書かれていてそのソースが公開されているコンパイラフレームワーク生成系の SableCC[3] を改造して，上記のアルゴリズムに従って最短一致の機能を実装し，正しく最短一致の字句解析器が生成されることを確認した．

なお，この方式で

```
.*R@
```

から得られる DFA は，最初に正規表現 R のパターンに一致するものを探すパターンマッチングのアルゴリズムに相当するものである[5]．

4. 拡　　張

ところで，以上の方式では，最短一致の記号 @ は正規表現の末尾にだけあることにしているが，正規表現の途中にあってもよいように拡張することは簡単ではない．たとえば，

```
r1 = R
r2 = T@U
```

という字句定義を考える．T@U にマッチするのは，T に最短一致するものに U の形が続くものである．ここで，最短一致とは，T に最初に一致した時点で，T に一致するそれ以外の可能性は捨てることである．ただし，T に一致しただけでは，まだ T@U に一致するか分からないから，この @ に対して前記のように優先順位を高くすることはあまり意味がない．したがって，T に一致した時点でも，r1 とマッチする可能性があればそれを考慮しなければならない．たとえば

```
r1 = "abcd"
r2 = "a"[a-z]*"c"@"/"
```

という定義があるとき，前記の方式では，入力から abc まで読んだ時点で r2 の @ まで到達したとして，その他のケースをすべて無視してしまう．それでは r1 を認識できない．r1 も認識できるようにするためには，前記の方式で D2 から取り除いたり，D2 に付け加えないことにする NFA の状態としては，r2 の NFA の状態だけを考え，r1 の NFA の状態には適用しないようにする必要がある．

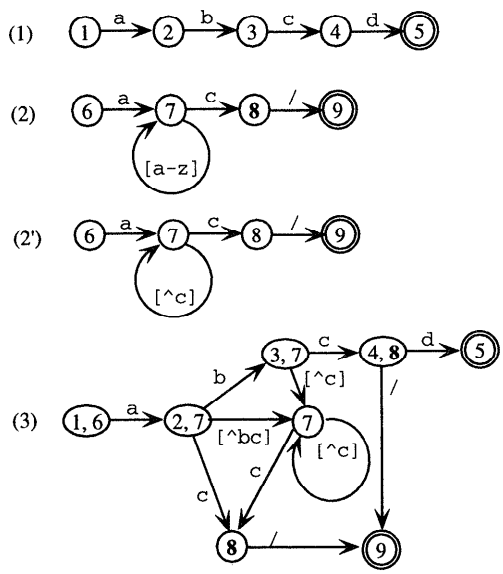


図 2 途中に @ がある場合の NFA と DFA：(1) r1 の NFA, (2) r2 の NFA, (3) r1 と r2 の DFA
Fig. 2 Example of shortest-match NFA/DFA: (1) NFA for r1="abcd", (2) NFA for r2="a"[a-z]*"c"@"/", (3) DFA for r1 and r2.

この方式によれば，今の例からは**図 2** のような NFA と DFA が得られる．その図で太字になっている状態 8 が最短一致の印のついた状態である．2 重丸は最終状態を表す．また，[^c] はここでの便宜上の記法であり，[a-z] で c 以外のものを表す．この DFA で ac を読んだときの状態は，通常は {7,8} となるところであるが，最短一致の指定があるので {8} となっている．同様に，abc を読んだときは，通常は {4, 7, 8} となるところであるが，7 だけが除かれている．

実際の実現の方法としては，まず最初に，途中に @ を含んだ正規表現に対してだけ独立に前記のアルゴリズムを適用し（今の例では図 2 の (2')），その後で（途中に @ はないものとして），全体に前記のアルゴリズ

ムを適用することが考えられる．

5. おわりに

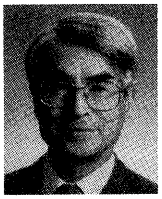
字句解析器生成系で字句を定義するとき，最短一致を指定できる機能を提案し，その実現法を示した．この機能を使えば，C 言語のコメントなどが簡潔に定義できる．実際にコンパイラフレームワーク生成系の SableCC を改造して，その機能を実装し，正しく最短一致の字句解析器が生成されることを確認した．

謝辞 SableCC の開発者であり，最短一致の方式を実験するための SableCC の改造にご協力いただいた Etienne Gagnon 氏に感謝する．

参考文献

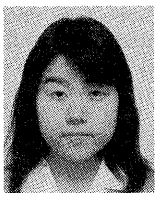
1) Aho, A.V., Sethi, R. and Ullman, J.: *Compilers – Principles, Techniques, and Tools*, Addison-Wesley. 原田賢一（訳）：コンパイラ—原理・技法・ツール I, II, サイエンス社 (1990).
2) Appel, A.W.: *Modern Compiler Implementation in Java*, Cambridge University Press (1998).
3) Gagnon, E.: SableCC, An Object-Oriented Compiler Framework, Master's thesis, School of Computer Science, McGill University (1998). http://www.sable.mcgill.ca/sablecc/.
4) Levine, J.R., Mason, T. and Brown, D.: *lex & yacc, 2nd edition*, O'Reilly & Associates (1990). 村上 列（訳）：lex & yacc プログラミング，アスキー出版局 (1995).
5) 中田育男：拡張正規表現によるパターンマッチングアルゴリズムの生成，コンピュータソフトウェア，Vol.10, No.1, pp.63–67 (1993).
6) 中田育男：コンパイラ，オーム社 (1995).
7) Schreiner, A.T. and Friedman Jr, H.G.: *Introduction to Compiler Construction with UNIX*, Prentice-Hall, Englewood Cliffs, New Jersey (1985). 矢吹道郎ほか（訳）：C コンパイラ設計—yacc/lex の応用，啓学出版 (1987).

(平成 11 年 3 月 25 日受付)
(平成 11 年 7 月 1 日採録)



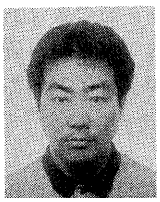
中田 育男（正会員）

1935 年生．1960 年東京大学大学院数物系研究科数学専攻修士課程修了．1960～1979 年（株）日立製作所中央研究所，システム開発研究所勤務．1979～1997 年筑波大学電子・情報工学系教授．1997～図書館情報大学教授．理学博士．プログラム言語，言語処理系，ソフトウェア工学等に興味を持っている．著書「コンパイラ」（産業図書），「基礎 FORTRAN」（岩波書店），「コンパイラ」（オーム社）．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE 各会員．



田村 祐子

1999 年 3 月図書館情報大学卒業．1999 年 4 月（株）富士通ターミナルシステムズ入社．



中井 央（正会員）

昭和 43 年生．平成 9 年筑波大学大学院工学研究科電子情報工学専攻修了．同年より図書館情報大学助手．工学博士．日本ソフトウェア科学会，ACM 各会員．