

第3章 正規表現と有限オートマトン

3.1 正規表現

_____ (_____ とも言う、regular expression) は、言語 (language, 文字列・記号列の集合) の定義の方法の一つである。正規表現で記述可能な言語を正規言語 (正則言語とも言う、regular language) という。

- 連接・選択・反復の3つの演算 (構成法) を持つ。BNF と異なり再帰はない。
- BNF より表現力は弱い。つまり、正規表現で表現できる言語は BNF でも表現できる。
- ただし、BNF より効率よく実装できる。そのために、字句解析と構文解析をわけける。

3.1.1 正規表現の定義

$\mathcal{A}$ を想定する文字の集合とする。

1. $\mathcal{A}$ の要素 a は $\mathcal{A}$ の上の正規表現である。
2. 空記号列 (「 」と書くことがある) は $\mathcal{A}$ の上の正規表現である。
3. x と y が $\mathcal{A}$ の上の正規表現であるとき、

1. $xy \cdots$ x と y の _____
2. $x|y \cdots$ x と y の _____
3. $x^* \cdots$ x の _____ (x の 0 回以上の繰り返し)

も $\mathcal{A}$ の上の正規表現である。

演算の順序を指定するために適宜丸括弧を使用する。

(正規表現の書き方はいろいろな流儀があるが、ここでは flex の記法に準じたものを紹介する。例えば、教科書では x の 0 回以上の繰り返しを $\{x\}$ と書く流儀を採用している。)

優先順位

正規表現の演算は「 $*$ 」、(連接)、「 $|$ 」の順に優先する。演算の順番を変えるには適宜、丸括弧「 $(\sim)$ 」を使用する。

$a|b^*c$ は、 _____ のことである。
 $a|bc^*$ は、 _____ のことである。

省略記法

$x+$	は、 xx^* (x の 1 回以上の繰り返し)
$x?$	は、 $x \varepsilon$ (x の 0 回または 1 回の出現)
$[abc]$	は、 $a b c$ つまり列挙した文字のいずれか
$[a-z]$	は、 $a b \dots z$ 一般に c_1-c_2 は c_1 から c_2 の範囲の任意の文字を表す。 $[0-9a-zA-Z]$ のように組み合わせることもある。
$^{\wedge}abc$	は、 a, b, c 以外の任意の文字
$^{\wedge}a-z$	は、 $a, b, \dots, z$ 以外の任意の文字

ここで x は任意の正規表現、 a, b, c, z は文字である。なお「+」と「?」は「*」と同じ優先順位である。

例

$a(a b)a$	は、 $\{aaa, aba\}$
$a(ba)^*a$	は、 $\{aa, abaa, ababaa, \dots\}$
$(^{\wedge}+ ^{\wedge}-)^*[0-9]^+$	は、整数リテラル
$[a-zA-Z_][a-zA-Z0-9_]^*$	は、C 言語で使える変数名

「+」や「-」は正規表現で特別な意味を持つ記号なので、「 $^{\wedge}+$ 」や「 $^{\wedge}-$ 」のように二重引用符で囲むことによって、「+」「-」という文字そのものを表している。「 $\backslash$ 」のように「 $\backslash$ 」(バックスラッシュ)を使ってエスケープすることもある。

問 3.1.1 次の正規表現が表す文字列のうち、文字数が 5 以下のものをすべて列挙せよ。

1. $(a|bc)^*ab?$
2. $(a?b)^*b?c?$

3.2 有限オートマトン

正規表現をプログラムで認識することを考える。

_____ (finite automaton, FA) は正規表現を認識できる抽象機械である。

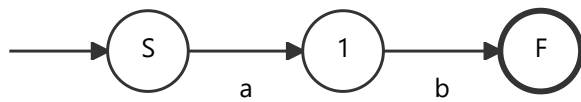
- いくつかの状態 (すごろくの“マス”) を持つ。
- 入力データ (空の場合も含む) によって状態を移る。
- 開始状態 (start state) (すごろくの“ふりだし”) は一つだけある。
- 終了状態 (final state) (すごろくの“あがり”) は一つ以上ある。

_____ (state transition diagram) の一種である。

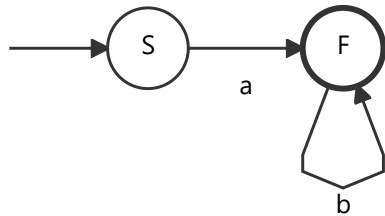
↑ 枝分かれの多い“すごろく”のようなものである。

例

"ab" を認識する FA

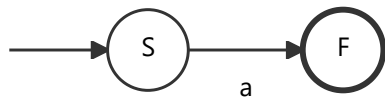


"ab*" を認識する FA



3.2.1 構成法

1 文字や ϵ を認識する FA は明らかである。例えば、"a" を認識する FA は次のようになる。



それらをベースにして、以下のように正規表現を認識する FA を構成していく。

接続 $x y$

選択 $x | y$

反復 x^*

このように正規表現に対応する文字列を認識する（受理する — つまり開始状態から終了状態に移る）FA を構成することができる。ただし、この構成法でできるのは、非決定性有限オートマトンである。

3.2.2 非決定性有限オートマトン

(nondeterministic finite automaton,) は入力によって移り先が一つに定まらないオートマトンである。(いくつかの可能性のうち、一つでも終了状態にたどりつけば受理となる。) これは、コンピュータプログラムとして実装するのには少し困る。

3.2.3 決定性有限オートマトン

_____ (deterministic finite automaton, _____) は移り先が一つに定まるオートマトンである。

3.3 部分集合構成法

NFA を“同等の”（つまり同じ記号列を受理する）DFA に変換するアルゴリズム（ ）を紹介する。

アイデア: NFA の状態の集まり (部分集合) を DFA の一つの状態とみなす。

記号

$\varepsilon\text{-closure}(S)$	NFA の状態の集合 S から ε だけで遷移できる状態の集合
$\text{move}(T, a)$	NFA の状態の集合 T から a と ε だけで遷移できる状態の集合

$\varepsilon\text{-closure}(S_0)$ を部分集合として付け加えて、それまでに付け加えられた部分集合 T に対して $\text{move}(T, a)$ を新しい状態として加えていく。 (S_0 はもとの NFA の開始状態)

例

正規表現 $(a|b)^*abb$ に対応する NFA

これに部分集合構成法を適用する。

$\varepsilon\text{-closure}(\{0\}) = \underline{\hspace{2cm}} \equiv \underline{\hspace{1cm}} \text{とおく}$
 $\text{move}(A, a) = \underline{\hspace{2cm}} \equiv \underline{\hspace{1cm}} \text{とおく}$
 $\text{move}(A, b) = \underline{\hspace{2cm}} \equiv \underline{\hspace{1cm}} \text{とおく}$
 $\text{move}(B, a) = \underline{\hspace{2cm}} = \underline{\hspace{1cm}}$
 $\text{move}(B, b) = \underline{\hspace{2cm}} \equiv \underline{\hspace{1cm}} \text{とおく}$
 $\text{move}(C, a) = \underline{\hspace{1cm}}$
 $\text{move}(C, b) = \underline{\hspace{1cm}}$
 $\text{move}(D, a) = \underline{\hspace{1cm}}$
 $\text{move}(D, b) = \underline{\hspace{2cm}} \equiv \underline{\hspace{1cm}} \text{ (9 は入らない)}$
 $\text{move}(E, a) = \underline{\hspace{1cm}}$
 $\text{move}(E, b) = \underline{\hspace{1cm}}$

まとめると次の図のようになる。

これがもとの NFA と同等な DFA になっているが、その証明は割愛する。

また、この例では a と b の 2 文字しか使っていないので状態数はそう多くないが、一般的には NFA から作った DFA は状態数がとても多くなる。

3.4 DFA の状態数の最小化

DFA を同じ文字列を受理する状態数最小の DFA に変換することができる。

- まず、状態を終了状態とそれ以外の状態の 2 つの部分集合にわけろ。
- 前のステップの分割を“区別”する入力があれば、さらに部分集合に分割する。これを、区別する入力なくなるまで繰り返す。

例

E だけが終了状態なので、先ほどの DFA はまず、次のような 2つの部分集合に分けられる。

_____ と _____

ここで、 $D \xrightarrow{b} E$ だが A, B, C からは b によって E に移動できないので、次の分割は次のようになる。

_____ と _____ と $\{E\}$

さらに、 $B \xrightarrow{b} D$ だが A, C からは b によって D に移動できないので、次の分割は次のようになる。

_____ と _____ と $\{D\}$ と $\{E\}$

ここで、 $A \xrightarrow{a} B$ かつ $C \xrightarrow{a} B$ で $A \xrightarrow{b} C$ かつ $C \xrightarrow{b} C$ なので、 A と C を区別する入力はない。つまり、これ以上は分割できない。

結局、 A と C は一つの状態としてまとめることができ、先ほどの DFA を最小化したものは次のようになる。

また、これも証明は割愛するが、同じ言語を受理する状態数最小の DFA はただ一つであることを示すことができる。このことから、DFA の状態数を最小化することで、正規表現の同値性を証明することもできる。

状態遷移表

以上の結果は次のように表 (_____) にまとめることができる。

	a	b
AC		
B		
D		
E		

あとはこれをプログラムにするのは容易である。

問 3.4.1 上の状態遷移表を用いて、 $(a|b)^*abb$ にマッチする文字列には 1 を、マッチしない文字列には 0 を返す関数 `int ababb(char* str)` を定義せよ。

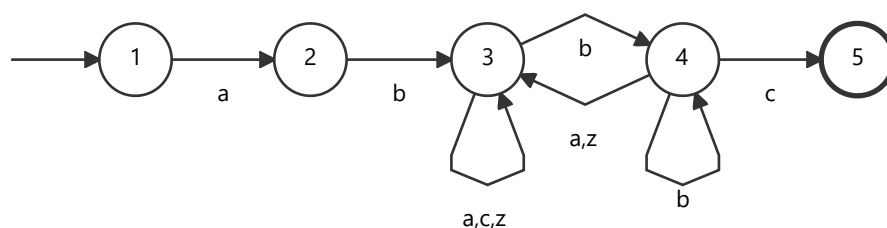
3.5 字句解析系の自動生成

以上の作業は自動化できる。lex や flex などのツールは、正規表現（とそれに対応する動作）から C 言語などで記述された字句解析部を自動生成する。ただし、自動生成に頼らず、手書きする場合もある。

3.6 有限オートマトンと正規表現の同値性

決定性有限オートマトン (DFA) から同じ文字列を認識する正規表現を構成することができる。つまり、正規表現で表現できる言語全体（つまり、正規言語）と有限オートマトンで受理できる言語全体は一致する。例を使って、構成法を説明する。

例:



決定性有限オートマトンの状態に 1 から n までの番号がつけられているとする。そして R_{ij}^k という記号を途中で k を超える状態を通ることなしに、状態 i から状態 j へ遷移させる文字列の集合を表す正規表現とする。特に R_{ij}^0 は途中に他の状態を通らずに状態 i から状態 j へ遷移させる文字列の集合を表す正規表現である。

上のオートマトンの場合、次のようになる。

$$R_{11}^0 = \varepsilon, R_{12}^0 = a, R_{22}^0 = \varepsilon, R_{23}^0 = b, R_{33}^0 = \varepsilon | a | c | z, R_{34}^0 = b, \\ R_{44}^0 = \varepsilon | b, R_{43}^0 = a | z, R_{45}^0 = c, R_{55}^0 = \varepsilon,$$

それ以外の i, j については $R_{ij}^0 = \emptyset$ である。（空集合 $\emptyset$ に対応する正規表現を、やはり $\emptyset$ で表すことにする。）

上の例で開始状態を 1、終了状態を 5 とするとき、 R_{15}^5 とそれを表す正規表現を構成できればよい。ここで一般に $k > 0$ のとき

$$R_{ij}^k = R_{ij}^{k-1} \mid R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$$

という関係が成り立つ。

これにより、まずすべての i, j に対して $R_{ij}^1 = R_{ij}^0$ となる。

次に $R_{13}^2 = R_{13}^1 \mid R_{12}^1 R_{22}^1 * R_{23}^1 = ab$ となり、それ以外の i, j に対しては $R_{ij}^2 = R_{ij}^1$ となる。

以下では、必要なものだけ計算していくと、

$$\begin{aligned}
R_{14}^3 &= R_{14}^2 \mid R_{13}^2 R_{33}^2 * R_{34}^2 = ab(a|c|z)*b, \\
R_{44}^3 &= R_{44}^2 \mid R_{43}^2 R_{33}^2 * R_{34}^2 = \varepsilon|b|(a|z)(a|c|z)*b, \\
R_{15}^4 &= R_{15}^3 \mid R_{14}^3 R_{44}^3 * R_{45}^3 = ab(a|c|z)*b(b|(a|z)(a|c|z)*b)*c \\
R_{15}^5 &= R_{15}^4
\end{aligned}$$

となり、求める正規表現は $ab(a|c|z)*b(b|(a|z)(a|c|z)*b)*c$ である。
 ちなみに DFA の状態につける番号の順を逆にすると、この手順で求まる正規表現は $ab(a|c|z|bb^*(a|z))*bb^*c$ となるが、どちらも同等の正規表現である。

3.7 (正規言語の) 反復補題

正規表現では表現できない記号列の集合がある。例えば、正規表現はかっこの入れ子（かっこの数が釣り合っている文字列の集合）は表現できない。

より、一般的に次の補題が成り立つ。

正規言語の反復補題 (pumping lemma for regular languages)

反復補題は _____ ともいう。

正規表現が表現する（つまり有限オートマトンが受理する）言語 L に対して、次のような条件を満たす自然数 $p (\geq 1)$ が存在する。

L に属する長さ p 以上の任意の文字列 w は $w = xyz$ と書けて、

1. y の長さは 1 以上である。
2. xy の長さは p 以下である。
3. すべての $i (\geq 0)$ に対して、 $xy^i z$ も L に属する。

証明は、有限オートマトンの状態数が $p - 1$ ならば、 p 以上の長さの記号列を読み込んだときに、必ず以前と同じ状態に到達することから言える。

例

$\{\varepsilon, (), ((())), (((()))), \dots\}$ のように「(」と「)」を同数個含む記号列の集合 L は、正規表現では表せない。

証明: 正規表現で表せたとして矛盾を導く。 L を正規表現で表せたとすると、ポンプの補題により、上のような条件を満たす自然数 p が存在する。 $w = ({}^p)^p$ とする。すると、 $w = xyz$ と分解したときに、 y の部分には「(」しか存在しない。このとき、 $xy^2 z$ は「(」と「)」の数が異なるが、 L に属することになり矛盾する。

3.8 正規言語の閉包性

正規表現と有限オートマトンの同値性から、次のような事実もすぐに導かれる。

- 正規言語の補集合も正規言語である。
- 2 つ以上の正規言語の共通部分も正規言語である。

などである。

