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（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. $x\ y$ ・・・ x と y の＿＿
    2. $x|y$ ・・・ x と y の＿＿
    3. $x*$ ・・・ 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。

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括弧「(～)」を使用する。

a¦b*c　　は、＿＿＿＿のことである。
a¦bc*　　は?＿＿＿＿

省略記法

$x+$　　は、$xx*$（x の 1 回以上の繰り返し）
$x?$　　は、$x|\epsilon$（x の 0 回または 1 回の出現）
[abc]　　は、a¦b¦c
[a-z]　　は、a¦b¦...¦z
[^abc]　　は、a, b, c 以外の任意の文字

[^a-z]　　は、a, b, ..., 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。

例

a(a¦b)a	は、{ aaa, aba }
a(ba)*a	は、{ aa, abaa, ababaa, ... }
("+"¦"-")?[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、「"+"」や「"-"」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。「\+」のように「\」（バックスラッシュ）を使って表す流儀もある。

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。

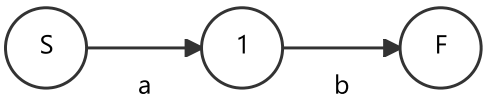
(finite automaton, FA) は正規表現を認識できる抽象機械である。

- 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。

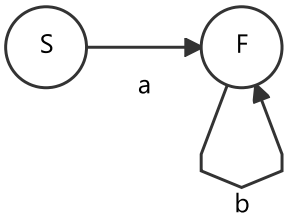
例

"ab" を認識する FA



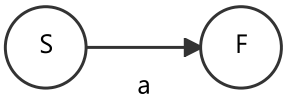
seisoku

"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。

連接 $x\ y$

選択　$x \mid y$

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、コンピューターの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

ϵ-$closure(s)$	NFA の状態 s から ϵ だけで遷移できる状態の集合
$move(T, a)$	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

ϵ-$closure(S_0)$ 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 $move(T, a)$ 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。

ϵ-$closure(0)$ = _____ $\equiv$ A とおく
$move(A, a)$ = _____ $\equiv$ B
$move(A, b)$ = _____ $\equiv$ C
$move(B, a)$ = _____ = B
$move(B, b)$ = _____ $\equiv$ D
$move(C, a)$ = _

$move(C, b) \quad =$
$move(D, a) \quad =$
$move(D, b) \quad =$ $\equiv \quad E$（9 は入らない）
$move(E, a) \quad =$
$move(E, b) \quad =$

まとめると次の図のようになる。

この例ではそうではないが、一般に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。（変換方法の説明は割愛する。）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。

例

さきほどの DFA を最小化したものは次のようになる。

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。

	AC	B	D	E
a				
b				

あとはプログラムにするだけである。

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

証明は省略するが、有限オートマトンから同じ文字列を認識する正規表現を構成することも可能である。 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。

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^iz 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

$\{\ \epsilon, (), (()), ((())), (((()))), \dots\ \}$ 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^2z 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。