第 5 講 正規言語と正規表現

林　恒俊

正規言語

- 正規言語は取扱いが最も容易な言語のクラスであり、言語に関する様々な性質を決定的に判断することができる。
- 正規言語は言語の関する最も基本的と見なされる演算を組み合わせて構成される。
- 通常正規言語は帰納的な定義技法により定義される。

正規言語の定義

- あるアルファベット Σ 上の正規言語は次のように定義される。
  - 空集合 $\emptyset$ は正規言語である。
  - $\sigma \in \Sigma$ であるすべての σ について 1 個の記号のみからなる言語 $\{\sigma\}$ は正規言語である。
  - もし $R_1, R_2, \ldots, R_n$ が正規言語なら $R_1 \cup R_2 \cup \cdots \cup R_n$ も正規言語である。ただし $n \geq 2$
  - もし $R_1, R_2, \ldots, R_n$ が正規言語なら $R_1 \circ R_2 \circ \cdots \circ R_n$ も正規言語である。ただし $n \geq 2$
  - もし R_1 が正規言語なら R_1^* も正規言語である。
  - 正規言語はこれらの規則のみを適用して得られたものである。
- これは典型的な帰納型定義である。
- 空列からなる言語 $\{\Lambda\}$ も正規言語である。なぜなら $\emptyset^* = \{\Lambda\}$。

- なお言語に関する演算 $(^*, \circ, \cup)$ はこの順に優先するので定義中の演算順を明示する場合括弧でグループ化する。

正規言語の例

- 要素の数が有限な言語はすべて正規言語である。(なぜか)
- アルファベット $\{a\}$ 上の言語

$$\{a^n \mid n \geq 0\} = \{\Lambda, a, aa, aaa, \ldots\} = \{a\}^*$$

  は正規言語である。
- アルファベット $\{a, b\}$ 上のすべての記号列を含む言語 $\{a, b\}^*$ あるいは $(\{a\} \cup \{b\})^*$ は正規言語である。
- アルファベット $\{a, b\}$ 上の a を偶数個含む記号列からなる言語は正規言語である。次の言語 $\{b\}^*(\{a\}\{b\}^*\{a\}\{b\}^*)^*$ をよく考察すること。

記号列のパターンによる言語定義

- 言語はその要素の記号列のパターンを与えることにより定義することも可能である。$\{a^n b^n \mid n \geq 0\}$ という言語定義型式も要素記号列パターンを定義しているといってもよい。
- つぎに要素記号列がパターンで定義されるもう一つの言語のクラスを考察する。

正規表現とはなにか

- Unix のコマンド行やテキストの検索や言語処理系の字句処理などでファイル名や文字列のパターンを指定するためにいわゆる**正規表現** (**regular expression**) が利用されている。

  - 例えば英文のテキスト中ある単語がいくつ出現するか調べようとする場合を考える。このような作業を誤りなく実行するためには機械を利用するとよい。英文は通常テキスト形式のデータで用意されているので、その単語もテキストデータで用意し英文データと適合すればよい。
  - 単語を “say” とするとこれは文字 “s” と “a” と “y” を連結したものと考えられる。つまり “s ∘ a ∘ y” あるいは “say” で検索する。
  - 単語 say には変化があり says や said も say の検索対象にしなければならない。今選択パターンを “|” で代表すると検索対象は “say|says|said” と表示できる。あるいはる略記すると “sa(y|ys|id)” である。これも記号列パターンの一部と考えられる。
  - 繰返しパターンを “*” で代表すると例えば “a” が偶数個連続するようなパターンは “(aa)*” と表示される。
  - このように連結、選択、繰返しで構成された記号列パターンを正規表現と呼んでいる。

- しかしこれらの正規表現と呼ばれるものの内容は応用の場毎に少しづつ異なったものになっている。例えば Unix のファイル名の表現と grep コマンドのパラメータの正規表現を比較してみるとよい。
- このように正規表現はテキストのパターンを定義している。これはそのパターンを満たす記号列の集合を定義していることと同等である。すなわちパターンを満たす記号列を要素とする言語を定義していることになる。
- 以下では正規表現が定義する言語について検討する。(実はこの言語のクラスは正規言語と同じである)
- ここでは正規表現を他の表現や式と区別するため専用の文字種を利用する。

正規表現

- アルファベット Σ 上の正規表現は Σ 上の言語を定義する。パターンを表現するためには決まった書き方 (文法) に従わなければならない。
- 正規表現は $\Sigma \cup \{(,), |, *\}$ というアルファベット上の記号列であり次の**文法** (**grammar**) に従う。
  - 記号列 () は正規表現である
  - $\sigma \in \Sigma$ であるすべての σ は正規表現である
  - E が正規表現なら (E) も正規表現である
  - E_1, E_2 が正規表現なら $(E_1|E_2)$ も正規表現である
  - E_1, E_2 が正規表現なら $(E_1\,E_2)$ も正規表現である
  - E が正規表現なら $E*$ も正規表現である
  - 以上の規則に基づいて構成された記号列のみが正規表現である
- なおこの文法も帰納法によって記述されていることに注意。

考察

正規表現は次のように BNF で与えることもできる。

<regular expression> ::=
(<regular expression>) |
(<regular expression>|<regular expression>) |
(<regular expression> <regular expression>) |
<regular expression>*

ただし <alphabet> はアルファベットの要素を | で選択するようにしたものである。

正規表現の例

- アルファベット $\{a, b\}$ 上の正規表現のいくつかを示す。
  - `()`
  - `a*`
  - `(a|b)*`
  - `((((a|b)*a)b)a)`
  - `(b*((ab*)(ab*))*)`
- 最後の 2 例は `(a|b)*aba`、`b*(ab*ab*)*` と表記することもある。

正規表現が与える記号パターン

- 正規表現に含まれる括弧 (と) は記号パターンを纏めて曖昧にならないように使われる。曖昧でない場合括弧は省略してもよい。
- `((ab)a)` は a と b と a がこの順に連結していることを示している。`aba` と略記することができる。そしてこのパターンは記号列 aba に適合する。
- 記号 | はこの記号の両側のパターンのいずれかを選択して全体のパターンとしてもよいことを示している。したがってパターン `(a|b)` は記号 a あるいは b に適合する。
- 記号 * は零回以上の繰返しのパターンを表している。例えばパターン `a*` は記号列 $\Lambda, a, aa, aaa, \ldots$ と適合する。
- なおこれらのパターン操作は *, 連結操作, | の順に優先する。
- `(a|b)*` は a と b からなるすべての記号列と適合する。(何故か)
- `(a|b)*aba` はアルファベット $\{a, b\}$ 上の記号列で接尾辞が aba である記号列と適合する。言換えると `(a|b)*aba` は接尾辞が aba からなる記号列から構成される言語 $\{aba, aaba, baba, aaaba, \ldots\}$ を定義している。

正規表現が定義する言語

- 正規表現に言語を対応づけることは正規表現の構文に対して**意味定義** (**semantics**) を与えるのと同様である。この点で正規表現は一種のプログラミング言語と見なされる。
- あるアルファベット Σ 上の正規表現 E が定義する言語を $L(E)$ とすると $L(E)$ は次のように与えられる。
  - $E = \mathtt{()}$ なら $L(E) = \emptyset$ である。
  - $\sigma \in \Sigma$ なる σ について $E = \sigma$ なら $L(E) = \{\sigma\}$ である。
  - 正規表現 E_1 について $E = \mathtt{(}E_1\mathtt{)}$ なら $L(E) = L(E_1)$ である。
  - 正規表現 E_1, E_2 について $E = \mathtt{(}E_1\mathtt{|}E_2\mathtt{)}$ なら

    $$L(E) = L(E_1) \cup L(E_2)$$

    である。
  - 正規表現 E_1, E_2 について $E = \mathtt{(}E_1\,E_2\mathtt{)}$ なら

    $$L(E) = L(E_1) \circ L(E_2)$$

    である。
  - 正規表現 E_1 について $E = E_1\mathtt{*}$ なら $L(E) = L(E_1)^*$ である。
- $\emptyset^* = \{\Lambda\}$ から $L(\mathtt{()*}) = \emptyset^* = \{\Lambda\}$
- この定義から正規表現が与えられると必ず対応する正規言語が存在することが判る。なぜなら上記の定義は正規言語の定義そのものだからである。
- 以上から $\mathcal{L}_{\mathbf{RE}} \subseteq \mathcal{L}_{\mathbf{RL}}$

正規表現から正規言語構成例

- 上記の正規表現の例は、それぞれ次のような言語を示している。
  - $\emptyset$

- $\{a\}^*$
- $(\{a\} \cup \{b\})^*$ または $\{a, b\}^*$
- $(\{a\} \cup \{b\})^*\{a\}\{b\}\{a\}$ または $\{a, b\}^*aba$
- $\{b\}^*(\{a\}\{b\}^*\{a\}\{b\}^*)^*$ または $b^*(ab^*ab^*)^*$

正規言語から正規表現

- 次に正規言語を与えられた場合その言語を定義する正規表現は次のような規則に従って構成すればよい。ただし正規言語を R とすると対応する正規表現を $E(R)$ で表示する。
  - 正規言語 R が空集合 $\emptyset$ なら

    $$E(R) = E(\emptyset) = ()$$

    とする。
  - 正規言語 R が 1 個の記号 σ のみからなる言語 $\{\sigma\}$ の場合

    $$E(R) = E(\{\sigma\}) = \sigma$$

    とする。
  - 正規言語 R が $R = R_1 \cup R_2 \cup R_3 \cup \cdots \cup R_n$ なら対応する正規表現は

    $$((\cdots((E(R_1)\,|\,E(R_2))\,|\,E(R_3))\,|\,\cdots)\,|\,E(R_n))$$

    とする。
  - 正規言語 R が $R = R_1 \circ R_2 \circ R_3 \circ \cdots \circ R_n$ なら対応する正規表現は

    $$((\cdots((E(R_1)\,E(R_2))\,E(R_3))\,\cdots)\,E(R_n))$$

    とする。
  - 正規言語 R が $R = R_1^*$ なら対応する正規表現は $E(R_1)*$ とする。
- 以上の規則により正規言語に対応する正規表現を必ず構成可能である。すなわち $\mathcal{L}_{\mathbf{RL}} \subseteq \mathcal{L}_{\mathbf{RE}}$

- なお $\emptyset^* = \{\Lambda\}$ から $E(\{\Lambda\}) = E(\emptyset^*) = ()*$ である。
- $\mathcal{L}_{\mathbf{RE}} \subseteq \mathcal{L}_{\mathbf{RL}}$ かつ $\mathcal{L}_{\mathbf{RL}} \subseteq \mathcal{L}_{\mathbf{RE}}$ から $\mathcal{L}_{\mathbf{RL}} = \mathcal{L}_{\mathbf{RE}}$。すなわち正規言語と正規表現が定義する言語は同じ種類の言語である。
- 正規言語と正規表現の定義する言語のクラスが同一であることが証明された。すなわち
  - 任意の正規言語は必ず正規表現で表現可能
  - 任意の正規表現は必ず正規言語で記述可能

  である。

考察

前述のように正規表現は $\Sigma \cup \{(,), |, *\}$ 上の記号列である。従って正規表現は可付番ということになる。ところが Σ 上のすべての言語の集合は可付番ではない。結論として正規言語ではない言語が無数に存在することになる。

正規表現の応用

- 正規表現は比較的簡単な記号列を定義するのに向いているため広く利用されている。以下に典型的な応用例を示している。
- Unix のファイル式。Unix が典型的であるがすべての行コマンド型制御プログラムでは何らかの方法でファイルを指定する式が定義されている。そしてこのような定義には擬似的な正規表現が利用されることが多い。例えば `img*.jpg` というようなパラメータが認められている。なおこの表現の処理は普通コマンドシェルに実装される。
- テキスト編集時の文字パターン探索。これも Unix の `grep` コマンドが有名であるが、それ以外にも大抵のテキスト編集プログラムは一定の文字パターンを検出するため正規表現による探索が可能である。例えば行コマンド型エディタのサブコマンド `find [a-zA-Z][0-9]*` は 1 個のアルファベットの次に数字が 0 個以上続く文字列を検出する。

- 言語処理系において字句処理の字句を定義するために利用される。演算記号や予約語なら比較的単純な正規表現で定義できるが、例えば浮動小数点数のようなものを正規表現で定義しようとするとかなり工夫が必要である。一度試みてみよ。

構造化プログラムと正規表現

- Dijkstra の提案した**構造化プログラム** (**structured programming**) では次の 3 通りの制御構造のみからプログラムを構成する。
- 文を S, S_1, S_2, …, S_n で、論理式を B, B_1, B_2, …, B_n で代表すると
  - 列 (sequence) **begin** S_1; S_2; …; S_n **end**
  - 選択 (selection) **if** B **then** S
if B **then** S_1 **else** S_2
  - 繰返し (repetition) **while** B **do** S
repeat S **until** B
- これらの制御構造で実行される式と文の列はそれぞれ次の正規表現で示される。
  - 列の表現 $(\cdots((S_1S_2)S_3)\cdots S_n)$
  - 選択表現 $(B(S|()*))$
$(B(S_1|S_2))$
  - 繰返し表現 $(B(SB)*)$
$(SB(SB)*)$
- 構造化プログラムでは文の実行列を正規表現で表示することができる。