第 1 章

準備

1.1　正規表現

正規表現とは言語 (文字列の集合) を一つの文字列で表現する方法である．正規表現は文字列のパターンマッチングなどに使われ，その実用性の高さから多くのプログラミング言語で扱うことができる．実世界のプログラミング言語で使用されるような正規表現の中には，正規言語 (ある正規表現にマッチする文字列全体の集合) 以上の表現力を持つものもある (後方参照の機能を提供している正規表現など) が，本論文では次に示すように純粋な正規表現を対象とする．

1.1.1　正規表現の構文

本論文で扱う正規表現の構文を以下に示す．

$$
\begin{array}{lll}
\text{正規表現} \quad r & := & \varepsilon \\
 & & \mid c \\
 & & \mid r_1 \mid r_2 \\
 & & \mid r_1 r_2 \\
 & & \mid r^*
\end{array}
$$

図 1.1　正規表現の構文

ε は空列を表す定数，c は任意の 1 文字 (本論文では ASCII コードを想定している) を表す定数，r, r_1, r_2 は正規表現を表す変数とする．$r_1 \mid r_2$ は選択，すなわち r_1 に含まれ

る文字列の集合と r_2 に含まれる文字列の集合の和集合を表す．r_1r_2 は連接，すなわち r_1 に含まれる文字列に r_2 に含まれる文字列をつなげてできる文字列の集合を表す．r^* はクリーネ閉包，すなわち r に含まれる文字列を 0 個以上つなげてできる文字列の集合を表す．

また，本論文では先の正規表現で表現可能ないくつかの拡張機能を搭載している．以下にどのような拡張を搭載したか述べる．

- . 任意の 1 文字にマッチすることを表す．
- $[a_ia_{i+1}\dots a_j]$ $a_i, a_{i+1}\dots a_j$ のうちのどれか 1 文字にマッチすることを表す．特に，$[a_i\text{-}a_j]$ は文字コード上で i から j 番目にある文字のうちどれか 1 文字にマッチすることを表す．
- $[\hat{}a_ia_{i+1}\dots a_j]$ $a_i, a_{i+1}\dots a_j$ 以外のどれか 1 文字にマッチすることを表す．
- $r?$ 正規表現 r に属する文字列が 0 回または 1 回出現することを表す．

1.1.2 正規表現を用いた文字列のパターンマッチング例

正規表現を用いた文字列のパターンマッチング例を見る．$a \mid b$ という正規表現には a, b といった文字列がマッチする．ab という正規表現には ab といった文字列がマッチする．$\varepsilon, a, aa, aaa\dots$ といった文字列がマッチする．a^* という正規表現には $\varepsilon, a, aa, aaa, \dots$ といった文字列がマッチする．最後に，$(a|bc)^*$ という正規表現には $\varepsilon, a, bc, aa, abc, bca, bcbc, \dots$ といった文字列がマッチする．

また，本論文で扱うような正規表現においてパターンマッチングに成功した (正規表現に受理されるとも言う) 文字列集合を正規言語と呼ぶ．そして，ある正規表現 $\mathcal{A}$ に受理される言語のことを $\mathcal{L}(\mathcal{A})$ と書くことにする．

1.2 NFA

NFA(Nondeterministic Finite Automaton) は有限オートマトンの一種で，ある状態から入力を受け取ったとき，次の状態への遷移が一意に決まらないことがあるものである．正規表現はそれと等価な NFA へと変換することが可能であるので，今後は正規表現のままではなくそれと等価な NFA を用いることにより議論を進める．

1.2.1 NFA の定義

先行研究で紹介されていた NFA の定義を以下に示す.

定義 1（(ε 遷移なし)NFA） NFA $\mathcal{A}$ は 5 つの組 $(Q, \Sigma, \Delta, q_0, F)$ からなる．Q は状態の有限集合，Σ はアルファベット (文字) の有限集合，$\Delta : Q \times \Sigma \to 2^Q$ は遷移関数である．$q_0 \in Q$ を初期状態，$F \subseteq Q$ を受理状態の集合とする．また，もし $q' \in \Delta(q, l)$ なら (q, l, q') をラベルを経由した遷移と呼ぶこととする.

また，関連してパスという用語の定義も行う.

定義 2（(ε 遷移なし NFA における) パス） NFA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ のパス π を遷移列，すなわち $(q_1, l_1, q_2), \ldots, (q_{m-1}, l_{m-1}, q_m)$ とする．ただし，$q_i \in Q$,$l_i \in \Sigma$,$q_{i+1} \in \Delta(q_i, l_i)$ である．つまり，π は q_i から始まって q_m で終わるパスを表す．また，$\mathrm{labels}(\pi)$ はラベル列，$(l_1, \ldots, l_{m-1})$ を意味するものとする.

この NFA の定義は一般的ではない．Σ に空列 ε を加えた $\Sigma \cup \{\varepsilon\}$ をアルファベット集合として 5 つ組 $(Q, \Sigma \cup \{\varepsilon\}, \Delta, q_0, F)$ を NFA の定義とすることが多い．すなわち,

定義 3（NFA） NFA $\mathcal{A}$ は 5 つの組 $(Q, \Sigma \cup \{\varepsilon\}, \Delta, q_0, F)$ からなる．Q は状態の有限集合，Σ はアルファベット (文字) の有限集合，ε は空列，$\Delta : Q \times \Sigma \to 2^Q$ は遷移関数である．$q_0 \in Q$ を初期状態，$F \subseteq Q$ を受理状態の集合とする．また，もし $q' \in \Delta(q, l)$ なら (q, l, q') をラベルを経由した遷移と呼ぶこととする.

また，パスも以下のように書き換える.

定義 4（パス） NFA $\mathcal{A} = ((Q, \Sigma \cup \{\varepsilon\}, \Delta, q_0, F))$ のパス π を遷移列，すなわち $(q_1, l_1, q_2), \ldots, (q_{m-1}, l_{m-1}, q_m)$ とする．ただし，$q_i \in Q$,$l_i \in \Sigma \cup \{\varepsilon\}$,$q_{i+1} \in \Delta(q_i, l_i)$ である．つまり，π は q_i から始まって q_m で終わるパスを表す．また，$\mathrm{labels}(\pi)$ はラベル列，$(l_1, \ldots, l_{m-1})$ を意味するものとする.

今後はただ NFA とだけ記載した場合，ε 遷移ありの NFA を指す.

NFA に文字列を入力したとき，初期状態 q_0 から始まって一つずつ文字を読み取って遷移していき，ちょうど受理状態に到達するとき，その文字列は NFA に受理されるという．受理状態以外の状態で読み取りが終了した場合，その文字列は NFA に拒絶されるという．NFA に受理される文字列全体の集合を NFA の受理する言語という.

1.2.2 ε 遷移なし NFA の例

(ϵ 遷移なし)NFA $\mathcal{A}_1$ を $(\{q, q'\}, \{a, b\}, \Delta, q, \{q'\})$ とする．なお，遷移関数 Δ は以下のような状態遷移表で表されるものとする．表の縦軸が状態，横軸がアルファベットを表す．ある状態とあるアルファベットが交差するセルは，ある状態からあるアルファベットを読んで遷移する状態の集合を表している．例えば，状態 q からアルファベット a を読んで遷移する状態の集合は $\{q, q'\}$ である．

表 1.1　遷移関数 Δ

	a	b
q	$\{q, q'\}$	$\{\}$
q'	$\{\}$	$\{q, q'\}$

NFA $\mathcal{A}_1$ をグラフにすると以下のようになる．

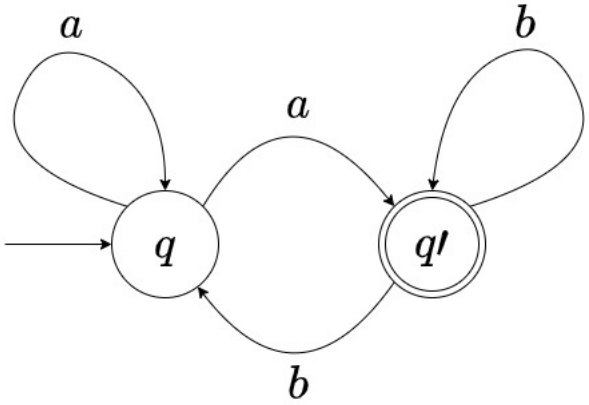

図 1.2　NFA $\mathcal{A}_1$

この NFA が受理する言語は $\{a, aa, ab, aaa, aab, aba, abb, \ldots\}$ である．これは正規表現 $a(a|b)^*$ に対応する．

1.3　正規表現の等価性判定

2 つの正規表現が等価であるとはどういうことか，また等価性判定がどのようなアルゴリズムで行われているのかを述べる．

1.3.1 正規表現の等価性

ある 2 つの正規表現 $\mathcal{A}, \mathcal{B}$ が存在し，それらの正規表現にマッチする文字列の集合 (つまり正規言語) が等しいとき，すなわち $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ であるとき，$\mathcal{A}, \mathcal{B}$ は意味的に等価であるという．以下に例を挙げる．

2 つの正規表現が構文的に等しい場合，例えば $\mathcal{A} = x^*, \mathcal{B} = x^*$ としたとき，$\mathcal{L}(x^*) = \{\varepsilon, x, xx, xxx, \ldots\}$ よりこれらは意味的にも等しい．

2 つの正規表現が構文的に等しくなく，意味的に等しい場合について考える．例えば $\mathcal{A} = x^*, \mathcal{B} = x^*x^*$ としたとき，これらは構文的に等しくないが，$\mathcal{L}(x^*) = \mathcal{L}(x^*x^*) = \{\varepsilon, x, xx, xxx, \ldots\}$ より意味的には等しい．

1.3.2 正規表現の等価性判定アルゴリズム

正規表現の等価性判定を行う方法はいくつか知られている．

例えば，有名なアルゴリズムとして次のようなものがある．まず 2 つの正規表現を NFA 化する．次に 2 つの NFA を DFA(Deterministic Finite Automaton，ある状態からの遷移先が一意に定まった有限オートマトン) 化する．さらに 2 つの DFA を最小化 (不要な状態の削減) する．最小化した際に得られる DFA は一意に定まることが知られている．この事実を利用し，2 つの最小 DFA が一致するかどうかを確認することで等価性判定を行うことができる．上記のようなアルゴリズムは [1] で紹介されている．

また，他にも正規表現の微分 ([2],[3]) を利用した等価性判定アルゴリズム ([4]) もある．

しかし，本研究では先の 2 つのアルゴリズムではなく，次のようなアルゴリズムを用いることにする (このアルゴリズムも有名で [5] で紹介されている)．

1. 2 つの正規表現 $\mathcal{A}, \mathcal{B}$ を NFA 化し，さらに DFA 化し，それらを $\mathcal{A}_{DFA}, \mathcal{B}_{DFA}$ とする．
2. $\mathcal{A}_{DFA}$ と $\mathcal{B}_{DFA}$ の否定を取った DFA($\overline{\mathcal{B}_{DFA}}$ と表記する) との積を取り，これを $\mathcal{A}_{DFA} \cap \overline{\mathcal{B}_{DFA}}$ と表記する．また，同様に $\overline{\mathcal{A}_{DFA}} \cap \mathcal{B}_{DFA}$ も構成する．なお，正規言語は積集合演算，補集合演算について閉じているので，これらもまた DFA である．
3. $\mathcal{L}(\mathcal{A}_{DFA} \cap \overline{\mathcal{B}_{DFA}}) = \varnothing$ かつ $\mathcal{L}(\overline{\mathcal{A}_{DFA}} \cap \mathcal{B}_{DFA}) = \varnothing$ なら 2 つの正規表現 $\mathcal{A}, \mathcal{B}$ は等価，そうでないなら等価ではない．

ここで DFA の積，否定を取る工程が発生しているが，具体的な構成方法は [5] を参照

されたい．

　このアルゴリズムのメリットとして 2 つの正規表現が等価でないことを示す反例を探しやすいということが挙げられる．後に提案する手法ではこの反例を利用する．

参考文献

[1] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[2] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, Vol. 11, No. 4, pp. 481–494, oct 1964.

[3] SCOTT OWENS, JOHN REPPY, and AARON TURON. Regular-expression derivatives re-examined. *Journal of Functional Programming*, Vol. 19, No. 2, pp. 173–190, 2009.

[4] Alexander Krauss and Tobias Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Automated Reasoning*, Vol. 49, pp. 95–106, 2012. published online March 2011.

[5] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.