

第 1 章

背景

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。

正規表現	r	$:=$	ε
			$ $ c
			$ $ $r_1 r_2$
			$ $ $r_1 r_2$
			$ $ r^*

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。 $r_1 | r_2$ 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。 $r_1 r_2$ は接続，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。
- $[a_i a_{i+1} \dots a_j]$ $a_i, a_{i+1} \dots a_j$ のうちのどれか 1 文字にマッチすることを表す。特に， $[a_i \dots a_j]$ 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。
- $[\wedge a_i a_{i+1} \dots a_j]$ $a_i, a_{i+1} \dots a_j$ 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 $\varepsilon, a, bc, aa, abc, bca, bcabc, \dots$ といった文字列がマッチする。

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 (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， $\Delta : Q \times \Sigma \rightarrow 2^Q$ 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.

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 組 $(Q, \Sigma \cup \{\varepsilon\}, \Delta, q_0, F)$ を NFA の定義とすることが多い.

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 の受理する言語と言う.

1.2.2 NFA の例

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なになる.

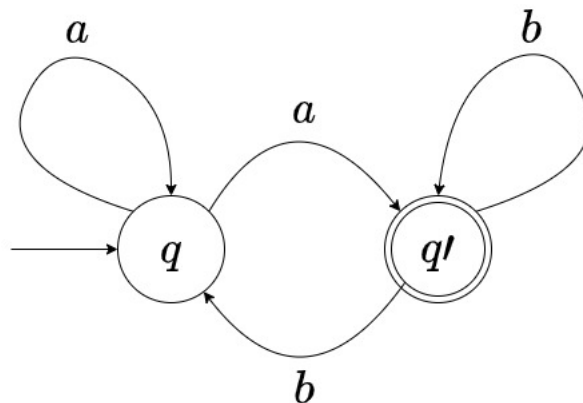


図 1.2 NFA \mathcal{A}_1

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

1.3 ReDoS 脆弱性

文字列のパターンマッチングにかなり時間がかかってしまうような正規表現が存在することが知られている. そのような正規表現を ReDoS(regular expression denial-of-service) 脆弱な正規表現と呼ぶ. 本節ではまず文字列のパターンマッチングはどのようなアルゴリズムで行われているかを紹介し, その次に正規表現の脆弱性について定式化を行う.

1.3.1 バックトラック探索

大抵のプログラミング言語が正規表現ライブラリを提供している. そのうち多くの正規表現ライブラリはバックトラック探索アルゴリズムを用いて文字列のパターンマッチングを行っている. バックトラック探索アルゴリズムがどのようなものなのかを [1] の論文で紹介されていた例を用いて説明する. バックトラック探索アルゴリズムについての詳細は [2] を参照されたい (プログラミング言語 Java におけるバックトラック探索アルゴリズムを用いた文字列パターンマッチングについて詳細に述べられている).

参考文献

- [1] Cristian-Alexandru Staicu and Michael Pradel. Freezing the web: A study of redos vulnerabilities in javascript-based web servers. In *Proceedings of the 27th USENIX Conference on Security Symposium*, 2018.
- [2] Martin Berglund, Frank Drewes, and Brink Van Der Merwe. Analyzing catastrophic backtracking behavior in practical regular expression matching. *Electronic Proceedings in Theoretical Computer Science*, Vol. 151, , 05 2014.