第 1 章

実験

本章では 4 章で提案した手法を実装し，実際に脆弱な正規表現が非脆弱な正規表現へと修正されていることを確認する．

1.1 実装における諸注意および実行環境

基本的には 4 章で提案した手法をプログラムにしただけである．しかし，RegEq では character set $[a_i a_{i+1} \ldots a_j]$ を直接扱うことができないため，これを union でつないだ形で表現している．具体的に言えば，$a_i|a_{i+1}|\ldots|a_j$ としている．any . に関しても同様である．

また，実験には計算機として「Xeon Gold6254(3.1GHz)x 2CPU (36 コア), 96GB RAM」を使用した．

1.2 結果

3 章で紹介した，図**??**の NFA に対応する正規表現 $c(ab)^*a(ba)^*$ を例に取って実験を行う．当然この正規表現は脆弱である．本研究で実装したプログラムに正規表現 $c(ab)^*a(ba)^*$，positive example $= \varnothing$，negative example $= \varnothing$ を入力すると，以下のような正規表現が出力として返ってきた (実行結果のログは付録を参照されたい)．

Listing 1.1 $c(ab)^*a(ba)^*$ を非脆弱化した正規表現

```
1 ca(?:(ba)*)
```

なお，non-capturing group ?: が含まれているが，本研究で扱う純粋な正規表現では意味

を持たないので削除してしまって構わない．従って，整形すると $ca(ba)^*$ という正規表現が得られる．

そして，$ca(ba)^*$ は $c(ab)^*a(ba)^*$ と意味的に等価で，かつ非脆弱 (量化子 $*$ が 1 つしかない，すなわち NFA に変換したときループを 1 つしか持たないため，3 章で紹介した定理 1 を満たさないことが直感的に分かる) であることが分かる．また，修正にかかった時間は 0.533 秒だった．以上のことを表にまとめると以下のようになる．

表 1.1　脆弱な正規表現 $c(ab)^*a(ba)^*$ の修正結果

修正前の正規表現	修正後の正規表現	修正にかかった時間 [s]
$c(ab)^*a(ba)^*$	$ca(ba)^*$	0.533

しかし，この例は REMEDY による初回の修正後の正規表現が偶然修正前の正規表現と意味的に等価であったため，1 ループ目で終了してしまった (REMEDY を実行，その後 RegEq を実行という工程を 1 ループとする)．数ループした果てに目的の正規表現が得られる例を見る．そこで $.^*a^*=$ という正規表現を考える．$.^*a^*=$ を入力として本研究で実装したプログラムを走らせると 10 ループ目にして以下のような結果が得られた (実行結果のログは付録を参照されたい)．

Listing 1.2　$.^*a^*=$ を非脆弱化した正規表現

```
1 (?:([^=])*)[=](?:((?:([^=])*)[=])*)
```

整形すると，$[\hat{}=]^*=([\hat{}=]^*=)^*$ のようになる．修正にかかった時間は 55.000 秒だった．以上のことを表にまとめると以下のようになる．

表 1.2　脆弱な正規表現 $[\hat{}=]^*=([\hat{}=]^*=)^*$ の修正結果

修正前の正規表現	修正後の正規表現	修正にかかった時間 [s]
$.^*a^*=$	$[\hat{}=]^*=([\hat{}=]^*=)^*$	55.000

参考文献