文字列抽出用正規表現の自動修正技術

はじめに

処理例を用いて文字列抽出用の正規表現を自動で修正することを可能とした世界初の技術について紹介します。

概要：

どのように修正したいのかという意図を処理例として与えることで、その意図を全て満たす形に文字列抽出用の正規表現を自動で修正する技術を考案

参考：

Nariyoshi Chida and Tachio Terauchi. **Repairing Regular Expressions for Extraction**. In Proceedings of the 44th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2023), PACMPL 7(PLDI), pp. 1633-1656. ACM, June, 2023.

論文へのリンク: https://dl.acm.org/doi/10.1145/3591287

正規表現 (Regular Expression, Regex)

正規表現は文字列のパターンを表す記法として現代のソフトウェア開発において**文字列チェック**と**文字列抽出**を行うために広く利用されている

正規表現:

キャプチャ

http:[/][/][^?]+[?](.*)$_1$

正規表現
エンジン

入力文字列:

http://example.com?key=value

文字列チェック:

受理 or 拒否

受理

文字列抽出:

部分文字列 or 拒否

key=value

1番のキャプチャにより抽出された部分文字列

正規表現は（便利だが）難しい！

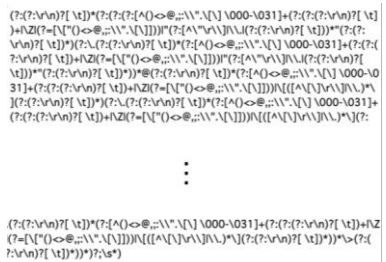
意図したパターンを表す正規表現を書くのは難しく、間違えやすい

文字列チェックの難しさ:

[Michael+ 2019; Wang+ 2020]

- 正規表現が複雑

82行



- 演算子の間違った利用

✕ [19|20][0-9]{2}
○ (?:19|20)[0-9]{2}

✕ [a-z]*.com
○ [a-z]*[.]com **etc...**

文字列抽出の難しさ:

正規表現エンジンの決定的な振る舞い

正規表現は世の中で広く利用されているため、間違いがあると
正規表現を利用しているプログラムのバグや脆弱性になりうる

自動修正技術

この状況を改善するため、意図した通りに文字列抽出用の
正規表現を自動で修正する自動修正技術を考案

修正対象となる正規表現

$a^*(aa|a)_1a^{*?}a$

受理される文字列

aaa⦇aa⦈$_1$a

⦇aa⦈$_1$a

拒否される文字列

a

誤りがないことが理論的に
保証された正規表現

自動修正技術

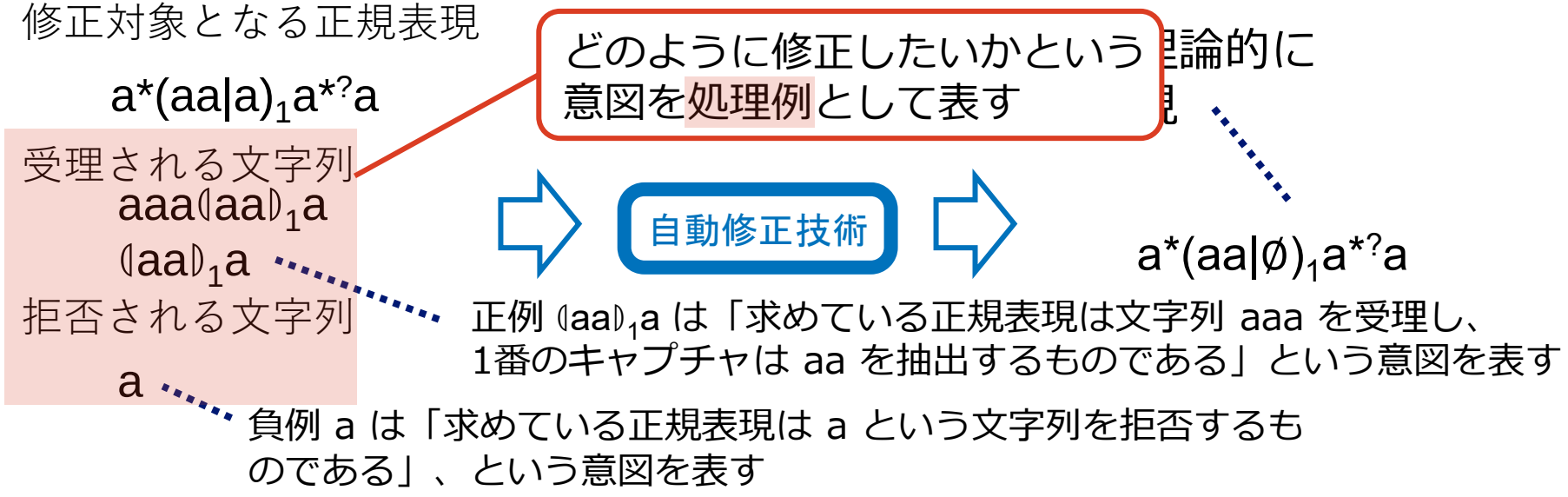
$a^*(aa|\emptyset)_1a^{*?}a$

正例 ⦇aa⦈$_1$a は「求めている正規表現は文字列 aaa を受理し、
1番のキャプチャは aa を抽出するものである」という意図を表す

負例 a は「求めている正規表現は a という文字列を拒否するも
のである」、という意図を表す

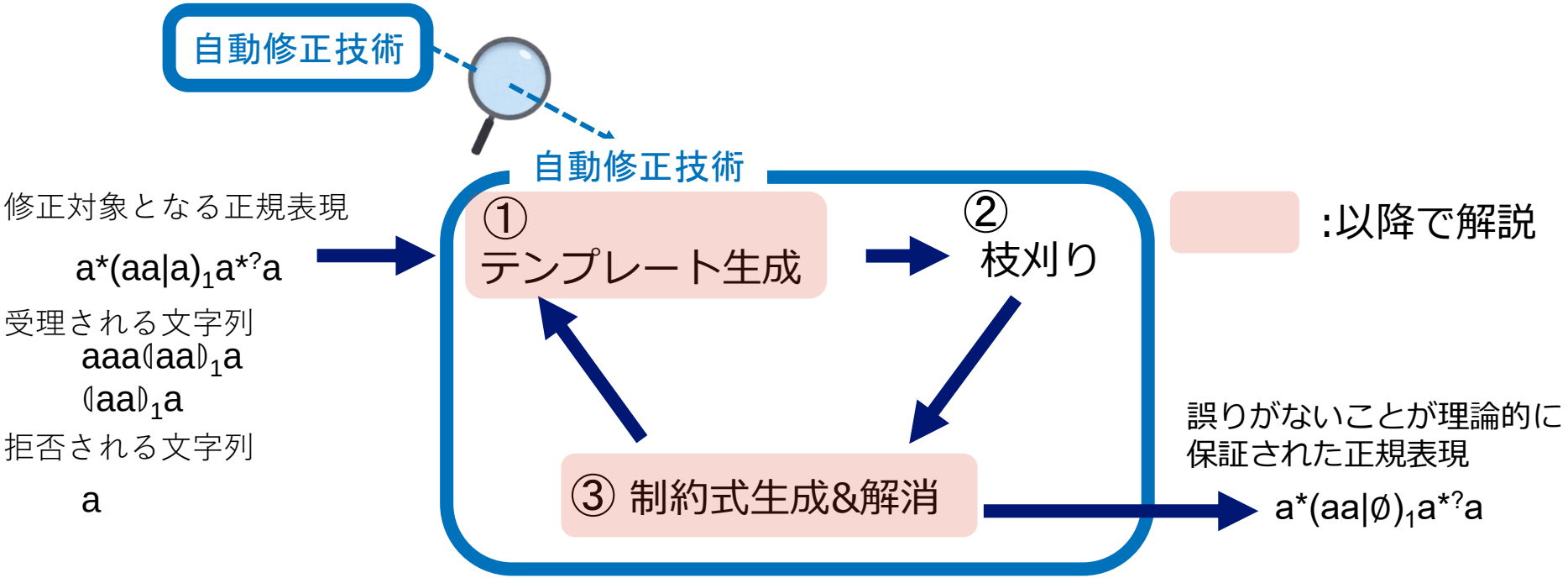
自動修正技術

この状況を改善するため、意図した通りに文字列抽出用の
正規表現を自動で修正する自動修正技術を考案



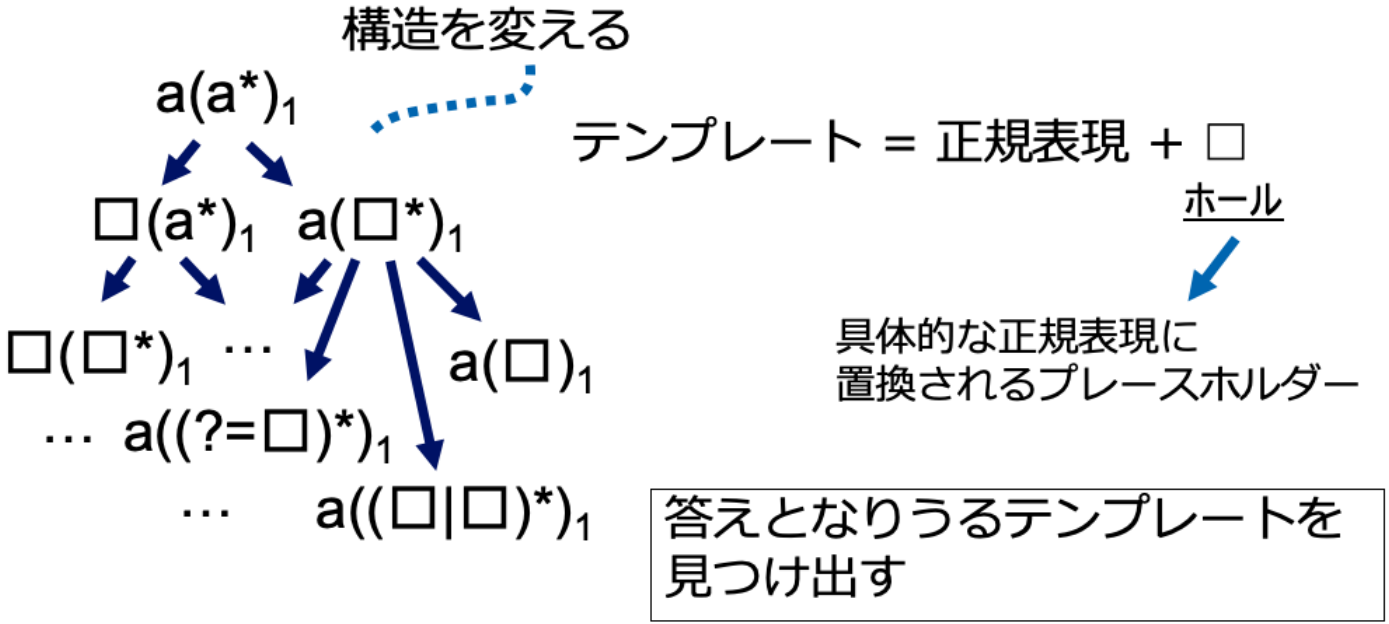
技術概要

自動修正技術の修正アルゴリズムは主に以下の３ステップからなる



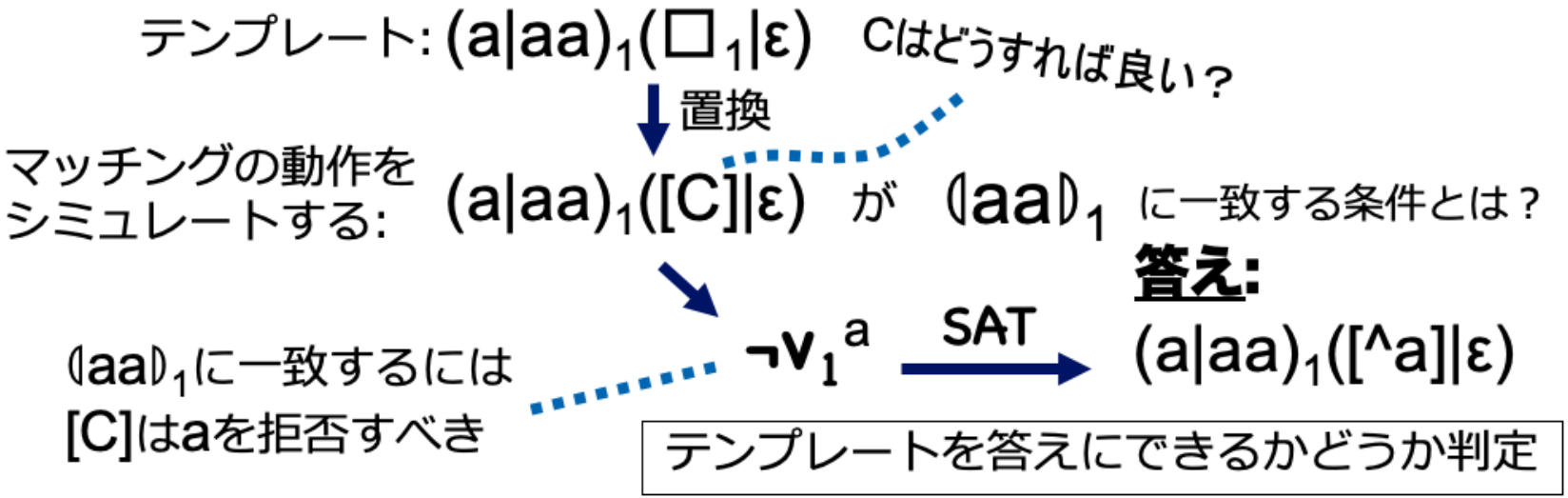
ステップ 1：テンプレート生成

正規表現の形を変え、全ての例に一致させることが可能なものを探す



ステップ3: 制約式生成&解消

テンプレートの中のホールを、全ての例に一致するように範囲文字に置き換えることが可能かどうか確認する



制約式生成に利用される正規表現エンジンの振る舞いの定義

Set of Characters

$$\frac{d = \text{forward} \quad p < |w| \quad w[p] \in C}{([C], r_c, w, p, \Gamma, d, l) \Downarrow (p+1, \Gamma)} \qquad \frac{d = \text{backward} \quad 0 \le p-1 \quad w[p-1] \in C}{([C], r_c, w, p, \Gamma, d, l) \Downarrow (p-1, \Gamma)}$$

Concatenation and Union

$$\frac{d = \text{forward} \quad (r_1, r_2 r_c, w, p, \Gamma, d, l) \Downarrow (p_1, \Gamma_1) \quad (r_2, r_c, w, p_1, \Gamma_1, d, l) \Downarrow (p_2, \Gamma_2)}{(r_1 r_2, r_c, w, p, \Gamma, d, l) \Downarrow (p_2, \Gamma_2)} \qquad \frac{d = \text{backward} \quad (r_2, r_c r_1, w, p, \Gamma, d, l) \Downarrow (p_1, \Gamma_1) \quad (r_1, r_c, w, p_1, \Gamma_1, d, l) \Downarrow (p_2, \Gamma_2)}{(r_1 r_2, r_c, w, p, \Gamma, d, l) \Downarrow (p_2, \Gamma_2)}$$

$$\frac{d = \text{forward} \quad (r_1 r_c, \epsilon, w, p, \Gamma, d, l) \Downarrow (p_1, \Gamma_1) \quad (\neg l \wedge p_1 = |w|) \vee l \quad (r_1, r_c, w, p, \Gamma, d, l) \Downarrow (p_2, \Gamma_2)}{(r_1 | r_2, r_c, w, p, \Gamma, d, l) \Downarrow (p_2, \Gamma_2)}$$

$$\frac{d = \text{forward} \quad (r_1 r_c, \epsilon, w, p, \Gamma, d, l) \Downarrow \tau \quad \tau = \text{failed} \vee (\neg l \wedge \tau = (p'', \Gamma'') \text{ where } p'' \neq |w|) \quad (r_2, r_c, w, p, \Gamma, d, l) \Downarrow (p', \Gamma')}{(r_1 | r_2, r_c, w, p, \Gamma, d, l) \Downarrow (p', \Gamma')}$$

Big-step operational semantics により定義

Greedy Kleene Star with Guards

$$\frac{d = \text{forward} \quad (r\langle r^* : p\rangle r_c, \epsilon, w, p, \mathit{reset}(r, \Gamma), d, l) \Downarrow (p_1, \Gamma_1) \quad p_1 = |w| \vee l \quad (r, \langle r^* : p\rangle r_c, w, p, \mathit{reset}(r, \Gamma), d, l) \Downarrow (p_2, \Gamma_2) \quad (r^*, r_c, w, p_2, \Gamma_2, d, l) \Downarrow (p_3, \Gamma_3)}{(r^*, r_c, w, p, \Gamma, d, l) \Downarrow (p_3, \Gamma_3)}$$

$$\frac{p' = p}{(\langle r : p'\rangle, r_c, w, p, \Gamma, d, l) \Downarrow \text{failed}} \qquad \frac{p' \neq p \quad (r, r_c, w, p, \Gamma, d, l) \Downarrow (p', \Gamma')}{(\langle r : p'\rangle, r_c, w, p, \Gamma, d, l) \Downarrow (p', \Gamma')}$$

Lazy Kleene Star with Guards

$$\frac{d = \text{forward} \quad (r_c, \epsilon, w, p, \Gamma, d, l) \Downarrow \tau \quad \tau = \text{failed} \vee (\neg l \wedge \tau = (p', \Gamma') \text{ where } p' \neq |w|) \quad (r\langle r^{*?} : p\rangle r_c, \epsilon, w, p, \mathit{reset}(r, \Gamma), d, l) \Downarrow (p_3, \Gamma_3) \quad p_3 = |w| \vee l \quad (r, \langle r^{*?} : p\rangle r_c, w, p, \mathit{reset}(r, \Gamma), d, l) \Downarrow (p_1, \Gamma_1) \quad (r^{*?}, r_c, w, p_1, \Gamma_1, d, l) \Downarrow (p_2, \Gamma_2)}{(r^{*?}, r_c, w, p, \Gamma, d, l) \Downarrow (p_2, \Gamma_2)}$$

Capturing Group, Backreference, and Positive Lookbehind

$$\frac{d = \text{forward} \quad (r\$_i, r_c, w, p, \Gamma[i \mapsto (p, \bot)], d, l) \Downarrow (p', \Gamma')}{((r)_i, r_c, w, p, \Gamma, d, l) \Downarrow (p', \Gamma')}$$

$$\frac{\Gamma(i) = (p', p'') \quad (w[p'..p''), r_c, w, p, \Gamma, d, l) \Downarrow (p''', \Gamma')}{(\backslash i, r_c, w, p, \Gamma, d, l) \Downarrow (p''', \Gamma')} \qquad \frac{i \notin \mathit{dom}(\Gamma) \vee \Gamma(i) = (p', \bot)}{(\backslash i, r_c, w, p, \Gamma, d, l) \Downarrow (p, \Gamma)}$$

$$\frac{d = \text{forward} \quad \Gamma(i) = (p, \bot)}{(\$_i, r_c, w, p', \Gamma, d, l) \Downarrow (p, \Gamma[i \mapsto (p, p')])} \qquad \frac{(r, \epsilon, w, p, \Gamma, \text{backward}, \text{true}) \Downarrow (p', \Gamma')}{((?<=r), r_c, w, p, \Gamma, d, l) \Downarrow (p, \Gamma')}$$

まとめ

- 処理例を用いて文字列抽出用の正規表現を自動で修正する世界初の技術を考案した

- 本技術を用いることで処理例として表された意図を必ず満たす形に正規表現を自動で修正できる