ON REGULAR EXPRESSIONS WITH BACKREFERENCES AND TRANSDUCERS

Frank Drewes

Martin Berglund Brink van der Merwe

UNIVERSITEIT
STELLENBOSCH
UNIVERSITY

Regular expressions with backreferences

REGULAR EXPRESSION

`(.+)\1`

Regular expressions with backreferences

REGULAR EXPRESSION

$(.+)\backslash 1$

Regular expressions with backreferences

REGULAR EXPRESSION

```
(.+)\1
(?'first'.+)\k'first'
```

Regular expressions with backreferences

REGULAR EXPRESSION

`(.+)\1`

`(?'first'.+)\k'first'`

TEST STRING

abab

Regular expressions with backreferences

REGULAR EXPRESSION	`(.+)\1` `(?'first'.+)\k'first'`
TEST STRING	`abab`
MATCH INFORMATION	`Full match  0-4  ` `` `abab` `` `Group 1.  0-2  ` `` `ab` ``

REGULAR EXPRESSION

```
[0-9]+\.\d*(\d+)\1+
```

REGULAR EXPRESSION

```
[0-9]+\.\d*(\d+)\1+
```

TEST STRING

```
0.818181
```

REGULAR EXPRESSION

```
[0-9]+\.\d*(\d+)\1+
```

TEST STRING

```
0.818181
```

MATCH INFORMATION

Full match	0-8	`0.818181`
Group 1.	4-6	`81`

REGULAR EXPRESSION `((?'one'.+)|(?'two'c+))\k'two'`

TEST STRING `abab`

MATCH INFORMATION Your regular expression does not match the subject string.

But how do we handle the following example:

In most matching engines the subexpression `(?i)` matches the empty string, but enables *case-insensitive* matching.

Thus `(?i)(.*)\1` matches any $\alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_n$ where, α_i and β_i are the same letter up to one (perhaps) being lowercase and the other uppercase.

But how do we handle the following example:

In most matching engines the subexpression `(?i)` matches the empty string, but enables *case-insensitive* matching.

Thus `(?i)(.*)\1` matches any $\alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_n$ where, α_i and β_i are the same letter up to one (perhaps) being lowercase and the other uppercase.

We permit *transducer subexpressions*, obtained by allowing the application of some string-to-string transducer to subexpressions.

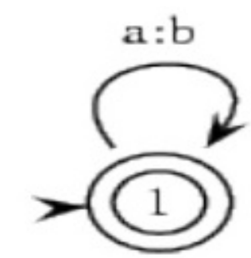
A transducer subexpression $t(E)$ describes the language of strings obtained by applying the transducer t to the language matched by E.

A transducer subexpression $t(E)$ describes the language of strings obtained by applying the transducer t to the language matched by E.

We call these extended expressions, obtained by adding backreferences and transducers, **regular expressions with backreferences and transducers** (REbt).

Example

A simple class of transducers over Σ^*, corresponding to transducers with only one state, is the set of all $t = (\alpha_1 : \beta_1, \ldots, \alpha_k : \beta_k)$ where $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \Sigma \cup \{\varepsilon\}$.

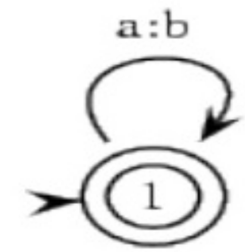


The transduction denoted by t is

$$\mathcal{L}(t) = \{(\alpha_{i_1} \cdots \alpha_{i_n}, \beta_{i_1} \cdots \beta_{i_n}) \mid n \in \mathbb{N}\}.$$

Example

A simple class of transducers over Σ^*, corresponding to transducers with only one state, is the set of all $t = (\alpha_1 : \beta_1, \dots, \alpha_k : \beta_k)$ where $\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \Sigma \cup \{\varepsilon\}$.



The transduction denoted by t is

$$\mathcal{L}(t) = \{(\alpha_{i_1} \cdots \alpha_{i_n}, \beta_{i_1} \cdots \beta_{i_n}) \mid n \in \mathbb{N}\}.$$

If $t_b = a : b$, $t_c = a : c$ and $t_d = b : d$ then

- $\mathcal{L}(\boxed{[_1 a^*]_1} t_b(\boxed{\uparrow_1}) t_c(\boxed{\uparrow_1})) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$
- $\mathcal{L}([_1 a^*]_1 [_2 b^*]_2 t_c(\uparrow_1) t_d(\uparrow_2)) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$

Example

A simple class of transducers over Σ^*, corresponding to transducers with only one state, is the set of all $t = (\alpha_1 : \beta_1, \ldots, \alpha_k : \beta_k)$ where $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \Sigma \cup \{\varepsilon\}$.

The transduction denoted by t is

$$\mathcal{L}(t) = \{(\alpha_{i_1} \cdots \alpha_{i_n}, \beta_{i_1} \cdots \beta_{i_n}) \mid n \in \mathbb{N}\}.$$

If $t_b = a : b$, $t_c = a : c$ and $t_d = b : d$ then

- $\mathcal{L}([_1 a^*]_1 t_b(\uparrow_1) t_c(\uparrow_1)) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$
- $\mathcal{L}(\boxed{[_1 a^*]_1}[_2 b^*]_2 t_c(\boxed{\uparrow_1}) t_d(\uparrow_2)) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$

Example

A simple class of transducers over Σ^*, corresponding to transducers with only one state, is the set of all $t = (\alpha_1 : \beta_1, \ldots, \alpha_k : \beta_k)$ where $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \Sigma \cup \{\varepsilon\}$.

The transduction denoted by t is

$$\mathcal{L}(t) = \{(\alpha_{i_1} \cdots \alpha_{i_n}, \beta_{i_1} \cdots \beta_{i_n}) \mid n \in \mathbb{N}\}.$$

If $t_b = a : b$, $t_c = a : c$ and $t_d = b : d$ then

- $\mathcal{L}([_1 a^*]_1 t_b(\uparrow_1) t_c(\uparrow_1)) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$

- $\mathcal{L}(\boxed{[_1 a^*]_1}\boxed{[_2 b^*]_2} t_c(\boxed{\uparrow_1}) t_d(\boxed{\uparrow_2})) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$

The Bad News

Theorem

For a recursively enumerable language L there exists an expression E with backreferences and transducers (i.e. $E \in \mathrm{REbt}$), such that $\mathcal{L}(E) = L$. Consequently the membership problem is undecidable for REbt.

The Bad News

Theorem

For a recursively enumerable language L there exists an expression E with backreferences and transducers (i.e. $E \in \text{REbt}$), such that $\mathcal{L}(E) = L$. Consequently the membership problem is undecidable for REbt.

Proof

For Turing machine M we construct the following transducers:

Theorem

For a recursively enumerable language L there exists an expression E with backreferences and transducers (i.e. $E \in \text{REbt}$), such that $\mathcal{L}(E) = L$. Consequently the membership problem is undecidable for REbt.

Proof

For Turing machine M we construct the following transducers:

- a transducer t_{init} such that $(w, c) \in \mathcal{L}(t_{\text{init}})$ if $c \in \Sigma^*$ is the initial configuration of M when starting with w as input,

The Bad News

Theorem

For a recursively enumerable language L there exists an expression E with backreferences and transducers (i.e. $E \in \text{REbt}$), such that $\mathcal{L}(E) = L$. Consequently the membership problem is undecidable for REbt.

Proof

For Turing machine M we construct the following transducers:

- a transducer t_{init} such that $(w, c) \in \mathcal{L}(t_{\text{init}})$ if $c \in \Sigma^*$ is the initial configuration of M when starting with w as input,
- a transducer t_{acc} such that $(c, c) \in \mathcal{L}(t_{\text{acc}})$ if c is the concatenation of configurations of M, with only the last configuration being accepting, and

Theorem

For a recursively enumerable language L there exists an expression E with backreferences and transducers (i.e. $E \in$ REbt), such that $\mathcal{L}(E) = L$. Consequently the membership problem is undecidable for REbt.

Proof

For Turing machine M we construct the following transducers:

- a transducer t_{init} such that $(w, c) \in \mathcal{L}(t_{\text{init}})$ if $c \in \Sigma^*$ is the initial configuration of M when starting with w as input,

- a transducer t_{acc} such that $(c, c) \in \mathcal{L}(t_{\text{acc}})$ if c is the concatenation of configurations of M, with only the last configuration being accepting, and

- a transducer t_{step} such that $(c, c') \in \mathcal{L}(t_{\text{step}})$ if M can go from the configuration c to the configuration c' in a single step.

Proof

For Turing machine M we construct the following transducers:

- a transducer t_{init} such that $(w, c) \in \mathcal{L}(t_{\text{init}})$ if $c \in \Sigma^*$ is the initial configuration of M when starting with w as input,

- a transducer t_{acc} such that $(c, c) \in \mathcal{L}(t_{\text{acc}})$ if c is the concatenation of configurations of M, with only the last configuration being accepting, and

- a transducer t_{step} such that $(c, c') \in \mathcal{L}(t_{\text{step}})$ if M can go from the configuration c to the configuration c' in a single step.

$E = \boxed{[_\phi \Gamma^*]_\phi} D([_\phi t_{\text{init}}(\uparrow_\phi)]_\phi t_{\text{acc}}([_\phi t_{\text{step}}(\uparrow_\phi)]_\phi{}^*))$, where D is a transducer that deletes the entire input, and outputs ε.

The first subexpression selects and captures any input string w.
The subexpression $D(\cdots)$ simulates a computation of M on w to either fail or, if M accepts, yield ε.

$E = [_\phi \Gamma^*]_\phi \boxed{D([_\phi t_{\mathrm{init}}(\uparrow_\phi)]_\phi t_{\mathrm{acc}}([_\phi t_{\mathrm{step}}(\uparrow_\phi)]_\phi{}^*)})$, where D is a transducer that deletes the entire input, and outputs ε.

The first subexpression selects and captures any input string w.
The subexpression $D(\cdots)$ simulates a computation of M on w to either fail or, if M accepts, yield ε.

$E = [_\phi \Gamma^*]_\phi D(\boxed{[_\phi t_{\mathrm{init}}(\uparrow_\phi)]_\phi} t_{\mathrm{acc}}([_\phi t_{\mathrm{step}}(\uparrow_\phi)]_\phi{}^*))$, where D is a transducer that deletes the entire input, and outputs ε.

$E = [_\phi \Gamma^*]_\phi D([_\phi t_{\mathrm{init}}(\uparrow_\phi)]_\phi \boxed{t_{\mathrm{acc}}([_\phi t_{\mathrm{step}}(\uparrow_\phi)]_\phi{}^*)})$, where D is a transducer that deletes the entire input, and outputs ε.

Consider various restrictions

- Allow backreferences but not transducers.

Consider various restrictions

- Allow backreferences but not transducers.

- Allow only a single top-level transducer.

Consider various restrictions

- Allow backreferences but not transducers.

- Allow only a single top-level transducer.

- Allow transducers but not backreferences.

Consider various restrictions

- Allow backreferences but not transducers.
- Allow only a single top-level transducer.
- Allow transducers but not backreferences.
- Do not allow the capture of transducer preimages.

Consider various restrictions

- Allow backreferences but not transducers.
- Allow only a single top-level transducer.
- Allow transducers but not backreferences.
- Do not allow the capture of transducer preimages.
- Allow only non-deleting transducers.

Consider various restrictions

- Allow backreferences but not transducers.

- Allow only a single top-level transducer.

- Allow transducers but not backreferences.

- Do not allow the capture of transducer preimages.

- Allow only non-deleting transducers.

- Allow only functional transducers.

Consider various restrictions

- Allow backreferences but not transducers.
- Allow only a single top-level transducer.
- Allow transducers but not backreferences.
- Do not allow the capture of transducer preimages.
- Allow only non-deleting transducers.
- Allow only functional transducers.

Consider some examples w.r.t. these restrictions

```
(?i)(.*)\1
```

Consider various restrictions

- Allow backreferences but not transducers.
- Allow only a single top-level transducer.
- Allow transducers but not backreferences.
- Do not allow the capture of transducer preimages.
- Allow only non-deleting transducers.
- Allow only functional transducers.

Consider some examples w.r.t. these restrictions

```
(?i)(.*)\1
```

If $t_b = a:b$, $t_c = a:c$ and $t_d = b:d$ then

- $\mathcal{L}([_1 a^*]_1 t_b(\uparrow_1) t_c(\uparrow_1)) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$
- $\mathcal{L}([_1 a^*]_1 [_2 b^*]_2 t_c(\uparrow_1) t_d(\uparrow_2)) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$

Consider various restrictions

- Allow backreferences but not transducers.
- Allow only a single top-level transducer.
- Allow transducers but not backreferences.
- Do not allow the capture of transducer preimages.
- Allow only non-deleting transducers.
- Allow only functional transducers.

Consider some examples w.r.t. these restrictions

```
(?i)(.*)\1
```

If $t_b = a:b$, $t_c = a:c$ and $t_d = b:d$ then

- $\mathcal{L}([_1 a^*]_1 t_b(\uparrow_1) t_c(\uparrow_1)) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$
- $\mathcal{L}([_1 a^*]_1 [_2 b^*]_2 t_c(\uparrow_1) t_d(\uparrow_2)) = \{a^m b^n c^m d^n \mid m, n \in \mathbb{N}\}$

$E = [_\phi \Gamma^*]_\phi D([_\phi t_{\text{init}}(\uparrow_\phi)]_\phi t_{\text{acc}}([_\phi t_{\text{step}}(\uparrow_\phi)]_\phi{}^*))$, where $D \in \text{FST}$ deletes the entire input and outputs ε.

Theorem

For expressions E without transducers we may decide whether $w \in \mathcal{L}(E)$ in time polynomial in $|w|$ and $|E|$, with the degree of the polynomial a constant times the number of backreference symbols in E.

Theorem

For expressions E without transducers we may decide whether $w \in \mathcal{L}(E)$ in time polynomial in $|w|$ and $|E|$, with the degree of the polynomial a constant times the number of backreference symbols in E.

Theorem

For expressions E with transducers but without backreferences, it is PSPACE-complete to decide whether $\varepsilon \in \mathcal{L}(E)$. In general, uniform membership testing for this class of expressions is PSPACE-complete.

Theorem

For expressions E without transducers we may decide whether $w \in \mathcal{L}(E)$ in time polynomial in $|w|$ and $|E|$, with the degree of the polynomial a constant times the number of backreference symbols in E.

Theorem

For expressions E with transducers but without backreferences, it is PSPACE-complete to decide whether $\varepsilon \in \mathcal{L}(E)$. In general, uniform membership testing for this class of expressions is PSPACE-complete.

Proof

Automata $\{A_1, \ldots, A_n\}$ – Question: $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n) = \emptyset$.

Theorem

For expressions E without transducers we may decide whether $w \in \mathcal{L}(E)$ in time polynomial in $|w|$ and $|E|$, with the degree of the polynomial a constant times the number of backreference symbols in E.

Theorem

For expressions E with transducers but without backreferences, it is PSPACE-complete to decide whether $\varepsilon \in \mathcal{L}(E)$. In general, uniform membership testing for this class of expressions is PSPACE-complete.

Proof

Automata $\{A_1, \ldots, A_n\}$ – Question: $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n) = \emptyset$.

Construct trunsducers t_i with $\mathcal{L}(t_i) = \{(w, w) \mid w \in \mathcal{L}(A_i)\}$.

Theorem

For expressions E without transducers we may decide whether $w \in \mathcal{L}(E)$ in time polynomial in $|w|$ and $|E|$, with the degree of the polynomial a constant times the number of backreference symbols in E.

Theorem

For expressions E with transducers but without backreferences, it is PSPACE-complete to decide whether $\varepsilon \in \mathcal{L}(E)$. In general, uniform membership testing for this class of expressions is PSPACE-complete.

Proof

Automata $\{A_1, \ldots, A_n\}$ – Question: $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n) = \emptyset$.

Construct trunsducers t_i with $\mathcal{L}(t_i) = \{(w, w) \mid w \in \mathcal{L}(A_i)\}$.

Let $E = D(t_1(\cdots t_n(\Sigma^*) \cdots))$ where D takes every input to ε.

Theorem

For expressions E without transducers we may decide whether $w \in \mathcal{L}(E)$ in time polynomial in $|w|$ and $|E|$, with the degree of the polynomial a constant times the number of backreference symbols in E.

Theorem

For expressions E with transducers but without backreferences, it is PSPACE-complete to decide whether $\varepsilon \in \mathcal{L}(E)$. In general, uniform membership testing for this class of expressions is PSPACE-complete.

Proof

Automata $\{A_1, \ldots, A_n\}$ – Question: $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n) = \emptyset$.

Construct trunsducers t_i with $\mathcal{L}(t_i) = \{(w, w) \mid w \in \mathcal{L}(A_i)\}$.

Let $E = D(t_1(\cdots t_n(\Sigma^*) \cdots))$ where D takes every input to ε.

The intersection is non-empty if and only if $\varepsilon \in \mathcal{L}(E)$.

Theorem

Uniform membership for the class of expressions E with backreferences and non-deleting transducers is EXPSPACE-complete.

Theorem

Uniform membership for the class of expressions E with backreferences and non-deleting transducers is EXPSPACE-complete.

Theorem

The uniform membership problem for the class of expressions with only a top-level transducer is PSPACE-complete.

Theorem

Uniform membership for the class of expressions E with backreferences and non-deleting transducers is EXPSPACE-complete.

Theorem

The uniform membership problem for the class of expressions with only a top-level transducer is PSPACE-complete.

For the subclass of expressions where the top-level transducer is also non-deleting, the uniform and non-uniform membership problems are NP-complete.

Summary

We have:

(i) proposed an extension of regular expressions with backreferences, by adding transducers;

Summary

We have:

(i) proposed an extension of regular expressions with backreferences, by adding transducers;

(ii) established that this makes membership testing undecidable; and

Summary

We have:

(i) proposed an extension of regular expressions with backreferences, by adding transducers;

(ii) established that this makes membership testing undecidable; and

(iii) explored various restrictions to form a practical basis for use in software.

Future work

The precise expressiveness of the classes should be considered – several gaps exist beyond what follows naturally from what we have done here.

Future work

The precise expressiveness of the classes should be considered –
several gaps exist beyond what follows naturally from what we have done here.

The subclasses should also be compared with respect to succinctness,
and there remain some open questions regarding computational complexity
of for example (uniform) membership in certain cases.

[2b | !2b]

That is the expression.