ON REGULAR EXPRESSIONS WITH BACKREFERENCES AND TRANSDUCERS

Martin Berglund$^{(A)}$ Frank Drewes$^{(B)}$
Brink van der Merwe$^{(C)}$

$^{(A)}$Department of Information Science, Center for AI Research (CSIR), Stellenbosch University, South Africa

$^{(B)}$Department of Computing Science, Umeå University, Sweden

$^{(C)}$Department of Computer Science, Stellenbosch University, South Africa

Abstract
Modern regular expression matching software features many extensions, some general while some are very narrowly specified. Here we consider the generalization of adding a class of operators which can be described by, e.g. finite-state transducers. Combined with backreferences they enable new classes of languages to be matched. The addition of finite-state transducers is shown to make membership testing undecidable. Following this result, we study the complexity of membership testing for various restricted cases of the model.

1. Introduction

In this paper we consider generalizations of various common feature additions in practical regular expression matching software. Notably we include expression with backreferences (which we abbreviate REb here), an extension which allows the regular expression to “capture” literal substrings as part of its matching procedure, and then “backreference” a previously captured string to match an exact copy of it in a different position of the string. Furthermore, in most matching engines (Java, Perl, etc.) the subexpression `(?i)` matches the empty string, but enables *case-insensitive* matching for a subexpression, meaning that `(?i)(.*)\1` matches any $\alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_n$ where, for each i, α_i and β_i are the same letter up to one (perhaps) being lowercase and the other uppercase. Several similar features exist (such as collating different representations of Unicode symbols), which can all be naturally expressed as a transduction of the matched string. To generalize this we here 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. We call these extended expressions, obtained by adding backreferences and transducers, regular expressions with backreferences and transducers (REbt). For the most part, the transducers considered will be finite-state transducers or restrictions thereof.

Beyond the transducer-like features of existing engines the REbt (and the various restricted subclasses we consider) can also describe some frequently encountered non-context-free languages. According to Dassow et al. [5] the three most commonly encountered non-context-free features in formal languages are reduplication, i.e. the ability to express languages of the form $L_{\mathrm{RD}} = \{ww \mid w \in \Sigma^*\}$, multiple agreements, described by languages of the form $L_{\mathrm{MA}} = \{a^n b^n c^n \mid n \geq 1\}$, and cross agreements, as given by languages of the form $L_{\mathrm{CA}} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$. The language L_{RD} can be described by REb, but neither L_{MA} nor L_{CA} can, which for a restricted class of REb follows from the pumping lemma in [4], and for REb in general, from the argument in [2] for why the language $\{a^n b^n \mid n \geq 0\}$ cannot be described by REb. The language of the example Java expression `(?i)(.*)\1` cannot be matched by REb either, as evidenced by the sublanguage $\{a^n b A^n B \mid n \geq 0\}$, using that REb is closed under intersection with regular languages as shown in Theorem 21 in [9]. The REbt matching these languages are quite simple, but the full formalism turns out to be very powerful. This establishes the goal of the paper, i.e. finding natural restrictions of REbt which can still match L_{MA} and L_{CA}, can be tested for membership with a computational complexity not too distant from REb, and may be considered "natural".

Reviewers: [2] is not yet published, see Lemma 7.1 in the Appendix for reproduction of the small argument.

After definitions given in Section 2, and the unrestricted case being shown to have undecidable membership in Section 3, the remaining sections explore various restrictions: Section 4 forbids the capture of transducer preimages and considers permitting only non-deleting transducers. Section 5 forbids transducers in capturing cycles (where a capturing cycle captures a submatch and then later backreferences this capture as part of another submatch by the same capturing subexpression), and requires the transducers occurring in captures to be functional. Finally, Section 6 considers permitting only a single top-level transducer.

2. Definitions

Denote by $\mathbb{N}$ the set of natural numbers, excluding 0, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, and by $[k]$, with $k \in \mathbb{N}$, the set $\{j \mid 1 \leq j \leq k\}$. An alphabet is a finite set of symbols. For sets S and T we write $S \uplus T$ to denote the union of these sets, assumed to be disjoint. Let ε denote the empty string and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, where Σ is an alphabet, and for a string $w \in \Sigma^*$, let $\mathrm{substr}(w)$ be the set of all substrings of w, i.e. $\mathrm{substr}(\varepsilon) = \{\varepsilon\}$, and if $w_i \in \Sigma$ for $1 \leq i \leq n$, then $\mathrm{substr}(w_1 \ldots w_n) = \{\varepsilon\} \cup \{w_i \ldots w_j \mid 1 \leq i \leq j \leq n\}$, in particular, $\varepsilon, w \in \mathrm{substr}(w)$. Given a notion of expressions, defined inductively, we denote by $\mathrm{subexps}(E)$ the set of all subexpressions of the expression E, that is, $\mathrm{subexps}(E)$ is the set of expressions used to obtain E inductively, including duplicate expressions when the same subexpression appears at different places in E. For a partial function $f\colon A \to B$, let $\mathrm{dom}(f)$ denote its domain, $\mathrm{range}(f)$ its range, and $g = f[x \mapsto y]$ denote the partial function such that $g(x) = y$ but $g(z) = f(z)$ for all $z \neq x$. A partial function f with $\mathrm{dom}(f) = \emptyset$ is denoted by $\perp$. Let $f[x \mapsto \perp]$ denote the function resulting when removing x from the domain of f. For a set A, we denote by $|A|$ the cardinality of A.

To keep our notion of regular expressions with backreferences and transducers general, we define it relative to a set Θ of string transducers. When we say that Θ is a class of transducers we mean that every element t of Θ denotes a transduction $\mathcal{L}(t)$ on Σ^* for some alphabet Σ, i.e. $\mathcal{L}(t)$ is a binary relation $\mathcal{L}(t) \subseteq \Sigma^* \times \Sigma^*$. As a transducer $t \in \Theta$ is a denotation of a transduction, it has a length, namely its length when written down as a string. We denote this length by $|t|$. Clearly, classes of transducers that specify the same transductions may differ regarding, e.g. their succinctness, and thus also with respect to their computational complexity.

We say that a transducer t is:

- *non-deleting* if there is a constant c such that $|u| \leq c|v| + c$ for all $(u, v) \in \mathcal{L}(t)$, we call the smallest such c the *non-deletion constant* of t,
- *non-generating* if the transducer defined by the inverse relation $\mathcal{L}(t)^{-1}$ is non-deleting, and
- *functional* if $\mathcal{L}(t)$ is a partial function, i.e. $|\{v \in \Sigma^* \mid (u, v) \in \mathcal{L}(t)\}| \leq 1$ for all $u \in \Sigma^*$.

Definition 2.1 *Let Θ be a class of transducers. For input and backreference alphabets Σ and Φ, $\alpha \in \Sigma_\varepsilon$, $\phi \in \Phi$, and $t \in \Theta$ a transducer on Σ, the set of* regular expressions with backreferences and transducers (over Θ)*, $REbt_{\Sigma,\Phi}$, is obtained inductively from the following subexpressions: (1) $\emptyset$; (2) α; (3) $(F \,|\, G)$; (4) $(F \cdot G)$; (5) (F^*); (6) $(\uparrow_\phi)$; (7) $([_\phi F]_\phi)$; and; (8) $t(F)$; where $F, G \in REbt_{\Sigma,\Phi}$.*

We call the REbt that can be constructed using rules 1–7 regular expressions with backreferences (REb or $\mathrm{REb}_{\Sigma,\Phi}$)*, using 1–5 and 8,* regular expressions with transducers (REt or REt_Σ)*, and, using 1–5,* regular expressions (RE or RE_Σ).

For $E \in$ REbt, we denote by $|E|$ the length of E as a string, but letting each transducer symbol t in $|E|$ contribute length $|t|$ (i.e. not just 1), and by $\mathrm{env}_{\Phi,\Sigma}$ (or simply env, when Φ and Σ is understood) the set of all partial functions from Φ to Σ^*. We refer to these partial functions as *environments*, since they keep track of which substring, in Σ^*, from the input string, is bound to a given backreference symbol $\phi \in \Phi$. The empty environment, i.e. the partial function in env with empty domain, is denoted by $\perp$.

Definition 2.2 *For $E \in REbt_{\Sigma,\Phi}$, we define the* matching relation $\mathcal{M}(E) \subseteq env_{\Sigma,\Phi} \times \Sigma^* \times env_{\Sigma,\Phi}$ *inductively on the structure of E, as follows.*

1. *$\emptyset$ if $E = \emptyset$;*
2. *$\{(f, \alpha, f) \mid f \in env_{\Sigma,\Phi}\}$ if $E = \alpha$ with $\alpha \in \Sigma_\varepsilon$;*
3. *$\mathcal{M}(F) \cup \mathcal{M}(G)$ if $E = (F \,|\, G)$;*
4. *$\{(f, vw, g) \mid (f, v, f') \in \mathcal{M}(F), (f', w, g) \in \mathcal{M}(G)\}$ if $E = (F \cdot G)$;*
5. *$\mathcal{M}(\varepsilon) \cup \{(f, vw, g) \mid (f, v, f') \in \mathcal{M}(F^*), (f', w, g) \in \mathcal{M}(F)\}$, or the least fixed point of $\mathcal{M}(E) = \mathcal{M}(\varepsilon) \cup M(E \cdot F)$, if $E = (F^*)$;*
6. *$\{(f, w, f'[\phi \mapsto w]) \mid (f, w, f') \in \mathcal{M}(F)\}$ if $E = ([_\phi F]_\phi)$ with $\phi \in \Phi$;*
7. *$\{(f, w, f) \mid f \in env_{\Sigma,\Phi}, f(\phi) = w\}$ if $E = (\uparrow_\phi)$;*
8. *$\{(f, w, f') \mid (f, v, f') \in \mathcal{M}(F), (v, w) \in \mathcal{L}(t)\}$ if $E = (t(F))$ for some $t \in \Theta$.*

The language matched *by a REbt E, denoted $\mathcal{L}(E)$, is defined as the set $\mathcal{L}(E) = \{w \mid (\perp, w, f) \in \mathcal{M}(E), f \in env_{\Sigma,\Phi}\}$.*

Let $\mathrm{td}(E)$ denote the set of all transducers occurring in E. For technical convenience, we assume that every transducer in $\mathrm{td}(E)$ is referred to only once in E. Hence, two distinct subexpressions $t(F)$ and $t'(F')$ have $t \neq t'$ even though we may of course have $\mathcal{L}(t) = \mathcal{L}(t')$.

As usual, when writing an expression as a string some parentheses may be elided using the rule that Kleene closure '*' takes precedence over concatenation '$\cdot$', which takes precedence over union '$|$'. In addition, outermost parenthesis and parenthesis in subexpressions of the form $([_\phi E]_\phi)$ and $(\uparrow_\phi)$, may be dropped, and $E_1 \cdot E_2$ abbreviated as E_1E_2. Naturally, the brackets which denote a capturing group may not be elided.

Example 2.3 *A simple class of transducers over Σ^*, corresponding to finite-state transducers with only one state, is the set of all $t = (\alpha_1 : \beta_1, \ldots, \alpha_k : \beta_k)$ where $k \in \mathbb{N}$ and $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \Sigma_\varepsilon$. The transduction denoted by t is*

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

Taking $E_{MA} = [_1a^*]_1 t_b(\uparrow_1) t_c(\uparrow_1)$ and $E_{CA} = [_1a^*]_1[_2b^*]_2 t_c(\uparrow_1) t_d(\uparrow_2)$ from $\mathrm{REbt}_{\{a,b,c,d\},\{1,2\}}$ with $t_b = a : b$, $t_c = a : c$ and $t_d = b : d$ yields $\mathcal{L}(E_{MA}) = L_{MA}$ and $\mathcal{L}(E_{CA}) = L_{CA}$, with L_{MA} and L_{CA} as given in the introduction.

We often let Σ and Φ indicate arbitrary input and backreference alphabets respectively, and may then also drop them, writing REbt instead of $\mathrm{REbt}_{\Sigma,\Phi}$.

The subset REb of REbt is equivalent to the semantics originally given by Aho in [1], which agrees fully with the behavior or many popular software implementations (e.g. Boost, the .NET standard library implementation, the PCRE library [3]), and form a superset of many more (e.g. the Java and Python implementations). The semantics considered by Schmid in [9] is also closely related, with one difference being that subexpressions of the form $[_\phi \cdots \uparrow_\phi \cdots]_\phi$ are not permitted by Schmid (but are here, in Aho, and in most implementations). Schmid also differs from Aho, while agreeing with other important theoretical work [4], in having $\uparrow_\phi$ match the empty string if ϕ has not yet been captured (i.e. they let $\mathcal{L}(E) = \{w \mid (\bot_\varepsilon, w, f) \in \mathcal{M}(E)\}$ where $\bot_\varepsilon(\phi) = \varepsilon$ for all $\phi \in \Phi$). We again adopt the Aho approach to align with the software practice, also noting that Aho semantics can simulate the use of $\bot_\varepsilon$, by first "initializing" all symbols from Φ to ε (using a leading sequence of subexpressions $[_\phi\varepsilon]_\phi$ for all $\phi \in \Phi$).

3. Unrestricted Language Classes

The use of transducers without severe restrictions unsurprisingly gives rise to a Turing complete formalism. To make this precise, let FST denote the class of all one-way finite-state string transducers. More precisely, FST is the set of all $t = (Q, \Sigma, q_0, \delta, F)$ where (1) Q is a *finite set of states*, (2) Σ is the *input and output alphabet*, (3) $q_0 \in Q$ is the *initial state*, (4) $\delta \subseteq Q \times \Sigma_\varepsilon \times \Sigma_\varepsilon \times Q$ is the *transition relation*, and (5) $F \subseteq Q$ is the set of *final states*.

A *computation* of such an FST t is a sequence $(q_1, \alpha_1, \beta_1, q_1'), \ldots, (q_n, \alpha_n, \beta_n, q_n')$ of zero or more transitions, having $q_i' = q_{i+1}$ for all $i \in [n-1]$. The transduction $\mathcal{L}(t) \subseteq \Sigma^* \times \Sigma^*$ consists of all (v, w) such that there exists a computation $(q_1, \alpha_1, \beta_1, q_1'), \dots, (q_n, \alpha_n, \beta_n, q_n')$ with $q_0 = q_1$, $q_n' \in F$, $v = \alpha_1 \cdots \alpha_n$ and $w = \beta_1 \cdots \beta_n$.

Theorem 3.1 *For every recursively enumerable language L there exists an $E \in$ REbt over* FST *such that $\mathcal{L}(E) = L$. Consequently the membership problem is undecidable for REbt over* FST.

Proof. For a Turing machine M with input alphabet Γ, choose a representation of the configurations of M as strings $w \in \Sigma^*$, where $\Sigma \supseteq \Gamma$, such that we can construct

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

This is easy for any reasonable string representation of configurations: t_{init} adds a tape head and state at the front, t_{acc} checks for an accepting state, and t_{step} performs one of a finite number of constant substring rewritings around the tape head, implementing the rules of M.

Then, take $\Phi = \{\phi\}$ and define $E_u \in \text{REbt}_{\Sigma,\Phi}$ to be

$$[_\phi \Gamma^*]_\phi D([_\phi t_{\text{init}}(\uparrow_\phi)]_\phi t_{\text{acc}}([_\phi t_{\text{step}}(\uparrow_\phi)]_\phi{}^*)),$$

where $D \in$ FST deletes the entire input (and outputs ε). Thus, 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. □

Corollary 3.2 *Theorem 3.1 holds even if E is required to be a REbt over functional non-generating* FST*s.*

Proof. The FSTs used in the proof of Theorem 3.1 are already non-generating. Further, we can without loss of generality pick M to be a deterministic Turing machine, at which point the natural way of constructing the transducers will make them functional. □

The rest of the paper studies restrictions of REbt which we consider to be natural, and which make matching more tractable while including REb and retaining the ability to match e.g. L_{MA} and L_{CA}. Tractability of restrictions must be judged relative to the known NP-completeness of the uniform membership problem for REb [1], forming a lower bound. The non-uniform membership problem for REb can be decided in PTIME, and if $|\Phi|$ is bounded, the same holds true for the uniform membership problem.

Lemma 3.3 *For $E \in \text{REb}_{\Sigma,\Phi}$ and $w \in \Sigma^*$ we may decide whether $w \in \mathcal{L}(E)$ by using Algorithm 1. This algorithm runs in time polynomial in $|w|$ and $|E|$, with a polynomial of degree $O(|\Phi|)$.*

Proof. The steps of the algorithm correspond directly to the semantics given in Definition 2.2.

Algorithm 1 Membership decision procedure for $\mathrm{REb}_{\Sigma,\Phi}$

```
procedure MEMBERSHIP($w \in \Sigma^*$,$E \in \mathrm{REb}_{\Sigma,\Phi}$)
    ▷ Returns true if $w \in \mathcal{L}(E)$
    Let $\mathrm{env}(w) = \{f \in \mathrm{env}_{\Sigma,\Phi} \mid \mathrm{range}(f) \subseteq \mathrm{substr}(w)\}$
    Let $T : \mathrm{env}(w) \times \mathrm{substr}(w) \times \mathrm{subexps}(E) \times \mathrm{env}(w) \to \{\mathrm{true}, \mathrm{false}\}$
    $T(f, v, E, f') \leftarrow$ false for all $(f, v, E, f') \in \mathrm{dom}(T)$
    $T(f, \alpha, \alpha, f) \leftarrow$ true for all $\alpha \in \Sigma_\varepsilon$ and $f \in \mathrm{env}(w)$
    $T(f, \varepsilon, F^*, f) \leftarrow$ true for all $F^*$ and $f$
    repeat
        if $T(f, v_1, F^*, g) \wedge T(g, v_2, F, f') =$ true for some $v_1v_2$ and $g$ then
            $T(f, v_1v_2, F^*, f') \leftarrow$ true
        if $T(f, v_1, F, g) \wedge T(g, v_2, G, f') =$ true for some $v_1v_2$ and $g$ then
            $T(f, v_1v_2, F \cdot G, f') \leftarrow$ true
        if $T(f, v, F, f') \vee T(f, v, G, f') =$ true then
            $T(f, v, F \mid G, f') \leftarrow$ true
        if $T(f, v, F, g) =$ true then
            $T(f, v, [_\phi F]_\phi, g[\phi \mapsto v]) \leftarrow$ true
        if $f(\phi) = v$ then
            $T(f, v, \uparrow_\phi, f) \leftarrow$ true
    until no additional function values of $T$ were set to true
    return true if $T(\bot, w, E, f)$ equals true for some $f$
```

The claimed bound $(|w|+|E|)^{O(|\Phi|)}$ on the running time can be verified by noting that $\mathrm{dom}(T)$ is of size $|\mathrm{env}(w)|^2|E|\binom{|w|+1}{2}$, and $|\mathrm{env}(w)| \leq (1 + \binom{|w|+1}{2})^{|\Phi|}$, since a function in $\mathrm{env}(w)$ maps each $\phi \in \Phi$ to one of the at most $\binom{|w|+1}{2}$ substrings, or leaves it undefined. With $|\Phi|$ bounded, this makes $|\mathrm{dom}(T)|$ polynomial in $|w|$ and $|E|$, and the algorithm can be performed in a polynomial number of steps (scanning $\mathrm{dom}(T)$ for a way to use one of the rules to set another cell to true, halting if a full scan results in no new true cells). □

Further, only regular languages are matched by REt over FST (as the class of regular languages is closed under FST), but the expressions are succinct.

Lemma 3.4 *For $E \in$ REt over* FST *it is PSPACE-complete to decide whether $\varepsilon \in \mathcal{L}(E)$, and in general, uniform membership testing for expressions in REt is PSPACE-complete.*

Proof. PSPACE-hardness can be seen by a reduction from the (complement of the) PSPACE-complete problem FINITE AUTOMATON INTERSECTION EMPTINESS [6], where the instances are sets of finite automata $\{A_1, \ldots, A_n\}$ and the question is whether $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n) = \emptyset$. For each A_i, $i \in [n]$, construct the FST t_i with $\mathcal{L}(t_i) = \{(w, w) \mid w \in \mathcal{L}(A)\}$, and let $E = D(t_1(\cdots t_n(\Sigma^*)\cdots))$ where D is the FST that takes every input to ε. Then the intersection is non-empty if and only if $\varepsilon \in \mathcal{L}(E)$.

The problem can be solved in a straightforward way by constructing a product automaton which simulates all active transducers at once. Explicitly constructing such an automaton requires exponential space (as its states would be the Cartesian product of the states of all transducers and automata for the subexpressions). However, one can incrementally construct this product automaton, remembering only a single (product) state and its outgoing transitions at every step, while doing a nondeterministic search (recall that deterministic and nondeterministic PSPACE are equal) for an accepting path, to solve the problem in PSPACE. □

4. Limiting Capturing and Deletions

In limiting deletions and nesting of transducers, we find further use for Algorithm 1 after extending it to handle transducers. A key observation to achieve this, is to note that for any $(f, v, F, f') \in \text{dom}(T)$ in Algorithm 1, where F contains no captures (and thus $f = f'$), the evaluation of $T(f, v, F, f)$ can be done without knowing the values of $T(f, v, F', f)$, for proper subexpressions F' of F, by constructing an NFA that is language equivalent to F (after replacing backreferences ϕ in F by $f(\phi)$).

In order to do this efficiently, the nesting of transducer applications must be bounded. We first formalize the notion of nesting depth.

Definition 4.1 *The* nesting depth *$nd(E)$ of $E \in REbt$ is defined inductively as $nd(E) = 0$ if $E \in REb$, and $nd(E) = \max\{1 + nd(F) \mid t \in td(E),\ t(F) \in subexps(E)\}$ otherwise.*

Algorithm 2 Membership decision procedure for the subclass of $\text{REbt}_{\Sigma,\Phi}$ in Theorem 4.2

```
procedure MEMBERSHIP($w \in \Sigma^*$, $E \in \text{REbt}_{\Sigma,\Phi}$)
    ▷ Returns true if $w \in \mathcal{L}(E)$
    ▷ Modify dom($T$) in Algorithm 1 as follows
    Let subexps′($E$) ⊆ subexps($E$), where subexps′($E$) excludes all proper
    subexpressions that do not contain captures.
    Let $T : \text{env}(w) \times \text{substr}(w) \times \text{subexps}'(E) \times \text{env}(w) \to \{\text{true}, \text{false}\}$.
    Evaluate $T(e', w', E', e'')$ for all $(e', w', E', e'')$ such that $E'$ contains no captures.
    ...
    repeat
        ...
    until no additional function values of $T$ were set to true
    return true if $T(\perp, w, E, f)$ equals true for some $f$
```

Theorem 4.2 *For all $E \in REbt_{\Sigma,\Phi}$ over* FST *such that no $t(F) \in subexps(E)$ with $t \in td(E)$ contains a capture, the non-uniform membership problem can be solved in polynomial time using Algorithm 2, whereas the uniform membership problem is NP-complete for $nd(E)$ bounded and PSPACE-complete in general.*

Proof. The requirement on subexpressions of E, of the form $t(E')$, ensures that all $(f, v, F, f') \in \text{dom}(T)$, in Algorithm 2, are such that v is a substring of w (as in Algorithm 1), and thus our environments can stay functions from Φ to $\text{substr}(w)$, instead of being functions from Φ to Σ^*.

The restriction on $nd(E)$ is required in order to evaluate cells of the form (f, v, E', f'), where E' contains no captures, in polynomial time (in $|E|$, where the degree of the polynomial is in $\mathcal{O}(nd(E))$). In the uniform case membership in REb is NP-complete as it is NP-hard [1] and in NP. For the latter, observe that accepting w requires only a polynomial (in $|E|$ and $|w|$) number of cells in T to be set, bounded by the number of subexpressions matching substrings of w for a given match, including duplicates of subexpressions F, where $F^* \in \text{subexps}(E)$, when F matches more than one substring, and also including subexpressions matching ε at a given position in the input string to potentially set a capture to the empty string. These cells can be nondeterministically chosen, verifying the match by applying Algorithm 2 to those cells only. Thus uniform membership is in PSPACE, and thus PSPACE-complete by Lemma 3.4. □

Next we restrict the type of transducers allowed in expressions.

Definition 4.3 *Let n-REbt denote the set of all $E \in$ REbt such that all $t \in td(E)$ are non-deleting.*

In the uniform case the complexity of the membership problem for n-REbt remains quite high, but it sheds some light on how one may further rein the complexity in.

Lemma 4.4 *Uniform membership for $E \in$ n-REbt over* FST *is EXPSPACE-hard in general and PSPACE-hard for all fixed $nd(E) \geq 2$.*

Proof. For any fixed Turing machine T running in space $f(|w|)$, construct t_{init}, t_{step} and t_{acc} as in the proof of Theorem 3.1. Given an input string w we argue how to construct an expression $E \in$ n-REbt such that $\varepsilon \in \mathcal{L}(E)$ if and only if T accepts w. For a polynomial and exponential f this characterizes PSPACE and EXPSPACE respectively.

For $k \in \mathbb{N}$, let $D_k \in$ FST be a non-deleting transducer such that $\mathcal{L}(D_k) = \{(u, a^{|u|/k}) \mid u \in \Sigma^*\}$, where a is an arbitrarily chosen symbol in Σ and '/' denotes integer division. Note that D_k can be constructed using $k+1$ states.

If f is a polynomial, let $k = f(|w|)$ and let

$$E = D_k([_\phi t_{\text{init}}(w)]_\phi)(D_k([_\phi t_{\text{step}}(\uparrow_\phi)]_\phi))^* D_k(t_{\text{acc}}(\uparrow_\phi)).$$

Then $\varepsilon \in \mathcal{L}(E)$ if and only if T accepts w using at most $f(|w|)$ tape cells, the simulation working the same way as in Theorem 3.1, with D_k deleting all remnants of the computation. Clearly, E can be constructed in polynomial time and has nesting depth 2.

If f is the exponential c^n construct E as above, but replace each subexpression $D_k(E')$ with the subexpression $D_c^{|w|}(E')$, i.e. using $|w|$ nested applications of D_c to reduce up to $c^{|w|}$ symbols to ε. This makes makes $nd(E) = |w| + 1$, but the reduction remains polynomial. □

Lemma 4.5 *The uniform membership problem for $E \in$ n-REbt over any class Θ of transducers can be decided in space $\mathcal{O}(|E|^2|w|^2c^{2nd(E)})$ where c is the largest non-deletion constant in $td(E)$, provided that the membership problem for transducers in Θ can be solved within this space bound.*

Proof (sketch). For any string w we can check whether $w \in \mathcal{L}(E)$ in the following way. Let $\{t_1, \ldots, t_k\} = \mathrm{td}(E)$, and let c be the largest non-deletion constant of the transducers, i.e. $c \geq 1$ such that $(u, v) \in \mathcal{L}(t_i)$ implies $|u| \leq c|v| + c$ for $1 \leq i \leq k$. In the following, let $m = c^{nd(E)}$.

Now note that for E to match w it must do so without any subexpression matching a string longer than $m(|w| + c)$. This is the case as every subexpression is, by definition, surrounded by at most $nd(E)$ transducers, and each transducer shrinks the string by at most a factor of c (after deleting up to c characters), meaning in total that any string matched by a subexpression is shrunk by at most a factor m (after deleting up to c characters). A string longer than $m(|w|+c)$ being matched by a subexpression thus results in the overall string matched being longer than w.

We can then determine whether $w \in \mathcal{L}(E)$ by performing a nondeterministic search across the expression, remembering only a single search state (p, f, e) consisting of three things:

1. The current position p reached in E (viewing E as a string).
2. Whenever entering a subexpression a string of length at most $m(|w| + c)$ is nondeterministically guessed and recorded in a table f mapping subexpressions to strings, inserting a marker $\triangleright$ in each $f(E')$ to record the position up to which the string has been matched so far.
3. The current environment $e \in \mathrm{env}_{\Phi,\Sigma}$, recording the strings so far captured (if any) and these too never need to be longer than $|\Phi| \cdot (m(|w| + c) + 1)$.

Reviewers: Many of the precise details are elided by this statement, but a more complete procedure is outlined in the appendix.

Informally we start in state $(p_0, f, \bot)$ where p_0 is the leftmost position in E, setting f to map E to $\triangleright w$, then we nondeterministically walk the expression, guessing a new string to enter into f whenever we enter a subexpression, verifying the guess, updating the parent marker (and the current environment if a capture), and simulating transducers as needed, whenever we exit a subexpression. If the rightmost position can be reached with $f(E) = w\triangleright$ then w is matched.

A state uses space $|E| + |E| \cdot m(|w| + c) + |\Phi| \cdot (m(|w| + c) + 1)$, so the procedure runs in nondeterministic space $\mathcal{O}(|E||w|c^{nd(E)})$ (since we assumed that the space needed for evaluating transducers fits into this bound). Thus, applying Savitch theorem [8] gives a deterministic procedure in $\mathcal{O}(|E|^2|w|^2c^{2nd(E)})$ space. □

Theorem 4.6 *Uniform membership for $E \in$ n-REbt over* FST *is EXPSPACE-complete in general and PSPACE-complete for all fixed $nd(E) \geq 2$.*

Proof. Combines Lemma 4.4 and 4.5. □

The non-uniform variant of the problem is not surprisingly a bit less complex, but remains NP-complete.

Lemma 4.7 *The non-uniform membership problem for n-REbt over Θ is in NP, provided that deciding membership is in NP for every $\theta \in \Theta$.*

Proof. For any (fixed) $E \in$ n-REbt, if w is an input string, then we can check whether $w \in \mathcal{L}(E)$ in nondeterministic polynomial time by the following procedure. Let c be the largest non-deletion constant in $\mathrm{td}(E)$ and $m = c^{nd(E)}$. By the same argument as in Lemma 4.5 no subexpression (more precisely: none of the matching relations inductively used in matching, as defined by Definition 2.2) in E matches a string longer than $m(|w| + c)$ when E matches w. Additionally, fewer than $(|w| + c) \cdot |\mathrm{subexps}(E)|$ instances of a subexpression matching a string longer than m can occur, as such strings contribute to the length of the overall string matched (the $|\mathrm{subexps}(E)|$ accounts for nested subexpressions involved in the match of part of a substring, or transducer preimage, matched by a larger subexpression).

Start by nondeterministically choosing a string v of length $m(|w| + c)^2 \cdot |\mathrm{subexps}(E)|$, and construct v' to be a string containing as a substring *every* string of length at most m (the length of v' is thus exponential in m, but is fixed as it depends only on E), and let $w' = vv'$. Then modify Algorithm 1 by adding the following step to the repeat-until loop:

```
if $T(f, u, F, f') =$ true and $(u, v) \in \mathcal{L}(t)$ then
    $T(f, v, t(F), f') \leftarrow$ true
```

The resulting algorithm, applied to the input string $|w'|$, sets $T(\perp, w, E, f)$ to true for some f if and only if E matches w. This procedure runs in nondeterministic polynomial time as w' is polynomial in length when E is taken to be fixed (as m is then constant). This works because any subexpression matching a string of length at most m can find that string in the v' section of w', and the at most $(|w|+c) \cdot |\mathrm{subexps}(E)|$ subexpressions matching strings of length at most $m(|w| + c)$ will have their strings nondeterministically generated in the v section of w'. □

Lemma 4.8 *The non-uniform membership problem for n-REbt$_{\Sigma,\Phi}$ over Θ is NP-hard if Θ contains all single state FST.*

Proof. We demonstrate NP-hardness by a reduction from the NP-hard LONGEST COMMON SUBSEQUENCE problem [7], the instances of which are the tuples $(\{w_1, \ldots, w_m\}, n)$, $\{w_1, \ldots, w_m\} \subset \mathcal{L}((a \mid b)^*)$, $n \in \mathbb{N}$, such that there exists a string v of length n which forms a subsequence of w_i for all i.

Take $\Sigma = \{a, b, \#, x\}$ and $\Phi = \{1\}$, let t and s be the transducers $a : x, b : x$ and $a : a, b : b, \varepsilon : a, \varepsilon : b$, then $E_{\mathrm{lcs}} = t([_{guess}(a \mid b)^*]_{guess})(\#(s(\uparrow_{guess}))^*$ matches the string $x^n\#w_1\# \cdots \#w_m$ if and only if $w_1, \ldots, w_m$ are strings in $\{a, b\}^*$ with a common subsequence of length n.

To see this, note that the initial x^n means that a string $w \in \{a, b\}^n$ must be captured by the capturing group '*guess*'. Thus, each w_i must be matched by inserting as and bs into w, making w a common subsequence of each of $w_1, \ldots, w_m$. As such any instance of LONGEST COMMON SUBSEQUENCE can be decided by checking whether $x^n\#w_1\# \cdots \#w_m \in \mathcal{L}(E_{\mathrm{lcs}})$. □

This expression E_{lcs}, used in the previous proof, will be reused near-verbatim to demonstrate NP-hardness of membership in fln- and nt-REbt (to be defined in the next section).

Theorem 4.9 *The non-uniform membership problem for n-REbt$_{\Sigma,\Phi}$ over Θ is NP-complete for every Θ containing all single state FST, provided that the membership problem of each transducer in Θ is in NP.*

Proof. Combines Lemma 4.7 and 4.8. □

5. Limiting Nesting and Non-Functional Behavior

In this section we consider restrictions to the way in which REbt may nest the application of transducers, and in the process also consider the restriction to functional transducers, the combination of which gives rise to a convenient normal form. The choice of restrictions is driven by the intuition that the construction in Theorem 3.1 relies on the subexpression $[_\phi t_{\text{step}}(\uparrow_\phi)]_\phi{}^*$ to match complex languages by the iterated application of t_{step}. By syntactically avoiding the iterated application of a transducer to previous output produced by the same transducer, we obtain a class of languages with much better properties.

Definition 5.1 *For $E \in REbt_{\Sigma,\Phi}$ (over an arbitrary class of transducers) let $\Sigma_{td(E)} = \Sigma \uplus \{\langle_t, \rangle_t \mid t \in td(E)\}$ and define $\tau(E) \in REb_{\Sigma_{td(E)},\Phi}$ as the expression obtained by replacing every subexpression of the form $t(F)$, where $t \in td(E)$, with $\langle_t \cdot F \cdot \rangle_t$. Let τ^{-1} be the inverse transformation, so $\tau^{-1}(\tau(E)) = E$ for all E.*

Note that each string in $\mathcal{L}(\tau(E))$ is a valid expression in $\text{RE}_{\Sigma_{\text{td}(E)}}$. Thus, $\tau^{-1}\mathcal{L}(\tau(E))$ denotes the set of expressions in REt_Σ obtained by applying τ^{-1} to each expression in $\mathcal{L}(\tau(E))$. Furthermore, $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$ is the union of all languages obtained by applying $\mathcal{L}$ to each expression in $\tau^{-1}\mathcal{L}(\tau(E))$.

Let us define the first restriction, which forbids the capture of the output of non-functional transducers.

Definition 5.2 *An expression $E \in REbt$ is* functional *if every transducer that occurs in a capturing subexpression is functional (i.e. $t \in td(F)$ for some $[_\phi F]_\phi \in subexps(E)$ only if t is functional). We denote the subset of REbt containing precisely the functional expressions as f-REbt.*

Lemma 5.3 *In general, $\mathcal{L}(E) \subseteq \mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$, but $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E))) = \mathcal{L}(E)$ for $E \in$ f-REbt.*

Proof (sketch). Strings $w \in \mathcal{L}(\tau(E))$ (potentially) contains a number of substrings of the form $\langle_t v\rangle_t$, precisely where a transducer t is applied to obtain strings in $\mathcal{L}(E)$. Applying τ^{-1} recovers the transducers (turns the substrings $\langle_t v\rangle_t$ back into $t(v)$), and then applying $\mathcal{L}$ evaluates all transducers. In general, $\mathcal{L}(E) \subseteq \mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$, since in $\mathcal{L}(E)$ subexpressions containing transducers first apply the transducer before (potentially) copying the output of the transducer, while in $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E)))$, subexpressions containing transducers (might) first get copied before applying the transducers. But the restrictions on transducers in f-REbt ensures equality. □

Example 5.4 *In this example we show that the equality $\mathcal{L}(\tau^{-1}\mathcal{L}(\tau(E))) = \mathcal{L}(E)$ of Lemma 5.3 does not hold for REbt in general. Let $E = [_1 f(a^*)]_1 \uparrow_1$, with f given by $a : a, a : b$. Then $\mathcal{L}(\tau(E)) = \{\langle_f a^n\rangle_f \langle_f a^n\rangle_f \mid n \in \mathbb{N}_0\}$, thus $\mathcal{L}(\tau^{-1}(\langle_f a^n\rangle_f \langle_f a^n\rangle_f)) = \mathcal{L}(f(a^n)f(a^n)) = (a \mid b)^{2n}$, whereas $\mathcal{L}(E) = \{ww \mid w \in \mathcal{L}((a \mid b)^*)\}$. That is, in $\mathcal{L}(E)$ the transducer is applied before it* *gets copied by the capturing group and backreference, whereas τ "hides" the transducer as a string, letting it be copied before it is applied, after which τ^{-1} is applied to recover the two transducers obtained by copying, and $\mathcal{L}$ then evaluates them independently.*

Next we construct a transducer T such that under certain restrictions, expressions over FST satisfy $\mathcal{L}(T(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$.

Definition 5.5 *For $c \in \mathbb{N}$ and $E \in REbt_{\Sigma,\Phi}$ over* FST*, with $td(E) = \{t_1, \ldots, t_n\}$, where $t_i = (Q_i, \Sigma, q_{0,i}, \delta_i, F_i)$, for $i \in [n]$, let $T_{E,c} = (Q, \Sigma', q_0, \delta, F)$ be the transducer defined as follows: (i) $Q = \{q \in (Q_1 \uplus \cdots \uplus Q_n)^* \mid |q| \leq c\}$; (ii) $\Sigma' = \Sigma_{td(E)} = \Sigma \uplus \{\langle_t, \rangle_t \mid t \in td(E)\}$; (iii) $q_0 = \varepsilon$; (iv) $F = \{\varepsilon\}$; and; (v) $\delta = \delta_\langle \cup \delta_\rangle \cup \delta_\varepsilon \cup \delta_\Sigma$ where:*

- $\delta_\langle = \{(q, \langle_{t_i}, \varepsilon, q \cdot q_{0,i}) \mid i \in [n], q \in Q, |q| + 1 \leq c\}$;
- $\delta_\rangle = \{(q_1 \cdots q_{k-1} q_k, \rangle_{t_i}, \varepsilon, q_1 \cdots q_{k-1}) \mid i \in [n], q_1 \cdots q_k \in Q, q_k \in F_i\}$;
- $\delta_\varepsilon = \{(\varepsilon, \alpha, \alpha, \varepsilon) \mid \alpha \in \Sigma'\}$;
- *δ_Σ is defined inductively over the length k of state sequences, as follows: for $\alpha, \beta \in \Sigma_\varepsilon$, $(q_1 \cdots q_k, \alpha, \beta, q'_1 \cdots q'_k) \in \delta_\Sigma$ if for some $\alpha' \in \Sigma_\varepsilon$ and $i \in [n]$,*
    - $(q_k, \alpha, \alpha', q'_k) \in \delta_i$; *and,*
    - $(q_1 \cdots q_{k-1}, \alpha', \beta, q'_1 \cdots q'_{k-1}) \in \delta_\varepsilon \cup \delta_\Sigma$.

Informally, without the length restriction enforced by c in $T_{E,c}$, i.e. letting $c = \infty$, we have $\mathcal{L}(T_{E,\infty}(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$. Next we define a restriction to ensure that a (finite) value can be selected for c such $\mathcal{L}(T_{E,c}(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$.

Definition 5.6 *An expression $E \in REbt$ such that every $t \in td(E)$ occurs only once in E, is* loop-free *if $\mathcal{L}(\tau(E))$ contains no subexpression of the form $\langle_t \cdots \langle_t \cdots \rangle_t \cdots \rangle_t$, i.e. there are no nested subexpressions $\langle_t \cdots \rangle_t$ for any $t \in td(E)$. We denote the set of all loop-free REbt by l-REbt.*

Lemma 5.7 *For $E \in l\text{-}REbt_{\Sigma,\Phi}$ over* FST*, we have $\mathcal{L}(T_{E,|td(E)|}(\tau(E))) = \mathcal{L}(\tau^{-1}(\mathcal{L}(\tau(E))))$.*

Proof. Set c to be the maximum number of nested transducers in $\tau^{-1}(v)$ over any $v \in \mathcal{L}(\tau(E))$, i.e. $c \leq |\text{td}(E)|$ by Definition 5.6. We prove that $T_{E,c}$ simulates all transducers running at each point of an input string. Thus $T_{E,c}$ produces the same output strings as would be obtained by first having τ^{-1} recover the transducers, and then evaluating them by using $\mathcal{L}$. This can be seen by induction on the number of transducers in E. Assume $T_{E,c}$ can go from state $q_1 \cdots q_k$ to $q'_1 \cdots q'_k$ while reading v and producing w as output, and that q_i (and thus also q'_i) is a state from t_i, for $i \in [k]$. Then there exists strings $v_0, \ldots, v_k$, with $v_0 = w$ and $v_k = v$, such that transducer t_i, for $i \in [k]$, can go from state q_i to q'_i when reading v_i and producing v_{i-1} as output. Add to this the brackets $\langle_{t_i}$, $\rangle_{t_i}$, instructing $T_{E,c}$ when to start and stop and check for acceptance of transducer t_i, and make $T_{E,c}$ act as the identity (the rules in δ_ε) when no transducers are simulated, to complete the picture.

It is sufficient to pick c to be equal to the $|\text{td}(E)|$, since this is an upper bound for the nesting depth for pairs of transducer brackets, i.e. symbols of the form $\langle_t$, $\rangle_t$, for strings matched by

$\tau(E)$, given that $E \in$ l-REbt$_{\Sigma,\Phi}$. The transducer $T_{E,c}$ reaches a state sequence $q_1 \cdots q_k$ precisely when the combined effect of k (distinct) transducers are being simulated by $T_{E,c}$, and $|\mathrm{td}(E)|$ places an upper bound on the number of transducers being applied simultaneously. □

Corollary 5.8 *For all $E \in$ fl-REbt$_{\Sigma,\Phi}$ (i.e. expressions fulfilling both Definitions 5.2 and 5.6) over* FST *we have $\mathcal{L}(E) = \mathcal{L}(T_{E,|td(E)|}(\tau(E))$, and thus every fl-REbt can be put in a normal form $t(E')$ where $E' \in$ REb (i.e. an expression with only a single top-level transducer).*

Proof. This result combines Lemma 5.3 with Lemma 5.7. □

Definition 5.9 *Let t-REbt denote the subset of REbt which are in the normal form of Corollary 5.8.*

Remark 5.10 *Lemma 4.8 demonstrates that non-uniform membership for fl-REbt and thus t-REbt (both over* FST*) is NP-hard, since E_{lcs} used in the proof of Lemma 4.8 is in fl-REbt (although the transducer s in E_{lcs} is not functional, output of s is not captured), and the above corollary can be used to convert E_{lcs} to an expression t-REbt.*

6. The Membership Problem for t-REbt

We show that the class t-REbt, and thus also fl-REbt, has a decidable membership problem, but it is complex, even with the input string fixed.

Corollary 6.1 (of Lemma 3.4) *For $E \in$ t-REbt$_{\Sigma,\Phi}$ over* FST *it is PSPACE-hard to decide whether $\varepsilon \in \mathcal{L}(E)$.*

Proof. Modify Lemma 3.4 by letting $E = D([_1\Sigma^*]_1 t_1(\uparrow_1) \cdots t_n(\uparrow_1)) \in$ fl-REbt, with $t_1, \ldots, t_n$ as in the proof of Lemma 3.4. Then $\varepsilon \in \mathcal{L}(E)$ iff the intersection $\mathcal{L}(A_1) \cap \ldots \cap \mathcal{L}(A_n)$, again with the A_i as in (the proof of) Lemma 3.4, is non-empty. The expression E can converted into t-REbt normal form by Corollary 5.8. Note that $T_{E,2}$ (i.e. $c = 2$) needs to be constructed, and the resulting expression is therefore polynomial in the size of E. Note that the proof of Lemma 5.7 in fact shows that we can set c to be the maximum number of nested transducers in $\tau^{-1}(v)$ over any $v \in \mathcal{L}(\tau(E))$, which in this case is $c = 2$, instead of using $|\mathrm{td}(E)| = n + 1$. □

Next we show that the membership problem for t-REbt (and by extension fl-REbt) can be decided in polynomial space. The approach works by, for a given input string and $t(F) \in$ t-REbt, computing the preimage of t on w, and intersecting this regular language with $\mathcal{L}(F)$. To achieve this within polynomial space, however, it is necessary to not expand captures and backreferences.

Definition 6.2 *For $E \in$ REbt$_{\Sigma,\Phi}$ we define $\sigma(E) \in$ REt$_{\Sigma_\Phi,\emptyset}$ with $\Sigma_\Phi = \Sigma \uplus \{[\![^i_\phi,]\!]^i_\phi, \uparrow_\phi \mid \phi \in \Phi, i \in [n]\}$, where n is equal to the maximum number of capturing expressions on the same capturing symbol, by making the following substitutions in E.*

1. *Replace every subexpression of the form $\uparrow_\phi$, with $\upharpoonleft_\phi$.*
2. *Replace the ith subexpression (ordered for example from left to right based on the position of the opening capturing bracket) of the form $[_\phi F]_\phi$, by $[\![^i_\phi \cdot F \cdot]\!]^i_\phi$.*

Let σ^{-1} be the inverse transformation of σ, i.e. $\sigma^{-1}(\sigma(E)) = E$, and extend σ^{-1} to sets of expressions in the obvious way.

For a finite automaton A, over alphabet Σ, and $n \in \mathbb{N}$, we define an automaton $C_{A,\Phi,n}$, which will be used to determine if $\mathcal{L}(A) \cap \mathcal{L}(E)$, for $E \in$ REb, is non-empty. To simplify our constructions, and since it will not make our results less general, we assume that A has no ε-transitions. Recall that we use $\perp$ to denote the partial function with empty domain.

Definition 6.3 *For Σ_Φ and n as in Definition 6.2, and an automaton $A = (Q, \Sigma, q_0, \delta, F)$, let $C_{A,\Phi,n} = (Q', \Sigma_\Phi, q_0', \delta', F')$ be the automaton where $Q' = Q \times (\Phi \to 2^{Q\times Q}) \times ((\Phi \times [n]) \to 2^{Q \times Q})$, $q_0 = (q_0, \perp, \perp)$, $F' = \{(q, C, M) \in Q \mid q \in F\}$, and $((q, C, M), \alpha, (q', C', M')) \in \delta'$ if one of the following holds:*

- *$(q, \alpha, q') \in \delta$, $C' = C$, and $M' = \{(\phi, i, (p, p'')) \mid (\phi, i, (p, p')) \in M, (p', \alpha, p'') \in \delta\}$,*
- *$\alpha = [^i_\phi$, $q = q'$, $C' = C$, and $M' = M[(\phi, i) \mapsto \{(p, p) \mid p \in Q\}]$,*
- *$\alpha =]^i_\phi$, $q = q'$, $M' = M[(\phi, i) \mapsto \perp]$, and $C' = C[\phi \mapsto M(\phi, i)]$ or*
- *$\alpha = \upharpoonleft_\phi$, $(q, q') \in C(\phi)$, $C' = C$, and, for all $\phi \in \Phi$ and $i \in [n]$ we have $M'(\phi, i) = \{(p, p'') \mid (p, p') \in M(\phi, i), (p', p'') \in C(\phi)\}$.*

In the next result we extend $\mathcal{L}$ to be also applied to a set of expressions, and to denote the union of languages defined by the expressions.

Lemma 6.4 *Let A be an automaton, $E \in$ REb, and $n \in \mathbb{N}$, with n equal to the the maximum number of capturing expressions on the same capturing symbol, in E. Then we have $\mathcal{L}(\sigma^{-1}(\mathcal{L}(C_{A,\Phi,n}) \cap \mathcal{L}(\sigma(E)))) = \mathcal{L}(A) \cap \mathcal{L}(E)$.*

Proof (sketch). For $w \in \mathcal{L}(\sigma(E))$, note that $\mathcal{L}(\sigma^{-1}(w))$ is a single string in $\mathcal{L}(E)$ (as σ^{-1} recovers the captures and backreferences, and $\mathcal{L}$ then evaluates them). $C_{A,\Phi,n}$, running on w, simulates A running on $\mathcal{L}(\sigma^{-1}(w))$. This can be demonstrated by induction on the length of w. If $C_{A,\Phi,n}$ reaches (q, C, M) on a prefix of w, then A can reach q on the corresponding prefix of $\mathcal{L}(\sigma^{-1}(w))$, and $(q, q') \in C(\phi)$ if and only if A can go from state q to the state q' on the string captured by $\phi \in \Phi$ (allowing the simulation of A on a $\upharpoonleft_\phi$, corresponding to a backreference). This follows from the way M is built as a parallel simulation of A, starting in any state, on all the currently ongoing captures (i.e. $M(\phi, i)$ simulates A on a partial capture and $C(\phi)$ records a completed capture). □

Theorem 6.5 *The emptiness of $\mathcal{L}(E) \cap \mathcal{L}(A)$, for $E \in$ REb and A a finite automaton, can be decided in PSPACE.*

Proof. As $\mathcal{L}(\sigma(E))$ and $\mathcal{L}(C_{A,\Phi,n})$ are regular languages, a standard product automaton can be constructed for $\mathcal{L}(\sigma(E)) \cap \mathcal{L}(C_{A,\Phi,n})$. While $C_{A,\Phi,n}$ is potentially large, emptiness can be decided in polynomial space by performing a nondeterministic search (as nondeterministic polynomial space equals polynomial space) for an accepting computation by incrementally constructing each, polynomially sized, state as it is visited, forgetting it again in the next step. □

Theorem 6.6 *The uniform membership problems for t-REbt and fl-REbt (both over* FST*) are PSPACE-complete.*

Proof. For t-REbt (and thus for fl-REbt) hardness is established in Corollary 6.1. To see that uniform membership for t-REbt is in PSPACE, consider an expression $E = t(F)$, for $t \in \mathrm{FST}$ and $F \in \mathrm{REb}$. To check whether $w \in \mathcal{L}(E)$, construct a finite automaton A with $\mathcal{L}(A) = \{v \mid (w, v) \in \mathcal{L}(t)\}$ (this can be done with standard techniques, producing an automaton A, polynomial in size in $|t| \cdot |w|$). Then if $\mathcal{L}(A) \cap \mathcal{L}(F) \neq \emptyset$, we have $w \in \mathcal{L}(E)$, which we can check in polynomial space by Theorem 6.5. This procedure extends to fl-REbt by additionally constructing $T_{E,c}$, as in Corollary 5.8, in an incremental fashion. □

A uniform membership problem in PSPACE improves vastly on the unrestricted case, and the top-level transducer appears to be a very natural formalism. More importantly, fairly minor further restrictions recover the easier membership problems established for REb.

Theorem 6.7 *The uniform and non-uniform membership problem for nt-REbt is NP-complete.*

Proof. Take $E = t(F) \in$ nt-REbt and let w be the input string. Since t is nondeleting, $|t| \cdot |w| \geq \max\{|v| \mid (w, v) \in \mathcal{L}(t)\}$, so we can nondeterministically choose a v with $(w, v) \in \mathcal{L}(t)$ and apply Lemma 3.3 to check in nondeterministic polynomial time if $v \in \mathcal{L}(F)$. NP-hardness in the non-uniform case is established by Lemma 4.8 (see Remark 5.10). □

7. Summary and Future Work

Summary. We have (i) proposed an extension of regular expressions with backreferences by additional transducers; (ii) established that this makes membership testing intractable; and (iii) explored various restrictions to form a practical basis for use in software. By Example 2.3 all restrictions can match L_{RD}, L_{MA}, and L_{CA}, but offer different levels of membership testing complexity and expressiveness. For immediate integration in an existing backtracking matching engine the restriction in Theorem 4.2 appears to be the obvious choice, with no transducer preimage ever captured, the matching procedure is largely the same as for REb.

Further, the relative tractability of the nondeleting class demonstrates that one source of intractability is the ability gained by an unrestricted use of transducers to erase every trace of an arbitrarily complex computation that has been made. However, the reduction of fl-REbt to t-REbt shows that the latter, despite being very simple, can capture many natural situations. Small additional restrictions can then be applied to obtain highly tractable subclasses.

Future work. The precise expressiveness of the classes should be considered, several gaps exist beyond what follows naturally from what we have done here; f-REbt $\equiv$ REbt; fl-REbt $\equiv$ t-REbt; n-, fl-/t-REbt all being strict subclasses of f-REbt/REbt and strict superclasses of REb.

There are further some gaps on the computational complexity (e.g. non-uniform membership for t-REbt), and relative succinctness should be considered.

References

[1] A. AHO, Algorithms for Finding Patterns in Strings. In: J. VAN LEEUWEN (ed.), *Handbook of Theoretical Computer Science (Vol. A)*. 1990, 255–300.
`http://dl.acm.org/citation.cfm?id=114872.114877`

[2] M. BERGLUND, B. VAN DER MERWE, On the semantics of regular expression parsing in the wild. *Theoretical Computer Science* 679 (2017), 69–82.

[3] M. BERGLUND, B. VAN DER MERWE, *Re-Examining regular expressions with backreferences*, 2018. Preprint.

[4] C. CÂMPEANU, K. SALOMAA, S. YU, A Formal Study of Practical Regular Expressions. *International Journal of Foundations of Computer Science* 14 (2003) 6, 1007–1018.

[5] J. DASSOW, G. PĂUN, Grammars with controlled derivations. In: G. ROZENBERG, A. SALOMAA (eds.), *Handbook of Formal Languages: Volume 2. Linear Modeling Background and Application*. 2010.

[6] D. KOZEN, Lower Bounds for Natural Proof Systems. In: *18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1977, 254–266.
`https://doi.org/10.1109/SFCS.1977.16`

[7] D. MAIER, The complexity of some problems on subsequences and supersequences. *Journal of the ACM* 25 (1978) 2, 322–336.

[8] W. SAVITCH, Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4 (1970) 2, 177 – 192.
`http://www.sciencedirect.com/science/article/pii/S002200007080006X`

[9] M. SCHMID, Characterising REGEX languages by regular languages equipped with factor-referencing. *Information and Computation* 249 (2016), 1–17.
`http://www.sciencedirect.com/science/article/pii/S0890540116000109`

Appendix

The language $a^n b^n$ is not in REb

Lemma 7.1 *There exists no $E \in$ REb with $\mathcal{L}(E) = \{a^n b^n \mid n \in \mathbb{N}\}$.*

Proof. Intuitively, capturing and backreferencing without the aid of transducers cannot keep track of the number of as in a way which is useful in generating the right number of bs. Next we provide the details.

We proceed by contradiction, and assume that $\mathcal{L}(E) = \{a^n b^n \mid n \in \mathbb{N}\}$. Now, note that the inductive definition of the matching relation in Definition 2.2 implies that the left-most b in any $w = a^n b^n$ is matched using $(f, b, f) \in \mathcal{M}(b)$, with range$(f) \subset \mathcal{L}(a^*)$, but as no further as occur in the input string, none of these captures, except possibly for empty captures, are ever backreferenced by any subexpressions used for matching the remainder of the input string. Since no other information reaches the second half of the match from the first half, there is no way to represent that exactly n consecutive bs should be matched. $\square$

Extended proof of Lemma 4.5

Proof. [of Lemma 4.5] Taking $E \in$ n-REbt$_{\Sigma,\Phi}$ and any string w we can check whether $w \in \mathcal{L}(E)$ in the following way. Let $\{t_1, \ldots, t_k\} = \text{td}(E)$, since all t_i are non-deleting there exists a constant $c \geq 1$ such that $(u, v) \in \mathcal{L}(t_i)$ implies $|u| \leq c|v| + c$ for $1 \leq i \leq k$, let $m = c^{nd(E)}$.

Now note that for E to match w it must do so without any subexpression matching a string longer than $m(|w| + c)$. This is the case as every subexpression is, by definition, surrounded by at most $nd(E)$ transducers, and each transducer shrinks the string by at most a factor of c (after deleting up to c characters), meaning in total that any string matched by a subexpression is shrunk by at most a factor m (after deleting up to c characters). A string longer than $m(|w|+c)$ being matched by a subexpression thus results in the overall string matched being longer than w.

Using this fact we can construct a directed graph $G = (V, D)$ with a distinguished start state G_S and a set of end states G_E such that there exists a state $v \in G_E$ reachable from G_s if and only if and only if $w \in \mathcal{L}(E)$. First construct

- $P = \{\text{left}(E'), \text{right}(E') \mid E' \in \text{subexps}(E)\}$,
- $S_{\text{guess}} = \text{subexps}(E) \to \{v \triangleright v' \mid v, v' \in \Sigma^*, |vv'| \leq m(|w| + c)\}$, and
- $S_{\text{env}} = \Phi \to \{v \mid v \in \Sigma^*, |v| \leq m(|w| + c)\} \cup \{\text{undefined}\}$,

and then let

- $V = P \times (\text{subexps}(E) \to S_{lm}) \times (\Phi \to \Sigma_l)$,
- $G_S = (\text{left}(E), \{(E, \triangleright w)\}, \bot)$,
- $G_E = \{\text{right}(E)\} \times \{f \in S_{\text{guess}} \mid f(E) = w\triangleright\} \times S_{\text{env}}$.

All that remains is to construct the directed edges in D, which will implement the semantics of E. D contains exactly the following edges (we write $v \to v'$ to denote an edge from v to v').

1. For every subexpressions $F = (G^*)$, $F = t(G)$, $F = [_\phi G]_\phi$, $F = (G \cdot H)$, $F = (G \mid H)$ and (symmetrically) $F = (H \mid G)$ we have $(\text{left}(F), V_\text{guess}, V_\text{env}) \to (\text{left}(G), V'_\text{guess}, V_\text{env})$ for all V_guess, V_env and $V'_\text{guess} = V_\text{guess}[G \mapsto \triangleright v]$ for any v.
2. For every subexpression $F = \alpha \in \Sigma_\varepsilon$ we have $(\text{left}(F), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V_\text{env})$, for $V_\text{guess}(F) = \triangleright\alpha$ and $V'_\text{guess} = V_\text{guess}[F \mapsto \alpha\triangleright]$.
3. For every subexpression $F = \uparrow_\phi$ we have $(\text{left}(F), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V_\text{env})$ for any V_guess, V_env and V'_guess such that $V_\text{guess} = v_1 \triangleright v_2 v_3$, $V'_\text{guess} = V_\text{guess}[F \mapsto v_1 v_2 \triangleright v_3]$, and $V_\text{env}(\phi) = v_2$ for some v_1, v_2 and v_3.
4. For every subexpression $F = t(G)$ we have $(\text{right}(G), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V_\text{env})$ for all V_guess, V_env, and V'_guess such that $V_\text{guess}(G) = u\triangleright$, $V_\text{guess}(F) = \triangleright v$, $(u, v) \in \mathcal{L}(t)$, $V'_\text{guess} = V_\text{guess}[F \mapsto v\triangleright]$, for some u and v.
5. For every subexpression $F = [_\phi G]_\phi$ we have $(\text{right}(G), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V'_\text{env})$ for all V_guess, V'_guess, V_env, and V'_env such that $V_\text{guess}(F) = \triangleright v$, $V_\text{guess}(G) = v\triangleright$, $V'_\text{guess} = V_\text{guess}[F \mapsto v\triangleright]$, and $V'_\text{env} = V_\text{env}[\phi \mapsto v]$, for some v.
6. For every subexpression $F = (G \cdot H)$ we have $(\text{right}(G), V_\text{guess}, V_\text{env}) \to (\text{left}(H), V'_\text{guess}, V_\text{env})$ for all V_guess, V'_guess and V_env such that $V_\text{guess}(G) = v_1\triangleright$, $V_\text{guess}(F) = \triangleright v_1 v_2$, $V'_\text{guess} = V_\text{guess}[F \mapsto v_1 \triangleright v_2]$, for some v_1 and v_2.
7. For every subexpression $F = (G \cdot H)$ we have $(\text{right}(H), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V_\text{env})$ for all V_guess, V_env, V'_guess such that $V_\text{guess}(F) = v_1 \triangleright v_2$, $V_\text{guess}(H) = v_2\triangleright$ and $V'_\text{guess} = V_\text{guess}[F \mapsto v_1 v_2 \triangleright]$ for some v_1 and v_2.
8. For every subexpression $F = (G \mid H)$ and $F = (H \mid G)$ (e.g. for either of the subexpressions) we have $(\text{right}(G), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V_\text{env})$ for all V_guess, V_env, V'_guess such that $V_\text{guess}(F) = \triangleright v$, $V_\text{guess}(G) = v\triangleright$, $V'_\text{guess} = V_\text{guess}[F \mapsto v\triangleright]$.
9. For every subexpression $F = (G^*)$ we have $(\text{right}(G), V_\text{guess}, V_\text{env}) \to (\text{left}(G), V'_\text{guess}, V_\text{env})$ for all V_guess, V_env and V'_guess, such that $V_\text{guess}(F) = v_1 \triangleright v_2 v_3$, $V_\text{guess}(G) = v_2\triangleright$, $V'_\text{guess} = V_\text{guess}[F \mapsto v_1 v_2 \triangleright v_3][G \mapsto \triangleright u]$ for any v_1, v_2, v_3 and u.
10. For every subexpression $F = (G^*)$ we have $(\text{right}(G), V_\text{guess}, V_\text{env}) \to (\text{right}(F), V'_\text{guess}, V_\text{env})$ for all V_guess, V_env and V'_guess, such that $V_\text{guess}(F) = v_1 \triangleright v_2$, $V_\text{guess}(G) = v_2\triangleright$, $V'_\text{guess} = V_\text{guess}[F \mapsto v_1 v_2 \triangleright]$ for any v_1, v_2.

Then simply note that we can construct this graph incrementally, remembering only the current state and its outgoing edges, to nondeterministically search for a path from G_S to any state in G_E. It remains to show that such a path exists if and only if $w \in \mathcal{L}(E)$. This can be seen by induction on the length of the path, relating it to the tree of matching relations implied by Definition 2.2 taken in postorder. The nodes reached relate to the position in the tree in the following way: the node $(p, V_\text{guess}, V_\text{env})$ records the subexpression the current position in the tree corresponds to (p being set to left(F) when passing F heading *down* in the tree, right(F) when heading *up*), V_env the current environment (exactly as in Definition 2.2), and V_guess encodes a *guess* of the strings that will be matched by the subexpressions above this position in the tree (with the marker encoding what has already been matched). This guess is made upon entering the subexpression, in item 1 above. For example, take items 9 and 10, for the subexpression $F = (G^*)$ from above: item 9 corresponds to G matching a prefix of the string guessed for the parent F, and then repeating the match, whereas item 10 corresponds to G matching all that remains of the string F should match (i.e. moving the marker all the way to the right in $V'_{\text{guess}}(F)$).

A single node in G uses space $2|E|+|E|\cdot m(|w|+c)+|\Phi|\cdot(m(|w|+c)+1)$, so the procedure runs in nondeterministic space $\mathcal{O}(|E||w|c^{nd(E)})$ (since we assumed that the space needed for evaluating transducers fits into this bound). Thus, applying Savitch theorem [8] gives a deterministic procedure in $\mathcal{O}(|E|^2|w|^2c^{2nd(E)})$ space. $\square$