Optimizing the Region Algebra is PSPACE-complete

Wouter Gelade[2], Frank Neven

Hasselt University and transnational University of Limburg,
School for Information Technology

Abstract

The Region Algebra is a set-at-a-time algebra for querying text regions. We show that satisfiability, inclusion, and equivalence testing of region algebra expressions are PSPACE-complete. This improves upon the previously known NP lower bounds and EXPTIME upper bounds.

Key words: Computational complexity, Region Algebra, Databases

1. Introduction

The region algebra was introduced by Consens and Milo [1], and is a set-at-a-time algebra for manipulating text regions based on the PAT-algebra [2, 3]. The expressiveness and complexity of the region algebra were investigated by Consens and Milo [1]. Among other things, they show that the satisfiability problem is decidable through a reduction to satisfiability of first-order logic formulas over trees. Testing the satisfiability of such logical formulas has non-elementary complexity [4]. Furthermore, they provide an NP lower bound. Neven [5] obtained an EXPTIME upper bound for the satisfiability, inclusion and equivalence problems through a reduction to the corresponding problems for extended attribute grammars.

In this paper, we provide a matching PSPACE upper and lower bound for the basic decision problems regarding region algebra expressions.

2. The Region Algebra

For the remainder of the paper we fix a finite alphabet Σ. A Σ-string w is a finite string over Σ. We denote its length by $|w|$. For a finite set of region names $\mathcal{R}$, an *$\mathcal{R}$-text* I is a pair (w, λ), where w is a Σ-string and λ is a function mapping each region name in $\mathcal{R}$ to a set of w-regions. Here, a *w-region* is a pair (i, j) where $1 \leq i \leq j \leq |w|$. When $\mathcal{R}$ is clear from the context, we usually write text instead of $\mathcal{R}$-text.

For instance, let $\mathcal{R} = \{\mathbf{Proc}, \mathbf{Func}, \mathbf{Var}\}$ and $\Sigma = \{a, b, c, \ldots, z\}$. Figure 1 shows a graphical illustration of the text $I = (w, \lambda)$ over $\mathcal{R}$ where $w =$ abcdefghijklmnop, $\lambda(\mathbf{Proc}) = \{(1, 16), (6, 10)\}$, $\lambda(\mathbf{Func}) = \{(12, 16)\}$, and $\lambda(\mathbf{Var}) = \{(2, 3), (6, 7), (12, 13)\}$.

We say that r is a *region in I* when $r \in \lambda(R)$, for some $R \in \mathcal{R}$. For a region $r = (i, j)$, we denote the beginning i and finish j of r by $b(r)$ and $f(r)$, respectively. Furthermore, $w(r)$ denotes the string $a_{b(r)} \ldots a_{f(r)}$, for $w = a_1 \cdots a_n$.

For two regions r and s in I, define:

- $r < s$ if $f(r) < b(s)$ (r *precedes* s); and
- $r \subset s$ if $b(s) \leq b(r)$, $f(r) \leq f(s)$, and $s \neq r$ (r is *included in* s).

We also allow the dual predicates $r > s$ and $r \supset s$ which have the obvious meaning. A text I is *hierarchical* if

- $\lambda(R) \cap \lambda(R') = \emptyset$ for all region names R, R' in $\mathcal{R}$, with $R \neq R'$; and
- for all regions r, s in I, with $r \neq s$, one of the following holds: $r < s$, $s < r$, $r \subset s$, or $s \subset r$.

The last condition simply says that if two regions overlap then one is strictly contained in the other.

Email addresses: wouter.gelade@uhasselt.be (Wouter Gelade), frank.neven@uhasselt.be (Frank Neven)

[1]We acknowledge the financial support of FWO-G.0821.09N and the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

[2]Research Assistant of the Fund for Scientific Research - Flanders (Belgium)

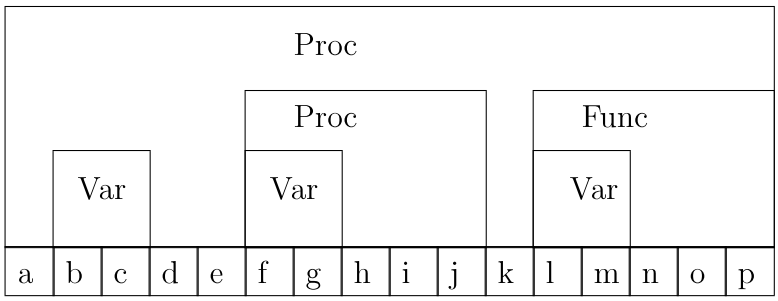

Figure 1: A text.

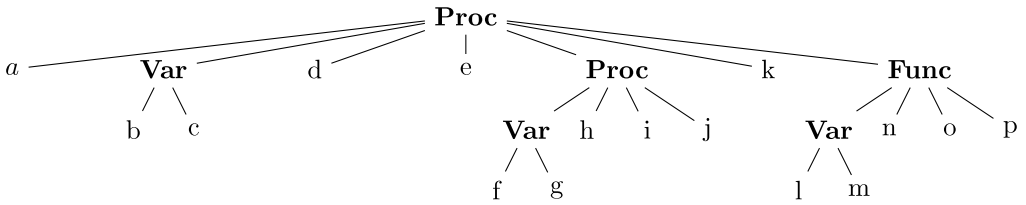

Figure 2: The text tree corresponding to the text in Figure 1.

The text in Figure 1 is hierarchical. Like in [1], we only consider hierarchical texts. The Region Algebra allows to query such hierarchical texts and thereto makes additional use of a *pattern language* $\mathcal{P}$. The latter consists of a set of *patterns* $p \in \mathcal{P}$, each of which defines a set of Σ-strings, denoted $L(p)$. An example of a pattern language is the set of all regular expressions, denoted RE.

We can now define the Region Algebra.

Definition 2.1. *Region Algebra expressions over a pattern language $\mathcal{P}$ and a set of region names $\mathcal{R}$* are inductively defined as follows:

- every region name R in $\mathcal{R}$ is a Region Algebra expression;
- if e_1 and e_2 are Region Algebra expressions then so are $e_1 \cup e_2$, $e_1 - e_2$, $e_1 \subset e_2$, $e_1 < e_2$, $e_1 \supset e_2$, and $e_1 > e_2$; and
- if e is a Region Algebra expression and p in $\mathcal{P}$ is a pattern then $\sigma_p(e)$ is a Region Algebra expression.

The semantics of a Region Algebra expression e on a text $I = (w, \lambda)$, denoted $[\![e]\!]^I$, is defined as follows:

$$
\begin{array}{lcl}
[\![R]\!]^I & := & \lambda(R); \\
[\![\sigma_p(e)]\!]^I & := & \{r \mid r \in [\![e]\!]^I \text{ and } w(r) \in L(p)\}; \\
[\![e_1 \cup e_2]\!]^I & := & [\![e_1]\!]^I \cup [\![e_2]\!]^I; \\
[\![e_1 - e_2]\!]^I & := & [\![e_1]\!]^I - [\![e_2]\!]^I;
\end{array}
$$

and for $\star \in \{<, >, \subset, \supset\}$:

$$
\begin{array}{lcl}
[\![e_1 \star e_2]\!]^I & := & \{r \mid r \in [\![e_1]\!]^I \text{ and } \exists s \in [\![e_2]\!]^I \\
 & & \text{such that } r \star s\}.
\end{array}
$$

We denote by RA($\mathcal{P}$) the set of Region Algebra expressions using only patterns of $\mathcal{P}$. A variety of pattern languages can be used. We focus on the classes RA($\emptyset$), RA(RE), and RA(NFA), where NFA denotes the class of non-deterministic finite automata.

For instance, the RA(RE) expression **Proc** $\supset \sigma_{\Sigma^* \text{start} \Sigma^*}$(**Proc**), defines all the **Proc** regions which contain a **Proc** region that contains the string start.

As all texts are required to be hierarchical, they can be modeled naturally as forests. But as we can always add a unique root, when no unique root is present, we will consider them to be trees. For a text I, we denote its *text tree* by $t(I)$. Figure 2 shows an example of the text tree corresponding to the text illustrated in Figure 1. Note that all leaf nodes are Σ-symbols, while the interior nodes are elements from $\mathcal{R}$. Moreover, as $\lambda(R) \cap \lambda(R')$ must be empty for all distinct region names, every interior node of $t(I)$ which has only non-leaf nodes as children, must have at least two children. Considering only text trees which satisfy this property, there is a one-to-

one correspondence between hierarchical texts and text trees. Therefore, we can restrict attention to text trees instead of hierarchical texts in the subsequent proofs. We usually call them just trees. Furthermore, for a tree t, $[\![e]\!]^t$ equals $[\![e]\!]^I$, where I is the unique text such that $t = t(I)$. We say that an expression e *selects* a tree t when $[\![e]\!]^t \neq \emptyset$, and *outputs* a node u of t when $u \in [\![e]\!]^t$. The set of nodes of t is denoted by Nodes(t).

3. Complexity

We study the complexity of the Region Algebra for the following decision problems.

Definition 3.1. *Let $\mathcal{P}$ be a pattern language.*

- SATISFIABILITY *for RA($\mathcal{P}$): Given an expression e in RA($\mathcal{P}$), does there exist a text I such that $[\![e]\!]^I \neq \emptyset$?*
- EQUIVALENCE *for RA($\mathcal{P}$): Given two expressions e, e' in RA($\mathcal{P}$), is $[\![e]\!]^I = [\![e']\!]^I$, for every text I?*
- INCLUSION *for RA($\mathcal{P}$): Given two expressions e, e' in RA($\mathcal{P}$), is $[\![e]\!]^I \subseteq [\![e']\!]^I$, for every text I?*

The following results are already known.

Theorem 3.2. 1. SATISFIABILITY *for RA($\emptyset$) is NP-hard [1].*
2. SATISFIABILITY, EQUIVALENCE, *and* INCLUSION *for RA(DFA) are in EXPTIME [5].*

We show that all these problems are in fact PSPACE-complete.

Theorem 3.3. 1. SATISFIABILITY, EQUIVALENCE, *and* INCLUSION *for RA($\emptyset$) are PSPACE-hard.*
2. SATISFIABILITY, EQUIVALENCE, *and* INCLUSION *for RA(NFA) are in PSPACE.*

It immediately follows from this theorem that SATISFIABILITY, EQUIVALENCE, and INCLUSION for RA($\emptyset$), RA(DFA), RA(RE), and RA(NFA) are all PSPACE-complete. The remainder of this article is devoted to proving Theorem 3.3.

3.1. Lower Bound

For the lower bound it suffices to show that SATISFIABILITY for RA($\emptyset$) is PSPACE-hard. Indeed, as an expression e is not satisfiable whenever it is equivalent to (respectively, included in) the expression $R - R$ for some R, the PSPACE-hardness then immediately carries over to EQUIVALENCE (respectively, INCLUSION) for RA($\emptyset$). We show that SATISFIABILITY for RA($\emptyset$) is PSPACE-hard by a reduction from quantified boolean formula (QBF), which is PSPACE-complete [6].

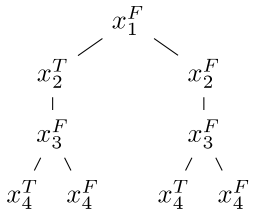

Figure 3: The assignment tree witnessing the satisfiability of the formula $\exists x_1 \forall x_2 \exists x_3 \forall x_4 (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4)$.

In short, QBF is the problem, given a boolean expression ϕ of the form $\exists x_1 \forall x_2 \cdots \exists x_{n-1} \forall x_n \bigwedge_{i=1}^m C_i$, where each C_i is a disjunction of three literals; does there exist a truth value for x_1, such that for all truth values for x_2, ..., and so up to x_n, such that ϕ is satisfied by the overall truth assignment. We then say that ϕ is *satisfiable*.

Note that each satisfiable formula can be witnessed by a tree of the form as shown in Figure 3. Here, every path from the root to a leaf encodes a satisfying truth assignment for $\bigwedge_{i=1}^m C_i$, where x_i^T (respectively, x_i^F) assigns true (respectively, false) to variable x_i. Furthermore, the branching structure of the tree conforms to the quantifier nesting in ϕ. That is, the set of root to leaf paths is precisely the set of variable assignments which need to be checked on $\bigwedge_{i=1}^m C_i$. The tree in Figure 3 witnesses the satisfiability of the formula $\exists x_1 \forall x_2 \exists x_3 \forall x_4 (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4)$. Given a formula ϕ, we construct $\mathcal{R}$ and e_ϕ such that e_ϕ is satisfiable if and only if ϕ is satisfiable. In particular, e_ϕ will select all assignment trees witnessing the satisfiability of ϕ.

Define $\mathcal{R} = \{x_j^T, x_j^F \mid j \in [1, n]\}$. We first define the expression e_{tree} selecting any tree which is *not* of the form depicted in Figure 3. That is, e_{tree} does not consider the actual truth values in the tree but checks its form. Thereto, we write a sequence of expressions each of which selects a tree when it violates one particular property. The formula e_{tree} is simply the union of the given formulas. Here, we abbreviate $\bigcup_{R \in \mathcal{R}} R$, selecting all nodes, by $\mathcal{R}$; $\mathcal{R} - ((\mathcal{R} - e_1) \cup (\mathcal{R} - e_2))$ by $e_1 \cap e_2$; and, for every $i \in [1, n]$, we abbreviate $x_i^T \cup x_i^F$ by x_i.

1. The root is not labeled x_1^T or x_1^F. The expression $\psi_1 = \mathcal{R} \subset \mathcal{R}$ outputs all non-root nodes. Then,

$$(\mathcal{R} - \psi_1) - x_1$$

   outputs the root when it is not labeled x_1^T or x_1^F.
2. For all i, j, with $i \leq j$, a node labeled x_i^T or x_i^F is a

descendant of a node labeled x_j^T or x_j^F:

$$x_i \subset x_j\,.$$

3. For every $i \in [1, \lfloor n/2 \rfloor]$, and $\alpha \in \{T, F\}$, a node labeled x_{2i-1}^α does not have both nodes labeled x_{2i}^T and x_{2i}^F as descendants:

$$x_{2i-1}^\alpha - [(x_{2i-1}^\alpha \supset x_{2i}^T) \cap (x_{2i-1}^\alpha \supset x_{2i}^F)]\,.$$

4. For every $i \in [1, \lfloor n/2 \rfloor - 1]$, and $\alpha \in \{T, F\}$, a node labeled x_{2i}^α does not have a node labeled x_{2i+1}^T or one labeled x_{2i+1}^F as descendant:

$$x_{2i}^\alpha - [(x_{2i}^\alpha \supset x_{2i+1}^T) \cup (x_{2i}^\alpha \supset x_{2i+1}^F)]\,.$$

Now, why is any tree not satisfying any of the above formula of the form as depicted in Figure 3? Because of the first formula, the root of such a tree must be either x_1^T or x_1^F. The third family of formulas guarantees that this root node has both x_2^T and x_2^F as descendants. Because of the second family of expressions, x_2^T and x_2^F must, in fact, be direct children of the root. These, in turn, must by the fourth family of expressions have a child labeled either x_3^T or x_3^F, and so on. Note, however, that this tree might well contain additional side branches not of the form of a proper assignment tree; for instance, there might be several identical copies of one subtree under a node. We are therefore only guaranteed that a subset of the nodes, including the root, forms a proper assignment tree, but the tree might contain additional redundant nodes. More specifically, any tree not selected by e_{tree} contains a proper assignment tree, and any tree exactly of the form of a proper assignment tree, is not selected by e_{tree}. This is sufficient for our purposes.

We next consider the actual truth values occurring in the tree. Thereto, we construct an expression e_{formula} which selects a tree whenever it contains a path from the root to a leaf encoding a truth assignment not satisfying $\bigwedge_i C_i$. If a tree contains such a path it is not an assignment tree witnessing the satisfiability of ϕ.

To construct e_{formula}, let C be any clause of the form $y_{i_1} \vee y_{i_2} \vee y_{i_3}$, with $i_1 < i_2 < i_3$ and where each y_{i_j} either equals x_{i_j} or $\neg x_{i_j}$. Let e_C be the expression $z_3 \subset z_2 \subset z_1$, where $z_j = x_{i_j}^F$ when $y_{i_j} = x_{i_j}$ and $z_j = x_{i_j}^T$ when $y_{i_j} = \neg x_{i_j}$, for $j \in [1, 3]$. Then, $e_{\text{formula}} = \bigcup_{i=1}^n e_{C_i}$.

Hence, the expression $e_{\text{wrong}} = e_{\text{formula}} \cup e_{\text{tree}}$ selects all trees which are or contain an assignment tree which does not witness the satisfiability of ϕ. Consequently, $e_{\text{all}} = e_{\text{wrong}} \cup \bigcup_{\star \in \{<,>,\subset,\supset\}} \mathcal{R} \star e_{\text{wrong}}$ selects the same trees as e_{wrong} and additionally outputs all nodes of these trees. Then, $e_\phi = \mathcal{R} - e_{\text{all}}$ selects those trees not selected by e_{all} which are all trees which contain an assignment tree witnessing the satisfiability of ϕ, and for which every subtree which forms an assignment tree also witnesses the satisfiability of ϕ. Obviously, this includes a normal assignment tree witnessing the satisfiability of ϕ and, hence, e_ϕ is satisfiable if and only if there exists an assignment tree witnessing the satisfiability of ϕ if and only if ϕ is satisfiable.

This concludes the proof for the PSPACE-hardness of SATISFIABILITY for RA($\emptyset$).

3.2. Upper Bound

We show that SATISFIABILITY for RA(NFA) is in PSPACE, as this also implies that INCLUSION and EQUIVALENCE for RA(NFA) are in PSPACE. Indeed, for two Region Algebra expressions e_1 and e_2 it holds that e_1 is included in e_2 (respectively, equivalent to e_2) whenever $e_1 \cap (\mathcal{R} - e_2)$ (respectively, $(e_1 \cap (\mathcal{R} - e_2)) \cup (e_2 \cap (\mathcal{R} - e_1))$) is not satisfiable.

To prove that SATISFIABILITY for RA(NFA) is in PSPACE, we use a crucial property of the Region Algebra proved by Consens and Milo [1].

Theorem 3.4 ([1]). *Let e be a Region Algebra expression. If e is satisfiable, then there exists a text I with $[\![e]\!]^I \neq \emptyset$ such that $t(I)$ has depth at most $2|e|$.*

Here, $|e|$ denotes the size of the expression. This theorem hence allows to restrict our attention to text trees of depth bounded by the size of the expression. Using this property, we solve SATISFIABILITY for RA(NFA) by reducing the problem to emptiness testing of *2-way alternating tree automata (2ATA)*, defined as follows.

For a set of propositions P, let $B(P)$ be the set of Boolean formulas over S, including true and false. A 2ATA A is a tuple (Q, δ, q_0) where Q is the set of states, q_0 is the initial state, and $\delta : Q \times \Sigma \to B(Q \times \{\text{-}, \to, \uparrow, \downarrow\})$ is the transition function. Elements in $\{\text{-}, \to, \uparrow, \downarrow\}$ denote directions in the tree. For a node u in the tree, u - equals u; $u \to$ denotes its right sibling; $u \uparrow$ its parent; and $u \downarrow$ its left most child, when these exist and are undefined otherwise.

Given a tree t and a node v of t, a *run of A on t starting from v* is a pair (s, f) where s is a finite tree in which each node is labeled by an element of $\text{Nodes}(t) \times Q$, and $f : \text{Nodes}(s) \to \{\text{true}, \text{false}\}$ is a function assigning truth values to nodes of s such that

- The label of the root of s is (v, q_0); and
- For every node x of s labeled (u, q), where the label of u in t is a, let $S = \{(d, p) \mid (d, p) \in \delta(q, a)$ such that ud is defined$\}$. Then,

– The set of labels of the children of x in s is exactly $\{(ud, p) \mid (d, p) \in S\}$, and no two children carry the same label; and

– Let S_{true} be the set of those $(d, p) \in S$ for which there exists a child y of x, such that y is labeled (ud, p) and $f(y) = \text{true}$. Then, $f(x) = \text{true}$ if and only if, in $\delta(q, a)$, setting all elements of S_{true} to true, and all others to false, satisfies $\delta(q, a)$.

A run (s, f) is *accepting* when $f(x) = \text{true}$, where x is the root of s. A tree t is *accepted* by A if there exists an accepting run of A on t starting from the root of t.

As mentioned before, we will translate Region Algebra expressions into 2ATA, thus reducing the SATISFIABILITY problem for RA(NFA) to the emptiness problem for 2ATA. In general, however, the emptiness problem for 2ATAs is EXPTIME-complete [7], but when restricted to bounded depth trees the problem becomes easier. Although it is not stated explicitly by Benedikt, Fan, and Geerts, the following theorem is implicit in the proof of Lemma 7.5 [8].

Theorem 3.5 ([8]). *Given a 2ATA A and a number n in unary, the problem of deciding whether A accepts a tree of depth at most n is in PSPACE.*

We next prove the lemma below. Combining this with Theorems 3.4 and 3.5 shows that SATISFIABILITY for RA(NFA) is in PSPACE.

Lemma 3.6. *Let e be a RA(NFA) expression. A 2ATA A_e accepting all trees of depth at most $2|e|$ selected by e, can be constructed in time polynomial in $|e|$.*

Proof. Let e be a RA(NFA) expression. The automaton A_e will be a combination of several automata. In particular, we first construct a 2ATA A_{tree} accepting proper text trees, i.e., trees which (1) are of depth at most $2|e|$, (2) have Σ-symbols as leaf nodes, and $\mathcal{R}$-symbols as interior nodes, and (3) do not have interior nodes which have only one child labeled with a $\mathcal{R}$-symbol. This can easily be done by constructing a 2ATA for each of these three properties and taking the intersection of these automata.

We next construct the 2ATA A_{formula} which, given a proper text tree t of depth at most $2|e|$, checks whether e selects t. Recall that a 2ATA accepts a tree if it has an accepting run starting from the root of the tree. To obtain A_{formula}, we construct, by induction on the structure of e, a 2ATA A which has an accepting run starting from a node u of t, if and only if e outputs this node of t. The automaton A_{formula} visits all nodes of t and runs A from each of these nodes; if A accepts at least one of them, A_{formula} accepts, otherwise, it does not.

Hence, it only remains to construct $A = (Q, \delta, q_0)$ by induction on the structure of e. We only consider the base case $e = R$, and the induction cases $e = e_1 \supset e_2$, $e = e_1 < e_2$, and $e = \sigma_B(e_1)$ and omit the others as they are either very similar or straightforward.

- $e = R$, for some $R \in \mathcal{R}$. Then, $Q = \{q_0\}$, $\delta(q_0, R) = \text{true}$, and $\delta(q_0, a) = \text{false}$, for all $a \in \Sigma \cup \mathcal{R} \setminus \{R\}$.

- $e = e_1 \supset e_2$, where $A_i = (Q_i, \delta_i, q_i)$ is the inductively obtained 2ATA for e_i, with $i = 1, 2$. In a node u, A needs to check whether e_1 outputs u, by running A_1, and whether there is a descendant of u which e_2 outputs. Thereto, let $Q = Q_1 \uplus Q_2 \uplus \{q_0, p\}$. Then, for every $a \in \Sigma \cup \mathcal{R}$, $\delta(q_0, a) = \delta_1(q_1, a) \wedge (p, \downarrow)$, and $\delta(p, a) = \delta(q_2, a) \vee (p, \rightarrow) \vee (p, \downarrow)$. Here, p is the state which visits all descendants of u to check whether one of them is output by e_2.

- $e = e_1 < e_2$, where $A_i = (Q_i, \delta_i, q_i)$ is the inductively obtained 2ATA for e_i, with $i = 1, 2$. In a node u, A needs to check whether e_1 outputs u, by running A_1, and whether there is a node v output by e_2 such that v follows u in the depth first traversal of t, but is not a descendant of u. Let $Q = Q_1 \uplus Q_2 \uplus \{q_0, p_\uparrow, p_\downarrow\}$. Then, for every $a \in \Sigma \cup \mathcal{R}$, $\delta(q_0, a) = \delta_1(q_1, a) \wedge [(p_\downarrow, \rightarrow) \vee (p_\uparrow, \uparrow)]$, $\delta(p_\downarrow, a) = \delta_2(q_2, a) \vee (p_\downarrow, \rightarrow) \vee (p_\downarrow, \downarrow)$, and $\delta(p_\uparrow, a) = (p_\downarrow, \rightarrow) \vee (p_\uparrow, \uparrow)$.

- $e = \sigma_B(e_1)$, where $A_1 = (Q_1, \delta_1, q_1)$ is the 2ATA recursively obtained for e_1 and $B = (Q_2, \delta_2, q_I, F_2)$ is an NFA. In a node u we now have to check whether it is output by e_1 (by running A_1) and whether the Σ-string below it is accepted by B. To check whether the latter holds we do a depth first traversal, starting from u, and run B over the string encountered at the leafs in this traversal (this is exactly the desired string). Here it is crucial to recognize the moment at which the depth first traversal is finished, i.e., when we arrive back at u. Thereto, when visiting a node v, we also keep track of the current depth with respect u, or, equivalently, the length of the path from u to v. Hence, once the depth is 0, we have finished the traversal and can check whether B accepts the string it has read. Here, we use the fact that we are only considering trees of depth at most $n = 2|e|$, which allows to encode the current depth in the states.

Formally, $Q = Q_1 \uplus \{q^i_\downarrow, q^i_\uparrow \mid i \in [0,n], q \in Q_2\} \uplus \{q_0, q_{\text{true}}\}$. Then:

- For every $a \in \Sigma \cup \mathcal{R}$,
  - $\delta(q_{\text{true}}, a) = \text{true}$, and $\delta(q_0, a) = \delta_1(q_1, a) \wedge (q^1_{I\downarrow}, \downarrow)$;
  - for all $q \in Q_1$, $\delta(q, a) = \delta_1(q, a)$;
  - for all $q \in F_2$, $\delta(q^0_\downarrow, a) = \delta(q^0_\uparrow, a) = \text{true}$; and
  - for all $q \in Q_2 \setminus F_2$, $\delta(q^0_\downarrow, a) = \delta(q^0_\uparrow, a) = \text{false}$.
- For every $q \in Q_2$, $i \in [1,n]$ and $R \in \mathcal{R}$,
  - $\delta(q^i_\downarrow, R) = (q^{i+1}_\downarrow, \downarrow)$ when $i < n$; and
  - $\delta(q^i_\uparrow, R) = (q^i_\downarrow, \rightarrow) \vee [\neg(q_{\text{true}}, \rightarrow) \wedge (q^{i-1}_\uparrow, \uparrow)]$, where $\neg(q_{\text{true}}, \rightarrow)$ is true in a node which does not have a right sibling.
- For every $q \in Q_2$, $i \in [1,n]$ and $b \in \Sigma$,
  - $\delta(q^i_\downarrow, b) = [\neg(q_{\text{true}}, \rightarrow) \wedge (p^{i-1}_\uparrow, \uparrow)] \vee \bigvee_{p \in \delta(q,b)} \delta(p^i_\downarrow, \rightarrow)$; and
  - $\delta(q^i_\uparrow, b) = \text{false}$.

□

4. Conclusion

The present paper pins down the theoretical complexity of optimization problems related to the region algebra. It remains open whether there are interesting fragments for which these problems become tractable. Of course, equivalence testing is just one of the possibilities for optimization. Other optimization procedures are known which avoid equivalence testing (e.g., [9, 10]).

References

[1] M. P. Consens, T. Milo, Algebras for querying text regions: Expressive power and optimization, J. Comput. Syst. Sci. 57 (3) (1998) 272–288.

[2] A. Salminen, F. Tompa, PAT expressions: an algebra for text search, in: Papers in Computational Lexicography (COMPLEX 92), 1992, pp. 309–332.

[3] G. Gonnet, Examples of PAT applied to the oxford english dictionary, Tech. Rep. OED-87-02, University of Waterloo (1987).

[4] E. Börger, E. Grädel, Y. Gurevich, The Classical Decision Problem, Perspectives in Mathematical Logic, Springer, 1997.

[5] F. Neven, Attribute grammars for unranked trees as a query language for structured documents, J. Comput. Syst. Sci. 70 (2) (2005) 221–257.

[6] L. J. Stockmeyer, A. R. Meyer, Word problems requiring exponential time: Preliminary report, in: STOC, 1973, pp. 1–9.

[7] M. Y. Vardi, Reasoning about the past with two-way automata, in: ICALP, 1998, pp. 628–641.

[8] M. Benedikt, W. Fan, F. Geerts, XPath satisfiability in the presence of DTDs, J. ACM 55 (2).

[9] M. Young-Lai, F. W. Tompa, One-pass evaluation of region algebra expressions, Inf. Syst. 28 (3) (2003) 159–168.

[10] J. A. List, V. Mihajlovic, G. Ramírez, A. P. de Vries, D. Hiemstra, H. E. Blok, Tijah: Embracing ir methods in xml databases, Inf. Retr. 8 (4) (2005) 547–570.