Containment of Nested Regular Expressions

Juan Reutter

Abstract

Nested regular expressions (NREs) have been proposed as a powerful formalism for querying RDFS graphs, but research in a more general graph database context has been scarce, and static analysis results are currently lacking. In this paper we investigate the problem of containment of NREs, and show that it can be solved in PSPACE, i.e., the same complexity as the problem of containment of regular expressions or regular path queries (RPQs).

1 Introduction

Graph-structured data has become pervasive in data-centric applications. Social networks, bioinformatics, astronomical databases, digital libraries, Semantic Web, and linked government data, are only a few examples of applications in which structuring data as graphs is, simply, essential.

Traditional relational query languages do not appropriately cope with the querying problematics raised by graph-structured data. The reason for this is twofold. First, in the context of graph databases one is typically interested in navigational queries, i.e. queries that traverse the edges of the graph while checking for the existence of paths satisfying certain conditions. However, most relational query languages, such as SQL, are not designed to deal with this kind of recursive queries. Second, current graph database applications tend to be massive in size (think, for instance, of social networks or astronomical databases, that may store terabytes of information). Thus, one can immediately dismiss any query language that cannot be evaluated in polynomial time (or even in linear time!). But then even the core of the usual relational query languages – conjunctive queries (CQs) – does not satisfy this property. In fact, parameterized complexity analysis tells us that – under widely-held complexity theoretical analysis – CQs over graph databases cannot be evaluated in time $0(|G|^c \cdot f(|\varphi|))$, where $c \geq 1$ is a constant and $f : \mathbb{N} \to \mathbb{N}$ is a computable function.

This raises a need for languages that are specific for the graph database context. The most commonly used core of these languages are the so-called regular path queries, or RPQs, that specify the existence of paths between nodes, with the restriction that the labels of such path belong to a regular language. The language of RPQs was later extended with the ability to traverse edge backwards, providing them with a 2-way functionality. This gives rise to the notion of 2RPQs.

Nested regular expressions are a graph database language that aims to extend the possibility of using regular expressions, or 2-way regular expressions, for querying graphs with an existential test operator $[\cdot]$, also known as nesting operator, similar to the one in XPath. This class of expressions was proposed for querying Semantic Web data, and have received a fair deal of attention in the last years.

We say that Here we study the problem of containment of NREs, which is the following problem:

| Problem: | NREContainment |
|----------|----------------|
| Input:   | NREs $Q_1$ and $Q_2$ over $\Sigma$. |
| Question:| Is $Q_1 \subseteq Q_2$? |

Note that we study this problem for the restricted case when all the possible input graphs are semipaths. The general case will be shown in an extended version of the manuscript.
2 Preliminaries

2.1 Graph Database and queries

Graph databases. Let \( V \) be a countably infinite set of node ids, and \( \Sigma \) a finite alphabet. A graph database \( G \) over \( \Sigma \) is a pair \((V, E)\), where \( V \) is a finite set of node ids (that is \( V \) is a finite subset of \( V \)) and \( E \subseteq V \times \Sigma \times V \). That is, \( G \) is an edge-labeled directed graph, where the fact that \((u, a, v)\) belongs to \( E \) means that there is an edge from node \( u \) into node \( v \) labeled \( a \). For a graph database \( G = (V, E) \), we write \((u, a, v) \in G\) whenever \((u, a, v) \in E\).

Nested Regular Expressions.

The language of nested regular expressions (NREs) were first proposed in [15] for querying Semantic Web data. Next we formalize the language of nested regular expressions in the context of graph databases.

Let \( \Sigma \) be a finite alphabet. The NREs over \( \Sigma \) extend classical regular expressions with an existential nesting test operator \([\cdot]\) (or just nesting operator, for short), and an inverse operator \( a^-\), over each \( a \in \Sigma \). The syntax of NREs is given by the following grammar:

\[
R ::= \varepsilon \mid a (a \in \Sigma) \mid a^- (a \in \Sigma) \mid R \cdot R \\
R^* \mid R + R \mid [R]
\]

As it is customary, we use \( n^+ \) as shortcut for \( n \cdot n^* \).

Intuitively, NREs specify pairs of node ids in a graph database, subject to the existence of a path satisfying a certain regular condition among them. That is, each NRE \( R \) defines a binary relation \([R]_G\) when evaluated over a graph database \( G \). This binary relation is defined inductively as follows, where we assume that \( a \) is a symbol in \( \Sigma \), and \( n, n_1 \) and \( n_2 \) are arbitrary NREs:

\[
\begin{align*}
[\varepsilon]_G &= \{(u, u) \mid u \text{ is a node id in } G\} \\
[a]_G &= \{(u, v) \mid (u, a, v) \in G\} \\
[a^-]_G &= \{(u, v) \mid (v, a, u) \in G\} \\
[n_1 \cdot n_2]_G &= [n_1]_G \circ [n_2]_G \\
[n_1 + n_2]_G &= [n_1]_G \cup [n_2]_G \\
[n^*]_G &= [\varepsilon]_G \cup [n]_G \cup [n \cdot n]_G \cup [n \cdot n \cdot n]_G \cup \cdots \\
[\llbracket n \rrbracket]_G &= \{(u, u) \mid \text{there exists } v \text{ s.t. } (u, v) \in [n]_G\}.
\end{align*}
\]

Here, the symbol \( \circ \) denotes the usual composition of binary relations, that is, \([n_1]_G \circ [n_2]_G = \{(u, v) \mid \text{there exists } w \text{ s.t. } (u, w) \in [n_1]_G \text{ and } (w, v) \in [n_2]_G\} \).
Example 2.1 Let $G_1$ be the graph database in Figure 4. The following is a simple NRE that matches all pairs $(x, y)$ such that $x$ is an author that published a paper in conference $y$:

$$n_1 = \text{creator}^- \cdot \text{partOf} \cdot \text{series}$$

For example the pairs (\texttt{:Jeffrey.D._Ullman,conf:focs}) and (\texttt{:Ronald.Fagin,conf:pods}) are in $[n_1]_G$. Consider now the following expression that matches pairs $(x, y)$ such that $x$ and $y$ are connected by a coauthorship sequence:

$$n_2 = (\text{creator}^- \cdot \text{creator})^+$$

For example the pair (\texttt{:John.E._Hopcroft,:Pierre_Wolper}), is in $[n_2]_G$. Finally the following expression matches all pairs $(x, y)$ such that $x$ and $y$ are connected by a coauthorship sequence that only considers conference papers:

$$n_3 = (\text{creator}^- \cdot [\text{partOf} \cdot \text{series}] \cdot \text{creator})^+$$

Let us give the intuition of the evaluation of this expression. Assume that we start at node $u$. The (inverse) edge $\text{creator}^-$ makes us to navigate from $u$ to a paper $v$ created by $u$. Then the existential test $[\text{partOf} \cdot \text{series}]$ is used to check that from $v$ we can navigate to a conference (and thus, $v$ is a conference paper). Finally, we follow edge $\text{creator}$ from $v$ to an author $w$ of $v$. The $(\cdot)^+$ over the expression allows us to repeat this sequence several times. For instance, (\texttt{:John.E._Hopcroft,:Moshe.Y._Vardi}) is in $[n_3]_G$, but (\texttt{:John.E._Hopcroft,:Pierre_Wolper}) is not in $[n_3]_G$.

Complexity and expressiveness of NREs The following result, proved in [15], shows a remarkable property of NREs. It states that the query evaluation problem for NREs is not only polynomial in combined complexity (i.e. when both the database and the query are given as input), but also that it can be solved linearly in both the size of the database and the expression. Given a graph database $G$ and an NRE $R$, we use $|G|$ to denote the size of $G$ (in terms of the number of egdes $(u, a, v) \in G$), and $|R|$ to denote the size of $R$.

Proposition 2.2 (from [15]). Checking, given a graph database $G$, a pair of nodes $(u, v)$, and an NRE $R$, whether $(u, v) \in [R]_G$, can be done in time $O(|G| \cdot |R|)$.

On the expressiveness side, NREs subsume several important query languages for graph databases. For instance, by disallowing the inverse operator $a^-$ and the nesting operator $[\cdot]$ we obtain the class of regular path queries (RPQs) [7, 13], while by only disallowing the nesting operator $[\cdot]$ we obtain the class of RPQs with inverse or 2RPQs [5]. (In particular, both expressions $n_1$ and $n_2$ in Example 2.1 are 2RPQs). In turn, NREs allow for an important increase in expressive power over those languages. For example, it can be shown that NRE expression $n_3$ in Example 2.1 cannot be expressed without the nesting operator $[\cdot]$, and hence it is not expressible in the language of 2RPQs (c.f. [15]).

On the other hand, the class of NREs fails capturing more expressive languages for graph-structured data that combine navigational properties with quantification over node ids. Some of the most paradigmatic examples of such languages are the classes of conjunctive RPQs and 2RPQs, that close RPQs and 2RPQs, respectively, under conjunctions and existential quantification. Both classes of queries have been studied in depth, as they allow identifying complex patterns over graph-structured data [6, 9, 14].

3 Containment of NREs over paths

Problem Definition. It is convenient for the proof to explain first how NRE’s are used to represent regular languages over $\Sigma$, and how to represent these languages using alternating two way automata. Let us begin with some notation.

Along the proof we assume that $\Sigma$ includes all reverse symbols. More precisely, if $\Sigma'$ is an alphabet, we work instead with the alphabet $\Sigma = \Sigma' \cup \{a^- \mid a \in \Sigma'\}$. Let $G = (V, E)$ be a graph over $\Sigma$. A semipath
in $G$ is a sequence $u_1, a_1, u_2, a_2, \ldots, u_m, a_m, u_{m+1}$, where each $u_i$ belongs to $V$, each $a_i$ belongs to $\Sigma$, and for each $u_i, a_i, u_{i+1}$, we have that $(u_i, a_i, u_{i+1})$ belongs to $E$, if $a_i$ is not a reverse symbol, and $(u_{i+1}, a_i, u_i)$ belongs to $E$ is $a_i$ is a reverse symbol, i.e., of form $a^-$ for some $a \in \Sigma'$. A semipath is simple if all of its nodes are distinct. Finally, a graph $G$ resembles a (simple) semipath if there is a (simple) semipath $\pi$ in $G$ of the form above such that the nodes of $G$ are precisely $\{u_1, \ldots, u_n\}$ and the edges of edges of $G$ are precisely those that witness the above definition.

As we have mentioned, we study $\text{NREContainment}$ only when the input graphs are semipaths. We are now ready to describe our goal which is to show that the following problem is in $\text{PSPACE}$: Given NREs $Q_1$ and $Q_2$ over $\Sigma$, decide whether $[Q_1]_G \subseteq [Q_2]_G$, for all graphs $G$ over $\Sigma$ such that $G$ resembles a simple semipath. In what follows, we refer to this problem as $\text{SP-NREContainment}$.

### 3.1 Alternating 2-way finite automata

Following \cite{12}, an Alternating 2-way finite automaton, or A2FA for short, is a tuple $A = (Q, q_0, U, F, \Sigma, \delta)$, where $Q$ is the set of states, $U \subseteq Q$ is a set of universal states, $q_0$ is the initial state, $F \subseteq Q$ is the set of final states, $\Sigma$ is the input alphabet (we also use symbols $\%$ and $\&$ not in $\Sigma$ as the start and end markers of the string), and the transition function is $\delta : Q \times (\Sigma \cup \{\%, \&\}) \to 2^{Q \times \{-1, 0, 1\}}$.

The numbers $-1, 0, 1$ in the transition stand for moving back, staying and moving forward, respectively. The input is delimited with $\%$ at the beginning and $\&$ at the end. For convenience, we assume that the automaton starts in state $q_0$ while reading the symbol $\&$ of the string.

**Semantics** Semantics are given in terms of computation trees over instantaneous descriptions. An instantaneous description (ID) is a triple of form $(q, w, i)$, where $q$ is a state, $w$ is a word in $\%\sigma^*(\varepsilon \parallel \&)$ and $1 \leq i \leq |w| + 1$. Intuitively, it represents the state of the current computation, the string it has already read, and the current position of the automata. An ID is universal if $q \in U$ and existential otherwise, and accepting IDs are of form $(q, w, |w| + 1)$ for $w \in \%\Sigma^*\&$ and $q \in F$.

Let $w = a_1, \ldots, a_n$, for each $a_i \in \Sigma \cup \{\%, \&\}$. The transition relation $\Rightarrow$ is defined as follows:

- $(q, w, i) \Rightarrow (p, w, i)$, if $(p, 0) \in \delta(q, a_i)$ and $1 \leq i \leq n$;
- $(q, w, i) \Rightarrow (p, w, i + 1)$, if $(p, 1) \in \delta(q, a_i)$ and $1 \leq i \leq n$; and
- $(q, w, i) \Rightarrow (p, w, i - 1)$, if $(p, -1) \in \delta(q, a_i)$ and $1 < i \leq n$.

A computation tree $\Pi$ of an A2FA $A = (Q, q_0, U, F, \Sigma, \delta)$ is a finite, nonempty tree with each of its nodes $\pi$ labelled with an ID $l(\pi)$, and such that

1. If $\pi$ is a non-leaf node and $l(\pi)$ is universal, let $I_1, \ldots, I_k$ be all IDs such that $l(\pi) \Rightarrow I_j$ for each $1 \leq j \leq k$. Then $\pi$ has exactly $k$ children $\pi_1, \ldots, \pi_k$, where $l(\pi_j) = I_j$; and
2. If $\pi$ is a non leaf node and $l(\pi)$ is existential, then $\pi$ has exactly one child $\pi'$ such that $l(\pi) \Rightarrow l(\pi')$.

Finally, an accepting computation tree of $A$ over $w$ is a computation tree $\Pi$ whose root is labelled with $(q_0, \%w\&\parallel |w| + 2)$ and each of it leaves are labelled with an accepting ID.

We need the following theorem. It follows immediately from the results in \cite{12}:

**Proposition 3.1** Given a A2FA $A$, it is $\text{PSPACE}$-complete to decide whether the language of $A$ is empty.

### 3.2 Proof of SP-NREContainment

The idea is to code acceptance of strings by NREs using alternating 2-way automata. More precisely, given an NRE $R$, we construct an A2FA $A_R$ such that the language of $A_R$ corresponds, in a precise sense, to all those words $w$ such that $[R]_G$ is nonempty for all those graphs $G$ that resemble the simple semipath $w$.

**Construction of $A_R$.** We define the translation by induction, all states are existential unless otherwise noted. Along the construction, we shall be marking, in each step, a particular state of the automata. We use
this mark in the construction. Furthermore, for the sake of readability we shall include \( \varepsilon \)-transitions. This is without loss of generality, as they can be easily simulated with two transitions (and an extra state), the first moving forward, and the second backwards.

- If \( R = a \), then \( A_R = (\{q_0, q_f, q_r\}, \emptyset, q_0, q_f, \Sigma, \delta) \), with \( \delta \) defined as:
  
  \[
  \begin{align*}
  \delta(q_0, a) &= \{(q_f, 1), (q_r, -1)\} \\
  \delta(q_0, b) &= \{(q_r, -1)\}, \text{ for each } b \in \Sigma, \; b \neq a \\
  \delta(q_r, a^-) &= \{(q_f, 0)\}
  \end{align*}
  \]

  State \( q_r \) and the two way functionality is added so that the automaton correctly accepts when the input is a word of form \( \Sigma^*a^-\Sigma^* \) (See [5] for a thorough explanation of this machinery). Moreover, state \( q_f \) is marked.

- Similarly, if \( R = a^- \), then \( A_R = (\{q_0, q_f, q_r\}, \emptyset, q_0, q_f, \Sigma, \delta) \), with \( \delta \) defined as:
  
  \[
  \begin{align*}
  \delta(q_0, a^-) &= \{(q_f, 1), (q_r, -1)\} \\
  \delta(q_0, b) &= \{(q_r, -1)\}, \text{ for each } b \in \Sigma, \; b \neq a^- \\
  \delta(q_r, a) &= \{(q_f, 0)\}
  \end{align*}
  \]

  State \( q_f \) is marked.

- Case when \( R = R_1 + R_2 \). Let \( A_{R_i} = (Q_i, U_i, q_0^i, F_i, \Sigma, \delta^i) \), for \( i = 1, 2 \), and assume that \( q_m^i \) is the marked state from \( A_{R_i} \). Define \( A_R = (Q, U, q_0, F, \Sigma, \delta) \), where \( Q = \{q_0, q_f\} \cup Q^1 \cup Q^2 \), \( U = U^1 \cup U^2 \), \( F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \cup (F^2 \setminus \{q_m^2\}) \) and \( \delta = \delta^1 \cup \delta^2 \), plus transitions
  
  \[
  \begin{align*}
  \delta(q_0, \varepsilon) &= \{(q_0^1, 0), (q_0^2, 0)\} \\
  \delta(q_m^1, \varepsilon) &= \{(q_f, 0)\} \\
  \delta(q_m^2, \varepsilon) &= \{(q_f, 0)\}
  \end{align*}
  \]

  For each \( i = 1, 2 \), remove al marks from \( A_{R_i} \), and mark state \( q_f \).

- In the case that \( R = R_1 \cdot R_2 \), let \( A_{R_i} = (Q_i, U_i, q_0^i, F_i, \Sigma, \delta^i) \), for \( i = 1, 2 \), and assume that \( q_m^i \) is the marked state from \( A_{R_i} \). For each \( i = 1, 2 \), remove all marks from \( A_{R_i} \). Define \( A_R = (Q, U, q_0, F, \Sigma, \delta) \), where \( Q = \{q_0, q_f\} \cup Q^1 \cup Q^2 \), \( U = U^1 \cup U^2 \), \( F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \cup (F^2 \setminus \{q_m^2\}) \) and \( \delta = \delta^1 \cup \delta^2 \), plus transitions
  
  \[
  \begin{align*}
  \delta(q_0, \varepsilon) &= \{(q_0^1, 0)\} \\
  \delta(q_m^1, \varepsilon) &= \{(q_0^2, 0)\} \\
  \delta(q_m^2, \varepsilon) &= \{(q_f, 0)\}
  \end{align*}
  \]

  For each \( i = 1, 2 \), remove al marks from \( A_{R_i} \), and mark state \( q_f \).

- For \( R = R_1^* \), let \( A_{R_i} = (Q_i, U_i, q_0^i, F_i, \Sigma, \delta^i) \), and assume that \( q_m^i \) is the marked state from \( A_{R_i} \). Define \( A_R = (Q, U, q_0, F, \Sigma, \delta) \), where \( Q = \{q_0, q_f\} \cup Q^1 \), \( F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \) and \( \delta = \delta^1 \) plus transitions
  
  \[
  \begin{align*}
  \delta(q_0, \varepsilon) &= \{(q_0^1, 0)\} \\
  \delta(q_m^1, \varepsilon) &= \{(q_f, 0)\} \\
  \delta(q_m^2, \varepsilon) &= \{(q_f, 0), (q_0^1, 0)\}
  \end{align*}
  \]

  Remove al marks from \( A_{R_i} \), and mark state \( q_f \).
Proof:
Let $A_{R_1} = (Q, U^1, q_0^1, F^1, \Sigma, \delta^1)$ be as constructed by this algorithm. To finish our construction we need to allow $A_R$ to (non-deterministically) move backwards from the end of the word, until it reaches a suitable starting point for the computation, and allow every final state to reach the end of the word in its computation. Formally, we define $A'_R = (Q \cup \{q_0^1\}, q_0^1, U, F, \Sigma \cup \{\epsilon\}, \delta')$, where $\delta'$ contains all transitions in $\delta$ plus transitions $\delta(q_0^1, a) = \{(q_0^1, 0)\}$ for each $a \in \Sigma$. Notice that the above construction can be computed in polynomial time with respect to $R$. Furthermore, let $q_m$ be the marked (final) state of $A'_R$. From its construction, it is clear that every accepting computation tree $\Pi$ of $A_R$ on input $w$ will have the following form: (1) For some $1 \leq i \leq |w|$ there is a single path from the root to a node $\pi_s$ such that $l(\pi) = (q_0, w, i)$ and no ancestor of $\pi$ is labelled with an ID using a state different from $q_0^1$; and (2) there is some $1 \leq j \leq |w|$ such that $\pi_f, \pi'_f, \pi''_f, \ldots$ is the maximal path of nodes labelled with $(q_m, w, j), (q_m, w, j+1), \ldots, (q_m, w, |w| + 1)$, i.e., the father of $(q_m, w, j)$ is not labelled with an ID using state $q_m$. Property (1) represents the automaton searching for its starting point, and (2) represents the end of the computation of the part of $A'_R$ that is representing the non-nesting part of $R$. We denote such nodes $\pi_s$ and $\pi_f$ as the tacit start and tacit ending of $\Pi$. With this definitions we can show the following.

Lemma 3.2 Let $S$ be a graph over $\Sigma$ that is a semipath, $w$ the label of the path $S$, and $R$ a NRE. Then a pair $(u_i, u_j)$ belongs to $[R]_S$ if and only if there is an accepting computation tree of $A'_R$ on input $w$ whose tacit start is labelled with $(q_0, w, i)$ and whose tacit ending is labelled with $(q_m, w, j)$.

Proof: Let $S$ be the semipath $u_1, a_1, u_2, a_2, \ldots, u_m, a_m, u_{m+1}$, and therefore $w = a_1 \cdots a_m$, and let $A'_R = (\Sigma, Q, U, q_0^1, \delta', F)$ constructed as explained above.

For the only if direction, assume that $[R]_S$ contains the pair $(u_i, u_j)$, $1 \leq i, j \leq m + 1$. We prove the above statement by induction on $R$.

- If $R = a$, for some $a \in \Sigma$, and $(u_i, u_j) \in [R]_S$, then either $j = i + 1$ and the edge $(u_i, a, u_j)$ is in $S$, or $j = i - 1$ and the edge $(u_j, a^-, u_i)$ is in $S$. In the former case the existence of a computation tree is obvious, for the latter case observe that one could use the transitions $(q_0, w, i) \Rightarrow (q_r, w, i - 1)$, and then $(u_j, a^-, u_i)$ is in $S$ we follow transition $(q_r, w, i - 1) \Rightarrow (q_f, w, i - 1)$.
- Case for $R = a^-$ is analogous to the previous one
- If $R = R_1 + R_2$ and $(u_i, u_j) \in [R]_S$, then $(u_i, u_j) \in [R_k]_S$ for $k = 1$ or $k = 2$, which entails a proper accepting computation tree for $A_{R_{1k}}$ ($A_{R_{2k}}$) on input $w$. The statement follows immediately from the construction of $A_R$.
- If $R = R_1 \cdot R_2$ and $(u_i, u_j) \in [R]_S$, then there is a node $u_k$ of $S$ such that $(u_i, u_k) \in [R_1]_S$ and $(u_k, u_j) \in [R_2]_S$. Assume that the initial and marked nodes of $A_{R_1}$ and $A_{R_2}$ are $q_0^1, q_m^1$ and $q_0^2, q_m^2$, respectively. From the induction hypothesis we have that there are accepting computation trees for $A_{R_1}$ and $A_{R_2}$ whose tacit starts are $(q_0^1, w, i)$ and $(q_0^2, w, k)$, respectively, and the tacit ending of the first tree is labelled with $(q_m^1, w, k)$. Since $A'_R$ has, by construction, the pair $(q_0^1, 0)$ in $\delta(q_m^1, \epsilon)$, we can cut the first tree in its tacit ending and plug in the computation tree for $A_{R_2}$, starting from its tacit start, which proves the statement.
- The case when $R = R_1^*$ goes along the same lines as the concatenation, except this time we may have to plug in a greater number of computation trees.
Finally, if \( R = [R_1] \) and \((u_i, u_j) \in [R]_S \), then \( u_i = u_j \), and there is some \( u_k \) such that \((u_i, u_k) \in [R_1]_s \). Let \( q_p \) be the universal state in \( A'_R \) that is not in \( A_R \). Then the only transitions associated to \( q_p \) are \( \delta(q_p, \varepsilon) = \{(q_0^1, 0), (q_f, 0)\} \), with \( q_f \) being the only marked (final) state of \( A_R \). Our accepting computation tree for \( A'_R \) has a path from the root to the tacit start, then a node labeled \((q_p, w, i)\) with children \((q_0^1, w, i)\) and \((q_f, w, i)\), with the computation tree for \( A_R \) (starting from its tacit start) plugged into the first of these children.

For the **if direction**, assume that there is an accepting computation tree of \( A_R \) on input \( w \) whose tacit start is labelled with \((q_0, w, i)\) and with its tacit ending labelled with \((q_m, w, j)\). We now prove that \((u_i, u_j)\) belong to \([R]_S \). The proof is again by induction

- For the base case when \( R = a \) (proof for \( R = a^- \) is analogous), there are two options for an accepting computation of \( A_R \). Either it is of form \((q_0, w, i) \Rightarrow (q_f, w, i + 1)\), in which case \( j = i + 1 \) and \( a_i = a \), or it is of form \((q_0, w, i) \Rightarrow (q_f, w, i - 1)\) in which case \( j = i - 1 \) and \( a_i = a^-\). For both cases we obtain that \((u_i, u_j)\) belong to \([R]_S \). The proof is again by induction.

- When \( R = R_1 + R_2 \), by the construction of \( A_R \), any computation tree of \( A_R \) can be pruned from its tacit start to obtain a computation tree for one of \( A_{R_1} \) or \( A_{R_2} \), from where the statement easily follows.

- When \( R = R_1 \cdot R_2 \), we can similarly obtain computation trees for \( A_{R_1} \) and \( A_{R_2} \), and then conclude that \((u_i, u_j)\) belong to \([R]_S \). Same hold when \( R = R_1^* \), except in this case we obtain multiple computation trees for \( A_1 \).

- Finally, if \( R = [R_1] \) and there is an accepting computation tree of \( A_R \) on input \( w \) whose tacit start is labelled with \((q_0, w, i)\) and with its tacit ending labelled with \((q_m, w, j)\), from the construction of \( A_R \) the top part of the computation tree is of form \((q_0, w, i) \Rightarrow (q_p, w, i) \Rightarrow (q_0^1, w, i), (q_m, w, i)\), where \( q_p \) is the only universal state of \( A_R \) not in \( A_{R_1} \), and \( q_0^1 \) is the initial state of \( A_{R_1} \). Then the part of the computation tree that follows from node \((q_0^1, w, i)\) comprises a computation tree for \( A_{R_1} \), i.e., there is a \( u_k \) such that \((u_i, u_k) \in [R_1]_S \). This entails that \((u_i, u_i)\) belong to \([R]_S \).

**Proof for containment** For our algorithm of containment, we need to be a little more careful, since for a word \( w \) accepted by \( A_R \) it is not necessarily the case that \( u_i \) and \( u_j \) are the start and finish nodes of the semipath \( S \). Thus, we have to distinguish the start/end of the word with the actual piece that is framed by nodes \( u_i \) and \( u_j \) in the semipath. In order to do that, we augment \( \Sigma \) with two extra symbols \( S, E \). Furthermore, if \( A_R = (\Sigma, Q, U, q_0, \delta, F) \), and \( q_m \) is the marked state of \( A_R \), we construct \( A_{R,S,E} = (\Sigma \cup \{S, E\}, Q \cup \{q_0^S, q_f^S\}, U, q_0^S, \delta_{S,E}(F \setminus \{q_m\}) \cup \{q_f^E\}) \), where \( \delta_{S,E} \) is defined as follows: for each state \( q \in Q \setminus U \), we add the pair \((q, 1) \to \delta(q, S) \) and \( \delta(q, E) \), if \( q \neq q_0 \) or \( q_m \), the pair \((q_f^E, 1) \to \delta(q, E) \), \((q_0, 1) \to \delta(q_0^S, S) \), plus the pair \((q_0^S, -1) \to \delta(q_0^S, a) \) for \( a \in \Sigma \cup \{E\} \) and \((q_f^E, 1) \to \delta(q_f^E, a) \) for \( a \in \Sigma \).

The intuition is the following. Let \( R \) be an NRE and \( A_R \) be the A2FA constructed as above. Now assume that there is a semipath \( w = u_1, a_1, u_2, \ldots, u_n, a_n, u_{n+1} \) and nodes \( u_i, u_j \) such that \((u_i, u_j) \in \([R]_S \). By the above Lemma, we have that there is a computation tree for \( A_R \) that tacitly starts in \((q_0, w, i)\) and tacitly ends in \((q_m, w, j)\). The idea of the symbols \( S \) and \( E \) is to specifically mark the tacit start and end of the piece \( a_1, \ldots, a_{j-1} \) labeling the semipath between \( u_i \) and \( u_j \). Thus, in this case, \( A_{R,S,E} \) accepts the word \( a_1 \cdots a_{i-1} S a_i \cdots a_{j-1} E a_j \cdots a_n \). It uses intuitively the same computation tree mentioned before, except now it moves backwards in state \( q_0^S \) until symbol \( S \) is reached, then proceeds with the computation, and the marked branch now ends in \( q_f^E \) instead of \( Q_m \), after checking there is a symbol \( E \) after \( a_{j-1} \). With this intuition, it is straightforward to show:

**Lemma 3.3** Let \( w = u_1, a_1, u_2, \ldots, u_n, a_n, u_{n+1} \) be a graph over \( \Sigma \) that is a simple semipath, \( w = a_1, \ldots, a_n \) the label of the path \( w \), and \( R \) a NRE. Then a pair \((u_i, u_j)\) belongs to \([R]_S \) if and only if \( A_{R,S,E} \) accepts the word \( a_1 \cdots a_{i-1} S a_i \cdots a_{j-1} E a_j \cdots a_n \)
We can now state our algorithm for solving SP-QUERYCONTAINMENT. On input NREs $R_1$ and $R_2$, we perform the following operations:

1. Compute an NFA $A^{S,E}$ that accepts only those words over $(\Sigma \cup \{S, E\})^*$ of form $w_1S w_2E w_3$, for each $w_1, w_2, w_3$ in $\Sigma^*$. 
2. Compute $A^{S,E}_{R_1}$ and $A^{S,E}_{R_2}$ as explained above. 
3. Compute the A2FA $A^c = (A^{S,E}_{R_2})^c$ whose language is the complement of $A^{S,E}_{R_2}$ 
4. Compute the A2FA $A$ whose language is the intersection of the languages $A^{S,E}, A^{S,E}_{R_1}$ and $A^c$. 
5. Check that the language of $A$ is empty.

We have seen how to perform the second step in polynomial time, and steps (1), (3), (4) can be easily performed in PTIME using standard techniques from automata theory. Finally, Proposition 3.4 shows that step (5) can be performed in PSPACE. Thus, all that is left to prove is that the language of the resulting automata $A$ is empty if and only if $R_1 \subseteq R_2$.

Assume first that $R_1 \subseteq R_2$, and assume for the sake of contradiction that there is a word $w \in L(A)$. We have that $w$ must be of form $a_1 \cdots a_i - 1 Sa_i \cdots a_j - 1 Ea_j \cdots a_n$, and $w$ is accepted by $A^{S,E}_{R_1}$, but not by $A^{S,E}_{R_2}$. Let $S$ be a graph consisting of the semipath $u_1, a_1, u_2, \ldots, u_n, a_n, u_{n+1}$. By Lemma 3.3, nodes $(u_i, u_j)$ must belong to $[R_2]_S$, but this would imply, again by the lemma, that $w$ is accepted by $A^{S,E}_{R_2}$.

On the other hand if $L(A)$ is empty but $R_1 \not\subseteq R_2$, then for some graph $S = u_1, a_1, u_2, \ldots, u_n, a_n, u_{n+1}$ that is a semipath and nodes $u_i, u_j$ it is the case that $(u_i, u_j) \notin \{R_2\}_S$. By Lemma 3.3 we have that $w = a_1 \cdots a_i - 1 Sa_i \cdots a_j - 1 Ea_j \cdots a_n$ is accepted by $A^{S,E}_{R_1}$, and it is not accepted by $A^{S,E}_{R_2}$, thus belonging to $A^c$. Since clearly $w$ is also in the language of $A^{S,E}$, this means that $w$ belongs to $L(A)$, which is a contradiction.

## 4 Containment of NREs

We now turn to the general problem. Let us begin with a few technical definitions.

**k-branch semipaths.** Of course, when dealing with general graph databases, we cannot longer use the construction of section 3 since it is specifically tailored for strings (or graph that look like paths). Nevertheless, we shall prove below that, even for the general case, we only need to focus on a very particular type of graphs, that we call here *k-branch semipaths*.

Fix a natural number $k$. A *k-branch domain* $D$ is a prefix closed subset of $1 \cdot \{1, \ldots, k\}^*$ such that

1. no element in $D$ is of form $\{1, \ldots, k\}^* \cdot i \cdot \{1, \ldots, k\}^* \cdot j \cdot \{1, \ldots, k\}^*$ with $i > j$. 
2. If $w \cdot i$ belongs to $D$ and there is a different element with prefix $w \cdot i$ in $D$, then $w \cdot i \cdot i$ belongs to $D$. 

A *k-branch semipath* over $\Sigma$ is a tuple $T = (D, E)$, where $D$ is a $k$-branch domain, and $E \subseteq D \times \Sigma \times D$ respects the structure of the tree: for each $u$ in $D$ there is a single edge to each of its *children* $u \cdot i$, $u \cdot j$, $u \cdot \ell$, etc. that belong to $D$, and there are no outgoing edges from the leaves of $D$. Note that $k$-branch domains have essentially $k$ types of elements: Each element of the class $[j]$ comprises all string of form $s \cdot j$, for $s \in D$, and these elements can have children only of classes $[j], \ldots, [k]$. We call each of this classes a *branch* of the semipath. Note that we impose that the element 1 must be always be the root of a $k$-branch semipath (instead of the usual $\varepsilon$).

**Canonical graphs for NREs.** Let $R$ be a NRE. We define the *nesting depth* of $R$ according to the following inductive definition:

- The nesting depth of $a$ or $a^+$ is 1, for $a \in \Sigma$. 

• If $R_1$ has nesting depth $i$, then $R'_1$ has nesting depth $i$.

• If $R_1$ has nesting depth $i$ and $R_2$ has nesting depth $j$, then $R_1 \cdot R_2$ and $R_1 + R_2$ have nesting depth $\max(i, j)$

• If $R_1$ has nesting depth $i$ then $[R_1]$ has nesting depth $i + 1$.

Let $R$ be an NRE and $T = (D, E)$ a $k$-branch semipath. We now need to define when $T$ is canonical for $R$. We do it in an inductive fashion.

• If $R = a$, for $a \in \Sigma$, then $T$ is canonical if it contains only two elements, $u$ and $u \cdot i$, and the edge from $u$ to $u \cdot i$ is labelled with $a$.

• If $R = R_1 + R_2$, then $T$ is canonical for $R$ if it is canonical for $R_1$ or for $R_2$.

• If $R = R_1 \cdot R_2$, then $T$ is canonical for $R$ if there exists an element $w$ in $T$ such that, if we define $T_1$ as the $k$-branch semipath induced by the set of elements $\{w\} \cup \{u \mid w \text{ is not a prefix of } u\}$ and $T_2$ the $k$-branch semipath induced by the set $\{u \mid w \text{ is a prefix of } u\}$, then $T_1$ is canonical for $R_1$ and $T_2$ is canonical for $R_2$ (in other words, $T$ is the concatenation of $T_1$ and $T_2$).

• If $R = [R]$, then $T$ is canonical for $[R]$ if it is canonical for $R$.

The following proposition highlights the importance of canonical graphs in our context.

**Proposition 4.1** Let $R_1$ and $R_2$ be NREs, and assume that the nesting depth of $R_1$ is $k$. Then $R_1$ is not contained in $R_2$ if and only if there is a $k$-branch semipath $T$ that is canonical for $R_1$ and two nodes of $T$ such that $(n_1, n_2) \in [R_1]_T$ yet $(n_1, n_2) \notin [R_2]_T$.

**Proof:** [Sketch] $(\Leftarrow)$: By definition.

$(\Rightarrow)$: Follows by monotonicity of NREs, using techniques similar to those in [3]. The idea is as follows. Assume that $R_1$ is not contained in $R_2$. Then there is a graph $G$ and two nodes of $G$ such that $(n_1, n_2) \in [R_1]_G$ yet $(n_1, n_2) \notin [R_2]_G$. By carefully following the construction of $R_1$, one can prune $G$ into a $k$-branch semipath $T$ (recall that $k$ is the nesting depth of $R$) that is canonical for $R_1$, and such that it still holds that $(n_1, n_2) \in [R_1]_T$. Since NREs are monotone and $T \subseteq G$ it must be the case that $(n_1, n_2) \notin [R_2]_T$. \[\square\]

**4.1 Main Proof**

We now proceed with the PSPACE upper bound for NREContainment. Let $R_1$ and $R_2$ be NREs over $\Sigma$ that are the inputs to this problem, and consider a symbol $\$ not in $\Sigma$. The roadmap of the proof is the following.

1. We will first show an encoding scheme $\text{trans}$ that transforms every $k$-branch semipath into a string over alphabet $\Gamma_k = \{1, \ldots, k\} \times (\Sigma \cup \{\$\}) \times \{1, \ldots, k\}$.

2. Afterwards, we show that one can construct, given an NRE $R$, an automaton $A_R$ over $\Gamma_k$ that accepts, in a precise sense, all encodings of $k$-branch semipaths that satisfy $R$.

3. Finally we proceed just as in Section [3] deciding whether $R_1 \subseteq R_2$ by taking the complement of $A_{R_2}$, intersecting it with $A_{R_1}$, and checking that the resulting automaton defines the empty language.
Coding $k$-branch semipaths as strings

In the following we show how $k$-branch semipaths over an alphabet $\Sigma$ can be coded into strings. Let $\$\$ be a symbol not in $\Sigma$, and $K = \{1, \ldots, k\}$. We use the alphabet $\Gamma_k = K \times (\Sigma \cup \{\$\}) \times K$, and define the translation inductively. Note that we maintain the assumption that strings begin and end with symbols $\%$ and $\&$, respectively. When $K$ is understood from context, we simply talk about $\Gamma$.

Let $T = (D, E)$ be a $k$-branch semipath. We define $\text{trans}(T)$ as $\% \cdot \text{trans}(1) \cdot \&$, where 1 is the root element of $T$. For each element in $D$, the relation $\text{trans}$ is defined as follows:

- If $w \cdot i$ is a leaf in $T$, then $\text{trans}(w \cdot i)$ is the single symbol $(i, \$, i)$.
- Otherwise, assume that the children of $w \cdot i$ are $w \cdot i \cdot \ell_1, \ldots, w \cdot i \cdot \ell_p$, and the label of each edge from $w \cdot i$ to $w \cdot i \cdot \ell_j$ is $a_j$. Then

$$\text{trans}(w \cdot i) = (i, a_2, \ell_2) \cdot \text{trans}(w \cdot i \cdot \ell_2) \cdot \cdots \cdot (i, a_p, \ell_p) \cdot \text{trans}(w \cdot i \cdot \ell_p) \cdot (i, a_1, \ell_1) \cdot \text{trans}(w \cdot i \cdot \ell_1)$$

Note that the number of characters in $\text{trans}(T)$ is precisely the sum of the number of edges and the number of leaves of the $k$-branch semipath $T$. We need a way to relate positions in trees with position in their translations. Formally, this is done via a function $\text{pos}$, that assigns to every node $w$ in a $k$-branch semipath $T$, the position in $\text{trans}(T)$ that corresponds to the point where the substring $\text{trans}(w)$ starts in $\text{trans}(T)$. Note then that all positions in $\text{trans}(T)$ will have a pre-image in $T$, except for those positions that are immediately after a symbol of form $(i, \$, i)$ in $\text{trans}(T)$.

Finally, the following proposition shows that the languages of strings represented by semipaths is regular. Moreover, an alternating automaton representing this language can be constructed in polynomial time with respect to $k$.

**Proposition 4.2** For each $k \geq 1$ there is an alternating automaton that accepts the language of all strings over $\Gamma_k$ that are encodings of a $k$-branch semipath.

**Proof:** It is useful to construct first an automaton that accepts the complement of the language in the statement of the proposition. Since AFA can be complemented in polynomial time, the proof then follows.

Fix then a number $k \geq 1$. We now sketch construct an AFA $A_k$ that accepts all strings over $\Gamma$ which are not encodings of a $k$-branch semipath. Essentially, we need $A_k$ to check for the following:

1. The string uses symbol $(i, \$, j)$ for some $i \neq j$ in $\{1, \ldots, k\}$, or any symbol $(i, a, j)$ for some $a \in \Sigma$ and $i > j$.
2. The string does not start with $\% \cdot (1, a, 1)$ for some $a \in \Sigma$
3. The string does not ends with the symbol $(1, \$, 1) \cdot \&$.
4. There is more than one appearance of the symbols $\%$ or $\&$.
5. For every $i, j \in \{1, \ldots, k\}$ and every $a \in \Sigma$, a symbol $(i, a, j)$ appears without a forthcoming symbol $(j, \$, j)$.
6. For every $i \in \{1, \ldots, k\}$ and every $a \in \Sigma$, the symbol $(i, \$, i)$ appears without a preceding symbol $(j, a, i)$, for some $j \in \{1, \ldots, k\}$.
7. There are two appearances of symbols of form $(i, \$, i)$ without any symbol of form $(j, a, i)$ in between them, for some $j < i$ and $a \in \Sigma$.
8. For some $i, j, \ell \in \{1, \ldots, k\}$, $j > i$, $\ell \leq i$ and $a, b \in \Sigma$ there are two symbols $(i, a, j)$ and $(i, b, j)$ between symbols $(\ell, a, i)$ and $(i, b, i)$, for some $a', b' \in \Sigma$.
9. For every $i, j \in \{1, \ldots, k\}$ and every $a \in \Sigma$, at some point after the symbol $(i, a, j)$ and before the symbol $(j, \$, j)$ there is a symbol of form $(i', a', j')$, for $a \in \Sigma$, with either $i'$ or $j'$ strictly lower than $j$. 

10.
10. For every \( i, j \in \{1, \ldots, k\} \) and every \( a \in \Sigma \), after a subword that starts with the symbol \((i, a, j)\), has only symbols of form \((\ell, b, \ell')\) for \( \ell, \ell' \geq j \) and \( b \in \Sigma \), and ends with symbol \((j, \$, j)\); there is a symbol of form \((p, c, p')\) with \( p \neq i \) and \( c \in \Sigma \cup \{\$\} \).

It is now straightforward to construct such automaton. Furthermore, since complementation in alternating automata can be performed in polynomial time, the proof follows. \( \square \)

**Alternating automata for NREs**

All that remains for the \( \text{PSPACE} \)-upper bound is to show how one can construct, given an NRE \( R \), an A2FA \( A_R \) that accepts all strings over \( \Gamma \) that are encodings of \( k \)-branch semipaths that satisfy \( R \).

**Construction of \( A_R \):**

We start with some technical definitions. For every \( i, j \in \{1, \ldots, k\} \), define the language \( L_{(i,j)} \) as follows:

\[
L_{(i,j)} = (i, a, j) \cdot ([\ell, b, \ell'] \in \Gamma | \ell, \ell' \geq j \text{ and } b \in \Sigma]^* \cdot (j, \$, j).
\]

Intuitively, each \( L_{(i,j)} \) defines path that departs from level \( i \) to level \( j \) in the \( k \)-branch semipath.

The main technical difficulty in this construction is to allow the automata to navigate through the encoding of the \( k \) branch semipath. This is mostly captured by the base cases of our inductive construction the idea is that one now has to allow the automata to skip words of form \( L_{(i,j)} \) when choosing the next symbol (or when looking for it when reaching backwards), or in fact allow it to jump to a different branch in the semipath. We shall therefore make repeated use of the languages \( L_{(i,j)} \). We also define, for each \( 1 \leq i \leq k \), the language

\[
B_i = (L_{(i,i)} + L_{(i,i+1)} + \cdots + L_{(i,k)})^*.
\]

Finally, we also work with a 2 way automaton \( A_{B_1} \) that reads backwards from symbol \((j, \$, j)\) to the first symbol of form \((i, a, j)\) for some \( i < j \). More precisely, \( B_1^{-} = (Q, q_0, \emptyset, \{q_f\}, \Gamma, \delta) \), where \( Q = \{q_0, q_1, q_2, q_f\} \) and \( \delta \) is as follows:

- For each symbol \( a \in \Gamma \), \( \delta(q_0, a) = (q_1, -1) \). This moves the automaton a step backwards, so we can start checking our language.
- In addition, \( \delta(q_1, (j, \$, j)) = \{(q_2, -1)\} \). This piece checks that we start with \((j, \$, j)\).
- For each \( a \in \Sigma \cup \{\$\} \) and \( k, k' \geq j \), \( \delta(q_2, (k, a, k')) = \{(q_2, -1)\} \). This forces the automaton to loop in this state if one does not find the start of the branch.
- Finally, for each \( a \in \Sigma \) and \( i < j \), \( \delta(q_2, (i, a, j)) = \{(q_f, 0)\} \). This simply checks that the last symbol marks the beginning of the branch.

With these definitions we can start describing the construction. Let \( R \) be an NRE. The automaton \( A_R \) for \( R \) is as follows:

- If \( R = a \) for some \( a \in \Sigma' \). For each \( 1 \leq i \leq k \), let \( A_{B_i} \) be a copy of an automaton accepting \( B_i \), using fresh states, and assume that their initial and final states, respectively, are \( p_0^i \) and \( p_f^i \). Furthermore, for each \( 1 \leq j \leq k \) create a fresh copy of the automaton \( A_{B_j}^{-} \), with initial and final states \((p_0^j)^-\) and \((p_f^j)^-\), respectively. then \( A_R = (Q, q_0, \emptyset, \{q_f\}, \Gamma, \delta) \), where \( Q \) contains \( \{q_0, q_1, q_2, q_f\} \) plus all the states of the automata \( A_{B_i} \) and \( A_{B_j}^{-} \) for \( i, j \in \{1, \ldots, k\} \), and \( \delta \) contains, apart from all the transitions in the \( A'_{B_i} \)'s and \( A'_{B_j}^{-} \)'s, the following transitions:
  - For each \( 1 \leq i \leq j \leq k \), \( \delta(q_0, (i, a, j)) = \{(q_f, 1), (q_1, -1)\} \) — item For each \( 1 \leq i \leq j \leq k \) and \( b \in \Sigma (b \neq a) \), \( \delta(q_0, (i, b, j)) = \{(q_1, -1)\} \)
  - \( \delta(q_0, \varepsilon) = \{(p_0^1, 0), \ldots, (p_0^k, 0)\} \)
For each $i$ of the automata $\mathcal{A} = (Q, \Gamma, q_0, F)$, let $\mathcal{B}_i$ be a copy of an automaton accepting $\mathcal{B}_i$, using fresh states, and assume that their initial and final state, respectively, are $p_0^i$ and $p_f^i$. Furthermore, for each $1 \leq j \leq k$ create a fresh copy of the automaton $\mathcal{A}_{B_i}$, with initial and final states $(p_0^i)^-$ and $(p_f^i)^-$, respectively. then $\mathcal{A}_R = (Q, q_0, \emptyset, \{q_f\}, \Gamma, \delta)$, where $Q$ contains $\{q_0, q_1, q_2, q_f\}$ plus all the states of the automata $\mathcal{A}_R$ and $\mathcal{A}_{B_i}$ for $i, j \in \{1, \ldots, k\}$, and $\delta$ contains, apart from all the transitions in the $\mathcal{A}_{B_i}$, the following transitions:

- For each $1 \leq i \leq j \leq k$, $\delta(q_0, (i, a^-, j)) = \{(q_f, 1), (q_1^i, -1)\}$ — item For each $1 \leq i \leq j \leq k$ and $b \in \Sigma (b \neq a^-)$, $\delta(q_0, (i, b, j)) = \{(q_1^i, -1)\}$
- $\delta(q_0, \varepsilon) = \{(p_0^i, 0), \ldots, (p_0^i, 0)\}$
- $\delta(p_i^j, \varepsilon) = \{(q_0, 0)\}$ for each $1 \leq i \leq k$.
- For each $1 \leq i \leq j \leq k$, $\delta(q_1^i, (i, a, j)) = \{(q_2^i, 0)\}$
- $\delta(q_2^i, \varepsilon) = \{(q_f, 0), (p_f^i)^-, 0), \ldots, (p_f^i)^-, 0)\}$
- $\delta((p_f^j)^-, \varepsilon) = \{(q_2^j, 0)\}$ for each $1 \leq i \leq k$.

State $q_f$ is marked.

Case when $R = R_1 + R_2$. Let $\mathcal{A}_{R_i} = (Q^i, U^i, q_0^i, F, \Gamma, \delta^i)$, for $i = 1, 2$, and assume that $q_m^i$ is the marked state from $\mathcal{A}_{R_i}$. Define $\mathcal{A}_R = (Q, U, q_0, F, \Gamma, \delta)$, where $Q = \{q_0, q_f\} \cup Q^1 \cup Q^2$, $U = U^1 \cup U^2$, $F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \cup (F^2 \setminus \{q_m^2\})$ and $\delta = \delta^1 \cup \delta^2$, plus transitions

\[
\delta(q_0, \varepsilon) = \{(q_0^1, 0), (q_0^2, 0)\} \\
\delta(q_m^1, \varepsilon) = \{(q_f, 0)\} \\
\delta(q_m^2, \varepsilon) = \{(q_f, 0)\}
\]

For each $i = 1, 2$, remove all marks from $\mathcal{A}_{R_i}$, and mark state $q_f$.

In the case that $R = R_1 \cdot R_2$, let $\mathcal{A}_{R_i} = (Q^i, U^i, q_0^i, F, \Gamma, \delta^i)$, for $i = 1, 2$, and assume that $q_m^i$ is the marked state from $\mathcal{A}_{R_i}$. For each $i = 1, 2$, remove all markings from $\mathcal{A}_{R_i}$. Define $\mathcal{A}_R = (Q, U, q_0, F, \Gamma, \delta)$, where $Q = \{q_0, q_f\} \cup Q^1 \cup Q^2$, $U = U^1 \cup U^2$, $F = \{q_f\} \cup (F^1 \setminus \{q_m^1\}) \cup (F^2 \setminus \{q_m^2\})$ and $\delta = \delta^1 \cup \delta^2$, plus transitions

\[
\delta(q_0, \varepsilon) = \{(q_0^1, 0)\} \\
\delta(q_m^1, \varepsilon) = \{(q_f, 0)\} \\
\delta(q_m^2, \varepsilon) = \{(q_f, 0)\}
\]

For each $i = 1, 2$, remove all marks from $\mathcal{A}_{R_i}$, and mark state $q_f$.

For $R = R_1^\ast$, let $\mathcal{A}_{R_i} = (Q^i, U^i, q_0^i, F^i, \Gamma, \delta^i, F^i)$, and assume that $q_m^1$ is the marked state from $\mathcal{A}_{R_i}$. Define $\mathcal{A}_R = (Q, U^1, q_0, F, \Gamma, \delta)$, where $Q = \{q_0, q_f\} \cup Q^1$, $F = \{q_f\} \cup (F^1 \setminus \{q_m^1\})$ and $\delta = \delta^1$ plus transitions

\[
\delta(q_0, \varepsilon) = \{(q_0^1, 0)\} \\
\delta(q_m^1, \varepsilon) = \{(q_f, 0)\} \\
\delta(q_m^2, \varepsilon) = \{(q_f, 0), (q_0^1, 0)\}
\]

Remove all marks from $\mathcal{A}_{R_i}$, and mark state $q_f$. 


Lemma 4.3
Let \( \pi \) state of the computation of the part of \( A_{i} \). Then \( A_{R} = (Q, U, q_{0}, F, \Gamma, \delta) \), where \( Q = \{q_{0}, p, q_{2}, q_{f}\} \cup Q^{1}, U = U^{1} \cup \{p\}, F = \{q_{f}\} \cup F^{1} \) and \( \delta = \delta^{1} \), plus transitions

\[
\begin{align*}
\delta(q_{0}, \varepsilon) &= \{(p, 0)\} \\
\delta(p, \varepsilon) &= \{(q_{f}, 0), (q_{f}^{1}, 0)\} \text{ (recall that } p \text{ is a universal state)} \\
\delta(q^{1}_{m}, a) &= \{(q^{1}_{m}, 1)\} \text{ for each } a \in \Gamma
\end{align*}
\]

Remove all marks from \( A_{R} \), and mark state \( q_{f} \).

Let \( A_{R} = (Q, q_{0}, U, F, \Gamma, \delta) \) be as constructed by this algorithm. To finish our construction we need to allow \( A_{R} \) to (non deterministically) move backwards from the end of the word, until it reaches a suitable starting point for the computation, and allow every final state to reach the end of the word in its computation. Formally, we define \( A'_{R} = (Q \cup \{q^{0}_{0}\}, q^{0}_{0}, U, F, \Gamma, \delta') \), where \( \delta' \) contains all transitions in \( \delta \) plus transitions \( \delta(q^{0}_{0}, a) = \{(q^{0}_{0}, 0), (q^{0}_{0}, -1)\} \) for each \( a \in \Gamma \) and \( \delta(q_{f}, a) = (q_{f}, 1) \) for each \( a \in \Gamma \). In the remainder of the proof, when speak of the automata for \( R \) we refer to this last automaton \( A'_{R} \), even if we use the clearer \( A_{R} \) instead.

The rest of the proof goes along the same lines as the version for semipaths. Notice that the above construction can be computed in polynomial time with respect to \( R \). Furthermore, let \( q_{m} \) be the marked state of \( A'_{R} \). From its construction, it is clear that every accepting computation tree \( \Pi \) of \( A_{R} \) on input \( w \) will have the following form: (1) For some \( 1 \leq i \leq |w| \) there is a single path from the root to a node \( \pi_{s} \) such that \( l(\pi) = (q_{0}, w, i) \) and no ancestor of \( \pi \) is labelled with an ID using a state different from \( q^{0}_{0} \); and (2) there is some \( 1 \leq j \leq |w| \) such that \( \pi_{f}, \pi_{f}', \pi_{f}'' \ldots \) is the maximal path of nodes (up to a leaf) labelled with \( (q_{m}, w, j), (q_{m}, w, j+1), \ldots, (q_{m}, w, |w| + 1) \), and where the father of \( (q_{m}, w, j) \) is not a configuration using state \( q_{m} \). Property (1) represents the automaton searching for its starting point, and (2) represents the end of the computation of the part of \( A'_{R} \) that is representing the non-nesting part of \( R \). We denote such nodes \( \pi_{s} \) and \( \pi_{f} \) as the tacit start and tacit ending of \( \Pi \).

With this definitions we can show the following.

Lemma 4.3 Let \( R \) a NRE, \( A_{R} \) the automaton constructed for \( R \), \( T \) a graph over \( \Sigma \) that is a \( k \)-branch semipath, where \( k \) is the nesting depth of \( R \), and \( w = \text{trans}(T) \) be the encoding of \( T \) as a string. Then a pair \( (u, v) \) belongs to \( [R]_{T} \) if and only if there is an accepting computation tree of \( A_{R} \) on input \( w \) whose tacit start is labelled with \( (q_{0}, w, \text{pos}(u)) \) and whose tacit ending is labelled with \( (q_{m}, w, \text{pos}(v)) \).

Proof: Let \( T \) be a \( k \)-branch semipath and let \( A_{R} = (Q, q_{0}, U, F, \Gamma, \delta) \) constructed as above. Let us start with the Only if direction. Assume that \( [R]_{T} \) contains the pair \( (u, v) \) for some nodes \( u \) and \( v \) of \( T \). We show the statement of the Lemma by induction on \( R \).

We only show the case when \( R = a \). The case when \( R = a^{-} \) is completely symmetrical, and the remaining ones follow from the proof of Lemma 3.2.

- If \( R = a \) for some \( a \in \Sigma' \) and \( (u, v) \) belong to \( [R]_{T} \), then either \( u \) is a prefix of \( v \) and the edge \( (u, a, v) \) is in \( T \), or \( v \) is a prefix of \( u \) and the edge \( (v, a^{-}, u) \) is in \( T \). For the former case, assume that \( u = w \cdot i \), all children of \( u \) are \( u_{1}, \ldots, u_{n} \), and \( v = u_{i} \) for some \( 1 \leq \ell \leq n \). If \( v = w \cdot i \cdot i \), then starting in \( \text{pos}(u) \) one can use the transitions that loop in some of the \( B_{i} \)'s until we reach symbol \( (i, a, i) \) in \( \text{trans}(T) \), from which we advance to the final state of \( A_{R} \). Otherwise, if \( v = w \cdot i \cdot j \) for some \( i < j \), we can also loop, but this time until we advance to the final state by means of symbol \( (i, a, j) \). For the latter case, assume that \( v = w \cdot i \), all children of \( v \) are \( v_{1}, \ldots, v_{n} \), and \( u = v_{i} \) for some \( 1 \leq \ell \leq n \). If \( u = w \cdot i \cdot i \), then by definition the symbol \( (i, a^{-}, i) \) is directly before \( \text{pos}(u) \). We can then non-deterministically jump to \( q_{i}^{1} \) in \( A_{R} \), check that effectively the symbol \( (i, a^{-}, j) \) exists, and move backwards according to the transitions looping in the copies of automata \( B_{j}^{-1} \)'s, until we reach \( \text{pos}(v) \). Otherwise if \( u = w \cdot i \cdot j \) with \( i < j \) then by definition again the symbol \( (i, a^{-}, j) \) is directly before \( \text{pos}(u) \), and we continue along the same lines as before.
Next, for the If direction, assume there is an accepting computation tree of $A'_R$ on input $w$ whose tacit start is labelled with $(q_0, w, \text{pos}(u))$ and whose tacit ending is labelled with $(q_m, w, \text{pos}(v))$. We show that $(u, v) \in R_R$ by induction. Once again, it suffices to show the base case when $R = a$.

- If $R = a$ for some $a \in \Sigma'$, there are two types of computation tree for $A_R$. Assume first that such tree does not mention any node labelled with an ID that corresponds to $q_1$ or $q_2$. Then, at some point in the computation tree, there must be a jump from an ID of form $(q_0, w, i)$ to an ID of form $(q_j, w, j)$ for some positions $i$ and $j$ in $w$, and when reading a symbol of form $(\ell_1, a, \ell_2)$. From the construction of $A_R$, we can only loop from state $q_j$ if we are directly after a symbol of form $(\ell', S, \ell')$, and thus in this case we can not loop; it must be that $j = \text{pos}(v)$. Furthermore, if one stays in state $q_0$ one can only move forward, in a way that the subword between position $\text{pos}(u)$ and $i$ must correspond to a concatenation of word in some $L(\ell_{i,j})$. Then either $i = \text{pos}(u)$ or $u$ has at least two children, and position $i$ corresponds to the position in $w$ after we have read the encoding for some of these children. It then follows from our translation $\text{trans}$ that the edge between $u$ and $v$ in $T$ is labelled $a$.

Next, assume that the tree does mention an ID going through $q_1^i$. In this case, there must be a step from $q_0$ to $q_1^i$ that is a move backwards, and then to advance to $q_2^i$ we need a symbol of form $(\ell_1, a', \ell_2)$. In other words, at some point in the tree we move from ID $(q_0, w, i)$ to $(q_1^i, w, i - 1)$ and then to $(q_2^i, w, i - 1)$, and such that the symbol between positions $i - 1$ and $i$ is of the form $(\ell_1, a', \ell_2)$. It follows that $i = \text{pos}(u)$, since by moving forwards in $q_0$ we shall never reach a point directly after a symbol with this form. A similar argument as the previous case also shows that either $\text{pos}(v) = i - 1$ or the path from $\text{pos}(v)$ to $i - 1$ must correspond to a concatenation of words in $L(\ell_{i,j})$. By inspecting our translation, we then have that there must be an edge $(v, a', u)$ in $T$, and therefore $(u, v) \in \llbracket R \rrbracket_T$.

Just as we saw for the case of semipaths, we need to be more careful, and explicitly mark with symbols $S$ and $E$ to positions in $\text{trans}(T)$, in order to distinguish the root and leaves of $T$ with the actual $k$-branch semipath that is framed by nodes $u$ and $v$. Formally, given a $k$-branch semipath $T$, and two nodes $u$ and $v$ of $T$, the expansion $T[u \to S, v \to E]$ is the $k$-branch semipath defined as follows. If $u = w \cdot i$, and its children $w \cdot i \cdot \ell_1, \ldots, w \cdot i \cdot \ell_n$, then rename all children to $w \cdot i \cdot \ell_1, \ldots, w \cdot i \cdot \ell_n$, and all of the descendants of $u$ accordingly, so that the domain remains prefix-closed. Now $u$ has a single child, $w \cdot i \cdot i$ connected by an edge labelled $S$, and this node is the father of all the nodes that were previously childrens of $u$. Repeat with $v$ and $E$. The intuition is that $T[u \to S, v \to E]$ is created by replacing node $u$ in $T$ with an edge labelled by $S$, and node $v$ by an edge labelled $E$. Let $R$ be an NRE. Using the ideas presented in the proof of Lemma 4.3, it is not difficult to define a translation from $A_R$ to an automaton $A_R^{S,E}$ such that the following holds:

**Lemma 4.4** Let $T$ be a $k$-branch semipath, and $R$ an NRE. Then a pair $(u, v)$ belongs to $\llbracket R \rrbracket_T$ if and only if $A_R^{S,E}$ accepts the semipath $T[u \to S, v \to E]$.

We can now state our algorithm for solving SP-QUERYCONTAINMENT. On input NREs $R_1$ and $R_2$ over $\Sigma$, we perform the following operations:

1. Compute an NFA $A^{S,E}$ that accepts only those words over $(\Sigma \cup \{S, E\})^*$ of form $w_1Sw_2Ew_3$, for each $w_1, w_2, w_3 \in \Sigma^*$.
2. Compute $A_k$ that accepts only those words which are translations of $k$-branch semipaths over $\Sigma$, where $k$ is the nesting depth of $R_1$.
3. Compute $A_{R_1}^{S,E}$ and $A_{R_2}^{S,E}$ as explained above.
4. Compute the A2FA $A^c = (A_{R_2}^{S,E})^c$ whose language is the complement of $A_{R_2}^{S,E}$.
5. Compute the A2FA $A$ whose language is the intersection of the languages $A^{S,E}$, $A_{R_1}^{S,E}$, $A^c$ and $A_{\text{trans}}$.
6. Check that the language of $A$ is empty.
We have seen how to perform the second step in polynomial time, and steps (1), (3), (4) can be easily performed in PTIME using standard techniques from automata theory. Finally, Proposition 3.1 shows that step (5) can be performed in PSPACE. Thus, all that is left to prove is that the language of the resulting automata \( A \) is empty if and only if \( R_1 \subseteq R_2 \).

Assume first that \( R_1 \subseteq R_2 \), and assume for the sake of contradiction that there is a word \( w \in L(A) \). This word is then accepted by \( A_{S,E}^{S,E} \) and \( A_k \). Then there is a \( k \)-branch semipath \( T \) over \( \Sigma \) and two nodes \( u \) and \( v \) of \( T \) such that \( w = \text{trans}(T[u \rightarrow S, v \rightarrow E]) \). Furthermore, \( w \) is accepted by \( A_{R_1}^{S,E} \), but not by \( A_{R_2}^{S,E} \). This implies, by Lemma 4.4, that \((u, v) \notin \llbracket R_1 \rrbracket_T \) but \((u, v) \in \llbracket R_2 \rrbracket_T \), which is a contradiction.

On the other hand if \( L(A) \) is empty but \( R_1 \nsubseteq R_2 \), then for some \( k \)-branch semipath \( T \) that is canonical for \( R_1 \) and two nodes \( u \) and \( v \) of \( T \) we have that \((u, v) \in \llbracket R_1 \rrbracket_T \) yet \((u, v) \notin \llbracket R_2 \rrbracket_T \). By Lemma 4.4 we have that \( w = \text{trans}(T[u \rightarrow S, v \rightarrow E]) \) is such that \( w \) belongs to \( A_{S,E}^{S,E} \), \( A_k \) and \( A_{R_1}^{S,E} \), but it is not accepted by \( A_{R_2}^{S,E} \), thus belonging to \( A^c \). This means that \( w \) is in the language of \( A \), which is a contradiction.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] P. Barceló, J. Pérez, and J. L. Reutter. Relative expressiveness of nested regular expressions. In *AMW*, pages 180–195, 2012.
[3] P. Barceló, J. Pérez, and J. L. Reutter. Schema mappings and data exchange for graph databases. In *to appear in ICDT*, page TBD, 2013.
[4] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Answering regular path queries using views. In *16th International Conference on Data Engineering (ICDE)*, pages 389–398, 2000.
[5] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Containment of conjunctive regular path queries with inverse. In *7th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 176–185, 2000.
[6] M. Consens and A. Mendelzon. Graphlog: A visual formalism for real life recursion. In *9th ACM Symposium on Principles of Database Systems (PODS)*, pages 404–416, 1990.
[7] I. Cruz, A. Mendelzon, and P. Wood. A graphical query language supporting recursion. In *ACM Special Interest Group on Management of Data 1987 Annual Conference (SIGMOD)*, pages 323–330, 1987.
[8] D2R DBLP bibliography database hosted at L3S research center. *http://dblp.l3s.de/d2r/*, 2013.
[9] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *17th ACM Symposium on Principles of Database Systems (PODS)*, pages 139–148, 1998.
[10] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.
[11] P. Hayes. RDF Semantics, W3C Recommendation. *http://www.w3.org/TR/rdf-mt*, February 2004.
[12] R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM J. Comput.*, 13(1):135–155, 1984.
[13] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.
[14] C. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.
[15] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.