Document Spanners: A Formal Approach to Information Extraction

RONALD FAGIN, BENNY KIMELFELD,and FREDERICK REISS, IBM Research – Almaden
STIJN VANSUMMEREN, Université Libre de Bruxelles (ULB)

An intrinsic part of information extraction is the creation and manipulation of relations extracted from text. In this article, we develop a foundational framework where the central construct is what we call a *document spanner* (or just *spanner* for short). A spanner maps an input string into a relation over the spans (intervals specified by bounding indices) of the string. The focus of this article is on the representation of spanners. Conceptually, there are two kinds of such representations. Spanners defined in a *primitive* representation extract relations directly from the input string; those defined in an *algebra* apply algebraic operations to the primitively represented spanners. This framework is driven by SystemT, an IBM commercial product for text analysis, where the primitive representation is that of regular expressions with capture variables.

We define additional types of primitive spanner representations by means of two kinds of automata that assign spans to variables. We prove that the first kind has the same expressive power as regular expressions with capture variables; the second kind expresses precisely the algebra of the *regular* spanners—the closure of the first kind under standard relational operators. The *core* spanners extend the regular ones by string-equality selection (an extension used in SystemT). We give some fundamental results on the expressiveness of regular and core spanners. As an example, we prove that regular spanners are closed under difference (and complement), but core spanners are not. Finally, we establish connections with related notions in the literature.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata (e.g., finite, push-down, resource-bounded), relations between models*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Algebraic language theory, classes defined by grammars or automata (e.g., context-free languages, regular sets, recursive sets), operations on languages*; H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.4 [**Database Management**]: Systems—*Textual databases, relational databases, rule-based databases*; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*

General Terms: Theory

Additional Key Words and Phrases: Information extraction, document spanners, regular expressions, finite-state automata

ACM Reference Format:
Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. 2015. Document spanners: A formal approach to information extraction. J. ACM 62, 2, Article 12 (April 2015), 51 pages.
DOI: http://dx.doi.org/10.1145/2699442

An abridged version of this article has been published in *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)* [Fagin et al. 2013].
B. Kimelfeld is currently affiliated with Technion – Israel Institute of Technology.
Authors' addresses: R. Fagin and F. Reiss, IBM Research – Almaden, 650 Harry Road, San Jose, CA 95120-6099; email: {fagin, frreiss}@us.ibm.com; B. Kimelfeld, Technion – Israel Institute of Technology, Haifa, Israel; email: bennyk@gmail.com; S. Vansummeren, Université Libre de Bruxelles, 50, Av. F. Roosevelt, CP 165/15, B-1050 Brussels; Belgium; email: stijn.vansummeren@ulb.ac.be.

0004-5411/2015/04-ART12 $15.00
DOI: http://dx.doi.org/10.1145/2699442

1. INTRODUCTION

Automatically extracting structured information from text is a task that has been pursued for decades. As a discipline, Information Extraction (IE) had its start with the DARPA Message Understanding Conference in 1987 [Grishman and Sundheim 1996]. While early work in the area focused largely on military applications, recent changes have made information extraction increasingly important to an increasingly broad audience. Trends such as the rise of social media have produced huge amounts of text data, while analytics platforms like Hadoop have at the same time made the analysis of this data more accessible to a broad range of users. Since most analytics over text involves the extraction of information items (at least as a first step), IE is nowadays an important part of data analysis in the enterprise.

Broadly speaking, there are two main schools of thought on the realization of IE: the *statistical* (machine-learning) methodology and the *rule-based* approach. The first started with simple models such as AutoSlog [Riloff 1993], CRYSTAL [Soderland et al. 1995] and SRV [Freitag 1998], then progressed to approaches based on probabilistic graph models [Leek 1997; McCallum et al. 2000; Lafferty et al. 2001]. Within the rule-based approach, most of the solutions (e.g., GATE/JAPE [Cunningham 2002]) built upon *cascaded finite-state transducers* [Appelt and Onyshkevych 1998]. Most systems in both categories were built for academic settings, where most users are highly-trained computational linguists, where workloads cover only a small number of very well-defined tasks and data sets, and where extraction throughput is far less important than the accuracy of results.

When IBM researchers, driven by the increasing importance of text data in the enterprise, attempted to use these existing tools to solve customers' analytics problems, they encountered a number of practical challenges. Users needed to have an intuitive understanding of machine learning or the ability to build and understand complex and highly interdependent rules. Determining why an extractor produced a given incorrect result was extremely difficult, which made impractical the reuse of extractors across different data sets and applications. Moreover, high CPU and memory requirements made extractors cost-prohibitive in deployment over large-scale data sets.

In 2005, researchers at the IBM Almaden Research Center began the design and development of a new system, specifically geared for practical information extraction in the enterprise. This effort led to SystemT, a rule-based IE system with an SQL-like declarative language named AQL (*Annotation Query Language*) [Chiticariu et al. 2010; Krishnamurthy et al. 2008; Reiss et al. 2008]. The declarative nature of AQL enables new kinds of tools for extractor development [Liu et al. 2010], and a cost-based optimizer for performance [Reiss et al. 2008]. In 2010, SystemT was released as a commercial IBM product.[1] An intensive study by Chiticariu et al. [2010] shows the value of SystemT, in particular the high extent to which it overcomes the difficulties mentioned earlier.

Conceptually, AQL can be viewed as built upon two main operations that were supported already in the original research prototype of SystemT [Reiss et al. 2008]. The first operation (expressed as "extract" statements) is the extraction of relations from the underlying text through simple mechanisms. The most commonly used of these mechanisms is that of regular expressions with capture variables. An important special case of that mechanism is the extraction of dictionary (gazetteer) matches that are distinguished from general regular expressions by their syntax and underlying implementation. The second operation (expressed as "select" statements) is the manipulation of the relations (from the first operation) through algebraic operators. There

[1]SystemT is included in IBM InfoSphere BigInsights.
http://www.ibm.com/software/data/infosphere/biginsights/.

are three types of algebraic operators: standard relational operators (e.g., union, projection, join), text-centric operators (e.g., string equality and containment), and conflict resolution (mainly, resolving cases of overlapping spans when those are undesired). In the actual (productized) AQL syntax, these operators are expressed as clauses of a Select-From-Where flavor.[2] In time, SystemT evolved to support additional facilities, like part-of-speech tagging, shallow parsing of XML tags, sorting and additional aggregate functions.

In this article, we embark on an investigation of the principles underlying AQL. Our ultimate goal is to establish a formal model that is robust enough to capture the principal capabilities of systems featuring AQL's principles, and yet, that is abstract enough to yield useful insights, and solutions with provable guarantees. Towards that, we develop here a framework that captures the core functionality of SystemT, and establish some fundamental results on expressiveness and on the relationship with existing literature. We believe that this work will be the basis of further investigation of tools for text analytics. We further believe that this work and its followups will shed light on the interplay between the textual and the relational querying models (in contrast to their traditional separation as distinct steps). In the remainder of this section, we give a more technical and detailed description of our framework and results.

A *span* of a string $\mathbf{s}$ (where $\mathbf{s}$ represents the text) represents the range of a substring of $\mathbf{s}$, and is given by two indices that specify where the range begins and ends within $\mathbf{s}$. For example, if $\mathbf{s}$ is `ACM_PODS_2013`, then the span $[5, 9\rangle$ refers to the part of $\mathbf{s}$ from the fifth to the eighth symbols inclusive, spanning the substring `PODS`. In this article we introduce *document spanners* (or just *spanners* for short), the central concept in our framework. Intuitively, a spanner extracts from a string $\mathbf{s}$ a relation over the spans of $\mathbf{s}$. It is formally defined as follows. An $\mathbf{s}$-*tuple* is associated with a finite domain V of *span variables*, and assigns a span of $\mathbf{s}$ to each variable in V. A *span relation* (*over* $\mathbf{s}$) is a set of $\mathbf{s}$-tuples, all over the same domain V of span variables. That set is naturally viewed as a relation, with the span variables playing the roles of the attribute names, and the spans themselves used as attribute values. A *spanner* is a function that maps each string $\mathbf{s}$ into a span relation over $\mathbf{s}$.

For illustration, consider Figure 1, that is used for our running example in this article. The figure shows two strings $\mathbf{s}$ and $\mathbf{t}$, and considers two spanners P_1 and P_2. The tables in the figure show the four span relations obtained by applying P_1 and P_2 to $\mathbf{s}$ and $\mathbf{t}$. For instance, the top row in the table of $P_1(\mathbf{s})$ shows the $\mathbf{s}$-tuple that assigns the spans $[1, 4\rangle$, $[5, 8\rangle$ and $[1, 8\rangle$ to the variables x, y and z, respectively.

This article focuses on the representation of spanners. Conceptually, we distinguish between two types of spanner representations. The first type is that of a *primitive* representation, which is a mechanism that extracts the relation directly from the input string $\mathbf{s}$. An example is a regular expression with span variables embedded as capture variables, as in AQL; here, we call such an expression a *regex formula*. The second type of a spanner representation is that of an *algebra*, which is the closure of primitive representations (of some specific class) under some algebraic operators.

Aside from regex formulas, we define two additional primitive spanner representations that are based on two corresponding types of automata. An automaton of each type is an ordinary nondeterministic finite automaton (NFA), except that it is associated with a finite set V of variables, and along a run on a string it can decide to open (i.e., begin the assigned span for) or close (i.e., end the assigned span for) a variable. In an accepting run, each variable in V must be opened and closed exactly once. The difference between the two types is in the data structures that maintain the variables. In a *variable-stack automaton* (*vstk-automaton* for short), that data structure is a stack,

[2]See http://publib.boulder.ibm.com/infocenter/bigins/v2r0/ for the complete reference of the AQL syntax.

String **s**

A	a	a	_	A	b	b	_	a	a	_	B	a	a	_	B	b	b	_	b	_	A	b	a	a	_	A	b	b
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

$P_1(\mathbf{s})$

	x	y	z
μ_1	$[1,4\rangle$	$[5,8\rangle$	$[1,8\rangle$
μ_2	$[12,15\rangle$	$[16,19\rangle$	$[12,19\rangle$
μ_3	$[22,26\rangle$	$[27,30\rangle$	$[22,30\rangle$

$P_2(\mathbf{s})$

	x_1	x_2
μ_4	$[1,4\rangle$	$[22,26\rangle$

String **t**

A	a	a	_	A	b	b	_	A	b	b
1	2	3	4	5	6	7	8	9	10	11

$P_1(\mathbf{t})$

	x	y	z
μ_5	$[1,4\rangle$	$[5,8\rangle$	$[1,8\rangle$
μ_6	$[5,8\rangle$	$[9,12\rangle$	$[5,12\rangle$

$P_2(\mathbf{t})$

	x_1	x_2
μ_7	$[1,4\rangle$	$[5,8\rangle$

Fig. 1. Running example: strings **s** and **t**, and the string relations obtained by applying two spanners P_1 and P_2.

and hence, the closed variable is always the most recently opened one. In a *variable-set automaton* (*vset-automaton* for short), that data structure is a set, and the automaton specifies the specific (previously opened) variable to close.

We begin by showing that regex formulas, vstk-automata and vset-automata are tightly related to each other. In particular, regex formulas and vstk-automata have the same expressive power. The vset-automata can express spanners that are not expressible by vstk-automata, since a spanner representable by the latter is necessarily *hierarchical*—the spans of every output **s**-tuple are nested like balanced parentheses. We prove that the spanners expressible by regex formulas are precisely the spanners that are both hierarchical and representable by vset-automata. Moreover, we prove that the expressive power of vset-automata is precisely that of the algebra that closes regex formulas under union, projection and natural join on spans. Finally, we prove that these algebraic operators do not increase the expressive power of vset-automata. We call the spanners expressible by vset-automata *regular spanners*. The name arises from the fact that, in the Boolean case, the languages recognizable by vset-automata are the regular ones.

An algebraic operator of AQL that was not mentioned in the previous paragraph is *string-equality* selection, which selects the **s**-tuples such that the spans for two specified variables x and y correspond to equal substrings of **s** (although x and y need not be the same span). The *core* spanners, which we view as capturing the core of AQL, are the ones expressible by regex formulas along with the operators union, projection, natural join on spans, and string-equality selection. In this language, one can also simulate selection operators for other common string relationships such as containment, prefix and suffix. Standard inexpressiveness results for regular expressions easily imply that core spanners are more expressive than regular spanners. We prove a key lemma for core spanners, the "core-simplification lemma," which states that every core spanner can be represented as a *single* vset-automaton, followed by string selections and then by a projection. This lemma is a crucial ingredient for our later proofs of inexpressiveness results.

Focusing on regular and core spanners, we also look at the ability to *simulate* selection operators based on string relations (relations whose entries are strings, not spans). More formally, for a string relation R, the corresponding selection operator selects all the $\mathbf{s}$-tuples such that the substrings corresponding to a specified sequence of variables (of the same arity as R) is in R. We say that R is *selectable* by a class of spanners (e.g., the regular or core spanners) if that class is closed under the selection operator for R. Like Barceló et al. [2012a], we look at three classes of string relations: the *recognizable relations* [Berstel 1979; Elgot and Mezei 1965], which are contained in the *regular relations* [Benedikt et al. 2003; Elgot and Mezei 1965], which are contained in the *rational relations* [Berstel 1979; Nivat 1968]. We show that every recognizable relation is selectable by the core spanners. We also show the existence of a regular (hence rational) relation that is not selectable by the core spanners, and the existence of a relation that is selectable by the core spanners but is not rational (hence not regular). As for regular spanners, it turns out that their selectable string relations are precisely the recognizable ones.

In Section 5, we investigate the incorporation of the *difference* operator in our setting. We prove that core spanners are not closed under difference. By analogy to the relational model, this may sound straightforward because all the other operators are monotonic. But this argument is invalid here, because regex formulas have the ability to simulate non-monotonic functionality. As evidence, it turns out that regular spanners are closed under difference. Moreover, as further evidence, some relations of a non-monotonic flavor are selectable by the core spanners, like inequality, nonprefix and nonsuffix. In contrast, we prove with the core-simplification lemma that non-substring is not selectable by the core spanners; with that, nonclosure under difference is a simple corollary.

Related Work. There is a large body of work on designing query languages for string databases (i.e., databases in which the atomic data values are strings) [Bonner and Mecca 1998; Benedikt et al. 2003; Grahne et al. 1999; Ginsburg and Wang 1998]. There are two important differences between that line of work and our work. First and foremost, the atomic data values within relations in a string database are strings, whereas the atomic data values within span relations are spans. This distinction is important because it yields a different semantics for natural join: in a string database, two tuples will join if they contain the same string in the shared attributes, whereas in span relations two tuples will join if they contain the same span. As we show in Section 5, it is exactly the capability of testing for equality on strings that causes loss of closure under difference. A second important difference is that query languages for string databases not only support pattern-matching for the purpose of extracting relevant information from strings, but also support powerful operations for the purpose of transforming strings. Typically, these transformation operations even make the query language Turing-complete in the class of string-to-string functions that can be expressed. In contrast, we focus on pattern matching, which has low complexity.

A database query language that is closely related to regular spanners is the language of *Conjunctive Regular Path Queries* (CRPQs) [Consens and Mendelzon 1990; Calvanese et al. 2000a, 2000b; Deutsch and Tannen 2001; Florescu et al. 1998]. We analyze in depth the relationship between CRPQs and our spanners in Section 6.

There is also a large body of work in extending finite state automata (or regular expressions) with mechanisms such as variables or registers. For example, Grumberg et al. [2010] study variable automata. These are simple extensions to finite state automata in which the finite alphabet consists not only of letters, but also of variables that range over an infinite additional alphabet in order to be able to accept strings formed over an infinite alphabet. In contrast, the automata we consider accept only strings

over a finite alphabet, and assign to each variable a span. Neven and Schwentick [2002] study the expressive power of *query automata* on strings and trees. These automata define mappings from input strings or trees to sets (i.e., unary relations) of positions in the input. Spanners, in contrast, define mappings from input strings to relations of arbitrary arity over the spans of the input. Barceló et al. [2013] study the extension of regular expressions with variables. In this extension, a variable can be substituted for a single alphabet letter only. In contrast, our variables bind to spans. A different extension of regular expressions with variables is given by the so called *extended regular expressions* [Aho 1990; Câmpeanu et al. 2003; Carle and Narendran 2009; Freydenberger 2011; Friedl 2006]. Here, variables can not only bind to a substring during matching, but can also be used to repeat a previously matched substring. We analyze in depth the relationship between extended regular expressions and spanners in Section 6.

Classic rule-based information extraction systems build upon the *Common Pattern Specification Language* (or CPSL) [Appelt and Onyshkevych 1998], where information extraction rules are specified based on *cascaded finite-state transducers*. The idea behind these transducers is similar to the notion of *attribute grammars* [Knuth 1968, 1971]: rules are used to parse (parts of) the input, and each rule can be assigned an action defining the values of attributes to be associated to the matched part of the input. (These attributes are considered to be the "extracted information".) While Neven and Van den Bussche [2002] have investigated the expressive power of attribute grammars in querying derivation trees generated by a fixed context-free grammar, we are not aware of any formal investigation of the expressive power of the cascaded finite-state string transducers employed by CPSL. This is probably due to the fact that CPSL does not have a formal semantics. Instead, it explicitly leaves important details to the discretion of the implementation system designer. In addition, CPSL provides many extensions to standard finite state transducers, most notably a complex disambiguation policy and the ability to write rule actions in a Turing complete language through calls to arbitrary user-defined functions. For these reasons, we do not directly compare our framework against CPSL.

Finally, there is a body of research rooted in Allen's seminal paper on interval algebra [Allen 1983]. In particular, while spans can be viewed as intervals, and spanners can hence be viewed as defining relations over intervals, Allen's interval algebra focuses on reasoning over relationships between intervals, but is not concerned with strings or string matching.

2. DOCUMENT SPANNERS

At its core, our focus system (SystemT) implements a textual query language (AQL) that translates the input string into a collection of relations; in turn, those relations are manipulated in a relational-database manner [Chiticariu et al. 2010]. The values in those relations are spans of the input string. Here we model the creation of those relations by the notion of a *document spanner*, which we formally define in this section. In the following section, we discuss the representation of document spanners, as well as extensions by relational operators. We begin with some preliminary concepts and terminology.

2.1. String Basics

Strings and Spans. We fix a finite alphabet Σ of *symbols*. We denote by Σ^* the set of all finite strings over Σ, and by Σ^+ the set of all finite strings of length at least one over Σ. We denote by ϵ the empty string. A *language over* Σ is a subset of Σ^*. Let $\mathbf{s} = \sigma_1 \cdots \sigma_n$ be a string with $\sigma_1, \ldots, \sigma_n \in \Sigma$. The length n of $\mathbf{s}$ is denoted by $|\mathbf{s}|$. A *span* identifies a substring of $\mathbf{s}$ by specifying its bounding indices. Formally, a span of $\mathbf{s}$ has

the form $[i, j\rangle$, where $1 \leq i \leq j \leq n+1$. If $[i, j\rangle$ is a span of $\mathbf{s}$, then $\mathbf{s}_{[i,j\rangle}$ denotes the substring $\sigma_i \cdots \sigma_{j-1}$. Note that $\mathbf{s}_{[i,i\rangle}$ is the empty string, and that $\mathbf{s}_{[1,n+1\rangle}$ is $\mathbf{s}$. We note that the more standard notation would be $[i, j)$, but we use $[i, j\rangle$ to distinguish spans from intervals. For example, $[1, 1)$ and $[2, 2)$ are both the empty interval, hence equal, but in the case of spans we have $[i, j\rangle = [i', j'\rangle$ if and only if $i = i'$ and $j = j'$ (and in particular, $[1, 1\rangle \neq [2, 2\rangle$). We denote by $\mathsf{Spans}(\mathbf{s})$ the set of all the spans of $\mathbf{s}$. Two spans $[i, j\rangle$ and $[i', j'\rangle$ of $\mathbf{s}$ *overlap* if $i \leq i' < j$ or $i' \leq i < j'$, and are *disjoint* otherwise. Finally, $[i, j\rangle$ *contains* $[i', j'\rangle$ if $i \leq i' \leq j' \leq j$.

Example 2.1. In a running example that we will use throughout the paper, we fix the alphabet $\Sigma = \{\mathtt{A}, \mathtt{a}, \mathtt{B}, \mathtt{b}, \_\}$ where we think of $\_$ as representing a space between words. Figure 1 shows two strings $\mathbf{s}$ and $\mathbf{t}$ in Σ^*. Later we discuss the tables in this figure. To clarify the meaning of the spans we mention, we write the index under each character of the strings. The span $[22, 26\rangle$ is a span of $\mathbf{s}$ (but not of $\mathbf{t}$, since $22 > |\mathbf{t}| + 1 = 12$) and we have $\mathbf{s}_{[22,26\rangle} = \mathtt{Abaa}$. Also, $\mathbf{s}_{[1,4\rangle}$ and $\mathbf{t}_{[1,4\rangle}$ are both $\mathtt{Aaa}$.

Regular Expressions. Regular expressions over Σ are defined by the language

$$\gamma := \emptyset \mid \epsilon \mid \sigma \mid \gamma \vee \gamma \mid \gamma \cdot \gamma \mid \gamma^*$$

where $\emptyset$ is the empty set, ϵ is the empty string, and $\sigma \in \Sigma$. Note that "$\vee$" is the disjunction operator, "$\cdot$" is the concatenation operator, and "*" is the Kleene-star operator. We use γ^+ as an abbreviation of $\gamma \cdot \gamma^*$. Moreover, by abuse of notation, if $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$, then we use Σ itself as an abbreviation of the regular expression $\sigma_1 \vee \cdots \vee \sigma_k$. The language recognized by a regular expression γ (i.e., the set of strings $\mathbf{s} \in \Sigma^*$ that γ matches) is denoted by $\mathcal{L}(\gamma)$. A language L over Σ is *regular* if $L = \mathcal{L}(\gamma)$ for some regular expression γ.

String Relations. An *n-ary string relation* is a (possibly infinite) subset of $(\Sigma^*)^n$. We will refer to the following well-known classes of string relations: recognizable relations, regular relations (sometimes also called synchronized relations), and rational relations. Here, a k-ary string relation R is called recognizable if it is a finite union of Cartesian products $L_1 \times \cdots \times L_k$, where each L_i is a regular language over Σ. A *regular* string relation is, informally, a relation that is recognized by an automaton with a head on each string in the tuple of question, such that the heads advance in a synchronized manner. A *rational* string relation is similarly defined, except that the heads can advance in an asynchronous manner. We refer the reader to Barceló et al. [2012a] for formal definitions of these classes, as well as a discussion on the relationships between these classes. We denote by REC the class of all recognizable string relations, and by REC_k the class of all recognizable relations of arity k. Similarly, we denote by REG (REG_k) the class of all (k-ary) regular relations, and by RAT (RAT_k) the class of all (k-ary) rational relations. It is known that $\mathrm{REC}_1 = \mathrm{REG}_1 = \mathrm{RAT}_1$ (they all give the regular languages), and that $\mathrm{REC}_k \subsetneq \mathrm{REG}_k \subsetneq \mathrm{RAT}_k$ for all $k > 1$.

Span Relations. We fix an infinite set SVars of *span variables*, which may be assigned spans. The sets Σ^* and SVars are disjoint. For a finite set $V \subseteq \mathsf{SVars}$ of variables and a string $\mathbf{s} \in \Sigma^*$, a $(V, \mathbf{s})$-*tuple* is a mapping $\mu \colon V \to \mathsf{Spans}(\mathbf{s})$ that assigns a span of $\mathbf{s}$ to each variable in V. If V is clear from the context, or V is irrelevant, we may write just "$\mathbf{s}$-tuple" instead of "$(V, \mathbf{s})$-tuple." A set of $(V, \mathbf{s})$-tuples is called a $(V, \mathbf{s})$-*relation*. A $(V, \mathbf{s})$-relation is also called a *span relation* (*over* $\mathbf{s}$). Note that a span relation is always finite, since there are only finitely many $(V, \mathbf{s})$-tuples (given that V and $\mathbf{s}$ are both finite).

2.2. Document Spanners

A *document spanner* (or just *spanner* for short) is an operator that transforms a given string into a span relation over that string. More formally, a spanner P is a function that is associated with a finite set V of variables, and that maps every string $\mathbf{s}$ to a $(V, \mathbf{s})$-relation $P(\mathbf{s})$. We denote the set V by $\mathsf{SVars}(P)$. We say that a spanner P is *n-ary* if $|\mathsf{SVars}(P)| = n$.

Example 2.2. In our running example (started in Example 2.1), we use two spanners: a ternary spanner P_1 and a binary spanner P_2. Later, we will specify what exactly each spanner extracts from a given string. For now, the span relations (tables) in Figure 1 show the results of applying the two spanners to the strings $\mathbf{s}$ and $\mathbf{t}$ (also in the figure).

Following are some special types of spanners that we use throughout this article.

Boolean Spanners. A spanner P is *Boolean* if $\mathsf{SVars}(P) = \emptyset$. In that case, $P(\mathbf{s}) = \mathbf{true}$ means that $P(\mathbf{s})$ consists of the empty $\mathbf{s}$-tuple, and $P(\mathbf{s}) = \mathbf{false}$ means that $P(\mathbf{s}) = \emptyset$. If P is Boolean, then we say that P *recognizes* the language of strings that evaluate to **true**.

Hierarchical Spanners. Let P be a spanner. Let $\mathbf{s} \in \Sigma^*$ be a string, and let $\mu \in P(\mathbf{s})$ be an $\mathbf{s}$-tuple. We say that μ is *hierarchical* if for all variables $x, y \in \mathsf{SVars}(P)$ one of the following holds: (1) the span $\mu(x)$ contains $\mu(y)$, (2) the span $\mu(y)$ contains $\mu(x)$, *or* (3) the spans $\mu(x)$ and $\mu(y)$ are disjoint. As an example, the reader can verify that all the tuples in Figure 1 are hierarchical. We say that P is *hierarchical* if μ is hierarchical for all $\mathbf{s} \in \Sigma^*$ and $\mu \in P(\mathbf{s})$. Observe that for two variables x and y of a hierarchical spanner, it may be the case that, over the same string, one tuple maps x to a subspan of y, another tuple maps y to a subspan of x, and a third tuple maps x and y to disjoint spans. We denote by **HS** the class of all hierarchical spanners.

Universal Spanners. Let P be a spanner. We say that P is *total on* $\mathbf{s}$ if $P(\mathbf{s})$ consists of all the $\mathbf{s}$-tuples over $\mathsf{SVars}(P)$. (Note that over a finite set of variables, there are only finitely many $\mathbf{s}$-tuples.) We say that P is *hierarchically total on* $\mathbf{s}$ if $P(\mathbf{s})$ consists of all the hierarchical $\mathbf{s}$-tuples. Let $Y \subseteq \mathsf{SVars}$ be a finite set of variables. The *universal spanner over* Y, denoted Υ_Y, is the unique spanner P such that $\mathsf{SVars}(P) = Y$ and P is total on every $\mathbf{s} \in \Sigma^*$. The *universal hierarchical spanner over* Y, denoted $\Upsilon_Y^{\mathbf{H}}$, is the unique spanner P such that $\mathsf{SVars}(P) = Y$ and P is hierarchically total on every $\mathbf{s} \in \Sigma^*$.

3. SPANNER REPRESENTATION

In our system of focus (SystemT), querying an input string $\mathbf{s}$ entails two steps (conceptually) [Chiticariu et al. 2010]. In the first step, span relations over $\mathbf{s}$ are extracted by standard string-oriented tools like regular expressions with capture variables or dictionary matchers. In the second step, the final result is obtained by applying algebraic operators to the relations of the first step. We model these two steps by two corresponding types of representations for spanners. The first type is that of *primitive spanner representations*. The second type extends the first type by including operators of a relational algebra.

3.1. Primitive Spanner Representations

We introduce here three types of primitive spanner representations. The first is that of *regular-expression formulas* that extend regular expressions by including variables. The second and third are special automata that we call *variable-stack* and *variable-set* automata.

3.1.1. Regex Formulas. A *regular expression with capture variables*, or just *variable regex* for short, is an expression in the following syntax that extends that of regular expressions:

$$\gamma := \emptyset \mid \epsilon \mid \sigma \mid \gamma \vee \gamma \mid \gamma \cdot \gamma \mid \gamma^* \mid x\{\gamma\}. \tag{1}$$

The added alternative is $x\{\gamma\}$, where $x \in \mathsf{SVars}$. The abbreviations γ^+ and Σ that we introduced for regular expressions naturally carry over to regex formulas. We denote by $\mathsf{SVars}(\gamma)$ the set of variables that occur in γ. Before we formally define how a variable regex represents a spanner, we give an example.

Example 3.1. We continue with our running example. Consider the variable regex γ_1 that is defined by

$$(\Sigma^* \cdot \_)^* \cdot z\{x\{\gamma_{\mathsf{1stCap}}\} \cdot \_ \cdot y\{\gamma_{\mathsf{1stCap}}\}\} \cdot (\_ \cdot \Sigma^*)^* \tag{2}$$

where γ_{1stCap} is the regular expression $(\mathtt{A} \vee \mathtt{B}) \cdot (\mathtt{a} \vee \mathtt{b})^*$. After we define the spanner represented by γ_1, it will turn out that γ_1 has the result of P_1 in Figure 1 on the strings $\mathbf{s}$ and $\mathbf{t}$. Note that $\mathsf{SVars}(\gamma_1) = \{x, y, z\}$.

We now formally define when a variable regex represents a spanner, and which spanner is represented in that case. The definition is based on the notion of a *parse tree*. In general, a *tree* is associated with an alphabet Λ of labels, and is recursively defined as follows: if $t_1, \ldots, t_n$ are trees (where $n \geq 0$) and $\lambda \in \Lambda$, then $\lambda(t_1 \cdots t_n)$ is a tree.

Let Λ be the alphabet $\Sigma \cup \mathsf{SVars} \cup \{\epsilon, \vee, \cdot, ^*\}$. Let γ be a variable regex, and let $\mathbf{s}$ be a string. We use the following inductive definition. A tree t over the alphabet Λ is a *γ-parse for* $\mathbf{s}$ if one of the following holds.

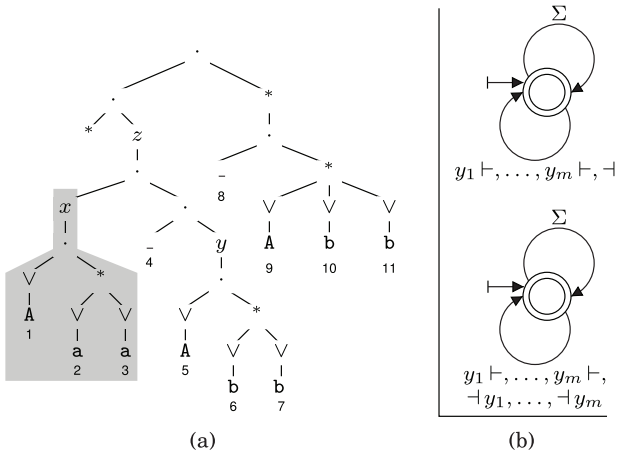
—$\gamma = \epsilon$, $\mathbf{s} = \epsilon$, and $t = \epsilon()$.
—$\gamma = \sigma \in \Sigma$, $\mathbf{s} = \sigma$, and $t = \sigma()$.
—$\gamma = \gamma_1 \vee \gamma_2$, and $t = \vee(t')$ where t' is either a γ_1-parse or a γ_2-parse for $\mathbf{s}$.
—$\gamma = \gamma_1 \cdot \gamma_2$, and $t = \cdot(t_1 t_2)$ where t_i is a γ_i-parse for $\mathbf{s}_i$ $(i = 1, 2)$ for some strings $\mathbf{s}_1$ and $\mathbf{s}_2$ such that $\mathbf{s} = \mathbf{s}_1\mathbf{s}_2$.
—$\gamma = \delta^*$ and there are strings $\mathbf{s}_1, \ldots, \mathbf{s}_n$ $(n \geq 0)$ such that $\mathbf{s} = \mathbf{s}_1 \cdots \mathbf{s}_n$, $t = {}^*(t_1 \cdots t_n)$, and each t_i is a δ-parse for $\mathbf{s}_i$ $(i = 1, \ldots, n)$.
—$\gamma = x\{\delta\}$ and $t = x(t_\delta)$ where t_δ is δ-parse for $\mathbf{s}$.

Example 3.2. We continue with our running example. Figure 2(a) shows a γ_1-parse for $\mathbf{t}$ for the variable regex γ_1 of Example 3.1 and the string $\mathbf{t}$ of Figure 1. As we did with Figure 1, we write the index under each character.

Note that there is no parse tree for the variable regex $\emptyset$. Clearly, a string $\mathbf{s}$ matches the regex γ, when variables are ignored, if and only if there exists a γ-parse for $\mathbf{s}$. In principle, a γ-parse t for $\mathbf{s}$ should determine one assignment for $\mathsf{SVars}(\gamma)$, as we later define. But for that, we need t to have *exactly* one occurrence of each variable in $\mathsf{SVars}(\gamma)$. So we restrict our variable regex to those that guarantee such a behavior of t, a property we call *functional*.

Definition 3.3. A variable regex γ is *functional* if for every string $\mathbf{s} \in \Sigma^*$ and γ-parse t for $\mathbf{s}$, each variable in $\mathsf{SVars}(\gamma)$ has precisely one occurrence in t.

Note that a variable regex can be functional even though it contains multiple occurrences of a variable. An example is the regex formula γ given by $x\{\mathtt{a}\} \vee x\{\mathtt{b}\}$, which has two occurrences of the variable x, although each γ-parse has only one occurrence of x.



(a) (b)

Fig. 2. (a) A γ_1-parse for $\mathbf{t}$ for the regex formula γ_1 of (2) (Example 3.1) and the string $\mathbf{t}$ of Figure 1 (b) A vstk-automaton A with $[\![A]\!] = \Upsilon^{\mathbf{H}}_Y$ (top) and a vset-automaton B with $[\![B]\!] = \Upsilon_Y$ (bottom) for $Y = \{y_1, \ldots, y_m\}$.

Example 3.4. Consider again the variable regex γ_1 of Example 3.1. Recall that $\mathsf{SVars}(\gamma_1) = \{x, y, z\}$. Observe that in the γ_1-parse of Figure 2(a), each variable in $\mathsf{SVars}(\gamma_1)$ has indeed exactly one occurrence. In fact, it can be easily verified that this is the case for every γ_1-parse. Consequently, γ_1 is functional.

Although Definition 3.3 is nonconstructive, functionality is a property that can be tested in polynomial time.

PROPOSITION 3.5. *Whether a given variable regex is functional can be tested in polynomial time.*

PROOF. We introduce the following inductive definition. Let γ be a regex formula, and let $Y \subseteq \mathsf{SVars}$ be a finite set of variables. We say that γ is *syntactically Y-functional* if (at least) one of the following holds.

—$\gamma = \emptyset$.
—γ is ϵ or σ for some $\sigma \in \Sigma$, and $Y = \emptyset$.
—$\gamma = \gamma_1 \vee \gamma_2$, where γ_1 and γ_2 are regex formulas that are both syntactically Y-functional.
—$\gamma = \gamma_1 \cdot \gamma_2$, where γ_1 and γ_2 are regex formulas, and there is a subset Y_1 of Y such that γ_1 is syntactically Y_1-functional, and γ_2 is syntactically Y_2-functional for $Y_2 = Y \setminus Y_1$.
—$\gamma = \delta^*$, where δ is a regex formula without variables, and $Y = \emptyset$.
—$\gamma = x\{\delta\}$, where $x \in Y$ and δ is a regex formula that is syntactically $(Y \setminus \{x\})$-functional.

A straightforward induction on the structure of γ shows that γ is functional if and only if it is syntactically $\mathsf{SVars}(\gamma)$-functional. Moreover, syntactic Y-functionality can be tested in polynomial time, given γ and Y. Consequently, whether γ is functional can be tested in polynomial time. □

The variable regexes that represent spanners are those that are functional, and we call those *regex formulas*.

Definition 3.6. A *regex formula* is a functional variable regex.

Let γ be a regex formula, and let p be a γ-parse for a string $\mathbf{s}$. If v is a node of p, then the subtree that is rooted at v naturally maps to a span p_v of $\mathbf{s}$. By μ^p we denote the

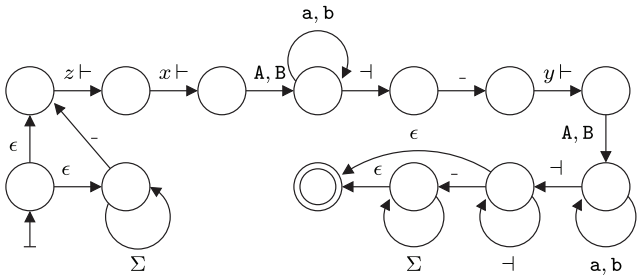


Fig. 3. A vstk-automaton A with $[\![A]\!] = [\![\gamma_1]\!]$ for the regex formula γ_1 of (2) (Example 3.1).

assignment that maps each variable x to the span $\mu^p(x) = p_v$, where v is the unique node of t that is labeled by x.

Example 3.7. Let p be the γ_1-parse of $\mathbf{t}$ depicted in Figure 2(a), where γ_1 is defined in Example 3.1 and $\mathbf{t}$ is shown in Figure 1. The subtree of p rooted at the node labeled x is shaded grey. We have $\mu^p(x) = [1, 4\rangle$, $\mu^p(y) = [5, 8\rangle$, and $\mu^p(z) = [1, 8\rangle$. Hence, μ^p is the $\mathbf{t}$-tuple μ_5 of Figure 1.

The spanner $[\![\gamma]\!]$ that is represented by the regex formula γ is the one where $\mathsf{SVars}([\![\gamma]\!])$ is the set $\mathsf{SVars}(\gamma)$ and where $[\![\gamma]\!](\mathbf{s})$ is the span relation $\{\mu^p \mid p \text{ is a } \gamma\text{-parse for } \mathbf{s}\}$.

Example 3.8. Consider again the regex formula γ_1 of Example 3.1, the strings $\mathbf{s}$ and $\mathbf{t}$ of Figure 1, and the spanner P_1 mentioned in that figure. The reader can verify that $[\![\gamma_1]\!](\mathbf{s}) = P_1(\mathbf{s})$ and that $[\![\gamma_1]\!](\mathbf{t}) = P_1(\mathbf{t})$.

3.1.2. Variable-Stack Automata. In this section, we define an automaton representation of a spanner. We call this automaton a *variable-stack automaton*, or just *vstk-automaton* for short. Later, we will show that vstk-automata capture precisely the expressive power of regex formulas (i.e., the two classes of spanner representations can express the same set of spanners).

Formally, a vstk-automaton is a tuple (Q, q_0, q_f, δ), where: Q is a finite set of *states*, $q_0 \in Q$ is an *initial state*, $q_f \in Q$ is an *accepting state*, and δ is a finite transition relation consisting of triples, each having one of the forms (q, σ, q'), (q, ϵ, q'), $(q, x\vdash, q')$ or $(q, \dashv, q')$, where $q, q' \in Q$, $\sigma \in \Sigma$, $x \in \mathsf{SVars}$, $\vdash$ is a special *push* symbol, and $\dashv$ is a special *pop* symbol.

Example 3.9. Figure 3 is a representation of a vstk-automaton A. Each circle represents a state, the double circle represents an accepting state, and a label a on an edge from q to q' represents the transition (q, a, q'). Conventionally, as a shorthand notation, we write the sequence $a_1, \ldots, a_k$ of labels on the edge from q to q' instead of the k edges $(q, a_1, q'), \ldots, (q, a_k, q')$. Moreover, if $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$, then we write the label Σ instead of $\sigma_1, \ldots, \sigma_m$. Later, we will link the vstk-automaton A to our running example.

Let A be a vstk-automaton. We denote by $\mathsf{SVars}(A)$ the set of variables that occur in the transitions of A. A *configuration* of a vstk-automaton A is a tuple $c = (q, \vec{v}, Y, i)$, where $q \in Q$ is the *current state*, $\vec{v}$ is a finite sequence of variables called the *current variable stack*, $Y \subseteq \mathsf{SVars}(A)$ is the set of *available variables*, and i is an index in $\{1, \ldots, n+1\}$ (pointing to the next character to be read from $\mathbf{s}$).

Let $\mathbf{s} = \sigma_1 \cdots \sigma_n$ be a string and let A be a vstk-automaton. A *run* ρ of A on $\mathbf{s}$ is a sequence $c_0, \ldots, c_m$ of configurations, such that $c_0 = (q_0, \epsilon, \mathsf{SVars}(A), 1)$, and

for all $j = 0, \ldots, m-1$ one of the following holds for $c_j = (q_j, \vec{v}_j, Y_j, i_j)$ and $c_{j+1} = (q_{j+1}, \vec{v}_{j+1}, Y_{j+1}, i_{j+1})$.

(1) $\vec{v}_{j+1} = \vec{v}_j$, $Y_{j+1} = Y_j$, and either
   (a) $i_{j+1} = i_j + 1$ and $(q_j, s_{i_j}, q_{j+1}) \in \delta$ (ordinary transition), or
   (b) $i_{j+1} = i_j$ and $(q_j, \epsilon, q_{j+1}) \in \delta$ (epsilon transition).
(2) $i_{j+1} = i_j$, and for some $x \in \mathsf{SVars}(A)$, either
   (a) $\vec{v}_{j+1} = \vec{v}_j \cdot x$, $x \in Y_j$, $Y_{j+1} = Y_j \setminus \{x\}$ and $(q_j, x\vdash, q_{j+1}) \in \delta$ (variable push), or
   (b) $\vec{v}_j = \vec{v}_{j+1} \cdot x$, $Y_{j+1} = Y_j$ and $(q_j, \dashv, q_{j+1}) \in \delta$ (variable pop).

An easy observation is that every configuration $(q, \vec{v}, Y, i)$ in a run is such that $\vec{v}$ and Y do not share any common variable.

The run $\rho = c_0, \ldots, c_m$ is *accepting* if $c_m = (q_f, \epsilon, \emptyset, n+1)$. We let $\mathsf{ARuns}(A, \mathbf{s})$ denote the set of all accepting runs of A on $\mathbf{s}$. If $\rho \in \mathsf{ARuns}(A, \mathbf{s})$, then for each $x \in \mathsf{SVars}(A)$ the run ρ has a unique configuration $c_b = (q_b, \vec{v}_b, Y_b, i_b)$ where x occurs in the current version of $\vec{v}$ (i.e., $\vec{v}_b$) for the first time; and later than that ρ has a unique configuration $c_e = (q_e, \vec{v}_e, Y_e, i_e)$ where x is occurs in the current version of $\vec{v}$ (i.e., $\vec{v}_e$) for the last time; the span $[i_b, i_e\rangle$ is denoted by $\rho(x)$. By μ^ρ, we denote the $\mathbf{s}$-tuple that maps each variable $x \in \mathsf{SVars}(A)$ to the span $\rho(x)$. The spanner $[\![A]\!]$ that is represented by A is the one where $\mathsf{SVars}([\![A]\!])$ is the set $\mathsf{SVars}(A)$, and where $[\![A]\!](\mathbf{s})$ is the span relation $\{\mu^\rho \mid \rho \in \mathsf{ARuns}(A, \mathbf{s})\}$.

Example 3.10. Consider the vstk-automaton A of Figure 3, described in Example 3.9. Observe that $\mathsf{SVars}(A) = \{x, y, z\}$. Note that in a run ρ, when reaching the final transition $(q, \dashv, q')$ (the leftmost occurrence of $\dashv$ in the bottom row), there is only one variable that is open, namely z. Hence, that transition can take place at most once. Moreover, if ρ is accepting, then ρ must take that transition *exactly* once, since otherwise z would not be closed.

Continuing with our running example, now consider again the regex-formula γ_1 of (2), introduced in Example 3.1. The reader can verify that A and γ_1 define the same spanner, that is, $[\![\gamma_1]\!] = [\![A]\!]$.

Example 3.11. The top part of Figure 2(b) depicts a single-state vstk-automaton A where we have $\mathsf{SVars}(A) = Y$, with $Y = \{y_1, \ldots, y_m\}$. The reader can verify that $[\![A]\!]$ is the universal hierarchical spanner $\Upsilon^{\mathbf{H}}_Y$. In particular, this example shows that the universal hierarchical spanners are expressible by vstk-automata.

3.1.3. Variable-Set Automata. A *variable-set automaton* (or *vset-automaton*) is defined to be a tuple (Q, q_0, q_f, δ) like a vstk-automaton, except δ does not have triples $(q, \dashv, q')$; instead, δ has triples $(q, \dashv x, q')$ where $x \in \mathsf{SVars}$. We denote by $\mathsf{SVars}(A)$ the set of variables that occur in the transitions of A.

The difference between the two types of automata is also in the definition of a *configuration* and a *run*. In a vset-automaton, a *set* of variables is used rather than a *stack*. More precisely, a configuration of a vset-automaton A is a tuple $c = (q, V, Y, i)$, where $q \in Q$ is the *current state*, $V \subseteq \mathsf{SVars}(A)$ is the *active variable set*, $Y \subseteq \mathsf{SVars}(A)$ is the set of *available variables*, and i is an index in $\{1, \ldots, n+1\}$.

For a string $\mathbf{s} = s_1, \ldots, s_n$, a *run* ρ of A on $\mathbf{s}$ is a sequence $c_0, \ldots, c_m$ of configurations, where $c_0 = (q_0, \emptyset, \mathsf{SVars}(A), 1)$, and for $j = 0, \ldots, m-1$ one of the following holds for $c_j = (q_j, V_j, Y_j, i_j)$ and $c_{j+1} = (q_{j+1}, V_{j+1}, Y_{j+1}, i_{j+1})$.

(1) $V_{j+1} = V_j$, $Y_{j+1} = Y_j$, and either
   (a) $i_{j+1} = i_j + 1$ and $(q_j, s_{i_j}, q_{j+1}) \in \delta$ (ordinary transition), or
   (b) $i_{j+1} = i_j$ and $(q_j, \epsilon, q_{j+1}) \in \delta$ (epsilon transition).
(2) $i_{j+1} = i_j$ and for some $x \in \mathsf{SVars}(A)$, either

(a) $x \in Y_j$, $V_{j+1} = V_j \cup \{x\}$, $Y_{j+1} = Y_j \setminus \{x\}$, and $(q_j, x\vdash, q_{j+1}) \in \delta$ (variable insert), or
(b) $x \in V_j$, $V_{j+1} = V_j \setminus \{x\}$, $Y_{j+1} = Y_j$ and $(q_j, \dashv x, q_{j+1}) \in \delta$ (variable remove).

Note that in a run, each configuration (q, V, Y, i) is such that V and Y are disjoint. The run $\rho = c_0, \ldots, c_m$ is *accepting* if $c_m = (q_f, \emptyset, \emptyset, n+1)$. The definitions of $\mathsf{ARuns}(A, \mathbf{s})$ and $[\![A]\!]$ are similar to those for a vstk-automaton (except that we replace the stack $\vec{v}$ with the set V).

Example 3.12. Consider again Figure 2(b). The bottom part depicts a single-state vset-automaton B with $\mathsf{SVars}(B) = Y$, where $Y = \{y_1, \ldots, y_m\}$. The reader can verify that $[\![B]\!] = \Upsilon_Y$. In particular, this example shows that the universal spanners are expressible by vset-automata. This example also shows that vset-automata can express spanners that regex formulas and vstk-automata cannot. In particular, an easy observation (that we later state formally in Proposition 3.14) is that the spanner defined by a regex formula, or a vstk-automaton, is necessarily hierarchical. But $[\![B]\!]$ is certainly not hierarchical.

3.1.4. Primitive Spanner Representations. We have defined three types of spanner representations. By RGX we denote the class of regex formulas, by $\mathsf{VA}_{\mathsf{stk}}$ we denote the class of vstk-automata, and by $\mathsf{VA}_{\mathsf{set}}$ we denote the class of vset-automata. We will refer to these three as our *primitive spanner representations* (to contrast with algebraic extensions of these representations).

If SR is any class spanner representations, like the primitive classes RGX, $\mathsf{VA}_{\mathsf{stk}}$, or $\mathsf{VA}_{\mathsf{set}}$, then $[\![SR]\!]$ represents the set of all the spanners representable by SR; that is, $[\![SR]\!] = \{[\![r]\!] \mid r \in SR\}$. For example, $[\![\mathsf{RGX}]\!]$ is the set of all the spanners $[\![\gamma]\!]$, where γ is a regex formula.

As mentioned in Example 3.12, every spanner defined by a regex formula or vstk-automaton is hierarchical. In our terminology, it is stated as $[\![\mathsf{RGX}]\!] \subseteq \mathbf{HS}$ and $[\![\mathsf{VA}_{\mathsf{stk}}]\!] \subseteq \mathbf{HS}$. In Example 3.12, we also mentioned that $[\![\mathsf{VA}_{\mathsf{set}}]\!] \not\subseteq \mathbf{HS}$. Later, we will show that $[\![\mathsf{RGX}]\!] = [\![\mathsf{VA}_{\mathsf{stk}}]\!]$. In fact, we will show that the class of spanners definable by a vstk-automaton is *precisely* the class of hierarchical spanners definable by a vset-automaton, or in our notation, $[\![\mathsf{VA}_{\mathsf{stk}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!] \cap \mathbf{HS}$.

3.2. Spanner Algebras

Consider a class SR of spanner representations (e.g., one of our primitive representations). We extend SR with *algebraic operator symbols* to form a *spanner algebra*. More formally, each operator symbol corresponds to a *spanner operator*, which is a function that takes as input a fixed-length sequence of spanners (usually one or two, depending on whether the operator is unary or binary), and outputs a single spanner. We now define the spanner operators we focus on in this article. Let P, P_1 and P_2 be spanners, and let $\mathbf{s}$ be a string.

—*Union*. The union $P_1 \cup P_2$ is defined when P_1 and P_2 are *union compatible*, that is, $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$. In that case, $\mathsf{SVars}(P_1 \cup P_2) = \mathsf{SVars}(P_1)$ and $(P_1 \cup P_2)(\mathbf{s}) = P_1(\mathbf{s}) \cup P_2(\mathbf{s})$.
—*Projection*. If $Y \subseteq \mathsf{SVars}(P)$, then $\pi_Y P$ is the spanner with $\mathsf{SVars}(\pi_Y P) = Y$, where $\pi_Y P(\mathbf{s})$ is obtained from $P(\mathbf{s})$ by restricting the domain of each $\mathbf{s}$-tuple to Y.
—*Natural Join*. The spanner $P_1 \bowtie P_2$ is defined as follows. We have $\mathsf{SVars}(P_1 \bowtie P_2) = \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$, and $(P_1 \bowtie P_2)(\mathbf{s})$ consists of all $\mathbf{s}$-tuples μ that agree with some $\mu_1 \in P_1(\mathbf{s})$ and $\mu_2 \in P_2(\mathbf{s})$; note that the existence of μ implies that μ_1 and μ_2 agree on the common variables of P_1 and P_2, that is, $\mu_1(x) = \mu_2(x)$ for all $x \in \mathsf{SVars}(P_1) \cap \mathsf{SVars}(P_2)$.

—*String selection.* Let R be a k-ary string relation. The string-selection operator ς^R is parameterized by k variables $x_1, \ldots, x_k$ in $\mathsf{SVars}(P)$, and may then be written as $\varsigma^R_{x_1,\ldots,x_k}$. If P' is $\varsigma^R_{x_1,\ldots,x_k} P$, then the span relation $P'(\mathbf{s})$ is taken to be the restriction of $P(\mathbf{s})$ to those $\mathbf{s}$-tuples μ such that $(\mathbf{s}_{\mu(x_1)}, \ldots, \mathbf{s}_{\mu(x_k)}) \in R$.

Regarding the natural join, observe that here pairs of tuples are joined based on having equal *spans* in shared variables. This is distinct from the natural join in query languages for string databases [Bonner and Mecca 1998; Benedikt et al. 2003; Grahne et al. 1999; Ginsburg and Wang 1998], where tuples are joined if they have the equal *substrings* in shared attributes. Also observe that in the special case where P_1 and P_2 are union compatible, the spanner $P_1 \bowtie P_2$ produces the intersection $P_1(\mathbf{s}) \cap P_2(\mathbf{s})$ for the given string $\mathbf{s}$; in that case, we denote $P_1 \bowtie P_2$ also as $P_1 \cap P_2$. As another special case, if $\mathsf{SVars}(P_1)$ and $\mathsf{SVars}(P_2)$ are disjoint, then $P_1 \bowtie P_2$ produces the Cartesian product of $P_1(\mathbf{s})$ and $P_2(\mathbf{s})$; in that case, we denote $P_1 \bowtie P_2$ also as $P_1 \times P_2$.

In this work, we focus mainly on one particular string-selection operator, namely the binary $\varsigma^=$. As defined previously, $\varsigma^=_{x,y} P(\mathbf{s})$ restricts $P(\mathbf{s})$ to those $\mathbf{s}$-tuples μ with $\mathbf{s}_{\mu(x)} = \mathbf{s}_{\mu(y)}$. Later, we also consider other string selections (featuring other binary string relations). We do not include the *difference* operator yet, but rather dedicate to it a separate discussion in Section 5.

For clarity of presentation, we will abuse notation by using the operator symbol itself to represent the spanner operator. As an example, if γ_1 and γ_2 are regex formulas, then the expression $\gamma_1 \bowtie \gamma_2$ is well formed, and it represents the spanner $[\![\gamma_1]\!] \bowtie [\![\gamma_2]\!]$. Similarly, if A_1 and A_2 are vstk-automata then $A_1 \cup A_2$ is well formed assuming union compatibility, that is, $\mathsf{SVars}(A_1) = \mathsf{SVars}(A_2)$. Similarly, if A is a vset-automaton, then $\pi_Y A$ is well formed assuming $Y \subseteq \mathsf{SVars}(A)$, and similarly $\varsigma^=_{x,y} A$ is well formed assuming $x, y \in \mathsf{SVars}(A)$.

Example 3.13. We continue with our running example. Let γ_{12} be the regex formula that captures all spans x_1 and x_2 such that x_1 ends before x_2 begins, that is

$$\gamma_{12}(x_1, x_2) \stackrel{\text{def}}{=} \Sigma^* \cdot x_1\{\Sigma^*\} \cdot \Sigma^* \cdot x_2\{\Sigma^*\} \cdot \Sigma^*$$

The following algebraic expression is denoted as γ_2.

$$\pi_{x_1,x_2}\big(\varsigma^=_{y_1,y_2}\big(\gamma_1(x_1, y_1, z_1) \bowtie \gamma_1(x_2, y_2, z_2) \bowtie \gamma_{12}(x_1, x_2)\big)\big),$$

where we use $\gamma_1(x_i, y_i, z_i)$ as the regex-formula that is obtained from γ_1 of (2) (Example 3.1) by replacing x, y and z with x_i, y_i and z_i, respectively. Observe that γ_2 selects all the spans x_1 and x_2 that occur in tuples of γ_1, such that the corresponding y_1 and y_2 span the same substrings (though y_1 and y_2 themselves are not required to be equal as spans), and moreover, x_1 ends before x_2 begins. Consider the strings $\mathbf{s}$ and $\mathbf{t}$ in Figure 1. The reader can verify that $[\![\gamma_2]\!]$ has the output of P_2 (also shown in the figure) for these two strings.

A *spanner algebra* is a finite set of spanner operators. If SR is a class of spanner representations and O is a spanner algebra, then SR^O denotes the class of all the spanner representations defined by applying (compositions of) the operators in O to the representations in SR. In other words, S^O is the closure of SR under O (when O is taken as a set of operator symbols); consequently, $[\![SR^O]\!]$ is the closure of $[\![SR]\!]$ under O (when O is now taken as a set of spanner operators). For example, one of the algebras we later explore is $\mathsf{VA}^{\{\cup,\pi,\bowtie,\varsigma^=\}}_{\mathsf{set}}$. As another example, the expression γ_2 of Example 3.13 is in $\mathsf{RGX}^{\{\pi,\bowtie,\varsigma^=\}}$.

We conclude this section with the following proposition, which relates the notion of hierarchical spanners to some of the definitions we gave here.

PROPOSITION 3.14. *Let SR be a class of spanner representations.*

(1) *If SR is* RGX *or* $\mathsf{VA}_{\mathsf{stk}}$*, then every spanner represented in SR is hierarchical (i.e.,* $[\![SR]\!] \subseteq \mathbf{HS}$*). On the other hand,* $\mathsf{VA}_{\mathsf{set}}$ *contains nonhierarchical spanners.*
(2) *The operators union, projection and string selection preserve the property of being hierarchical; that is, if* $[\![SR]\!] \subseteq \mathbf{HS}$*, then* $[\![SR^{\{\cup,\pi,\varsigma^R\}}]\!] \subseteq \mathbf{HS}$*. On the other hand, the natural join does not preserve the property of being hierarchical; that is, there are hierarchical spanners* P_1 *and* P_2*, such that* $P_1 \bowtie P_2$ *is nonhierarchical.*

PROOF. We begin with Part 1. The fact that $[\![SR]\!] \subseteq \mathbf{HS}$, when SR is one of RGX and $\mathsf{VA}_{\mathsf{stk}}$, follows straightforwardly from the way a spanner is defined in these representations. An example of a vset-automaton that represents a non-hierarchical spanner was given in Example 3.12.

For Part 2, the fact that union, projection and string selection preserve the property of being hierarchical follows straightforwardly from the definitions of these operators. Finally, a simple example of hierarchical spanners P_1 and P_2, such that $P_1 \bowtie P_2$ is non-hierarchical is $P_1 = \Upsilon^{\mathbf{H}}_X$ and $P_2 = \Upsilon^{\mathbf{H}}_Y$, where X and Y are nonempty, disjoint sets of variables (hence, $P_1 \bowtie P_2$ is a Cartesian product). □

4. REGULAR AND CORE SPANNERS

In this section, we define the classes of regular and core spanners, and study their relative expressive power.

4.1. Regular Spanners

We call a spanner *regular* if it is definable by a vset-automaton. In this section, we explore expressiveness aspects of the class of regular spanners, and of its restriction to the hierarchical spanners.

Observe that vstk-automata, vset-automata and NFAs are basically the same objects in the Boolean case. In particular, a language $L \subseteq \Sigma^*$ is recognized by some Boolean vstk-automaton if and only if it is recognized by some Boolean vset-automaton if and only if L is regular. Hence, the results of this section are of interest only in the non-Boolean case.

Key constructs that we later utilize for establishing our results here are those of a *transition graph* and the special case of a *path union*, both introduced in the next section.

4.1.1. Transition Graphs and Path Unions. We define two types of transition graphs, which function similarly to vstk-automata and vset-automata, respectively, except that in a single transition a whole substring (matching a specified regular expression) can be read, and moreover, every transition to a non-accepting state involves a single operation of opening or closing a variable. Those graphs are similar to the extended automata obtained by the known *state-removal* technique, that is used to convert an automaton into a regular expression [Linz 2001]. Recall that, throughout this article we fix the alphabet Σ for the input string language.

A *variable-stack transition graph*, or *vstk-graph* for short, is a tuple $G = (Q, q_0, q_f, \delta)$ defined similarly to a vstk-automaton, except that now δ consists of edges of three forms: $(q, \gamma, x\vdash, q')$, $(q, \gamma, \dashv, q')$ and (q, γ, q_f); here, $q, q' \in Q \setminus \{q_f\}$, γ is a regular expression over Σ, and $x \in \mathsf{SVars}$. Note that the accepting state q_f has only incoming transitions, and none of these transitions involve a variable. For example, the middle and bottom graphs in Figure 6 are vstk-graphs.

As usual, $\mathsf{SVars}(G)$ denotes the set of variables that occur in G. A *configuration* $c = (q, \vec{v}, Y, i)$ is defined exactly as in the case of a vstk-automaton, but the definition of a run changes: a run ρ of G on a string $\mathbf{s}$ is a sequence $c_0, \ldots, c_m$ of configurations,

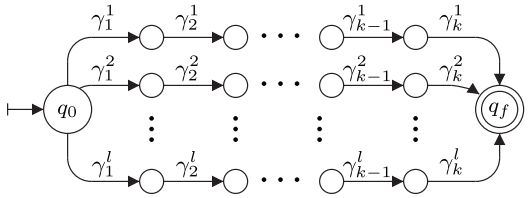


Fig. 4. An illustration of a vstk-path union or a vset-path union.

such that for all $j = 0, \ldots, m-1$, the configurations $c_j = (q_j, \vec{v}_j, Y_j, i_j)$ and $c_{j+1} = (q_{j+1}, \vec{v}_{j+1}, Y_{j+1}, i_{j+1})$ satisfy the following. First, $i_j \leq i_{j+1}$. Second, one of the following holds.

—δ contains a tuple $(q, \gamma, x\vdash, q')$, such that $q = q_j$, the string $\mathbf{s}_{[i_j, i_{j+1}\rangle}$ is in $\mathcal{L}(\gamma)$, and $q' = q_{j+1}$; moreover, $x \in Y_j$, $\vec{v}_{j+1} = \vec{v}_j \cdot x$ and $Y_{j+1} = Y_j \setminus \{x\}$. Semantically, the variable x opens right after i_{j+1}.
—δ contains a tuple $(q, \gamma, \dashv, q')$, such that $q = q_j$, the string $\mathbf{s}_{[i_j, i_{j+1}\rangle}$ is in $\mathcal{L}(\gamma)$, and $q' = q_{j+1}$; moreover, $\vec{v}_j = \vec{v}_{j+1} \cdot x$ and $Y_{j+1} = Y_j$ for some variable x. Semantically, the variable x closes right after i_{j+1}.
—δ contains a tuple (q, γ, q_f), such that $q = q_j$, the string $\mathbf{s}_{[i_j, i_{j+1}\rangle}$ is in $\mathcal{L}(\gamma)$, and $q_f = q_{j+1}$; moreover, $\vec{v}_j = \vec{v}_{j+1}$ and $Y_{j+1} = Y_j$.

The definition of an *accepting configuration* is similar to that for vstk-automata. Moreover, the definitions of $\mathsf{ARuns}(G, \mathbf{s})$ and $[\![G]\!]$ are similar to those of $\mathsf{ARuns}(A, \mathbf{s})$ and $[\![A]\!]$ in the case of a vstk-automaton A.

A vstk-graph $G = (Q, q_0, q_f, \delta)$ is a *vstk-path* if we can write Q as $\{q_0, q_1, \ldots, q_k = q_f\}$ where δ contains exactly k edges: from q_0 to q_1, from q_1 to q_2, and so on, until q_k. A vstk-path is *consistent* if the variables open and close in a balanced manner (which we define in the natural way like grammatical parentheses). We say that G is a *vstk-path union* if G is the union of consistent vstk-paths, such that: (1) every two vstk-paths have the same set of variables, namely $\mathsf{SVars}(G)$, and (2) every two vstk-paths share precisely the states q_0 and q_f, as illustrated in Figure 4 (where we omit the opening and closing of variables). For example, the bottom graph of Figure 6 is a vstk-path union.

Similarly to the vstk case, we define a *vset-graph* to be a variation of a vset-automaton. In particular, $\mathsf{ARuns}(G, \mathbf{s})$ and $[\![G]\!]$ are now defined when G is a vset-graph. Also similarly we define a *vset-path*, a *consistent vset-path* (where parenthetical balance is not required, but every variable needs to be opened and later closed exactly once), and a *vset-path union*.

By $\mathsf{VG}_{\mathsf{stk}}$ and $\mathsf{VG}_{\mathsf{set}}$ we denote the set of all vstk-graphs and vset-graphs, respectively, and by $\mathsf{PU}_{\mathsf{stk}}$ and $\mathsf{PU}_{\mathsf{set}}$ we denote the class of vstk-path unions and the class of vset-path unions, respectively.

4.1.2. Relative Expressive Power. We can now give some results on the (relative) expressive power of the regular spanners. A key lemma is the following.

LEMMA 4.1. *The following hold.*

(*1*) *Every vstk-automaton can be translated into a vstk-graph and vice-versa; that is:*

$$[\![\mathsf{VA}_{\mathsf{stk}}]\!] = [\![\mathsf{VG}_{\mathsf{stk}}]\!].$$

(*2*) *Every vset-automaton can be translated into a vset-graph and vice-versa; that is,*

$$[\![\mathsf{VA}_{\mathsf{set}}]\!] = [\![\mathsf{VG}_{\mathsf{set}}]\!].$$

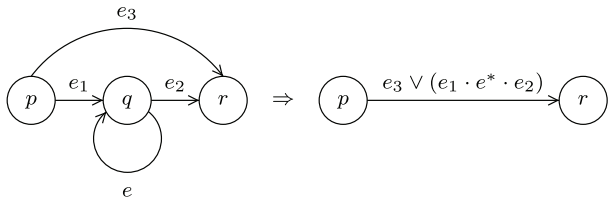


Fig. 5. Contraction of an automaton state q.

PROOF. We prove only Part 1, since the proof of Part 2 is similar. We first prove that every vstk-automaton can be translated into a vstk-graph. Let $A = (Q, q_0, q_f, \delta)$ be a vstk-automaton. We will transform A into a vstk-graph G. By using extra empty transitions (ϵ-transitions), we can assume that Q is the disjoint union of two sets Q_s and Q_v, such that:

—every triple $(q, \tau, q') \in \delta$, where τ is either ϵ or a symbol in Σ, satisfies $q' \in Q_s$;
—every triple $(q, x\vdash, q') \in \delta$ with $x \in \mathsf{SVars}$, as well as every triple $(q, \dashv, q')$, satisfies $q' \in Q_v$.

So, we make that assumption. By a similar argument, we further assume that q_0 has no incoming edges (i.e., triples with q_0 as the third element), that q_f has no outgoing edges (i.e., triples with q_f as the first element), and that q_0 and q_f are both in Q_s.

Next, we make use of a slight modification of the well-known *state-removal* procedure [Linz 2001] for translating an automaton into a regular expression. Before giving our modification, we briefly describe the original procedure. Throughout the execution of this procedure, we allow the automaton to have regular expressions on edges, and we repeatedly contract (remove) states, leaving only q_0 and q_f in the end. When a state q is contracted, we consider every edge e from an incoming neighbor p of q to an outgoing neighbor r of q (we assume that such an edge e always exists, since its expression could be $\emptyset$); we then update the regular expression on that edge to accommodate the contraction of v. For illustration, see Figure 5, and for more details see Linz [2001].

Here, when A in a vstk-automaton, we apply the very same procedure with one exception: we contract all the states in Q_s except for q_s and q_f, and *none* of the states in Q_v. Clearly, in the end, the result is a vstk-graph.

For illustration, the top box of Figure 6 depicts an example of the vstk-automaton A, and the middle box depicts the resulting vstk-graph. The states of Q_v are colored grey.

The other direction, that every vstk-graph can be translated into a vstk-automaton, is straightforward. Assume that $G = (Q, q_0, q_f, \delta)$ is a vstk-graph. Then, we simply replace every regular expression γ with a collection of vstk-automaton triples by injecting an ordinary automaton (with a fresh set of states) that corresponds to the expression γ. □

LEMMA 4.2. *The following holds:*

(*1*) *Every vstk-graph can be translated into a vstk-path unions, that is,*

$$[\![\mathsf{VG}_{\mathsf{stk}}]\!] = [\![\mathsf{PU}_{\mathsf{stk}}]\!].$$

(*2*) *Every vset-graph can be translated into a vset-path unions, that is,*

$$[\![\mathsf{VG}_{\mathsf{set}}]\!] = [\![\mathsf{PU}_{\mathsf{set}}]\!].$$

PROOF. Again, we will prove only Part 1, since the proof of Part 2 is similar. We need to prove that every vstk-graph G can be translated into a vstk-path union G'. So, let $G = (Q, q_0, q_f, \delta)$ be a vstk-graph. Let $\mathcal{P}(G)$ be the set of all the paths from q_0 to q_f in G. Even though G is finite, it could happen that $\mathcal{P}(G)$ is infinite, because of the possible presence of cycles in G. We view $\mathcal{P}(G)$ as a collection of vstk-paths. Let $\mathcal{P}^c(G)$ be the

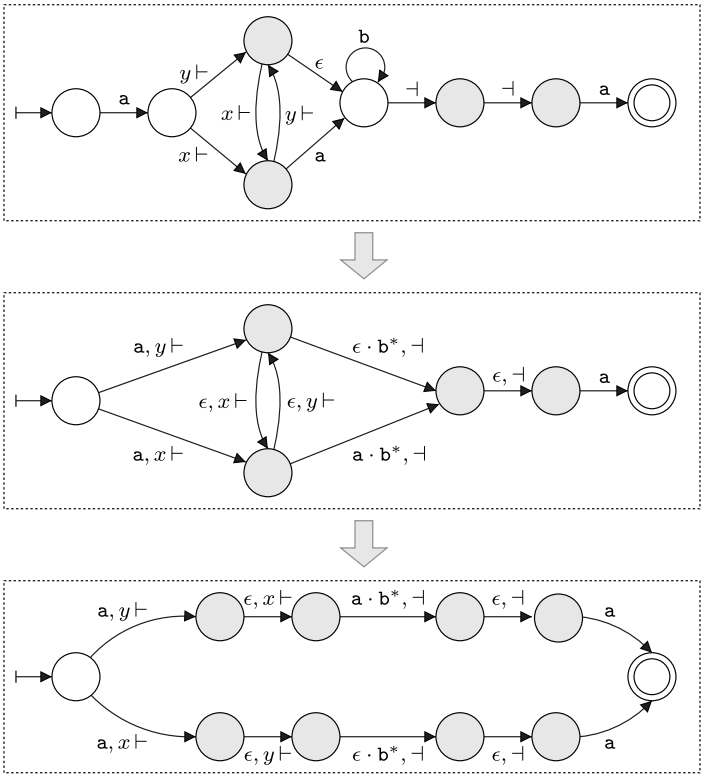


Fig. 6. Translating a vstk-automaton into a vstk-graph and then into a vstk-path union.

subset of $\mathcal{P}(G)$ that consists of all the vstk-paths P, such that: (1) $\mathsf{SVars}(P) = \mathsf{SVars}(G)$, and (2) P is consistent. The following are easy observations.

(1) $\mathcal{P}^c(G)$ is finite (since every path in $\mathcal{P}^c(G)$ has precisely $2k+1$ edges, where k is the number of variables in G).
(2) $[\![G]\!] = \bigcup_{P \in \mathcal{P}^c(G)} [\![P]\!]$. Indeed, obviously, $[\![G]\!] \supseteq \bigcup_{P \in \mathcal{P}^c(G)} [\![P]\!]$. Moreover, since, by definition of run of a vstk-automaton, every variable must be opened and closed in a run of G and since this must happen in a hierarchical manner, every run of G must also be a run of some $P \in \mathcal{P}^c$. Therefore, $[\![G]\!] \subseteq \bigcup_{P \in \mathcal{P}^c(G)} [\![P]\!]$.

So, we construct the vstk-path union G' by simply merging the first state of every path into the single initial state q_0, and merging the last state of every path into the single accepting state q_f, while treating each path as if its set of states is disjoint from that of any other path, except for the first and last states (namely q_0 and q_f). For illustration, the bottom box of Figure 6 depicts an example of the vstk-path union G', when starting with the vstk-graph G from the middle box. Due to Observation 1, the resulting G' is indeed a (finite) vstk-path union, and due to Observation 2, we have that $[\![G]\!] = [\![G']\!]$. This completes the proof. □

By combining Lemmas 4.1 and 4.2, we get the following.

LEMMA 4.3. *The following hold.*

(*1*) *Every spanner definable by a vstk-automaton is definable by a vstk-path union and vice-versa; that is,* $[\![\mathsf{VA}_{\mathsf{stk}}]\!] = [\![\mathsf{PU}_{\mathsf{stk}}]\!]$.

(2) *Every spanner definable by a vset-automaton is definable by a vset-path union and vice-versa; that is,* $[\![\mathsf{VA}_{\mathsf{set}}]\!] = [\![\mathsf{PU}_{\mathsf{set}}]\!]$.

Our first theorem states that regex formulas and vstk-automata have the same expressive power.

THEOREM 4.4. *A spanner is definable by a vstk-automaton if and only if it is definable by a regex formula; that is,* $[\![\mathsf{VA}_{\mathsf{stk}}]\!] = [\![\mathsf{RGX}]\!]$.

PROOF. We first prove that $[\![\mathsf{VA}_{\mathsf{stk}}]\!] \subseteq [\![\mathsf{RGX}]\!]$. Let A be a vstk-automaton. Part 1 of Lemma 4.2 implies that A can be translated into a vstk-path union. Converting a consistent vstk-path into an equivalent regex formula is straightforward—every state with an incoming $x\vdash$ becomes "$\cdot x\{$" and every state with an incoming $\dashv$ becomes "$\}\cdot$." Hence, translating a vstk-path union into a regex formula is also straightforward using the disjunction operator.

Next, we prove that $[\![\mathsf{RGX}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{stk}}]\!]$. This is done by a straightforward adaptation of the standard construction by Thompson (see, e.g., Linz [2001]), namely, incremental construction of an automaton from a regular expression through a bottom-up traversal of the parse of a regular expression. □

Next, we prove that the spanners definable by vstk-automata are precisely the spanners that are both regular and hierarchical. We do so by combining Lemma 4.3 with the following lemma.

LEMMA 4.5. *If P is a vset-path such that $[\![P]\!]$ is hierarchical, then there is a vstk-path P' such that $[\![P']\!] = [\![P]\!]$.*

PROOF. Let P be a vset-path such that $[\![P]\!]$ is hierarchical. We denote P as follows:

$$q_s[\gamma_0] \to v_1 q_1[\gamma_1] \to \cdots \to v_k q_k[\gamma_k] \to v_{k+1} q_{k+1}[\gamma_{k+1}] \to q_f.$$

where q_s, q_f and the q_i are the states, each γ_i is the regular expression on the edge between its preceding and following states, and each v_i is either $x\vdash$ or $\dashv x$ for some variable $x \in \mathsf{SVars}(P)$. We say that.

—v_i *precedes* v_j if $i < j$;
—v_i *weakly precedes* v_j if either $i \leq j$ or each of the regular expressions γ_l after v_j and before v_i is equivalent to ϵ;
—v_i *strongly precedes* v_j if v_i precedes v_j and v_j does not weakly precede v_i (i.e., some regular expression γ_l after v_i and before v_j is not equivalent to ϵ).

We now define two relations, $\preceq$ and $\rightsquigarrow$, over $\mathsf{SVars}(P)$. Let x and y be variables in $\mathsf{SVars}(P)$. We define.

—$x \preceq y$ if $x\vdash$ weakly precedes $y\vdash$ and $\dashv y$ weakly precedes $\dashv x$;
—$x \rightsquigarrow y$ if x and y are incomparable by $\preceq$ (i.e., neither $x \preceq y$ nor $y \preceq x$ holds) and $x\vdash$ precedes $y\vdash$.

Intuitively, $x \preceq y$ says that it is possible to reorder the operations of opening and closing the variables x and y so that y opens and closes inside the span where x opens and closes. Observe the following. First, every x and y are comparable by exactly one of $\preceq$ and $\rightsquigarrow$. Second, if $x \rightsquigarrow y$, then $x\vdash$ strongly precedes $y\vdash$ and $\dashv x$ strongly precedes $\dashv y$. Consequently, from the fact that P is hierarchical we conclude that $\dashv x$ weakly precedes $y\vdash$; otherwise, we can construct a counterexample (i.e., a string $\mathbf{s}$ and an output $\mathbf{s}$-tuple that is not hierarchical) by replacing each regular expression γ_l with a string that is nonempty whenever possible. Another observation is that $\preceq$ is a partial order (regardless of P being hierarchical); we fix a linear extension $\ll$ of $\preceq$.

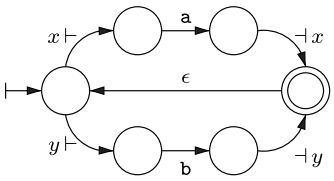


Fig. 7. A vset-automaton A illustrating the problem with naively eliminating variables.

We define the order $\rightsquigarrow$ over the operations v_i as follows. For all variables x and y, we have.

—if $x \rightsquigarrow y$, then $(x\vdash) \rightsquigarrow (\dashv x) \rightsquigarrow (y\vdash) \rightsquigarrow (\dashv y)$.
—if $x \ll y$, then $(x\vdash) \rightsquigarrow (y\vdash) \rightsquigarrow (\dashv y) \rightsquigarrow (\dashv x)$.

From these observations, we conclude that $\rightsquigarrow$ is a linear order over the operations v_i. Let P' be obtained from P by reordering the operations v_i according to $\rightsquigarrow$. Observe that to obtain P', we switched between operations only when all the regular expressions in between are equivalent to ϵ. Consequently, we have not changed the semantics (i.e., the spanner defined by) P, or in other words, $[\![P']\!] = [\![P]\!]$. Moreover, the operations in P' are balanced in a hierarchical manner, and consequently, by replacing each $\dashv x$ with $\dashv$ we get an equivalent vstk-path. □

We then get the following theorem.

THEOREM 4.6. *A spanner is definable by a vstk-automaton if and only if it is both regular and hierarchical; that is,* $[\![\mathsf{VA}_{\mathsf{stk}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!] \cap \mathbf{HS}$.

Next, we prove that union, projection and natural-join operators do not increase the expressive power of vset-automata. We begin with the union operator, which is straightforward to handle due to the fact that a vset-automaton is allowed to have ϵ-transitions. Therefore, we omit the proof of the following lemma.

LEMMA 4.7. $[\![\mathsf{VA}_{\mathsf{set}}^{\{\cup\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

Next, we consider the projection operator.

LEMMA 4.8. $[\![\mathsf{VA}_{\mathsf{set}}^{\{\pi\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

PROOF. Let A be a vset-automaton, and let Y be a subset of $\mathsf{SVars}(A)$. We need to construct an automaton A' such that $[\![A']\!] = [\![\pi_Y A]\!]$. One would be tempted to believe that the construction of A is straightforward: simply ignore the variables that are not in Y by replacing the transitions that involve them with empty transitions. However, this operation may result in a vset-automaton A'' such that $[\![A'']\!]$ is actually a strict superset of $[\![\pi_Y A]\!]$, since the need to assign spans to all the variables in $\mathsf{SVars}(A)$ restricts the set of accepting paths. We illustrate this issue next.

Consider the vset-automaton A of Figure 7, and suppose that $Y = \{x\}$. This automaton maps only `ab` and `ba` to nonempty sets of assignments (where in each x is assigned the span of `a` and y is assigned the span of `b`). In particular, $[\![A]\!]$ maps the string `a` to the empty set of assignments, and so does $[\![\pi_Y A]\!]$. But if we replaced $y\vdash$ with ϵ and $\dashv y$ with ϵ, then the resulting automaton A'' would be such that $[\![A'']\!]$ maps `a` to a nonempty set of assignments, namely the singleton mapping x to $[1, 2\rangle$.

Nevertheless, it is easy to verify that this simplistic approach (i.e., simulating projection by removing variables in a straightforward manner) would work correctly on a vset-path union; the only difference would be that some obvious node contractions would need to take place to eliminate the new states that involve no opening or closing

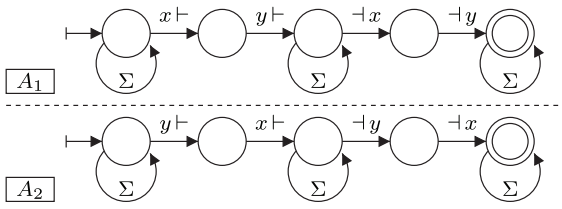


Fig. 8. Two vset-automata with equal spanners.

of variables. Consequently, we get the following procedure to push π_Y into A. First, translate A into a vset-path union G with $[\![A]\!] = [\![G]\!]$, which we can do according to Lemma 4.3. Second, apply the projection to G and get the graph G' with $[\![G']\!] = \pi_Y[\![G]\!]$. Finally, translate G' into a vset-automaton A' with $[\![A']\!] = [\![G']\!]$, which we can do, again by Lemma 4.3. We then get the vset-automaton A' with $[\![A']\!] = [\![\pi_Y A]\!]$, as required. □

The final operator we consider is the natural join. This proof involves a subtlety. The expected approach is similar to intersecting two NFAs: a vset-automaton for $A_1 \bowtie A_2$ runs on A_1 and A_2 in parallel. Moreover, when a variable x is common to both automata, the two parallel runs must open and close x together (after all, x must be the same span in both runs in taking the join). This approach, however, fails, for a subtle reason. As an example, A_1 and A_2 of Figure 8 are such that $[\![A_1]\!] = [\![A_2]\!] = [\![A_1 \bowtie A_2]\!]$. However, our construction for A_1 and A_2 will result in the empty spanner, since A_1 requires x to open before y (with an epsilon transition in between), and A_2 requires x to open after y. We solve this problem by converting A_1 and A_2 into a *normalized form* where common tuples necessarily correspond to "similar" runs (and we will again use Lemma 4.3 for that). More precisely, we use the notion of a *lexicographic vset-automaton*, which we define next.

Let $\mathbf{s} \in \Sigma^*$ be a string, and let μ be a $(V, \mathbf{s})$-tuple for some finite set V of variables. A *V-operation* is an expression of the form $x\vdash$ or $\dashv x$, where $x \in V$. If $\mu(x) = [i, j\rangle$, then we define $pos(x\vdash) = i$ and $pos(\dashv x) = j$. A *storyline* of μ is a sequence $\lambda = \langle o_1, \ldots, o_m\rangle$ such that

—$o_1, \ldots, o_m$ consists of all the V-operations, without repetition;
—For all $x \in V$, the operation $x\vdash$ occurs before $\dashv x$;
—For all V-operations o and o', if $pos(o) < pos(o')$, then o occurs before o' in λ.

Observe that whenever $pos(o) = pos(o')$ and o and o' involve different variables, we can switch between o and o' and still get a storyline. In particular, an assignment can have multiple storylines.

As an example, suppose that $V = \{x, y, z\}$ and that $\mu(x) = [1, 5\rangle$, $\mu(y) = [1, 3\rangle$ and $\mu(z) = [3, 5\rangle$. Following are some of the storylines for $\mathbf{s}$.

—$\lambda_1 = \langle x\vdash, y\vdash, \dashv y, z\vdash, \dashv z, \dashv x\rangle$.
—$\lambda_2 = \langle y\vdash, x\vdash, \dashv y, z\vdash, \dashv z, \dashv x\rangle$.
—$\lambda_3 = \langle y\vdash, x\vdash, z\vdash, \dashv y, \dashv z, \dashv x\rangle$.
—$\lambda_4 = \langle x\vdash, y\vdash, \dashv y, z\vdash, \dashv x, \dashv z\rangle$.

We denote by $SL(\mu)$ the set of all storylines for μ. Let A be a vset-automaton with $\mathsf{SVars}(A) = V$. For a run $\rho \in \mathsf{ARuns}(A, \mathbf{s})$, we denote by $sl(\rho)$ the sequence of V-operations of ρ, in their chronological order.

We fix a linear order $\preceq$ on the set of all operations $x\vdash$ and $\dashv x$, such that for every variable y we have $y\vdash \preceq \dashv y$. We extend $\preceq$ to the set of all storylines, lexicographically. A story line λ is said to be *$\preceq$-minimal* for μ if $\lambda \preceq \lambda'$ for all $\lambda' \in SL(\mu)$. As an example, suppose that $x\vdash \preceq \dashv x \preceq y\vdash \preceq \dashv y \preceq z\vdash \preceq \dashv z$. Then storyline λ_4 from our previous example

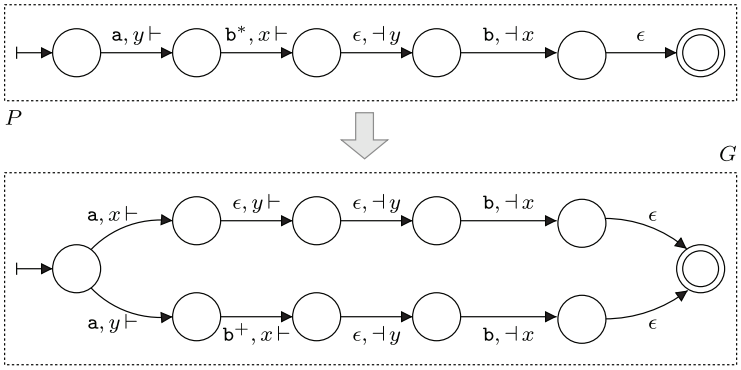


Fig. 9. An example of the transformation of a vset-path P into a lexicographic vset-path union G.

is $\preceq$-minimal for μ. Since $\preceq$ is a linear order over the storylines, every $\mathbf{s}$-tuple μ has a unique $\preceq$-minimal storyline; so we refer to this storyline as *the minimal storyline* of μ.

Let A be a vset-automaton. We say that A is *lexicographic* if for all strings $\mathbf{s} \in \Sigma^*$ and $\mathbf{s}$-tuples $\mu \in A(\mathbf{s})$, the set $\mathsf{ARuns}(A, \mathbf{s})$ contains a run ρ such that $sl(\rho)$ is the minimal storyline of μ.

LEMMA 4.9. *If A is a vset-automaton, then there is a lexicographic vset-automaton A' such that $[\![A]\!] = [\![A']\!]$.*

PROOF. We naturally extend the definition of $sl(\rho)$ to the case where ρ is a run of a vset-graph. We also define a vset-graph to be *lexicographic* similarly to the case of a vset-automaton. Let P be a vset-path. We will prove that there exists a lexicographic vset-path union G, such that $[\![P]\!] = [\![G]\!]$. This is enough, for the following reason. Lemma 4.3 states that a vset-automaton A can be translated into a path-union U. We show here that every path P of U can be translated into a lexicographic vset-path union G, and consequently, we can translate U into a lexicographic vset-path union U'. Finally, it is an easy observation that U' (just like any vset-graph) can be translated into a vset-automaton in a storyline-preserving manner.

Recall that the edges of the vset-path P contain regular expressions γ. Let G' be the vset-path union that consists of all the paths P' that are obtained from P by replacing each γ with γ' where;

—$\gamma' = \gamma$ if $\epsilon \notin \mathcal{L}(\gamma)$,
—γ' is either γ'' or ϵ if $\epsilon \in \mathcal{L}(\gamma)$, where γ'' is a regular expression such that $\mathcal{L}(\gamma'') = \mathcal{L}(\gamma) \setminus \{\epsilon\}$. Note that γ'' can be avoided if $\mathcal{L}(\gamma) = \{\epsilon\}$ (since then γ'' is empty).

(In the second case, G' gets two paths, one with γ' as γ'', and one with γ' as ϵ.) Notice that, even though G' can be exponentially larger than P, we still have $[\![G']\!] = [\![P]\!]$.

Next, we obtain G from G' by applying the following operation to G' until we can no longer change it. Find an edge e of G' that emanates from an operation o to an operation o', such that $o' \preceq o$, and the regular expression on e is ϵ; then switch between o and o'.

For illustration, Figure 9 shows an example of the vset-path P (top box), and the resulting vset-path union G (bottom box). In this example we assume that $x\vdash \preceq \dashv x \preceq y\vdash \preceq \dashv y$. Note that the top path in G is obtained from P by replacing $\mathtt{b}^*$ with ϵ, and switching between $y\vdash$ and $x\vdash$. The bottom path in G is obtained from P by replacing $\mathtt{b}^*$ with $\mathtt{b}^+$, and no switching took place. Also note that we never switched between $\dashv y$ and $\dashv x$, since the regular expression between them (namely $\mathtt{b}$) does not accept ϵ.

Easy observations are that this switching process terminates, and that the resulting G satisfies $[\![G]\!] = [\![G']\!]$. To complete the proof, we need to show that G is indeed

lexicographic. So, let ρ be a run of G, and let λ be $sl(\rho)$. Then, ρ is a run over one of the vset-paths that comprise G, say P. Suppose that o and o' are two consecutive operations in λ that we can switch between and still retain a storyline for μ^ρ. This means that o and o' involve different variables, and $pos(o) = pos(o')$. To prove that G is lexicographic, we need to show that $o \preceq o'$. The fact that $pos(o) = pos(o')$ means that the span between o to o' is empty. So, it must be the case that the regular expression in the P, between o and o', is ϵ, since no other regular expression in G accepts ϵ. Consequently, $o \preceq o'$ must hold, or otherwise $o' \preceq o$, and we can still switch between o and o' (in contradiction to the fact that our switching operation is applied until no change is possible). □

LEMMA 4.10. *Let A_1 and A_2 be two lexicographic vset-automata. There is a vset-automaton A, such that $[\![A]\!] = [\![A_1 \bowtie A_2]\!]$.*

PROOF. Let $A_1 = (Q_1, q_1^0, q_1^f, \delta_1)$ and $A_2 = (Q_2, q_2^0, q_2^f, \delta_2)$ be two vset-automata. We will construct a vset-automaton A such that $[\![A]\!] = [\![A_1]\!] \bowtie [\![A_2]\!]$. Let $Y_1 = \mathsf{SVars}(A_1)$ and $Y_2 = \mathsf{SVars}(A_2)$. The construction is similar to the construction of the intersection of two NFAs: we run on both automata in parallel, making only steps that are allowed by both automata. The difference is in handling the variables. When A_1 wants to open or close a variable in $Y_1 \setminus Y_2$, we allow it to do so without a state change for A_2. Similarly, when A_2 wants to open or close a variable in $Y_2 \setminus Y_1$, we allow it to do so without a state change for A_1. However, a variable in $Y_1 \cap Y_2$ must be opened and closed simultaneously by both automata. Next, we give a more formal construction of A.

We define $A = (Q, q^0, q^f, \delta)$ where

—$Q = Q_1 \times Q_2$;
—$q^0 = \langle q_1^0, q_2^0 \rangle$; and
—$q^f = \langle q_1^f, q_2^f \rangle$.
—δ has the following transitions.

(i) $(\langle q_1, q_2 \rangle, \sigma, \langle q_1', q_2' \rangle)$ whenever $\sigma \in \Sigma$, $(q_1, \sigma, q_1') \in \delta_1$ and $(q_2, \sigma, q_2') \in \delta_2$.
(ii) $(\langle q_1, q_2 \rangle, \epsilon, \langle q_1', q_2' \rangle)$ whenever (1) $(q_1, \epsilon, q_1') \in \delta_1$ and $q_2 = q_2'$, *or* (2) $q_1 = q_1'$ and $(q_2, \epsilon, q_2') \in \delta_2$.
(iii) $(\langle q_1, q_2 \rangle, x\vdash, \langle q_1', q_2' \rangle)$ whenever one of the following holds:
  (1) $x \in Y_1 \setminus Y_2$, $(q_1, x\vdash, q_1') \in \delta_1$ and $q_2 = q_2'$;
  (2) $x \in Y_2 \setminus Y_1$, $q_1 = q_1'$ and $(q_2, x\vdash, q_2') \in \delta_2$.
(iv) $(\langle q_1, q_2 \rangle, \dashv x, \langle q_1', q_2' \rangle)$ whenever one of the following holds:
  (1) $x \in Y_1 \setminus Y_2$, $(q_1, \dashv x, q_1') \in \delta_1$ and $q_2 = q_2'$;
  (2) $x \in Y_2 \setminus Y_1$, $q_1 = q_1'$ and $(q_2, \dashv x, q_2') \in \delta_2$.
(v) $(\langle q_1, q_2 \rangle, x\vdash, \langle q_1', q_2' \rangle)$ whenever we have $x \in Y_1 \cap Y_2$ and $(q_i, x\vdash, q_i') \in \delta_i$ for $i = 1, 2$.
(vi) $(\langle q_1, q_2 \rangle, \dashv x, \langle q_1', q_2' \rangle)$ whenever we have $x \in Y_1 \cap Y_2$ and $(q_i, \dashv x, q_i') \in \delta_i$ for $i = 1, 2$.

The proof that $[\![A]\!] = [\![A_1]\!] \bowtie [\![A_2]\!]$ has two directions. To show that $[\![A]\!] \subseteq [\![A_1]\!] \bowtie [\![A_2]\!]$, we split a run of A on a string $\mathbf{s}$ into two consistent runs of A_1 and A_2. This is straightforward, and omitted.

To show that $[\![A_1]\!] \bowtie [\![A_2]\!] \subseteq [\![A]\!]$, let $\mathbf{s} \in \Sigma^*$ be a string, and let $\mu_1 \in A_1(\mathbf{s})$ and $\mu_2 \in A_2(\mathbf{s})$ be two $\mathbf{s}$-tuples that agree on the common variables. Let μ be the $\mathbf{s}$-tuple that produces all the assignments of both μ_1 and μ_2. We need to show the existence of a run $\rho \in \mathsf{ARuns}(A, \mathbf{s})$ that produces μ. Let $\rho_1 \in \mathsf{ARuns}(A_1, \mathbf{s})$ and $\rho_2 \in \mathsf{ARuns}(A_2, \mathbf{s})$ be runs for μ_1 and μ_2, respectively. It is easy to construct the desired run ρ by a simple combination of ρ_1 and ρ_2, if we assume that $sl(\rho_1)$ and $sl(\rho_2)$ are *consistent* with each other: if an operation o occurs before o' in ρ_1 and both o and o' are in ρ_2, then o should occur before o' in ρ_2 as well. We omit the obvious details of this construction in this

case. But generally, $sl(\rho_1)$ and $sl(\rho_2)$ need not be consistent with each other. So here, we use the assumption that A_1 and A_2 are both lexicographic, and select ρ_1 and ρ_2 as these with the minimal storylines; so now, $sl(\rho_1)$ and $sl(\rho_2)$ are indeed consistent. □

By combining Lemmas 4.9, and 4.10, we get the following lemma.

LEMMA 4.11. $[\![\mathsf{VA}_{\mathsf{set}}^{\{\bowtie\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

By combining Lemmas 4.7, 4.8, and 4.11, we get the following theorem.

THEOREM 4.12. *The class of regular spanners is closed under union, projection and natural join; that is,* $[\![\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

Finally, we prove that to express all regular spanners, it suffices to enrich the vstk-automata with union, projection and join. We use the following lemma.

LEMMA 4.13. $[\![\mathsf{VA}_{\mathsf{set}}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{stk}}^{\{\cup,\pi,\bowtie\}}]\!]$.

PROOF. We will prove the following. Let P be a consistent vset-path. Then there is an expression E in $\mathsf{RGX}^{\{\pi,\bowtie\}}$, such that $[\![P]\!] = [\![E]\!]$. That suffices for proving the lemma, since Lemma 4.3 states that every vset-automaton can be converted into a union of consistent vset-paths, and Theorem 4.4 states that regex formulas and vstk-automata are equivalent in terms of expressive power.

We denote P in the following natural way:

$$q_s[\gamma_0] \to v_1 q_1[\gamma_1] \to \cdots \to v_k q_k[\gamma_k] \to v_{k+1} q_{k+1}[\gamma_{k+1}] \to q_f,$$

where q_s, q_f and the q_i are the states, each γ_i is the regular expression on the edge between its preceding and following states, and each v_i is the operation in $\{x\vdash, \dashv x\}$ for some variable x, that takes place when entering q_i.

We first construct a regex formula γ. The spanner $[\![\gamma]\!]$ is not equivalent to $[\![P]\!]$ (and in fact has different variables), but later we will join $[\![\gamma]\!]$ with other regex-defined spanners (and apply needed projection) to get a spanner that is indeed equivalent to $[\![P]\!]$.

The variables of γ have the form y_O^C, where O and C are (disjoint) subsets of $\mathsf{SVars}(P)$. In a run of P, the set O represents the set of variables of P that are *open* (i.e., have been opened and not closed yet), and C represents the set of variables of P that are *closed* (i.e., have been opened and later closed). The regex-formula γ is the following:

$$y_{O_0}^{C_0}\{\gamma_0\} \cdot y_{O_1}^{C_1}\{\gamma_1\} \cdot y_{O_2}^{C_2}\{\gamma_2\} \cdots y_{O_k}^{C_k}\{\gamma_k\} \cdot y_{O_{k+1}}^{C_{k+1}}\{\gamma_{k+1}\},$$

The O_i and C_i are inductively defined as follows.

—O_0 and C_0 are both $\emptyset$.
—For $1 \le i \le k+1$, we consider two cases:
  —If v_i is $x\vdash$, then $O_i = O_{i-1} \cup \{x\}$ and $C_i = C_{i-1}$.
  —If v_i is $\dashv x$, then $O_i = O_{i-1} \setminus \{x\}$ and $C_i = C_{i-1} \cup \{x\}$.

As an example, suppose that P is the following vset-path:

$$q_s[\gamma_0] \to x\vdash q_1[\gamma_1] \to z\vdash q_2[\gamma_2] \to \dashv x\ q_3[\gamma_3] \to \dashv z\ q_4[\gamma_4] \to q_f. \tag{3}$$

Then γ will be the following regex formula.

$$y_{\emptyset}^{\emptyset}\{\gamma_0\} \cdot y_{\{x\}}^{\emptyset}\{\gamma_1\} \cdot y_{\{x,z\}}^{\emptyset}\{\gamma_2\} \cdot y_{\{z\}}^{\{x\}}\{\gamma_3\} \cdot y_{\emptyset}^{\{x,z\}}\{\gamma_4\}. \tag{4}$$

An important observation about our construction of γ is that for every variable $x \in \mathsf{SVars}(P)$, there are indices i and j with $i \le j$, such that the variables y_O^C with $x \in O$ form the sequence $y_{O_i}^{C_i}, \ldots, y_{O_j}^{C_j}$.

Let x be a variable in $\mathsf{SVars}(P)$. Let $y_{O_i}^{C_i}, \ldots, y_{O_j}^{C_j}$ be the sequence of y_O^C with $x \in O$. We now construct a new regex-formula γ_x that expresses the spanner $[\![\gamma_x]\!]$ with the following two properties:

—$\mathsf{SVars}([\![\gamma_x]\!]) = \{x, y_{O_i}^{C_i}, \ldots, y_{O_j}^{C_j}\}$;

—the spanner $[\![\gamma_x]\!]$ produces every assignment where the spans assigned to every $y_{O_l}^{C_l}$ and $y_{O_{l+1}}^{C_{l+1}}$ are consecutive and adjacent, for $l \in \{i, \ldots, j-1\}$, and moreover, x is the span that contains all the $y_{O_l}^{C_l}$ for $l \in \{i, \ldots, j\}$.

More formally, γ_x can be defined by the following regex-expression.

$$\Sigma^* \cdot x\{y_{O_i}^{C_i}\{\Sigma^*\} \cdots y_{O_j}^{C_j}\{\Sigma^*\}\} \cdot \Sigma^*.$$

For example, consider again the vset-path P of (3) and the resulting γ of (4). For the variable z, the regex formula γ_z is the following:

$$\Sigma^* \cdot z\{y_{\{x,z\}}^{\emptyset}\{\Sigma^*\} \cdot y_{\{z\}}^{\{x\}}\{\Sigma^*\}\} \cdot \Sigma^*.$$

Let $\mathsf{SVars}(P) = \{x_1, \ldots, x_n\}$. We construct a spanner F in $[\![\mathsf{RGX}^{\{\pi,\bowtie\}}]\!]$, equivalent to $[\![P]\!]$, by the following expression:

$$F \stackrel{\text{def}}{=} \pi_{\mathsf{SVars}(P)} \left([\![\gamma]\!] \bowtie [\![\gamma_{x_1}]\!] \bowtie \cdots \bowtie [\![\gamma_{x_n}]\!] \right).$$

By following our construction, one can verify that, indeed, we have $F = [\![P]\!]$. □

We then obtain the following theorem.

THEOREM 4.14. $[\![\mathsf{VA}_{\mathsf{stk}}^{\{\cup,\pi,\bowtie\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

PROOF. The proof is by the following argument.

$$[\![\mathsf{VA}_{\mathsf{stk}}^{\{\cup,\pi,\bowtie\}}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{stk}}^{\{\cup,\pi,\bowtie\}}]\!].$$

The first containment is due to Theorem 4.6, the equality is due to Theorem 4.12, and the second containment is due to Lemma 4.13. □

4.1.3. Simulation of String Relations. Let R be a k-ary string relation, and let $\mathcal{C}$ be a class of spanners. We say that R is *selectable by* $\mathcal{C}$ if for every spanner $P \in \mathcal{C}$ and sequence $\vec{x} = x_1, \ldots, x_k$ of variables in $\mathsf{SVars}(P)$, the spanner $\varsigma_{\vec{x}}^R P$ is also in $\mathcal{C}$. Let $\vec{x} = x_1, \ldots, x_k$ be a sequence of span variables, and let $X = \{x_1, \ldots, x_k\}$. The *R-restricted universal spanner over $\vec{x}$*, denoted $\Upsilon_{\vec{x}}^R$, is the spanner $\varsigma_{\vec{x}}^R \Upsilon_X$. (Recall that Υ_X is the universal spanner over X.) The following (straightforward) proposition states that under some assumptions (that hold in all the spanner classes of our interest), selectability of R is equivalent to the ability to define the R-restricted universal spanners. We will later use this proposition as a tool to decide whether or not a relation R is selectable by a class of spanners at hand. The proof is straightforward, hence omitted.

PROPOSITION 4.15. *Let R be a string relation, and let $\mathcal{C}$ be a class of spanners. Assume that $\mathcal{C}$ contains all the universal spanners, and that $\mathcal{C}$ is closed under natural join. The relation R is selectable by $\mathcal{C}$ if and only if $\Upsilon_{\vec{x}}^R \in \mathcal{C}$ for all $\vec{x} \in \mathsf{SVars}^k$.*

Let REC_k be as defined in Section 2.1. Thus, a k-ary string relation R is in REC_k if and only if it is a finite union of Cartesian products $L_1 \times \cdots \times L_k$, where each L_i is a regular language over Σ. Proposition 4.15 easily implies that every recognizable relation is selectable by the regular spanners. Interestingly, the other direction is also true.

THEOREM 4.16. *A string relation is selectable by the regular spanners if and only if it is recognizable. That is,* REC *is precisely the class of string relations selectable by* $[\![\mathsf{VA}_{\mathsf{set}}]\!]$.

PROOF.

The "If" Direction. Let R be a string relation in REC_k. Let $\vec{x} = x_1, \ldots, x_k$ be a sequence of k variables. By Proposition 4.15, it suffices to show that $\Upsilon^R_{\vec{x}}$ is a regular spanner. And due to Theorems 4.4 and 4.14, it suffices to show that $\Upsilon^R_{\vec{x}}$ is definable in the representation language $\mathsf{RGX}^{\{\cup,\pi,\bowtie\}}$. By definition, R can be represented as

$$R = \bigcup_{i=1}^{m} L_1^i \times \cdots \times L_k^i,$$

where each L_j^i is a regular language. Note that each of the Cartesian products is of the same number k of elements, where k is the number of variables in $\vec{x}$. For each L_j^i, we select a regular expression γ_j^i with $\mathcal{L}(\gamma_j^i) = L_j^i$. Then, $\Upsilon^R_{\vec{x}}$ is defined by the following spanner.

$$\bigcup_{i=1}^{m} \left(\Sigma^* \cdot x_1\{\gamma_1^i\} \cdot \Sigma^*\right) \times \cdots \times \left(\Sigma^* \cdot x_k\{\gamma_k^i\} \cdot \Sigma^*\right).$$

The "Only If" Direction. Suppose that R is a k-ary string relation that is selectable by the regular spanners. We need to show that $R \in \mathrm{REC}_k$. Let $x_1, \ldots, x_k$ be k distinct variables. Let P be the spanner defined by the following expression in $\mathsf{RGX}^{\{\cup,\pi,\bowtie\}}$.

$$x_1\{\Sigma^*\} \cdot x_2\{\Sigma^*\} \cdots x_k\{\Sigma^*\}. \tag{5}$$

That is, given a string $\mathbf{s}$, the spanner P breaks $\mathbf{s}$ into k consecutive spans and assigns the jth span to x_j. Since R is selectable by the regular spanners, the spanner $P^R = \varsigma^R_{x_1,\ldots,x_k}(P)$ is also regular. And due to Theorems 4.14 and 4.4 and Lemma 4.3, there is a vset-path union U, such that $[\![U]\!] = P^R$. We fix such a vset-path union U.

To prove that $R \in \mathrm{REC}_k$, we will show that R can be represented as a union $\bigcup_{i=1}^{m} L_1^i \times \cdots \times L_k^i$, where each L_j^i is a regular language. To do so, in the remainder of the proof we will show that this union can be obtain directly from U, where each path in U corresponds to a product $L_1^i \times \cdots \times L_k^i$, and each L_j^i is the language defined by the regular expression between $x_j\vdash$ and $\dashv x_j$.

Let $p_1, \ldots, p_m$ be the paths of U. Fix an i in $\{1, \ldots, m\}$. Let x and x' be two distinct variables in $\{x_1, \ldots, x_k\}$, and suppose that γ is a regular expression in the intersection of the scopes of x and x' inside p_i. Due to the definition of P^R, the expression γ matches only the empty string; otherwise, we can easily construct a string $\mathbf{s}$ and an $\mathbf{s}$-tuple μ, such that $\mu \in [\![p_i]\!](\mathbf{s})$, and $\mu(x)$ and $\mu(x')$ overlap, so consequently, by the special form of (5), we have $\mu \notin P^R(\mathbf{s})$. For a similar reason, we can assume that the edge emanating from the start state q_0 is labeled with the regular expression ϵ, as is the edge entering the final state q_f. As a result, we can reorder the nodes of p_i so that it has the form of each of the paths of the vset-path union in Figure 10. Consequently, we can assume that U is actually the vset-path union in that figure.

We refer to the regular expressions γ_j^i in Figure 10. Consider the following string relation R'.

$$R' \stackrel{\text{def}}{=} \bigcup_{i=1}^{m} \mathcal{L}\left(\gamma_1^i\right) \times \cdots \times \mathcal{L}\left(\gamma_k^i\right).$$

Clearly, $R' \in \mathrm{REC}_k$. To complete our proof, we will show that $R' = R$.

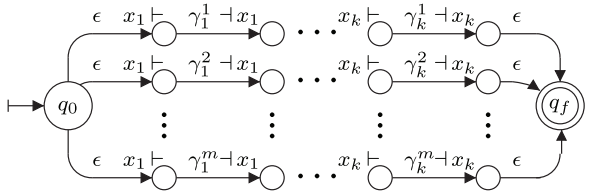


Fig. 10. The vset-path union U in the proof of Theorem 4.16.

We begin by showing $R' \subseteq R$. Let $t = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ be a member of R'. We need to prove that $t \in R$. Suppose that t belongs to $\mathcal{L}(\gamma_1^i) \times \cdots \times \mathcal{L}(\gamma_k^i)$. Let $\mathbf{s}$ be the string $\mathbf{s}_1 \cdots \mathbf{s}_k$. Clearly, p_i (hence, U) has an accepting run ρ on $\mathbf{s}$ that produces the $\mathbf{s}$-tuple μ^ρ, such that each x_j is assigned the span that corresponds to $\mathbf{s}_j$. In particular, $[\![U]\!] = P^R$ implies that $\mu^\rho \in P^R(\mathbf{s})$. But then, the definition of P^R implies that t is in R, as claimed.

We now prove that $R \subseteq R'$. Let $t = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ be a member of R. We need to prove that $t \in R'$. Let $\mathbf{s}$ be the string $\mathbf{s}_1 \cdots \mathbf{s}_k$. By the definition of P^R, the set $P^R(\mathbf{s})$ contains the $\mathbf{s}$-tuple μ such that each x_j is assigned the span that corresponds to $\mathbf{s}_j$. Consequently, since $[\![U]\!] = P^R$, there is an accepting run ρ of U on $\mathbf{s}$, such that $\mu^\rho = \mu$. Moreover, because U is a vset-path union, ρ is actually a run of one of the paths, say p_i, on $\mathbf{s}$. Therefore, it must be the case that each $\mathbf{s}_j$ belongs to $\mathcal{L}(\gamma_j^i)$. We conclude that $t \in \mathcal{L}(\gamma_1^i) \times \cdots \times \mathcal{L}(\gamma_k^i)$, and hence, $t \in R'$, as claimed. □

4.2. Core Spanners

As the core of AQL we identify the algebra $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}$. Henceforth, we call a spanner in $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$ a *core spanner*. A consequence of Theorems 4.4 and 4.14 is that the algebra $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}$ has the same expressive power as $\mathsf{VA}_{\mathsf{stk}}^{\{\cup,\pi,\bowtie,\varsigma^=\}}$ and $\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie,\varsigma^=\}}$. Therefore, the core spanners are obtained from the regular spanners by extending the algebra with the selection operator $\varsigma^=$.

To reason about the expressiveness of core spanners, we will use the following sequence of lemmas.

LEMMA 4.17. *Let F_1 and F_2 be two union-compatible spanners in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\varsigma^=\}}]\!]$. The spanner $F_1 \cup F_2$ is expressible in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\pi,\varsigma^=\}}]\!]$.*

PROOF. We denote each F_i as $S_i(A_i)$, where S_i is a sequence of string-equality selections and $A_i \in \mathsf{VA}_{\mathsf{set}}$.

Let $\varsigma^=_{x,y}$ be one of the string-equality selections in S_1. Let z be a fresh variable (that is not in the A_i). We construct from A_1 and A_2 two vset-automata A_1' and A_2', with $\mathsf{SVars}(A_1') = \mathsf{SVars}(A_2') = \mathsf{SVars}(A_1) \cup \{z\}$, as follows.

—A_1' is the same as A_1, with z taking exactly the span of y (i.e., z opens and closes exactly when y does).
—A_2' is the same as A_2, with z taking exactly the span of x.

Consider the string-equality selection $\varsigma^=_{x,z}$. When applied to A_1', it is equivalent to $\varsigma^=_{x,y}$. But $\varsigma^=_{x,z}$ is always true in A_2', since there x and z are always assigned the same span. Let S_1' be obtained from S_1 by removing $\varsigma^=_{x,y}$. Then, $F_1 \cup F_2$ is equal to the spanner

$$\pi_Y\, \varsigma^=_{x,z} \left(S_1'(A_1') \cup S_2(A_2')\right),$$

where Y is the set of variables in F_1 (and hence in F_2 since F_1 and F_2 are union-compatible). In particular, we managed to pull out one string-equality selection from

S_1. We continue doing so until we completely eliminate S_1. Note that we do not need to add the projection π_Y in the elimination of the remaining string-equality selections. We also eliminate S_2 in an analogous manner. Consequently, in the end, we get an expression of the form $\pi_Y S(B_1 \cup B_2)$, where S is a sequence of string-equality selections and the B_i are vset-automata. We then use Theorem 4.12 to replace $B_1 \cup B_2$ with a single vset-automaton B, and consequently get the expression $\pi_Y S(B) \in [\![\mathsf{VA}_{\mathsf{set}}^{\{\pi,\varsigma^=\}}]\!]$, as required. □

LEMMA 4.18. $[\![\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}^{\{\pi,\varsigma^=\}}]\!]$.

PROOF. We first make the following observation. Suppose that F is a spanner in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\pi,\varsigma^=\}}]\!]$. By definition, F is equal to some $Q(A)$ where Q is a sequence of projection and string-equality selections. Note that we can push all the projections to the beginning of Q. Furthermore, we can assume that Q contains exactly one projection, namely $\pi_{\mathsf{SVars}(F)}$, in the beginning. Consequently, we can assume that F has the form $\pi_{\mathsf{SVars}(F)}S(A)$ where S is a sequence of string-equality selections and $A \in \mathsf{VA}_{\mathsf{set}}$.

We associate with each spanner F in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$ a fixed algebraic expression that defines F. Now, let F be a spanner in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$. We need to prove that F is in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\pi,\varsigma^=\}}]\!]$. The proof is by induction on the number of algebraic operators used for defining F. The (trivially true) basis of the induction is where $F = [\![A]\!]$ for some $A \in \mathsf{VA}_{\mathsf{set}}$. For the inductive step, we consider several cases.

Case 1. $F = F_1 \cup F_2$. We denote $\mathsf{SVars}(F)$ by Y. Note that F_1 and F_2 are union compatible, that is, $\mathsf{SVars}(F_1) = \mathsf{SVars}(F_2) = Y$. By the induction hypothesis (and this observation), each F_i is equal to some $\pi_Y S_i(A_i)$, where S_i is a sequence of string-equality selections and $A_i \in \mathsf{VA}_{\mathsf{set}}$. Let $Y_1 = \mathsf{SVars}(A_1)$ and $Y_2 = \mathsf{SVars}(A_2)$. Without loss of generality, we can assume that $Y_1 \cap Y_2 = Y$; this is true since we can rename the variables of A_2 that are not in Y. Consequently, we get the following. (Recall that Υ_V is the universal spanner over the variable set V.)

$$\begin{aligned} F &= \pi_Y((S_1(A_1) \bowtie \Upsilon_{Y_2 \setminus Y}) \cup (S_2(A_2) \bowtie \Upsilon_{Y_1 \setminus Y})) \\ &= \pi_Y((S_1(A_1 \bowtie \Upsilon_{Y_2 \setminus Y})) \cup (S_2(A_2 \bowtie \Upsilon_{Y_1 \setminus Y})). \end{aligned}$$

Recall from Example 3.12 that every universal spanner Υ_V is in $[\![\mathsf{VA}_{\mathsf{set}}]\!]$. Consequently, using Theorem 4.12 we conclude that each of $S_1(A_1 \bowtie \Upsilon_{Y_2 \setminus Y})$ and $S_2(A_2 \bowtie \Upsilon_{Y_1 \setminus Y})$ is in $[\![\mathsf{VA}_{\mathsf{set}}^{\{\varsigma^=\}}]\!]$. Therefore, we get the stated claim as a consequence of Lemma 4.17.

Case 2. $F = \pi_Y(F')$. By the induction hypothesis F' is equal to some $\pi_{Y'}S'(A')$, where S' is a sequence of string-equality selections and $A' \in \mathsf{VA}_{\mathsf{set}}$. We assume that $F = \pi_Y(F')$ is well defined, which in particular implies that $Y \subseteq \mathsf{SVars}(F') = Y'$. We then get that F is equal to $\pi_Y S'(A')$.

Case 3. $F = F_1 \bowtie F_2$. By the induction hypothesis, each F_i is equal to some $\pi_Y S_i(A_i)$, where S_i is a sequence of string-equality selections and $A_i \in \mathsf{VA}_{\mathsf{set}}$. We have the following:

$$F = S_1(A_1) \bowtie S_2(A_2) = S_1 S_2(A_1 \bowtie A_2).$$

Case 4. $F = \varsigma^=_{x,y}(F')$. By the induction hypothesis F' is equal to some $\pi_{Y'}S'(A')$, where S' is a sequence of string-equality selections and $A' \in \mathsf{VA}_{\mathsf{set}}$. An easy observation is that we can apply the string equality only after the projection $\pi_{Y'}$ (note that both x and y are in Y'). Hence, we get that $F = \pi_{Y'}S(A')$, where S is obtained from S' by adding the operator $\varsigma^=_{x,y}$. □

The following lemma is a key tool for reasoning about the expressiveness of core spanners. This lemma, which we call the *core-simplification lemma*, states that every core spanner can be defined by a very simple expression: a single vset-automaton, on top of which we apply string-equality selections, and finally a single projection. The proofs of the inexpressibility results we later give for core spanners are inherently based on this result.

LEMMA 4.19 (CORE-SIMPLIFICATION LEMMA). *Every core spanner is definable by an expression of the form $\pi_V SA$, where A is a vset-automaton, $V \subseteq \mathsf{SVars}(A)$, and S is a sequence of selections $\zeta^{=}_{x,y}$ for $x, y \in \mathsf{SVars}(A)$.*

PROOF. Lemma 4.18 is not quite enough to prove this lemma, since an expression in $\mathsf{VA}^{\{\pi,\zeta^{=}\}}_{\mathsf{set}}$ is not necessarily of the form $\pi_V SA$, but rather OA, where O is a sequence of projection and string-equality operators. To obtain the special form $\pi_V SA$, we use the easy observation that in the case of OA, only one projection is needed, and that projection can be applied at the very end. □

Next, we discuss selectable relations. Observe that string equality, which is obviously selectable by the core spanners, is not selectable by the regular spanners, because string equality is not in REC (and because of Theorem 4.16). Another way of seeing that is as follows: if string equality were selectable by the regular spanners, then a Boolean regular spanner (which can be represented as an NFA) could recognize the nonregular language $\{\mathbf{s} \cdot \mathbf{s} \mid \mathbf{s} \in \Sigma^*\}$ by $\pi_\emptyset\, \zeta^{=}_{x,y}(x\{\Sigma^*\} \cdot y\{\Sigma^*\})$.

Let $\mathbf{s}$ and $\mathbf{t}$ be two strings. By $\mathbf{s} \sqsubseteq \mathbf{t}$, we denote that $\mathbf{s}$ is a (consecutive) substring of $\mathbf{t}$ (i.e., $\mathbf{s}$ is equal to some $\mathbf{t}_{[i,j\rangle}$). By $\mathbf{s} \sqsubseteq_{\mathsf{prf}} \mathbf{t}$, we denote that $\mathbf{s}$ is a prefix of $\mathbf{t}$ (i.e., $\mathbf{s}$ is equal to some $\mathbf{t}_{[1,j\rangle}$). By $\mathbf{s} \sqsubseteq_{\mathsf{sfx}} \mathbf{t}$, we denote that $\mathbf{s}$ is a suffix of $\mathbf{t}$ (i.e., $\mathbf{s}$ is equal to some $\mathbf{t}_{[i,|\mathbf{t}|+1\rangle}$).

Next, we will use Proposition 4.15 to show that the binary substring relation $\sqsubseteq$ is selectable by the core spanners. Due to Proposition 4.15, it suffices to show that the spanner $\Upsilon^{\sqsubseteq}_{x,y}$ is definable in $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\zeta^{=}\}}]\!]$. Let $\gamma(x', y)$ be the spanner that captures the property that x' is a subspan of y. We can define $\gamma(x', y)$ by $\Sigma^* \cdot y\{\Sigma^* \cdot x'\{\Sigma^*\} \cdot \Sigma^*\} \cdot \Sigma^*$. Then, $\Upsilon^{\sqsubseteq}_{x,y}$ is defined by

$$\pi_{\{x,y\}}\, \zeta^{=}_{x,x'} \left(\Upsilon_{\{x,x',y\}} \bowtie \gamma(x', y)\right).$$

Similar constructions show that the relations $\sqsubseteq_{\mathsf{prf}}$ and $\sqsubseteq_{\mathsf{sfx}}$ are also selectable by the core spanners. We record this as a proposition, for later use. We also include in the proposition the fact that every relation in REC is also selectable by the core spanners; the proof is by the same argument that precedes Theorem 4.16.

PROPOSITION 4.20. *Every string relation in* REC*, as well as each of the string relations $\sqsubseteq$, $\sqsubseteq_{\mathsf{prf}}$, and $\sqsubseteq_{\mathsf{sfx}}$, is selectable by the core spanners.*

The next theorem will be used for showing that the classes of regular and rational relations are incomparable with the class of relations selectable by the core spanners. Informally, a *regular* string relation is a relation that is recognized by an automaton with a head on each string in the tuple of question, such that the heads advance in a synchronized manner. A *rational* string relation is similarly defined, except that the heads can advance in an asynchronous manner. We refer the reader to Barceló et al. [2012a] for more formal definitions of these classes.

THEOREM 4.21. *The language $\{0^m 1^m \mid m \in \mathbb{N}\}$ is not recognizable by any Boolean core spanner.*

PROOF. Let $L = \{0^m 1^m \mid m \in \mathbb{N}\}$. Assume, by way of contradiction, that L is recognizable by a core spanner. Due to the core-simplification lemma, we can assume

that this core spanner is $\pi_\emptyset SA$, where A is a vset-automaton and S is a sequence of string selections $\varsigma^{=}_{x,y}$ for $x, y \in \mathsf{SVars}(A)$. Associate with each string $\mathbf{s} = 0^m1^m$ a run $\rho_\mathbf{s} \in \mathsf{ARuns}(A, \mathbf{s})$ such that the string selections hold for the $\mathbf{s}$-tuple defined by $\rho_\mathbf{s}$.

If (q, V, Y, i) is a configuration, let us refer to (q, V, Y) as a *semiconfiguration*. If $\mathbf{s} \in L$, and (q, V, Y, i) is the configuration in the run $\rho_\mathbf{s}$ where i is the location of the first 1 in $\mathbf{s}$, then call (q, V, Y, i) the *middle configuration*, and (q, V, Y) the *middle semiconfiguration*. Since there are only finitely many semiconfigurations, it follows that there is some infinite set $L' \subset L$ such that every member of L' has the same middle semiconfiguration. Let $\mathbf{s}_1 = 0^m1^m$ and $\mathbf{s}_2 = 0^n1^n$ be two distinct members of L'. Let $\mathbf{s} = 0^m1^n$. Then, $\mathbf{s} \notin L$.

It is easy to see that there is a run of A that accepts $\mathbf{s}$: this run (a) starts out with the configurations of the run $\rho_{\mathbf{s}_1}$ up to and including its middle configuration, and (b) continues with that portion of $\rho_{\mathbf{s}_2}$ that starts just after its middle configuration but which has the index values i' in its configurations (q', V', Y', i') suitably modified. Let ρ be such a run. Although ρ accepts $\mathbf{s}$, there is still the question of whether the selections $\varsigma^{=}_{x,y}$ continue to hold; we shall show that this is the case. This gives us a contradiction, since then our Boolean spanner accepts a string not in L.

Let ρ_1 and ρ_2 be the runs $\rho_{\mathbf{s}_1}$ and $\rho_{\mathbf{s}_2}$, respectively. Take one of the selections $\varsigma^{=}_{x,y}$. We need only show that $\varsigma^{=}_{x,y}$ holds in ρ. We now consider several possibilities. Assume first that x opens in 0^m in ρ_1 and closes in 1^m in ρ_1. In particular, the value of x contains both 0's and 1's. Since $\varsigma^{=}_{x,y}$ holds in ρ_1, necessarily y must open in the same position as x in ρ_1, in order for the values of x and y to have the same number of 0's. Since (a) ρ_1 and ρ are the same up to the middle configuration of ρ_1, and (b) x and y open in the same position in 0^m in $\mathbf{s}_1$, it follows that they also open in the same position in 0^m in ρ. Since ρ_1 and ρ_2 have the same middle semi-configuration, and since x opens in 0^m in ρ_1 and closes in 1^m in ρ_1, it follows that x opens in 0^n in ρ_2 and closes in 1^n in ρ_2. Since $\varsigma^{=}_{x,y}$ holds in ρ_2, necessarily y must close in the same position as x in ρ_2, in order for the values of x and y to have the same number of 1's. So by considering ρ_2 and ρ, which have the same run (with the index values suitably modified) starting with the middle configuration of ρ_2, it follows that x and y close in the same position in 1^n in ρ. So x and y open in the same position in ρ and close in the same position in ρ; hence, $\varsigma^{=}_{x,y}$ holds in ρ, as desired. By reversing the roles of x and y, we see that $\varsigma^{=}_{x,y}$ holds in ρ if y opens in 0^m in ρ_1 and closes in 1^m in ρ_1.

So we can now assume that x opens in 0^m in ρ_1 and closes in 0^m in ρ_1, or else x opens in 1^m in ρ_1 and closes in 1^m in ρ_1, and similarly for y. Let us say that x and y are *split* in ρ_1 if one of them opens and closes in 0^m in ρ_1 and the other opens and closes in 1^m in ρ_1. If x and y are not split, then assume without loss of generality, that they each open and close in 0^m in ρ_1. Since $\varsigma^{=}_{x,y}$ holds in ρ_1, it also holds in ρ.

Finally, assume that x and y are split in ρ_1. Assume without loss of generality, that x opens and closes in 0^m in ρ_1, and y opens and closes in 1^m in ρ_1. Since ρ_1 and ρ_2 have the same middle semiconfiguration, and this semiconfiguration reflects opening and closing of variables, also x opens and closes in 0^n in ρ_2, and y opens and closes in 1^n in ρ_2. Since (a) $\varsigma^{=}_{x,y}$ holds in ρ_1, (b) the value of x is of the form 0^i for some $i \geq 0$, and (c) the value of y is of the form 1^j for some $j \geq 0$, it follows that the values of x and y are both the empty string. Similarly, the values of x and y in ρ_2 are both the empty string. Now the value of x in ρ is the same as the value of x in ρ_1, namely the empty string. Further, the value of y in ρ is the same as the value of y in ρ_2, namely the empty string. Therefore, $\varsigma^{=}_{x,y}$ holds in ρ, as desired. □

The existence of a regular relation that is not selectable by the core spanners is due to the following theorem.

THEOREM 4.22. *Assume that the alphabet Σ contains at least two symbols. The string relation $\{(\mathbf{s}, \mathbf{t}) \mid |\mathbf{s}| = |\mathbf{t}|\}$ is not selectable by the core spanners.*

PROOF. Without loss of generality, we assume that Σ contains the symbols 0 and 1 (and, possibly, additional symbols). Let R be the relation $\{(\mathbf{s}, \mathbf{t}) \mid |\mathbf{s}| = |\mathbf{t}|\}$, and suppose, by way of contradiction, that R is selectable by the core spanners. So the following expression defines a core spanner:

$$\pi_{\emptyset}\, \varsigma^{R}_{x,y} \left(x\{0^*\} \cdot y\{1^*\}\right).$$

Then again, the spanner defined by this expression is precisely the Boolean spanner that recognizes $\{0^m 1^m \mid m \in \mathbb{N}\}$, contradicting Theorem 4.21. □

THEOREM 4.23. *There is a string relation that is selectable by the core spanners but is nonrational (and hence nonregular), and there is a regular (and hence rational) relation that is not selectable by the core spanners.*

PROOF. To show that there is a nonrational string relation that is selectable by the core spanners, we use the observation that every rational relation possesses the following property: the projection on each component gives a regular language. Using string equality, we can construct a relation (e.g., $\{(\mathbf{ss}, \mathbf{ss}) \mid \mathbf{s} \in \Sigma^*\}$) selectable by the core spanners, such that this property is violated. As for the other direction, we use Theorem 4.22 and the obvious fact that the same-length relation, $\{(\mathbf{s}, \mathbf{t}) \mid |\mathbf{s}| = |\mathbf{t}|\}$, is regular. □

5. DIFFERENCE

In this section, we discuss the difference operator. Let P_1 and P_2 be spanners that are union compatible (that is, $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$). The difference $P_1 \setminus P_2$ is defined as follows. First, $\mathsf{SVars}(P_1 \setminus P_2) = \mathsf{SVars}(P_1)$. Second, if $\mathbf{s}$ is a string, then $(P_1 \setminus P_2)(\mathbf{s}) = P_1(\mathbf{s}) \setminus P_2(\mathbf{s})$.

The result with the most involved proof in this section states that core spanners are not closed under difference. Recall that the core spanners are those spanners that are expressible in $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}$. One may be tempted to think that nonclosure of core spanners under difference should be trivial to prove due to some monotonicity properties, as in the case of ordinary relational algebra. But this is not the case, because our algebra does not involve ordinary relations, but rather spanners; and the primitive representation of spanners (e.g., regex formulas or vset-automata) can simulate nonmonotonic behavior (e.g., regular expressions are closed under complement). In fact, we later show that core spanners can simulate string relations of a nonmonotonic flavor. Moreover, *regular* (but not *core*) spanners are actually closed under difference.

THEOREM 5.1. *Regular spanners are closed under difference; that is, $[\![\mathsf{VA}^{\{\setminus\}}_{\mathsf{set}}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.*

PROOF. Recall from Example 3.12 that the universal spanner Υ_Y is regular. Let P be a spanner. The *complement* of P, denoted $\overline{P}$, is the spanner $\Upsilon_{\mathsf{SVars}(P)} \setminus P$. Clearly, if P and Q are union-compatible spanners, then $P \setminus Q = P \cap \overline{Q}$. Consequently, it suffices to prove that regular spanners are closed under complement, that is, if A is a vset-automaton A, then there is a vset-automaton A' such that $[\![A']\!] = \overline{[\![A]\!]}$. So, let A be a vset-automaton.

Let G be a vset-path union that such that $[\![G]\!] = [\![A]\!]$. Recall that G exists due to Lemma 4.3. By definition, G is the union of some finite set $\mathcal{P}$ of consistent vset-paths,

that is,

$$[\![G]\!] = \bigcup_{P \in \mathcal{P}} [\![P]\!].$$

Consequently, we get the following.

$$\overline{[\![A]\!]} = \overline{[\![G]\!]} = \bigcap_{P \in \mathcal{P}} \overline{[\![P]\!]}.$$

Hence, due to Theorem 4.12 (and the fact that $\cap$ is a special case of $\bowtie$), it suffices to show that if P is a consistent vset-path, then there exists a vset-automaton B such that $[\![B]\!] = \overline{[\![P]\!]}$.

So, let $P = (Q, q_0, q_f, \delta)$ be a consistent vset-path. Suppose that δ consists of the edges $(q_{i-1}, \gamma_i, o_i, q_i)$, for $i = 1, \ldots, k$, and the final edge (q_k, γ_{k+1}, q_f), where each o_i is an operation of the form $x\vdash$ or $\dashv x$ for some variable $x \in \mathsf{SVars}(P)$. By considering cases, it is not hard to verify that the disjunction of the following four conditions is exactly what is needed to produce the spanner $\overline{[\![P]\!]}$.

(1) The operation o_{i+1} occurs strictly prior to o_i (i.e., at least one character exists after o_{i+1} and before o_i; note that other operations o_j can take place between between o_{i+1} and o_i).
(2) The operation o_i occurs prior to (or at the same time as) o_{i+1} and the span in between the two is in the complement of the regular expression γ_{i+1}.
(3) The prefix before o_1 is in the complement of the regular expression γ_1.
(4) The suffix after o_k is in the complement of the regular expression γ_{k+1}.

Now, it is easy to show that for each individual property among the above, there exists a vset-automaton B' such that $\mathsf{SVars}(B') = \mathsf{SVars}(P)$ and, moreover, for each string $\mathbf{s}$, we have that the tuples in the relation $[\![B']\!](\mathbf{s})$ give exactly those spans that correspond to that property. Hence, due to Theorem 4.12, we conclude that the union of those vset-automata B' is expressible by a vset-automaton B, which is the vset-automaton we desire. □

In an attempt to prove that core spanners are not closed under difference (or, equivalently, complement), we tried to prove that the language $\{\mathbf{s}\#\mathbf{t} \mid \mathbf{s} \neq \mathbf{t}\}$, where $\mathbf{s}$ and $\mathbf{t}$ are over the alphabet $\{0, 1\}$, and # is a new symbol, is not recognizable by any Boolean core spanner. After multiple failing attempts, we were surprised to discover that our candidate language L is a wrong candidate, since it actually *is* recognizable by a Boolean core spanner, for the following reason.

PROPOSITION 5.2. *The binary string relation $\neq$ is selectable by the core spanners.*

PROOF. Building on Proposition 4.15, it suffices to show a definition of the spanner $\Upsilon^{\neq}_{x,y}$ in the language $\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^{=}\}}$. We use the following definition.

$$\gamma_1(x, y) \cup \gamma_1(y, x) \cup \bigcup_{\sigma \neq \tau} \gamma_{\sigma,\tau}(x, y).$$

Here, $\gamma_1(x', y')$ defines the spanner that produces all the spans x' and y' such that the string spanned by y' is a proper prefix of the one spanned by x', and can be given as follows.

$$\pi_{x',y'}\, \varsigma^{=}_{y',z}((\Sigma^* \cdot x'\{z\{\Sigma^*\} \cdot \Sigma^+\} \cdot \Sigma^*) \times (\Sigma^* \cdot y'\{\Sigma^*\} \cdot \Sigma^*)).$$

(Recall that $\times$ is used for $\bowtie$ when there are no variables in common.) The expression $\gamma_{\sigma,\tau}(x, y)$ defines the spanner that finds all the spans x and y such that immediately

after a common prefix, the string of x has σ and that of y has τ. The expression $\gamma_{\sigma,\tau}$ can be the following:

$$\pi_{x,y}\, \zeta^{=}_{z_x,z_y}((\Sigma^* \cdot x\{z_x\{\Sigma^*\} \cdot \sigma \cdot \Sigma^*\} \cdot \Sigma^*) \times (\Sigma^* \cdot y\{z_y\{\Sigma^*\} \cdot \tau \cdot \Sigma^*\} \cdot \Sigma^*)). \quad \square$$

We remark that a proof similar to that of Proposition 5.2 shows that the string relations $\not\sqsubseteq_{\mathsf{prf}}$ and $\not\sqsubseteq_{\mathsf{sfx}}$ are also selectable by the core spanners. Eventually, we were able to prove nonclosure of the core spanners under difference through the (complement of) the substring relation.

THEOREM 5.3. *Assume that the alphabet Σ contains at least two symbols. The string relation $\not\sqsubseteq$ is not selectable by the core spanners.*

We will prove Theorem 5.3 in the next section.

Recall from Proposition 4.20 that the string relation $\sqsubseteq$ is selectable by the core spanners. Theorem 5.3, on the other hand, states that $\not\sqsubseteq$ is not selectable by the core spanners. By combining, these two, we get the following.

THEOREM 5.4. *Assume that the alphabet Σ contains at least two symbols. Core spanners are not closed under difference; that is,* $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!] \subsetneq [\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\zeta^=,\setminus\}}]\!]$.

PROOF. Propositions 4.15 and 4.20 imply that $\Upsilon^{\sqsubseteq}_{x,y}$ is a core spanner. But then, if core spanners are closed under difference, then $P' = \Upsilon_{x,y} \setminus \Upsilon^{\sqsubseteq}_{x,y}$ is also a core spanner. However, P' is equal to $\Upsilon^{\not\sqsubseteq}_{x,y}$, and by Proposition 4.15, $\not\sqsubseteq$ would be selectable, contradicting Theorem 5.3. $\square$

Theorems 5.1 and 5.4 show an interesting contrast between regular and core spanners with respect to difference.

5.1. Proof of Theorem 5.3

Our alphabet is $\Sigma = \{0, 1\}$. Let $\mathbf{s}$ be a string in Σ. A 0-*chunk of* $\mathbf{s}$ is a maximal span of $\mathbf{s}$ that consists of only "0" symbols. Here, maximality is with respect to span containment. We similarly define a 1-chunk of $\mathbf{s}$. As an example, the string $\mathbf{s} = 111011000100$ has three 0-chunks, namely $[4, 5\rangle$, $[7, 10\rangle$ and $[11, 13\rangle$, and three 1-chunks, namely $[1, 4\rangle$, $[5, 7\rangle$ and $[10, 11\rangle$. We define P to be the Boolean spanner (i.e., $\mathsf{SVars}(P) = \emptyset$) such that $P(\mathbf{s}) = \mathbf{true}$ if and only if $\mathbf{s}$ ends with a 0-chunk that is strictly longer than all the other 0-chunks. As an example, P accepts 00111000 and 001101000 but neither 00110 nor 0001101000.

Observe that P is in $[\![\mathsf{VA}^{\{\pi,\zeta^{\not\sqsubseteq}\}}_{\mathsf{set}}]\!]$, since we can define P as follows.

$$\pi_{\emptyset}\, \zeta^{\not\sqsubseteq}_{x,y}(y\{(0 \vee 1)^*\} \cdot x\{0^+\}).$$

So, if $\not\sqsubseteq$ were selectable by the core spanners, then P would be a core spanner. So suppose, by way of contradiction, that P is indeed a core spanner. By our core-simplification lemma, we can assume that P is represented by $\pi_{\emptyset} SA$, where A is a vset-automaton and S is a sequence of string selections $\zeta^{=}_{x,y}$ for $x, y \in \mathsf{SVars}(A)$. A variable that occurs in S is called an *S-variable*. Let M_v be the number of variables of A, and let N_q be the number of states of A. We take M and N to be sufficiently large, where M depends only on M_v, and where N depends only on M_v and N_q. Later, we shall see what "sufficiently large" means.

For a number $i \in \{1, \ldots, N\}$, we define $\mathbf{s}^i \in \Sigma^*$ to be the following string:

$$\mathbf{s}^i \stackrel{\mathsf{def}}{=} 0^i 1^1 0^i 1^2 0^i \cdots 0^i 1^M 0^{i+1}.$$

Observe that, for each $\mathbf{s}^i$ we have $P(\mathbf{s}^i) = \mathbf{true}$. For all $i = 1, \ldots, N$, we fix an accepting path ρ^i of A over $\mathbf{s}^i$, such that all the string equalities of S are satisfied (i.e., for each

$\varsigma^{=}_{x,y}$ in S the strings $\mathbf{s}^i_{\rho(x)}$ and $\mathbf{s}^i_{\rho(y)}$ are equal). Note that there is such an accepting path ρ^i by the assumption that $P(\mathbf{s}^i) = \mathbf{true}$.

For $i \in \{1, \ldots, N\}$, we define the following.

—A variable x *i-overlaps* with a span $[b, e\rangle$ if the spans $\rho^i(x)$ and $[b, e\rangle$ are not disjoint.
—A variable x *i-wraps* a span $[b, e\rangle$ if $\rho^i(x) = [b', e'\rangle$ where $b' < b$ and $e' > e$.
—An S-variable x is *i-trivial* if for every string-equality selection $\varsigma^{=}_{x,y}$ or $\varsigma^{=}_{y,x}$ in S it is the case that $\rho^i(x)$ is actually the same span (and not just spans the same string) as $\rho^i(y)$; otherwise, x is *i-nontrivial*.
—For $j = 1, \ldots, M$, the index b^i_j is that where the chunk 1^j begins in $\mathbf{s}^i$.

LEMMA 5.5. *Assume $i \in \{1, \ldots, N\}$, and let x be an S-variable. If x i-wraps a 1-chunk, then x is i-trivial.*

PROOF. Consider a string-equality selection $\varsigma^{=}_{x,y}$ or $\varsigma^{=}_{y,x}$ in S. We need to show that $\rho^i(x) = \rho^i(y)$. Since $\mathbf{s}^i_{\rho^i(x)} = \mathbf{s}^i_{\rho^i(y)}$, both $\mathbf{s}^i_{\rho^i(x)}$ and $\mathbf{s}^i_{\rho^i(y)}$ contain a 1-chunk, say 1^j, preceded and followed by zeros. But $\mathbf{s}^i$ contains exactly one 1-chunk of length j, and hence, that common 1^j of $\mathbf{s}^i_{\rho^i(x)}$ and $\mathbf{s}^i_{\rho^i(y)}$ must be of the same span. Therefore, $\rho^i(x)$ and $\rho^i(y)$ must be the same span. □

As an immediate consequence of Lemma 5.5, we get the following lemma.

LEMMA 5.6. *Assume $i \in \{1, \ldots, N\}$, and let x be an S-variable. If x is i-nontrivial, then x i-overlaps at most two 1-chunks.*

Assume $i \in \{1, \ldots, N\}$. We say that a span $[b, e\rangle$ of $\mathbf{s}^i$ is *i-safe* if no i-nontrivial S-variable i-overlaps with $[b, e\rangle$. By using Lemma 5.6, we see that if M is sufficiently large, then we get the following.

LEMMA 5.7. *Assume $i \in \{1, \ldots, N\}$. There are j, k with $1 \leq j < k \leq M$, such that the span $[b^i_j, b^i_k\rangle$ is i-safe.*

Building on Lemma 5.7, we fix for each $i \in \{1, \ldots, N\}$ a span $[b^i_{j_i}, b^i_{k_i}\rangle$ that is i-safe. Let $(q^i_1, V^i_1, Y^i_1, l^i_1)$ be the configuration of ρ^i right before $b^i_{j_i}$ is read, and let $(q^i_2, V^i_2, Y^i_2, l^i_2)$ be the configuration of ρ^i right before $b^i_{k_i}$ is read. We define $T_i = \langle q^i_1, V^i_1, Y^i_1, q^i_2, V^i_2, Y^i_2\rangle$. If N is sufficiently large, then we get the following lemma.

LEMMA 5.8. *There are i and l with $1 \leq i < l \leq N$ such that $T_i = T_l$.*

Fix some i and l as in Lemma 5.8. Let $\mathbf{s}$ be the string that is obtained from $\mathbf{s}^i$ by replacing the span $[b^i_{j_i}, b^i_{k_i}\rangle$ with the substring of $\mathbf{s}^l$ in the span $[b^l_{j_l}, b^l_{k_l}\rangle$. From the fact that $i < l$ we conclude that $P(\mathbf{s}) = \mathbf{false}$ (i.e., the last 0-chunk of $\mathbf{s}$, having length $i + 1$, is at most as long as one of the other 0-chunks, having length l). We then get a contradiction using the following lemma.

LEMMA 5.9. $[\![\pi_\emptyset S(A)]\!](\mathbf{s}) = \mathbf{true}$.

PROOF. From the fact that $T_i = T_l$, we can build an accepting run of A on $\mathbf{s}$ by replacing in ρ^i the sub-run that corresponds to ρ^i with the sub-run of ρ^l that corresponds to T_l. Let ρ be the resulting run. We need to show that each $\varsigma^{=}_{x,y}$ in S holds for ρ. So, consider such a $\varsigma^{=}_{x,y}$. If both x and y are i-nontrivial then $\varsigma^{=}_{x,y}$ holds because $[b^i_{j_i}, b^i_{k_i}\rangle$ is i-safe, which implies that x and y opened and closed outside the replaced sub-run. Otherwise, assume (without loss of generulity) that x is i-trivial. Then, $\rho^i(x)$ and $\rho^i(y)$ are the same span.

If $\rho^i(x)$ does not i-overlap with $[b^i_{j_i}, b^i_{k_i}\rangle$, then, again, x and y are opened and closed outside the replaced sub-run. Otherwise, from the fact that $T_i = T_l$ we conclude that x and y l-overlap with $[b^l_{j_l}, b^l_{k_l}\rangle$, and consequently (due to l-safety), x and y are l-trivial. This means that wherever x and y are opened or closed (i.e., either in an original sub-run or the replaced sub-run), they open and close at the same time, and then $\varsigma^{=}_{x,y}$ holds. □

6. SPANNERS VS. OTHER FORMALISMS

We now discuss the relationship between (core and regular) spanners and two related formalisms in the literature.

6.1. Extended Regular Expressions

We first relate core spanners to *extended regular expressions* (xregex for short) [Aho 1990; Câmpeanu et al. 2003; Carle and Narendran 2009; Freydenberger 2011], which extend the variable regular expressions with *backreferences* (a.k.a. *variable references*) that specify repetitions of a previously matched substring. Their expressive power goes strictly beyond the class of regular languages and, due to their usefulness in practice, most modern regular expression matching engines actually support extended regular expressions [Friedl 2006]. From a theoretical perspective, the extended regular expressions were formalized by Aho [1990], and investigated with respect to the complexity of their membership problem [Aho 1990], their expressiveness and closure properties [Câmpeanu et al. 2003; Câmpeanu and Santean 2009; Carle and Narendran 2009], and their conciseness and decidability [Freydenberger 2011], among other properties.

Syntactically, an xregex can be viewed as a (not necessarily functional) variable regex that, in addition to the variable-binding expressions $x\{\gamma\}$ also allows variable *backreferences* of the form $\&x$. For example, if δ_1 is $x\{(0 \vee 1)^*\} \cdot \&x$, and δ_2 is $x\{(0 \vee 1)^*\} \cdot \&x \cdot x\{(0 \vee 1)^*\} \cdot \&x$, then δ_1 and δ_2 are xregexes. To determine if an input string $\mathbf{s}$ is accepted by an xregex, the xregex is interpreted from left to right on $\mathbf{s}$ in a manner we now describe (cf., e.g., Câmpeanu et al. [2003] and Freydenberger [2011]). For normal variable regexes (see Section 3.1.1), a binding subexpression $x\{\gamma\}$ matches a substring if γ matches the substring. In this case, x is bound to the corresponding span. A backreference $\&x$ matches a substring $\mathbf{s}'$ if $\mathbf{s}' = \mathbf{s}_{[i,j\rangle}$ with $[i, j\rangle$ the span previously bound to x. If x has been bound multiple times, then the last binding prior to the backreference is taken when matching $\&x$; and if x has not been bound before, $\&x$ matches the empty string. As an example, the above xregex δ_1 matches precisely the strings $\mathbf{ss}$ with $\mathbf{s} \in \{0, 1\}^*$, and δ_2 matches precisely the strings $\mathbf{sss}'\mathbf{s}'$ with $\mathbf{s}, \mathbf{s}' \in \{0, 1\}^*$. Observe that neither of these languages is regular.

The evaluation of an xregex over a string is not (naturally) mapped to an $\mathbf{s}$-tuple, since a variable can be assigned multiple spans. Therefore, we restrict our discussion to the comparison of xregexes with *Boolean* core spanners (where all of the variables are projected out). An important part of the expressive power of xregexes stems from the fact that both variable binders and backreferences can occur under the scope of a Kleene star (or plus). For example, $(x\{(0 \vee 1)^*\} \cdot \&x)^+$ matches all strings $\mathbf{s_1 s_1} \cdots \mathbf{s_n s_n}$ with $n \geq 1$ and every $\mathbf{s_i} \in \{0, 1\}^*$. Moreover,

$$1^+ \cdot x\{0^*\} \cdot (1^+ \cdot \&x)^* \cdot 1^+$$

matches all strings $\mathbf{s}_1\mathbf{t}\mathbf{s}_2\mathbf{t} \cdots \mathbf{s}_{n-1}\mathbf{t}\mathbf{s}_n$, where $\mathbf{t} \in 0^*$ and every $\mathbf{s}_i$ is in 1^+. In other words, it accepts the language of strings over $\{0, 1\}^*$ that start and end with 1, and where all maximal chunks of consecutive 0's are of equal length. We refer to this language as the

uniform-0-chunk language. As the following theorem states, this language is beyond the expressive power of core spanners.

THEOREM 6.1. *The uniform-0-chunk language is recognizable by an xregex but is not recognizable by any Boolean core spanner.*

PROOF. We already showed an xregex that recognizes the uniform-0-chunk language. It remains to prove that this language is not recognizable by any Boolean core spanner. Recall the proof of Theorem 5.3. There, we considered the language L of all strings $\mathbf{s} \in \{0,1\}^*$, such that $\mathbf{s}$ ends with a 0-chunk that is longer than all the other 0-chunks. We showed that L is not recognized by any Boolean core spanner. The proof that the uniform-0-chunk language is not recognized by any Boolean core spanner is almost identical to that for L. The only difference is the following. Recall that for a number $i \in \{1, \ldots, N\}$, we defined $\mathbf{s}^i \in \Sigma^*$ as follows:

$$\mathbf{s}^i \stackrel{\text{def}}{=} 0^i 1^1 0^i 1^2 0^i \cdots 0^i 1^M 0^{i+1}.$$

So now, we define $\mathbf{s}^i$ slightly differently:

$$\mathbf{s}^i \stackrel{\text{def}}{=} 1^1 0^i 1^2 0^i \cdots 0^i 1^M.$$

Except for that, the proof remains the same. □

It is currently still open whether every language recognized by a Boolean core spanner can also be recognized by an xregex. We do note the following. Consider a core spanner represented by $\pi_Y SA$, as in the core-simplification lemma (Lemma 4.19). If the variables of the vset-automaton A cover disjoint spans, then it is easy to prove that such a simulating xregex must exist. To illustrate, consider the regex formula $\gamma := x\{\gamma_1\} \cdot \gamma_2 \cdot y\{\gamma_3\}$, where x and y are variables, and γ_1, γ_2, and γ_3 are regular expressions. Then, the core spanner $\pi_\emptyset \zeta^=_{x,y}(\gamma)$ is specified by the xregex $x\{\delta\} \cdot \gamma_2 \cdot \&x$, where δ is the regular expression that recognizes the intersection of the regular expressions γ_1 and γ_3. The problem in finding an xregex that corresponds to a Boolean core spanner arises when the variables in the core spanner have overlapping spans.

6.2. CRPQs on Marked Paths

Regular expressions have been extensively used and studied in database theory as a means to express reachability queries in semistructured and graph databases since the late 1980s. Arguably, the simplest form of such queries is the *regular path query* (*RPQ* for short) on directed graphs with labeled edges [Consens and Mendelzon 1990; Cruz et al. 1987]. RPQs search for the existence of a path, such that the word formed by the edge labels belongs to a specified regular language. A *conjunctive regular path query* (*CRPQ* for short) applies conjunction and existential quantification (over nodes) to RPQs; this concept has been the subject of much investigation [Calvanese et al. 2000a, 2000b; Consens and Mendelzon 1990; Deutsch and Tannen 2001; Florescu et al. 1998].

Let Δ be a finite alphabet. A *Δ-labeled graph* is a pair $G = (V, E)$ where V is a finite set of nodes, and $E \subseteq V \times \Delta \times V$ is a set of labeled edges. A *path* from u to v in G is a sequence

$$\vec{e} = (v_0, \sigma_0, v_1), (v_1, \sigma_1, \sigma_2), \ldots, (v_{m-1}, \sigma_{m-1}, v_m)$$

of edges from E, with $v_0 = u$, and $v_m = v$. The word $\sigma_0 \cdots \sigma_{m-1} \in \Delta^*$ is called the *string formed by* $\vec{e}$, and is denoted by $str(\vec{e})$.

Fix an infinite set NVars of *node variables*, pairwise disjoint from SVars and Σ. A *regular path query* (RPQ) over Δ is a triple of the form (x, L, y) with $x, y \in$ NVars and $L \subseteq \Delta^*$ a regular language. A *conjunctive regular path query* (CRPQ) over Δ is a

formula φ of the form

$$\exists \vec{z} \bigwedge_{i=1}^{m} (x_i, L_i, y_i),$$

where the (x_i, L_i, y_i) are RPQs and $\vec{z}$ is a sequence of node variables. We denote by $\mathsf{NVars}(\varphi)$ the set of all node variables occurring in φ; by $\mathit{free}(\varphi)$ the set $\mathsf{NVars}(\varphi) \setminus \vec{z}$ of free node variables of φ; and by $\mathit{body}(\varphi)$ the set $\{(x_i, L_i, y_i) \mid 1 \leq i \leq n\}$ of all RPQs of φ. We refer to the elements of $\mathit{body}(\varphi)$ also as the *atoms* of φ.

Semantically, φ evaluates to a set of mappings $\mathit{free}(\varphi) \to V$ when evaluated on a Δ-labeled graph $G = (V, E)$. To formally define this semantics of CRPQs, let ν be a mapping from $\mathsf{NVars}(\varphi)$ to the set of nodes of a Δ-labeled graph G. We define the relationship $(G, \nu) \models \varphi$ to hold if for each atom (x_i, L_i, y_i) of φ there is a path $\vec{e}$ in G from $\nu(x_i)$ to $\nu(y_i)$ such that $\mathit{str}(\vec{e}) \in L_i$. Let $\nu|_{\mathit{free}(\varphi)}$ denote the restriction of ν to $\mathit{free}(\varphi)$. The semantics $\varphi(G)$ of φ on G is then the set of all mappings $\nu|_{\mathit{free}(\varphi)}$ such that $(G, \nu) \models \varphi$, for some ν.

Example 6.2. Consider the CRPQ $\varphi(x, y)$ and graph G.

$$\varphi(x, y) := \exists u\, (u, a^+, x) \wedge (x, b^*, y)$$

$$G = 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 4.$$

Then, $\varphi(G) = \{\nu_1, \nu_2, \nu_3\}$ with

$$\nu_1 \colon x \mapsto 2, y \mapsto 2$$
$$\nu_2 \colon x \mapsto 2, y \mapsto 3$$
$$\nu_3 \colon x \mapsto 2, y \mapsto 4.$$

A *union of CRPQs* (UCRPQ) is a formula φ of the form $\varphi_1 \vee \cdots \vee \varphi_k$ where every φ_i is a CRPQ, and $\mathit{free}(\varphi_1) = \cdots = \mathit{free}(\varphi_k)$. We define $\varphi(G)$ to be $\varphi_1(G) \cup \cdots \cup \varphi_k(G)$.

6.3. Evaluating UCRPQs on strings

A string $\mathbf{s} = \sigma_1 \cdots \sigma_k$ can be viewed as a special case of a graph, namely as the following simple path $p(\mathbf{s})$ over the nodes $\{1, \ldots, k+1\}$:

$$1 \xrightarrow{\sigma_1} 2 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{k-1}} k \xrightarrow{\sigma_k} k+1.$$

Under this representation of strings as graphs, however, UCRPQs cannot detect which node marks the start of the input string, nor which node marks the end of the input string. UCRPQs are therefore not able to verify that they have "processed" their entire input, as formalized by the following proposition.

If μ is a mapping from a set of variables to the integers and k is an integer, then we denote by μ^{+k} the mapping such that $\mu^{+k}(x) = \mu(x) + k$, for every x.

Proposition 6.3 (UCRPQ Monotonicity on Simple Paths). *Let φ be a UCRPQ and let $\mathbf{s}$ and $\mathbf{t}$ be strings such that $\mathbf{s} \sqsubseteq \mathbf{t}$, that is, $\mathbf{t} = \mathbf{s_1 s s_2}$ for some strings $\mathbf{s_1}$ and $\mathbf{s_2}$. If $\mu \in \varphi(p(\mathbf{s}))$, then $\mu^{+|\mathbf{s_1}|} \in \varphi(p(\mathbf{t}))$.*

The proof is straightforward.

Viewed on the Boolean level, Proposition 6.3 says that if a UCRPQ φ accepts string $\mathbf{s}$ (in the sense that $\varphi(\mathbf{s}) \neq \emptyset$), it must also accept all extensions $\mathbf{t}$ with $\mathbf{s} \sqsubseteq \mathbf{t}$. As such the language $\{\mathbf{s} \mid \varphi(\mathbf{s}) \neq \emptyset\}$ recognized by φ is closed under string extensions. Since the regular spanners can recognize all regular languages, and since obviously not all regular languages are closed under string extensions, it immediately follows

that with this representation of strings as graphs, an exact correspondence between regular spanners and UCRPQs cannot be obtained.

We will therefore represent strings by means of *marked paths*, where the nodes that represent the beginning and end of the string are marked with a loop labeled with a special symbol. Formally, if $\mathbf{s} = \sigma_1 \cdots \sigma_k$ then the marked path $G_\mathbf{s}$ is the graph over nodes $\{1, \ldots, k+1\}$ defined as

$$\overset{\rhd}{\circlearrowright}\, 1 \xrightarrow{\sigma_1} 2 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{k-1}} k \xrightarrow{\sigma_k} k+1 \,\overset{\lhd}{\circlearrowright}$$

That is, $\mathbf{s}$ is represented as a chain where the first node carries a $\rhd$-loop to denote the beginning of $\mathbf{s}$, and the last node carries a $\lhd$-loop to denote the end of $\mathbf{s}$. Here, we assume that $\rhd$ and $\lhd$ are two special symbols not in Σ. Note that, under this representation of strings as graphs, one can find the start node by means of the RPQ $(x, \rhd, x)$ and the end node by $(y, \lhd, y)$. Also note that UCRPQs are not monotonic on marked paths. For example, if we let $\mathbf{s} = aa$ and $\mathbf{t} = aab$ and

$$\varphi = (x, \rhd, x) \wedge (x, aa, y) \wedge (y, \lhd, y),$$

then $\mathbf{s} \sqsubseteq \mathbf{t}$, but $\varphi(G_\mathbf{s}) \neq \emptyset$ while $\varphi(G_\mathbf{t}) = \emptyset$.

6.4. Correspondence

We will establish two correspondences between regular spanners and UCRPQs. The first correspondence is in terms of the set of spanners definable by UCRPQs, while the second is in terms of the set of node assignments definable by regular spanners. In this section, we formally state these correspondences; they are proved in Section 6.5 and 6.6, respectively.

Fix, for every span variable x, two node variables $x^\vdash$ and $x^\dashv$. If V is a set of span variables, we denote by $\tilde{V}$ the set $\{x^\vdash, x^\dashv \mid x \in V\}$. Furthermore if $\mu \colon V \to \mathsf{Spans}(\mathbf{s})$ is a $(V, \mathbf{s})$-tuple, then we denote by $\tilde{\mu}$ the unique node assignment $\tilde{\mu} \colon \tilde{V} \to \{1, \ldots, |\mathbf{s}|+1\}$ on $G_\mathbf{s}$ such that $\mu(x) = [\tilde{\mu}(x^\vdash), \tilde{\mu}(x^\dashv)\rangle$.

Definition 6.4. A CRPQ or UCRPQ φ is said to *define* the spanner P over a set V of span variables if φ is a CRPQ or UCRPQ over alphabet $\Sigma \cup \{\rhd, \lhd\}$ such that $\tilde{V} = \mathit{free}(\varphi)$ and $\{\tilde{\mu} \mid \mu \in P(\mathbf{s})\} = \varphi(G_\mathbf{s})\}$, for every $\mathbf{s} \in \Sigma^*$.

The following theorem now establishes our first correspondence (to be proved in Section 6.5).

THEOREM 6.5. $[\![\mathsf{UCRPQ}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

Before moving to the second correspondence, we want to comment on the relationship between $[\![\mathsf{UCRPQ}]\!]$ and $[\![\mathsf{CRPQ}]\!]$. In particular, while it is obvious that $[\![\mathsf{CRPQ}]\!] \subseteq [\![\mathsf{UCRPQ}]\!]$, it is not a priori, obvious that this inclusion is strict. Indeed, CRPQs actually allow certain forms of disjunction. For example, the UCRPQ $\varphi_1 \vee \varphi_2$ with

$$\varphi_1 := \exists u, z\, ((u, \rhd b^*, x) \wedge (x, a, y) \wedge (y, b^* \lhd, z)),$$
$$\varphi_2 := \exists u, z\, ((u, \rhd a^*, x), (x, b, y), (y, a^* \lhd, z)),$$

can be equivalently expressed on marked paths by the CRPQ

$$\exists u, z\, ((u, (\rhd a^* b a^* \lhd) \vee (\rhd b^* a b^* \lhd), z) \wedge \big(x, (a \vee b), y\big) \wedge \big(u, (\rhd a^* b \vee \rhd b^* a), y\big)).$$

The question of whether $[\![\mathsf{CRPQ}]\!]$ is strictly contained in $[\![\mathsf{UCRPQ}]\!]$ is tightly linked to the question whether UCRPQs strictly extend the power of CRPQs. We can show

that UCRPQs are strictly more expressive than CRPQs,[3] although it is beyond the scope of this paper to include the proof. This immediately yields that $[\![\mathsf{CRPQ}]\!]$ is strictly contained in $[\![\mathsf{UCRPQ}]\!]$.

For the other correspondence, let $\vec{x} = x_1, \ldots, x_n$ be a sequence of node variables, and let $\vec{y} = y_1, \ldots, y_n$ be a sequence of span variables of the same arity. We say that a node assignment ν over $\{x_1, \ldots, x_n\}$ on $G_{\mathbf{s}}$ is $(\vec{x}, \vec{y})$-*compatible* with a $(\{y_1, \ldots, y_n\}, \mathbf{s})$-tuple μ if $\nu(x_i)$ is the first component of the span $\mu(y_i)$, for every $1 \leq i \leq n$. (That is, $\mu(y_i) = [\nu(x_i), j_i\rangle$ for some $j_i \geq \nu(x_i)$.) We write $\nu \ \sim_{(\vec{x},\vec{y})} \mu$ to denote that ν is $(\vec{x}, \vec{y})$-*compatible* with μ. Since $(\vec{x}, \vec{y})$ compatibility of ν with μ essentially states that we get exactly ν by looking only at the first component of each span in μ, the relationship $\sim_{(\vec{x},\vec{y})}$ defines an encoding of node assignments on $G_{\mathbf{s}}$ as $\mathbf{s}$-tuples.

The following proposition now establishes our second correspondence (to be proved in Section 6.6). It states that each node assignment definable by a UCRPQ can also be defined by a regular spanner, modulo this encoding of node assignments as tuples.

PROPOSITION 6.6. *Let φ be a UCRPQ with free variables $\vec{x} = x_1, \ldots, x_n$. Let $\vec{y}$ be a sequence of span variables of the same arity as $\vec{x}$. There exists a regular spanner P with* $\mathsf{SVars}(P) = \vec{y}$ *such that for all* $\mathbf{s} \in \Sigma^*$ *we have*

$$P(\mathbf{s}) = \{\mu \mid \exists \nu \in \varphi(G_{\mathbf{s}}) \text{ such that } \nu \ \sim_{(\vec{x},\vec{y})} \mu\}.$$

The proof appears in Section 6.6.

6.5. Proof of Theorem 6.5

We prove Theorem 6.5 in two steps. First we show in Proposition 6.7 that $[\![\mathsf{VA}_{\mathsf{set}}]\!] \subseteq [\![\mathsf{UCRPQ}]\!]$. Then, we show in Proposition 6.19 that $[\![\mathsf{UCRPQ}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

PROPOSITION 6.7. $[\![\mathsf{VA}_{\mathsf{set}}]\!] \subseteq [\![\mathsf{UCRPQ}]\!]$

PROOF. Let A be a vset-automaton. By Lemma 4.3, there exist consistent vset-paths $P_1, \ldots, P_n$ such that $[\![A]\!] = [\![P_1]\!] \cup \cdots \cup [\![P_n]\!]$.We will prove that every $[\![P_i]\!]$ is definable by a CRPQ φ_i, for $1 \leq i \leq n$. As such, $[\![A]\!]$ is defined by the UCRPQ $\varphi_1 \vee \cdots \vee \varphi_n$.

Fix i such that $1 \leq i \leq n$. By definition, vset-path P_i is of the form

$$q_s[e_0] \to v_1 q_1[e_1] \to \cdots \to v_k q_k[e_k] \to v_{k+1} q_{k+1}[e_{k+1}] \to q_f,$$

where q_s, q_f and the q_j are the states, each e_j is the regular expression on the edge between its preceding and following states, and each v_j is the operation in $\{x\vdash, \dashv x\}$, that takes place when entering q_j.

Let $V = \mathsf{SVars}(P_i)$. Define, for every j with $1 \leq j \leq k+1$, the node variable $\tilde{v_j} \in \tilde{V}$ by

$$\tilde{v}_j = \begin{cases} x^{\vdash} & \text{if } v_j = x\vdash \text{ for some } x \\ x^{\dashv} & \text{if } v_j = \dashv x \text{ for some } x. \end{cases}$$

Let y, z be node variables not in $\tilde{V}$. Then, define the CRPQ φ_i by

$$\exists y, z\ (y, \triangleright \cdot e_0, \tilde{v}_1) \wedge \bigwedge_{j=1}^{k} (\tilde{v}_j, e_j, \tilde{v}_{j+1}) \wedge (\tilde{v}_{k+1}, e_{k+1} \cdot \triangleleft, z).$$

It is now straightforward to verify that $\{\tilde{\mu} \mid \mu \in [\![P_i]\!](\mathbf{s})\} = \varphi_i(G_{\mathbf{s}})$, for every $\mathbf{s} \in \Sigma^*$. □

To complete the proof of Theorem 6.5, it remains to prove that $[\![\mathsf{UCRPQ}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{set}}]\!]$. Observe that, since $[\![\mathsf{VA}_{\mathsf{set}}]\!]$ is closed under union by Theorem 4.12, this immediately

[3]This result was obtained jointly with Pablo Barceló.

follows if we succeed in proving that $[\![\mathsf{CRPQ}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{set}}]\!]$. We devote the rest of this section to this proof, which proceeds in three steps.

(1) First, we show that the set of all pairs $(\mathbf{s}, \nu)$ with $\mathbf{s} \in \Sigma^*$ such that $\nu \in \varphi(G_{\mathbf{s}})$ can be encoded as a regular language of *free*(φ)-linear strings (Proposition 6.13). A *free*(φ)-linear string (which we define formally later) is intuitively a string in which it is recorded to which positions node variables of φ are mapped by ν.
(2) We then show that it is possible to transform this regular language into a regular language of *parses*. Intuitively, a parse is a string in which it is recorded where span variables should start to match, and where they should stop matching. Parse languages naturally define spanners, and we show, denoting by RParses the class of all regular parse languages, that $[\![\mathsf{VA}_{\mathsf{set}}]\!] = [\![\mathsf{RParses}]\!]$ (Corollary 6.17).
(3) From this, we finally obtain $[\![\mathsf{CRPQ}]\!] \subseteq [\![\mathsf{RParses}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$ (Proposition 6.18).

First, however, we recall some basic facts about regular languages. Let Σ and Δ be two finite alphabets. A *morphism* is a function $f\colon \Sigma^* \to \Delta^*$ such that $f(\mathbf{st}) = f(\mathbf{s})f(\mathbf{t})$, for all $\mathbf{s}, \mathbf{t} \in \Sigma^*$. Note that every morphism is uniquely determined by the values $f(\sigma)$ for $\sigma \in \Sigma$ since $f(\sigma_1 \cdots \sigma_n) = f(\sigma_1) \cdots f(\sigma_n)$. Also note that every morphism has $f(\epsilon) = \epsilon$ since otherwise $f(\epsilon) = f(\epsilon\epsilon) = f(\epsilon)f(\epsilon)$ cannot hold. It is well-known [Yu 1997] that the class of regular languages is closed under morphisms and inverse morphisms: if $K \subseteq \Sigma^*$ is regular, then so is $f(K) = \{f(\mathbf{s}) \mid \mathbf{s} \in K\}$ and if $L \subseteq \Delta^*$ is regular, then so is $f^{-1}(L) = \{\mathbf{s} \in \Sigma^* \mid f(\mathbf{s}) \in L\}$.

Let Δ and Λ be two disjoint alphabets. We denote by

$$del_{\Delta,\Lambda}\colon (\Delta \cup \Lambda)^* \to \Delta^*$$

the morphism that deletes all Λ-elements from its input, defined by

$$del_{\Delta,\Lambda}(a) = \begin{cases} a & \text{if } a \notin \Lambda, \text{ and} \\ \epsilon & \text{otherwise.} \end{cases}$$

We simply write del_Λ for $del_{\Delta,\Lambda}$ when Δ is clear from the context.

If L and K are two languages over alphabet Δ, then the *right quotient of L by K*, denoted L/K, is the language $\{\mathbf{s} \in \Delta^* \mid \exists \mathbf{t} \in K \text{ such that } \mathbf{st} \in L\}$. The *left quotient of L by K*, denoted $L \% K$, is the language $\{\mathbf{t} \in \Delta^* \mid \exists \mathbf{s} \in K \text{ such that } \mathbf{st} \in L\}$. It is well-known that the class of regular languages is closed under both left and right quotients [Yu 1997].

In what follows, we write KL for the concatenation of languages K and L. Also, if $L = \{\mathbf{s}\}$ then we simply write $\mathbf{s}$ for L. We write L^+ for $L^* - \{\epsilon\}$.

6.5.1. Linear Strings. Let Λ be a finite alphabet, disjoint from Σ. A string $\mathbf{w}$ is called Λ-*linear* if $\mathbf{w} \in (\Sigma \cup \Lambda)^*$ and every element of Λ occurs exactly once in $\mathbf{w}$. Let $\mathbf{w}$ be a Λ-linear string. Then we can write $\mathbf{w}$ as an alternation $\mathbf{w}_1\mathbf{v}_1\mathbf{w}_2\mathbf{v}_2 \cdots \mathbf{w}_n\mathbf{v}_n\mathbf{w}_{n+1}$ of strings $\mathbf{w}_i \in \Sigma^*$ for $1 \leq i \leq n+1$ and strings $\mathbf{v}_j \in \Lambda^*$ for $1 \leq i \leq n$. Define $\hat{\mathbf{w}}$ to be $\mathbf{w}_1 \cdots \mathbf{w}_n = del_{\Sigma,\Lambda}(\mathbf{w})$. To $\mathbf{w}$, we associate the function $[\mathbf{w}]\colon \Lambda \to \{1, \ldots, |\hat{\mathbf{w}}| + 1\}$ such that, for all $x \in \Lambda$,

$$[\mathbf{w}](x) = 1 + \sum_{i=1}^{k} |\mathbf{w}_i|,$$

where k is the unique element of $\{1, \ldots, n\}$ such that $x \in \Lambda$ occurs in $\mathbf{v}_k$.

Example 6.8. To illustrate, let $\Sigma = \{a, b, c\}$ and $\Lambda = \{x, y\}$. Then $\mathbf{w} = abbxcby$ is Λ-linear, $\hat{\mathbf{w}} = abbcb$ and $[\mathbf{w}]$ maps $x \mapsto 4$ and $y \mapsto 6$. Note that $y \mapsto 6$ and not $y \mapsto 7$ because y is the sixth element of the string $abbcby$ where x has been removed.

Let $\mathsf{Lin}(\Lambda)$ denote the set of all Λ-linear strings. Since Λ is finite, it is not difficult to check by means of a finite state automaton that a given input $\mathbf{w} \in (\Sigma \cup \Lambda)^*$ is Λ-linear. Hence, we have the following lemma.

LEMMA 6.9. $\mathsf{Lin}(\Lambda)$ *is regular, for all finite alphabets* Λ.

Note that for every $\mathbf{s} \in \Sigma^*$ and every node assignment $\nu\colon V \to \{1, \ldots, |\mathbf{s}| + 1\}$ on set V of node variables we can always find a V-linear string $\mathbf{w}$ such that $\hat{\mathbf{w}} = \mathbf{s}$ and $[\mathbf{w}] = \nu$. We say that $\mathbf{w}$ *encodes* the pair $(\mathbf{s}, \nu)$ in this case.

Example 6.10. Let $\varphi = (x, cb, y)$. Then $\mathbf{w} = abbxcby$ from Example 6.8 encodes the unique node assignment ν on G_{abbcb} with $\nu \in \varphi(G_{abbcb})$ in the sense that $\hat{\mathbf{w}} = abbcb$ and $[\mathbf{w}] = \nu$.

Let φ be a CRPQ. We define $linenc(\varphi)$ to be the language of all linear strings that encode pairs $(\mathbf{s}, \nu)$ with $\nu \in \varphi(G_{\mathbf{s}})$:

$$\begin{aligned} linenc(\varphi) &= \{\mathbf{w} \in \mathsf{Lin}(free(\varphi)) \mid \exists \mathbf{s} \in \Sigma^*, \exists \nu \in \varphi(G_{\mathbf{s}}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\mathbf{w}] = \nu\} \\ &= \{\mathbf{w} \in \mathsf{Lin}(free(\varphi)) \mid [\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})\}. \end{aligned}$$

We will show that $linenc(\varphi)$ is regular. We first require the following auxiliary result.

LEMMA 6.11. $linenc(\alpha)$ *is regular, for every RPQ* α.

PROOF. Let $\alpha = (x, L, y)$ with $L \subseteq (\Sigma \cup \{\triangleright, \triangleleft\})^*$. From L, we compute the following four languages:

$$\begin{aligned} R_{\triangleright\Sigma^*\triangleleft} &= \{\mathbf{s} \in \Sigma^* \mid \mathbf{vsw} \in L \text{ for some } \mathbf{v} \in \triangleright^+, \mathbf{w} \in \triangleleft^+\}, \\ R_{\triangleright\Sigma^*} &= \{\mathbf{s} \in \Sigma^* \mid \mathbf{vs} \in L \text{ for some } \mathbf{v} \in \triangleright^+\}, \\ R_{\Sigma^*\triangleleft} &= \{\mathbf{s} \in \Sigma^* \mid \mathbf{sw} \in L \text{ for some } \mathbf{w} \in \triangleleft^+\}, \\ R_{\Sigma^*} &= \{\mathbf{s} \in \Sigma^* \mid \mathbf{s} \in L\}. \end{aligned}$$

We claim that these four are all regular. Indeed, it is straightforward to verify the following.

$$\begin{aligned} R_{\triangleright\Sigma^*\triangleleft} &= ((L \,\%\, \triangleright^+)/\, \triangleleft^+) \cap \Sigma^*, \\ R_{\triangleright\Sigma^*} &= (L \,\%\, \triangleright^+) \cap \Sigma^*, \\ R_{\Sigma^*\triangleleft} &= (L/\, \triangleleft^+) \cap \Sigma^*, \\ R_{\Sigma^*} &= L \cap \Sigma^*. \end{aligned}$$

Hence, since L is regular and since the class of regular languages is closed under concatenation, intersection and both left and right quotients, the four languages are regular. Therefore, K defined as follows is also regular.

$$K = xR_{\triangleright\Sigma^*\triangleleft}y \cup xR_{\triangleright\Sigma^*}y\Sigma^* \cup \Sigma^*xR_{\Sigma^*\triangleleft}y \cup \Sigma^*xR_{\Sigma^*}y\Sigma^*.$$

Note that K is $\{x, y\}$-linear. We claim that $K = linenc(\alpha)$. We first show that $K \subseteq linenc(\alpha)$. Assume $\mathbf{w} \in K$. Then, w belongs to one of $xR_{\triangleright\Sigma^*\triangleleft}y$, $xR_{\triangleright\Sigma^*}y\Sigma^*$, $\Sigma^*xR_{\triangleright\Sigma^*\triangleleft}y$, or $\Sigma^*xR_{\Sigma^*}y\Sigma^*$. Assume that it belongs to $xR_{\triangleright\Sigma^*\triangleleft}y$ (the other cases are similar). Then $\mathbf{w} = x\mathbf{s}y$ with $\mathbf{s} \in \Sigma^*$ and $\mathbf{v}_1\mathbf{s}\mathbf{v}_2 \in L$, for some $\mathbf{v}_1 \in \triangleright^+$ and $\mathbf{v}_2 \in \triangleleft^+$. Then, clearly, $\hat{\mathbf{w}} = \mathbf{s}$, $[\mathbf{w}] = \{x \mapsto 1, y \mapsto |\mathbf{s}| + 1\}$ and $[\mathbf{w}] \in \alpha(G_{\mathbf{s}})$, as desired.

We now show that $linenc(\alpha) \subseteq K$. Assume $\mathbf{w} \in linenc(\alpha)$. Then, $\mathbf{w} \in \mathsf{Lin}(\{x, y\})$ and $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$. By definition of the semantics of CRPQs, there exists some path $\vec{e}$ from $\nu(x)$ to $\nu(y)$ in $G_{\hat{\mathbf{w}}}$ such that $str(\vec{e}) \in L$. Moreover, by definition of $G_{\hat{\mathbf{w}}}$, it follows that $str(\vec{e})$ must be of the form $\mathbf{v}_1\mathbf{s}\mathbf{v}_2$ with $\mathbf{v}_1 \in \triangleright^*$, $\mathbf{s} \in \Sigma^*$ and $\mathbf{v}_2 \in \triangleleft^*$. We distinguish four cases: (1) $\mathbf{v}_1 \neq \epsilon$ and $\epsilon \neq \mathbf{v}_2$; (2) $\mathbf{v}_1 \neq \epsilon = \mathbf{v}_2$; (3) $\mathbf{v}_1 = \epsilon \neq \mathbf{v}_2$; and (4) $\mathbf{v}_1 = \mathbf{v}_2 = \epsilon$. We illustrate

the reasoning only for the first case; the other cases are similar. Suppose that $\mathbf{v}_1 \neq \epsilon$ and $\mathbf{v}_2 \neq \epsilon$. Then, $\nu(x)$ must have an outgoing $\triangleright$-labeled edge. Since only the first node in $G_{\hat{\mathbf{w}}}$ has such an edge, $\nu(x) = 1$. Similarly, $\nu(y)$ must have an incoming $\triangleleft$-labeled edge and hence $\nu(y) = |\hat{\mathbf{w}}| + 1$. Then, clearly $\mathbf{w} = x\mathbf{s}y$. Since $\mathbf{v}_1\mathbf{s}\mathbf{v}_2 \in L$ with $\mathbf{v}_1 \in \triangleright^+$ and $\mathbf{v}_2 \in \triangleleft^+$, we have $\mathbf{s} \in R_{\triangleright\Sigma^*\triangleleft}$ by construction. Then, $\mathbf{w} = x\mathbf{s}y \in xR_{\triangleright\Sigma^*\triangleleft}y \subseteq K$. □

LEMMA 6.12. *linenc*(φ) *is regular, for every CRPQ* φ *with free*(φ) = NVars(φ) *(i.e., for every CRPQ without existential quantification).*

PROOF. Let $X = \mathsf{NVars}(\varphi) = \mathit{free}(\varphi)$. Since φ does not contain existential quantification, it is of the form

$$\varphi \equiv \bigwedge_{\alpha \in \mathit{body}(\varphi)} \alpha.$$

By Lemma 6.11, $\mathit{linenc}(\alpha) \subseteq \mathsf{Lin}(\mathit{free}(\alpha))$ is regular, for every $\alpha \in \mathit{body}(\varphi)$. Let $Y_\alpha = X \setminus \mathit{free}(\alpha)$, for every $\alpha \in \mathit{body}(\varphi)$. Then, define the set K of X-linear strings by

$$K = \left(\bigcap_{\alpha \in \mathit{body}(\varphi)} \mathit{del}^{-1}_{\Sigma \cup \mathit{free}(\alpha), Y_\alpha}(\mathit{linenc}(\alpha)) \right) \cap \mathsf{Lin}(X).$$

Note that K is regular since the class of regular languages is closed under inverse morphisms and since $\mathsf{Lin}(X)$ is regular by Lemma 6.9.

We claim that $K = \mathit{linenc}(\varphi)$. We first show that $K \subseteq \mathit{linenc}(\varphi)$. Assume $\mathbf{w} \in K$. Then, in particular, $\mathbf{w} \in \mathsf{Lin}(X)$. To show that $\mathbf{w} \in \mathit{linenc}(\varphi)$, we need to show that $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$. In this respect, first observe that, for $\mathbf{s} \in \Sigma^*$ we have $\nu \in \varphi(G_{\mathbf{s}})$ if and only if $\nu|_{\mathit{free}(\alpha)} \in \alpha(G_{\mathbf{s}})$, for every atom $\alpha \in \mathit{body}(\varphi)$. Then, let $\mathbf{w}_\alpha = \mathit{del}_{\Sigma \cup \mathit{free}(\alpha), Y_\alpha}(\mathbf{w})$, for every $\alpha \in \mathit{body}(\varphi)$. It is straightforward to check that $\hat{\mathbf{w}} = \hat{\mathbf{w}}_\alpha$ and $[\mathbf{w}]|_{\mathit{free}(\alpha)} = [\mathbf{w}_\alpha]$. Then, since by definition of K we have

$$\mathbf{w} \in \mathit{del}^{-1}_{\Sigma \cup \mathit{free}(\alpha), Y_\alpha}(\mathit{linenc}(\alpha)),$$

we obtain that $\mathbf{w}_\alpha \in \mathit{linenc}(\alpha)$. Therefore, $[\mathbf{w}]|_{\mathit{free}(\alpha)} \in \alpha(G_{\hat{\mathbf{w}}})$, for every $\alpha \in \mathit{body}(\varphi)$ and hence $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$, as desired.

We next show that $\mathit{linenc}(\varphi) \subseteq K$. Assume $\mathbf{w} \in \mathit{linenc}(\varphi)$. Then $\mathbf{w} \in \mathsf{Lin}(X)$ and $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$. Since $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$, we know that $[\mathbf{w}]|_{\mathit{free}(\alpha)} \in \alpha(G_{\hat{\mathbf{w}}})$, for every atom $\alpha \in \mathit{body}(\varphi)$. Then, let $\mathbf{w}_\alpha = \mathit{del}_{Y_\alpha}(\mathbf{w})$, for every $\alpha \in \mathit{body}(\varphi)$. It is straightforward to check that $\hat{\mathbf{w}} = \hat{\mathbf{w}}_\alpha$ and $[\mathbf{w}]|_{\mathit{free}(\alpha)} = [\mathbf{w}_\alpha]$. Hence, $\mathbf{w}_\alpha \in \mathit{linenc}(\alpha)$, for every $\alpha \in \mathit{body}(\varphi)$. As such, $\mathbf{w} \in \mathit{del}^{-1}_{\Sigma \cup \mathit{free}(\alpha), Y_\alpha}(\mathit{linenc}(\alpha))$, for every $\alpha \in \mathit{body}(\varphi)$. Hence, $\mathbf{w} \in K$. □

PROPOSITION 6.13. *linenc*(φ) *is regular, for every CRPQ* φ.

PROOF. Let $X = \mathsf{NVars}(\varphi)$ be the set of all node variables occurring in φ and let φ' be the CRPQ $\bigwedge_{\alpha \in \mathit{body}(\varphi)} \alpha$. That is, φ' is equal to φ, except that it does not contain the existential quantification of φ (if any). In particular, $X = \mathsf{NVars}(\varphi) = \mathit{free}(\varphi')$. By Lemma 6.12, $\mathit{linenc}(\varphi') \subseteq \mathsf{Lin}(X)$ is regular.

Let Z be the set $\mathsf{NVars} \setminus \mathit{free}(\varphi)$ of variables that are existentially quantified in φ. Let $L = \mathit{del}_Z(\mathit{linenc}(\varphi'))$. Note that L is regular since the class of regular languages is closed under morphisms. We now show that $L = \mathit{linenc}(\varphi)$. We first show that $L \subseteq \mathit{linenc}(\varphi)$. Assume $\mathbf{w} \in L$. Then, there exists $\mathbf{w}'$ in $\mathit{linenc}(\varphi')$ such that $\mathbf{w} = \mathit{del}_Z(\mathbf{w}')$. In particular, since $\mathbf{w}'$ is X-linear, $\mathbf{w}$ is $\mathit{free}(\varphi)$-linear. Since $\mathbf{w}' \in \mathit{linenc}(\varphi')$, we know that $[\mathbf{w}'] \in \varphi'(G_{\hat{\mathbf{w}}'})$. Hence, $[\mathbf{w}']|_{\mathit{free}(\varphi)} \in \varphi(G_{\hat{\mathbf{w}}'})$. It is straightforward to check that, since $\mathbf{w} = \mathit{del}_Z(\mathbf{w}')$, we have $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$ and $[\mathbf{w}] = [\mathbf{w}']|_{\mathit{free}(\varphi)}$. As such, $\mathbf{w} \in \mathit{linenc}(\varphi)$.

We next show that $L \supseteq \mathit{linenc}(\varphi)$. Assume $\mathbf{w} \in \mathit{linenc}(\varphi)$. Then $\mathbf{w} \in \mathsf{Lin}(\mathit{free}(\varphi))$ and $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$. Since $[\mathbf{w}] \in \varphi(G_{\hat{\mathbf{w}}})$, there exists $\nu' \in \varphi'(G_{\hat{\mathbf{w}}})$ with $[\mathbf{w}] = \nu'|_{\mathit{free}(\varphi)}$. Then, for all X-linear strings $\mathbf{w}'$ with $(\hat{\mathbf{w}}', [\mathbf{w}']) = (\hat{\mathbf{w}}, \nu')$ we have $\mathbf{w}' \in \mathit{linenc}(\varphi')$. At least one such $\mathbf{w}'$ must satisfy $\mathbf{w} = \mathit{del}_Z(\mathbf{w}')$ and hence $\mathbf{w} \in L$, as desired. □

6.5.2. Parses. Let V be a finite set of span variables. A string $\mathbf{w}$ is called a *V-parse* if $\mathbf{w} \in (\Sigma \cup \tilde{V})^*$, $\mathbf{w}$ is $\tilde{V}$-linear and, moreover, for every $x \in V$ it holds that $x^\vdash$ occurs before $x^\dashv$ in $\mathbf{w}$. (Recall that $\tilde{V} = \{x^\vdash, x^\dashv \mid x \in V\}$.) Clearly, if $\mathbf{w}$ is a V-parse, then $[\mathbf{w}](x^\vdash) \leq [\mathbf{w}](x^\dashv)$. Therefore, $[\mathbf{w}]$ naturally corresponds to the unique span assignment μ over V on $\hat{\mathbf{w}}$ such that $\mu(x) = [[\mathbf{w}](x^\vdash), [\mathbf{w}](x^\dashv)\rangle$, for every $x \in V$. We denote this μ by $[\![\mathbf{w}]\!]$ in what follows. Note that $\mu = [\![\mathbf{w}]\!]$ if and only if $\tilde{\mu} = [\mathbf{w}]$.

Let $\mathsf{Parses}(V)$ denote the set of all V-parses. A *V-parse language* is a set $L \subseteq \mathsf{Parses}(V)$. Since $\tilde{V}$ is finite, it is not difficult to check by means of a finite state automaton that $\mathbf{w} \in (\Sigma \cup \tilde{V})^*$ is a V-parse. Hence, we have the following lemma.

LEMMA 6.14. *$\mathsf{Parses}(V)$ is regular, for every finite set V of span variables.*

Define, for every spanner P over a finite set V of span variables, the V-parse language $\tilde{P}$ to be

$$\begin{aligned}\tilde{P} &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \mu \in P(\mathbf{s}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\![\mathbf{w}]\!] = \mu\} \\ &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid [\![\mathbf{w}]\!] \in P(\hat{\mathbf{w}})\}.\end{aligned}$$

PROPOSITION 6.15. *$\tilde{P}$ is regular, for every $P \in [\![\mathsf{VA}_{\mathsf{set}}]\!]$.*

PROOF. Let $V = \mathsf{SVars}(V)$. By Proposition 6.7, there exists a UCRPQ $\varphi = \varphi_1 \vee \cdots \vee \varphi_k$ that defines P. Then let $L = \mathsf{Parses}(V) \cap \bigcup_{i=1}^k \mathit{linenc}(\varphi_i)$. Since every $\mathit{linenc}(\varphi_i)$ is regular by Proposition 6.13 and since $\mathsf{Parses}(V)$ is regular by Lemma 6.14, we obtain that L is also regular. Then, by definition of $\mathit{linenc}(\varphi_i)$ and because $\varphi(G_{\mathbf{s}}) = \bigcup_{i=1}^m \varphi_i(G_{\mathbf{s}})$, we have

$$\begin{aligned}L &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \nu \in \varphi(G_{\mathbf{s}}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\mathbf{w}] = \nu\} \\ &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \mu \in P(\mathbf{s}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\mathbf{w}] = \tilde{\mu}\} \\ &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \mu \in P(\mathbf{s}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\![\mathbf{w}]\!] = \mu\} \\ &= \tilde{P}. \quad \square\end{aligned}$$

Conversely, define, for each $L \subseteq \mathsf{Parses}(V)$ the spanner $[\![L]\!]$ such that $[\![L]\!](\mathbf{s}) = \{[\![\mathbf{w}]\!] \mid \mathbf{w} \in L, \hat{\mathbf{w}} = \mathbf{s}\}$. Note that, since for every $\mathbf{s}$ and every span assignment μ over V on $\mathbf{s}$ we can find a V-parse $\mathbf{w}$ such that $\hat{\mathbf{w}} = \mathbf{s}$ and $[\![\mathbf{w}]\!] = \mu$, we have $[\![\tilde{P}]\!] = P$, for every spanner P.

PROPOSITION 6.16. *$[\![L]\!] \in [\![\mathsf{VA}_{\mathsf{set}}]\!]$, for every regular parse language L.*

PROOF. Let $A = (Q^A, q_0^A, q_f^A, \delta^A)$ be an NFA over $\Sigma \cup \tilde{V}$ such that $\mathcal{L}(A) = L$. We then define the vset-automaton $B = (Q^B, q_0^B, q_f^B, \delta^B)$ such that (1) the states Q^B of B are the same as the states Q^A of A; (2) $q_0^B = q_0^A$; (3) $q_f^B = q_f^A$; and (4) δ^B contains exactly the same transitions as A, except that $x^\vdash$ becomes $x\vdash$ and $x^\dashv$ becomes $\dashv x$. Specifically, δ^B contains all transitions

—(q, σ, q') with $(q, \sigma, q') \in \delta^A$ and $\sigma \in \Sigma$;
—(q, ϵ, q') with $(q, \epsilon, q') \in \delta^A$;
—$(q, x\vdash, q')$ with $(q, x^\vdash, q') \in \delta^A$ and $x \in \mathsf{SVars}$; and
—$(q, \dashv x, q')$ with $(q, x^\dashv, q') \in \delta^A$ and $x \in \mathsf{SVars}$.

It is now routine to check that $[\![L]\!] = [\![B]\!]$. □

Let RParses denote the class of all *regular* V-parse languages, for some finite set V of span variables. From Propositions 6.15 and 6.16, we obtain the following.

COROLLARY 6.17. $[\![\mathsf{RParses}]\!] = [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

Our next step is to prove the following proposition.

PROPOSITION 6.18. $[\![\mathsf{CRPQ}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

PROOF. Let φ be a CRPQ that defines a spanner P. Let $V = \mathsf{SVars}(P)$. In particular, $\tilde{V} = \mathit{free}(\varphi)$ and $\{\tilde{\mu} \mid \mu \in P(\mathbf{s})\} = \varphi(G_{\mathbf{s}})$, for every $\mathbf{s} \in \Sigma^*$. Then, let $L = \mathit{linenc}(\varphi) \cap \mathsf{Parses}(V)$. We have that L is regular since $\mathit{linenc}(\varphi)$ is regular by Proposition 6.13 and $\mathsf{Parses}(V)$ is regular by Lemma 6.14. Therefore, $[\![L]\!] \in [\![\mathsf{VA}_{\mathsf{set}}]\!]$ by Proposition 6.16. It remains to show that $P = [\![L]\!]$, for which it suffices to show that $\tilde{P} = L$ since then $P = [\![\tilde{P}]\!] = [\![L]\!]$. Now observe that, since φ defines P,

$$\begin{aligned} L &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \nu \in \varphi(G_{\mathbf{s}}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\mathbf{w}] = \nu\} \\ &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \mu \in P(\mathbf{s}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\mathbf{w}] = \tilde{\mu}\} \\ &= \{\mathbf{w} \in \mathsf{Parses}(V) \mid \exists \mathbf{s} \in \Sigma^*, \exists \mu \in P(\mathbf{s}) \text{ such that } \hat{\mathbf{w}} = \mathbf{s}, [\![\mathbf{w}]\!] = \mu\} \\ &= \tilde{P} \end{aligned}$$

as desired. □

PROPOSITION 6.19. $[\![\mathsf{UCRPQ}]\!] \subseteq [\![\mathsf{VA}_{\mathsf{set}}]\!]$.

PROOF. Let P be a spanner defined by UCRPQ $\varphi = \varphi_1 \vee \cdots \vee \varphi_k$. Then every φ_i defines a spanner P_i, and $P = P_1 \cup \cdots \cup P_k$. From Proposition 6.18, we know that $P_i \in [\![\mathsf{VA}_{\mathsf{set}}]\!]$, for every $1 \leq i \leq k$. Then, since $[\![\mathsf{VA}_{\mathsf{set}}]\!]$ is closed under union by Theorem 4.12, also $P \in [\![\mathsf{VA}_{\mathsf{set}}]\!]$. □

By combining Propositions 6.7 and 6.19, we obtain Theorem 6.5.

6.6. Proof of Proposition 6.6

The proof uses the technical tools developed in Section 6.5. Let $Y = \{y_1, \ldots, y_n\}$, let $Y^{\vdash} = \{y_1^{\vdash}, \ldots, y_n^{\vdash}\}$ and. let $Y^{\dashv} = \{y_1^{\dashv}, \ldots, y_n^{\dashv}\}$. Let $f\colon \Sigma \cup \{x_1, \ldots, x_n\} \to \Sigma \cup Y^{\vdash}$ be the morphism defined by

$$f(a) = \begin{cases} y_i^{\vdash} & \text{if } a = x_i, 1 \leq i \leq n, \\ a & \text{otherwise.} \end{cases}$$

Then consider $L = \mathit{del}_{Y^{\dashv}}^{-1}(f(\mathit{linenc}(\varphi))) \cap \mathsf{Parses}(V)$. Clearly, $L \subseteq \mathsf{Parses}(V)$. Moreover, because $\mathit{linenc}(\varphi)$ is regular by Proposition 6.13, and because the class of regular languages is closed under morphisms and inverse morphisms, L is regular. Therefore, $[\![L]\!] \in [\![\mathsf{VA}_{\mathsf{set}}]\!]$ by Proposition 6.16. It is now routine to check that $P = [\![L]\!]$ satisfies the claimed condition.

6.7. CRPQs with String Equality and Core Spanners

In light of the correspondence between UCRPQs and regular spanners given by Theorem 6.5, it is natural to ask whether there exists an extension of UCRPQs that corresponds to the core spanners. In this section, we show that such an extension exists: it suffices to add to UCRPQs the ability to check string equality.

To formally define this extension of UCRPQs, fix an infinite set PVars of *path variables*, pairwise disjoint from NVars, SVars, and Σ. A *conjunctive regular path query*

with string equality (CRPQ$^=$) over an alphabet Δ is a formula φ of the form

$$\exists \vec{z}\, \exists \vec{p}\, \left(\bigwedge_{i=1}^{m} (x_i, p_i : L_i, y_i) \wedge \bigwedge_{j=1}^{n} \left(p_j^1 = p_j^2\right) \right),$$

where (x_i, L_i, y_i) are RPQs; $p_1, \ldots, p_m$ are pairwise distinct path variables; $p_1^1, p_1^2, \ldots, p_n^1, p_n^2$ are path variables in $\{p_1, \ldots, p_m\}$; $\vec{z}$ is a sequence of node variables; and $\vec{p} = p_1, \ldots, p_m$. (Note, in particular, that all path variables are quantified in a CRPQ$^=$.)

Similarly to normal CRPQs, a CRPQ$^=$ formula φ evaluates to a set of mappings $\mathit{free}(\varphi) \to V$ when evaluated on a Δ-labeled graph $G = (V, E)$. To formally define these semantics, let ν be a mapping that associates to each node variable a node in G, and to each path variable a path in G. We define the relationship $(G, \nu) \models \varphi$ to hold if for each atom $(x_i, p_i : L_i, y_i)$ of φ it holds that $\nu(p_i)$ is a path from $\nu(x_i)$ to $\nu(y_i)$ in G such that $\mathit{str}(\nu(p_i)) \in L_i$ and, moreover, $\mathit{str}(\nu(p_j^1)) = \mathit{str}(\nu(p_j^2))$ for every j with $1 \leq j \leq n$. The semantics $\varphi(G)$ of CRPQ$^=$ φ on G are then the set of all mappings $\nu|_{\mathit{free}(\varphi)}$ such that $(G, \nu) \models \varphi$ for some ν.

A union of CRPQ$^=$ (UCRPQ$^=$) is a formula φ of the form $\varphi_1 \vee \cdots \vee \varphi_k$ where every φ_i is a CRPQ$^=$ and $\mathit{free}(\varphi_1) = \cdots = \mathit{free}(\varphi_k)$. We define $\varphi(G)$ to be $\varphi(G) \cup \cdots \cup \varphi_k(G)$.

UCRPQ$^=$ now define spanners similarly to UCRPQs (cf. Definition 6.4).

THEOREM 6.20. $[\![\mathsf{UCRPQ}^=]\!] = [\![\mathsf{RGX}^{\{\cup, \pi, \bowtie, \varsigma^=\}}]\!]$.

The inclusion $[\![\mathsf{RGX}^{\{\cup, \pi, \bowtie, \varsigma^=\}}]\!] \subseteq [\![\mathsf{UCRPQ}^=]\!]$ is easy to prove using the Core-Simplification Lemma (Lemma 4.19): since each core spanner can be written as an expression of the form $\pi_V SA$ where A is a vset automaton and S is a sequence of selections $\varsigma^=_{x,y}$ we can first translate A to a UCRPQ using Proposition 6.7; assign a unique path variable to each atom in the UCRPQ, and then translate the selections $\varsigma^=_{x,y}$ by corresponding string equalities among the corresponding path variables.

The converse inclusion is a bit trickier since the strings that we compare when evaluating a UCRPQ$^=$ on a marked path may contain the special marker symbols $\triangleright$ and $\triangleleft$, to which we do not have access to when comparing substrings using $\varsigma^=$. Fortunately, this difficulty can be done away with. To explain how this can be done, we need to introduce the following terminology. Recall that in a CRPQ$^=$ φ of the form $\exists \vec{z}\, \exists \vec{p}\, (\bigwedge_{i=1}^m (x_i, p_i : L_i, y_i) \wedge \bigwedge_{j=1}^n (p_j^1 = p_j^2))$ all the p_i, for $1 \leq i \leq m$, are required to be pairwise distinct. For each p_i the atom $(x_i, p_i : L_i, y_i)$ that introduces it is hence uniquely determined. We call x_i the start node variable of p_i, we call y_i the end node variable of p_i, and we call L_i the *range* of p_i. Now call a string comparison $p_j^1 = p_j^2$ in φ Σ-*restricted* if the ranges of both p_j^1 and p_j^2 are subsets of Σ^*. Intuitively, a Σ-restricted comparison compares only strings of paths in which $\triangleright$ and $\triangleleft$ do not occur. A CRPQ$^=$ is Σ-restricted if all of the comparisons $p_j^1 = p_j^2$ for $1 \leq j \leq n$ are Σ-restricted. A UCRPQ$^=$ is Σ-restricted if each of its CRPQ$^=$ is Σ-restricted.

LEMMA 6.21. *On marked paths, every UCRPQ$^=$ is equivalent to a Σ-restricted UCRPQ$^=$.*

PROOF. It suffices to show that, on marked paths, every CRPQ$^=$ is equivalent to a Σ-restricted UCRPQ$^=$. Then fix some CRPQ$^=$ $\exists \vec{z}\, \exists \vec{p}\, (\bigwedge_{i=1}^m (x_i, p_i : L_i, y_i) \wedge \bigwedge_{j=1}^n (p_j^1 = p_j^2))$

which we denote by φ. Then clearly, φ is equivalent on marked paths to

$$\exists \vec{z}\ \exists \vec{p}\ \left(\bigwedge_{i=1}^{m} \left(\bigvee_{K \in \{\Sigma^*, \lhd^+\Sigma^*, \Sigma^*\rhd^+, \lhd^+\Sigma^*\rhd^+\}} (x_i, p_i : L_i \cap K, y_i) \right) \wedge \bigwedge_{j=1}^{n} (p_j^1 = p_j^2) \right).$$

By converting the latter expression into disjunctive normal form we obtain a UCRPQ$^=$ φ' that is equivalent to φ on marked paths. We will now show that each disjunct ψ of φ' is equivalent on marked paths to a Σ-restricted CRPQ$^=$.

First, observe that if ψ contains an equality condition $q = q'$ where the range of q and q' are disjoint, then ψ is unsatisfiable, and we can simply replace it by the unsatisfiable but Σ-restricted CRPQ$^=$ $\bigwedge_{x \in free(\psi)}(x, \emptyset, x)$.

Otherwise, for every equality condition $q = q'$ in ψ, we know that the ranges of q and q' are not disjoint. By construction of φ', the range of q is a subset of some $K \in \{\Sigma^*, \lhd^+\ \Sigma^*, \Sigma^*\rhd^+, \lhd^+\ \Sigma^*\rhd^+\}$. Similarly, the range of q' is a subset of some K' in this set. Then, since the elements of $\{\Sigma^*, \lhd^+\ \Sigma^*, \Sigma^*\rhd^+, \lhd^+\ \Sigma^*\rhd^+\}$ are pairwise disjoint, while the ranges of q and q' are not, it follows that $K = K'$. By case analysis on K, we next show that we can convert each equality condition $q = q'$ to a Σ-restricted equality condition.

—*Case* $K = \Sigma^*$. Then this equality condition is already Σ-restricted.

—*Case* $K = \lhd^+\ \Sigma^*$. Let x (and y) be the start node variable (respectively end node variable) of q and x' (respectively, y') the start node (respectively, end node) variable of q'. Since $K = \lhd^+\ \Sigma^*$ is a superset of the range of both q and q we know that for every $\mathbf{s} \in \Sigma^*$ and every ν such that $(G_\mathbf{s}, \nu) \models \psi$, it must be the case that $str(\nu(q))$ and $str(\nu(q'))$ start with a number of $\lhd$ symbols. This implies that $\nu(x) = \nu(x') = 1$, the start position in $\mathbf{s}$. Moreover, it is easy to see that $str(\nu(q)) = str(\nu(q'))$ if, and only if, in addition, $\nu(y) = \nu(y')$. Therefore, the equality condition $q = q'$ in ψ is equivalent to demanding that x is bound to the same node as x' and y to the same node as y'. We can hence replace $q = q'$ by

$$(x, \lhd, x) \wedge (x, \epsilon, x') \wedge (y, \epsilon, y').$$

Here, the conjunct $(x, \lhd, x)$ ensures that x is bound to position 1, while $(x, \epsilon, x') \wedge (y, \epsilon, y')$ ensures that x is mapped to the same node as x' and y to the same node as y'. Note that no string equality is required in this case.

—The cases $K = \Sigma^*\rhd^+$ and $K = \lhd^+\ \Sigma^*\rhd^+$ are similar. □

Using this lemma, we can establish the inclusion $[\![\mathsf{UCRPQ}^=]\!] \subseteq [\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$ of Theorem 6.20 as follows.

PROPOSITION 6.22. $[\![\mathsf{UCRPQ}^=]\!] \subseteq [\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$

PROOF. Since $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$ is closed under union, it suffices to show that $[\![\mathsf{CRPQ}^=]\!] \subseteq [\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$. Towards establishing this inclusion, let φ be a CRPQ$^=$ that defines the spanner P, and let $V = \{v_1, \ldots, v_l\}$ be the span variables of P. In particular, $free(\varphi) = \tilde{V} = \{v_1^\vdash, v_1^\dashv, \ldots, v_l^\vdash, v_l^\dashv\}$, and, for every string $\mathbf{s}$, we have

$$P(\mathbf{s}) = \{V\text{-tuple } \mu \mid \exists \nu \text{ with } (G_\mathbf{s}, \nu) \models \varphi \text{ s. t. } \mu(v_i) = [\nu(v_i^\vdash), \nu(v_i^\dashv)\rangle \text{ for all } i \text{ with } 1 \le i \le l\}. \quad (6)$$

By Lemma 6.21, we may assume without loss of generality that φ is Σ-restricted. Let φ be $\exists \vec{z}\ \exists \vec{p}\ (\bigwedge_{i=1}^m (x_i, p_i : L_i, y_i) \wedge \bigwedge_{j=1}^n (p_j^1 = p_j^2))$. To express φ as a core spanner, we will obviously need to simulate the equality conditions $\bigwedge_{j=1}^n (p_j^1 = p_j^2)$ by means of the operator $\varsigma^=$ on spans. This is conceptually simple enough: define, for every path

variable p a spanner with a single span variable that starts at the same position as the start node variable of p, and ends at the same position as the end node variable of p. Then use this to enforce the equalities. Notice, however, that p may have endpoint node variables that do not occur in $\tilde{V} = \mathit{free}(\varphi)$. Since correspondence Theorem 6.5 gives us "access" to only the variables in $\tilde{V} = \mathit{free}(\varphi)$ (in the sense that it yields a spanner over V with no reference to the bound node variables $\vec{z}$ of φ), we will employ the correspondence given by Proposition 6.6 towards this end instead.

Formally, let φ' be the CRPQ obtained by removing the path variables, the equality conditions, and the quantification from φ, that is, $\varphi' = \bigwedge_{i=1}^{m}(x_i, L_i, y_i)$. Let X be the set of node variables occurring in φ'. Fix, for every node variable $x \in X$, a new span variable x' that is not in V. Let $X' = \{x' \mid x \in X\}$. By Proposition 6.6, there exists a regular spanner $P' \in [\![\mathsf{RGX}^{\{\cup,\pi,\bowtie\}}]\!]$ with $\mathsf{SVars}(P') = X'$ such that, for all $\mathbf{s} \in \Sigma^*$, we have

$$P'(\mathbf{s}) = \{X'\text{-tuple } \mu \mid \exists \nu \in \varphi'(G_{\mathbf{s}}) \text{ such that for all } x \in X \text{ there exists } k \text{ with } \mu(x') = [\nu(x), k\rangle\}. \quad (7)$$

In other words, the tuples in P' simulate the mappings of φ', including the node variables that are bound in φ. We will now modify P' so that it defines the same spanner as P. Towards this, assume that p_i is one of the path variables mentioned in one of the equality conditions in φ, where $1 \leq i \leq m$, and let x_i and y_i be its start and end node variable in φ, respectively. Since φ is Σ-restricted, we know that for every $\mathbf{s} \in \Sigma^*$ and every mapping ν with $(G_{\mathbf{s}}, \nu) \models \varphi$, it must be the case that ν assigns to p_i the unique path from $\nu(x_i)$ to $\nu(y_i)$ in $G_{\mathbf{s}}$ that traverses only edges labeled by elements of Σ. In other words, $\mathit{str}(\nu(p_i)) = \mathbf{s}_{[\nu(x_i),\nu(y_i)\rangle}$. Hence, we can simulate every equality condition $q = q'$ in φ by checking the equality of the substrings between the start and end node variables of q and q'. From this observation, it ensues that we can express P as follows in $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\zeta^=\}}]\!]$.

(1) Fix, for every path variable p_i in φ with $1 \leq i \leq m$ a new span variable p_i' not in $V \cup X$. Let $Y = X \cup \{p_1', \ldots, p_m'\}$. Define, for every path variable p_i in φ with start node variable x_i and end node variable y_i the regular spanner P_{p_i} by

$$P_{p_i} = \Sigma^* p_i' \{x_i' \{\Sigma^*\} \Sigma^*\} y_i' \{\Sigma^*\} \Sigma^*.$$

Note in particular that in every tuple output by P_{p_i}, the span assigned to p_i' starts at the same position as x_i' and ends at the start position of y_i'. Therefore, if we denote by Q the spanner $P' \bowtie P_{p_1} \bowtie P_{p_2} \bowtie \cdots \bowtie P_{p_m}$ with $\mathsf{SVars}(Q) = Y$, then, by (7), we have

$$Q(\mathbf{s}) = \{Y\text{-tuple } \mu \mid \exists \nu \in \varphi'(G_{\mathbf{s}}) \text{ such that for all } x \in X \text{ there exists } k \text{ with } \mu(x') = [\nu(x), k\rangle \text{ and } \mu(p_i') = [\nu(x_i), \nu(y_i)\rangle \text{ for every } i \text{ with } 1 \leq i \leq m\}.$$

Then, since, as observed previously, $\mathit{str}(\nu(p_i)) = \mathbf{s}_{[\nu(x_i),\nu(y_i)\rangle}$ for every assignment ν to node and path variables such that $(G_{\mathbf{s}}, \nu) \models \varphi$ we have

$$\begin{aligned}
&(\zeta^{=}_{p_1^1, p_1^2} \cdots \zeta^{=}_{p_n^1, p_n^2} Q)(\mathbf{s}) \\
&= \{\mu \mid \exists \nu \in \varphi'(G_{\mathbf{s}}) \text{ such that, for all } x \in X \text{ there exists } k \text{ with } \mu(x') = [\nu(x), k\rangle \\
&\qquad \text{and } \mu(p_i') = [\nu(x_i), \nu(y_i)\rangle \text{ for every } i \text{ with } 1 \leq i \leq m \\
&\qquad \text{and } \mathit{str}(\mu(p_j'^1)) = \mathit{str}(\mu(p_j'^2)) \text{ for every } j \text{ with } 1 \leq j \leq n\} \\
&= \{\mu \mid \exists \nu \text{ with } (G_{\mathbf{s}}, \nu) \models \varphi \text{ such that, for all } x \in X \text{ there exists } k \text{ with } \mu(x') = [\nu(x), k\rangle \\
&\qquad \text{and } \mu(p_i') = [\nu(x_i), \nu(y_i)\rangle \text{ for every } i \text{ with } 1 \leq i \leq m \\
&\qquad \text{and } \mathit{str}(\mu(p_j'^1)) = \mathit{str}(\mu(p_j'^2)) \text{ for every } j \text{ with } 1 \leq j \leq n\}.
\end{aligned}$$

Now let R denote the core spanner $\pi_{X'}(\varsigma^{=}_{p_1^1,p_1^2} \cdots \varsigma^{=}_{p_n^1,p_n^2} Q)$ with $\mathsf{SVars}(R) = X'$. Then

$$R(\mathbf{s}) = \{X\text{-tuple } \mu \mid \exists \nu \text{ with } (G_{\mathbf{s}}, \nu) \models \varphi \text{ such that for all } x \in X \text{ there exists } k \text{ with } \mu(x') = [\nu(x), k\rangle\}. \quad (8)$$

(2) To finish the proof, observe that R is a spanner over X' whereas the spanner P that we need to express is over V. To obtain the correct output, define, for every span variable $v \in V$, the spanner P_v by

$$P_v = \Sigma^* v\{v^{\vdash'}\{\Sigma^*\}\Sigma^*\}v^{\dashv'}\{\Sigma^*\}\Sigma^*.$$

Note in particular that in every tuple output by P_v, the span assigned to v starts at the same position as $v^{\vdash'}$ and ends at the start position of $v^{\dashv'}$. By combining this observation with Eqs. (8) and (6), we obtain, for every string $\mathbf{s}$,

$$\begin{aligned}
&\pi_{v_1,\ldots,v_l}(R \bowtie P_{v_1} \bowtie \cdots \bowtie P_{v_l})(\mathbf{s}) \\
&\quad = \{V\text{-tuple } \mu \mid \exists \mu' \in R(\mathbf{s}) \text{ such that } \mu(v_i) \\
&\quad = [\mu'(v_i^{\vdash'}), \mu'(v_i^{\dashv'})\rangle \text{ for all } i \text{ with } 1 \le i \le l\} \\
&\quad = \{V\text{-tuple } \mu \mid \exists \nu \text{ with } (G_{\mathbf{s}}, \nu) \models \varphi \text{ such that } \mu(v_i) \\
&\quad = [\nu(v_i^{\vdash}), \nu(v_i^{\dashv})\rangle \text{ for all } i \text{ with } 1 \le i \le l\} \\
&\quad = P(\mathbf{s}).
\end{aligned}$$

This finishes the proof since $\pi_{v_1,\ldots,v_l}(R \bowtie P_{v_1} \bowtie \cdots \bowtie P_{v_l})$ is a spanner in $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie,\varsigma^=\}}]\!]$, as desired. □

One could further extend this discourse, and pose the question whether the so-called *extended CRPQs* introduced by Barceló et al. [2012b], which extend CRPQs with the ability to check any regular relation between path variables (not just string equality) correspond, on marked paths, to $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie\}\cup O}]\!]$, where $O = \{\varsigma^R \mid R \text{ a regular relation}\}$. It is not difficult to see that, if the extended CRPQs can use only regular relations over Σ (which conforms to Σ-restriction) then the proof for CRPQ$^=$ can indeed be generalized. When the extended CRPQs can use regular relations over the extended alphabet $\Sigma \cup \{\triangleright, \triangleleft\}$, however, it is not clear that spanners defined by extended CRPQs can always be expressed in $[\![\mathsf{RGX}^{\{\cup,\pi,\bowtie\}\cup O}]\!]$. We leave an investigation of this question to future work.

7. SUMMARY AND DISCUSSION

We introduced the concept of a spanner, and investigated three primitive spanner representations: regex formulas, vstk-automata and vset-automata. As we showed, the classes of regex formulas and vstk-automata have the same expressive power, and vset-automata (defining the regular spanners) have the same expressive power as the closure of regex formulas under the relational operators union, natural join, and projection. By adding the string-equality operator, we get the core spanners. We gave some basic results on core spanners, like the core-simplification lemma. We discussed selectable string relations, and showed, among other things, that REC is precisely the class of relations selectable by the regular spanners. We showed that regular spanners are closed under difference, but core spanners are not (which we proved using the core-simplification lemma). Finally, we discussed the connection between core spanners and xregexes, and showed a tight connection between regular spanners and CRPQs.

From the perspective of system building, the designer of an extraction rule language negotiates a tradeoff between expressivity, conciseness, and performance. To be an effective tool for building extractors, a language needs to be expressive in the sense

that a rule developer can write typical extractors entirely inside the confines of the domain-specific language, without resorting to custom code. At the same time, the language needs to be concise: Small numbers of simple rules should cover important and common patterns within text. Performance is also important, both in terms of throughput (number of documents annotated per second) and in terms of memory consumption. This work provides a detailed exploration of the expressivity dimension of extraction language design. Our theoretical development captures core operations of a rule language in a way that bridges different semantics for rule languages, including finite state transducers and operator algebras. Certain important operations, such as cleaning and aggregation, are outside the scope of this work; but on the whole, we have established a good understanding of the expressive power that different components of the system provide. Moreover, a central aspect of system building is that of complexity—both *software complexity* (how complicated is it for a developer to build solutions, in terms of the number of rules and their level of sophistication?) and *computational complexity* (how costly is it to execute programs?). We believe that in this work we have set the theoretical framework that will enable the future investigation of such fundamental aspects.

Indeed, this work is our first step in embarking on the investigation of spanners. Many aspects remain to be considered, and many problems remain to be solved. As mentioned previously, one major aspect is that of complexity. For example, what is the complexity of the translations among spanner representations that were applied in this article? What is the (data and combined) complexity that query evaluation entails in each representation? Regarding the difference operator, an intriguing question is whether we can find a simple form, in the spirit of the core-simplification lemma, when adding difference to the representation of core spanners (i.e., the class $\mathsf{VA}_{\mathsf{set}}^{\{\cup,\pi,\bowtie,\zeta^{=},\setminus\}}$); as illustrated here, such a result would be highly useful for reasoning about the expressive power of that class. As another open problem, we repeat the one we mentioned in Section 6: can extended regular expressions express every Boolean core spanner?

Cleaning of *inconsistent tuples* has an important role in the practice of rule-based information extraction [Chiticariu et al. 2010]. As a simple example, on the string `John_Fitzgerald_Kennedy`, one component of an extraction program may identify the span of `John_Fitzgerald` as that of a person name, another may do so for `Fitzgerald_Kennedy`, and a third may do so for `John_Fitzgerald_Kennedy`. As only one of these is the mentioning of a person name, a cleanup resolution filters out two of the three annotations. In CPSL [Appelt and Onyshkevych 1998], for instance, this resolution takes place implicitly at every stage (cascade). A significant differentiator of SystemT's AQL is that it exposes inconsistency cleaning as an explicit relational operator, similarly to selection, and moreover, supports multiple resolution semantics. Yet, this operator is different from a standard selection, as it is not applied in a tuple-by-tuple basis, but rather in an aggregate manner. We have investigated the topic of inconsistency cleaning [Fagin et al. 2014] and established a framework for declarative cleaning through the database concept of inconsistent database *repairs* [Arenas et al. 1999]. Specifically, our framework adopts the notion of *prioritized repairs* of Staworko et al. [2012], and we have shown that our framework provides a unified formalism to express and generalize the ad-hoc cleaning strategies of various systems such as SystemT and CPSL.

ACKNOWLEDGMENTS

We are grateful to Pablo Barceló, Kenneth Clarkson, and Leonid Libkin for helpful discussions. We also thank the SystemT group their intensive work in establishing the system, and for useful input.

REFERENCES

Alfred V. Aho. 1990. Algorithms for finding patterns in strings. In *Handbook of Theoretical Computer Science*, Volume A: Algorithms and Complexity (A). North Holland, 255–300.

James F. Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26, 11, 832–843.

Douglas E. Appelt and Boyan Onyshkevych. 1998. The common pattern specification language. In *Proceedings of the TIPSTER Text Program: Phase III*. Association for Computational Linguistics, 23–30.

Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proceedings of PODS*. ACM, 68–79.

Pablo Barceló, Diego Figueira, and Leonid Libkin. 2012a. Graph logics with rational relations and the generalized intersection problem. In *Proceedings of LICS*. IEEE, 115–124.

Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. 2012b. Expressive languages for path queries over graph-structured data. *ACM Trans. Datab. Syst.* 37, 4, 31. DOI: http://dx.doi.org/10.1145/2389241.2389250

Pablo Barceló, Juan L. Reutter, and Leonid Libkin. 2013. Parameterized regular expressions and their languages. *Theoret. Comput. Sci.* 474, 21–45.

Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. 2003. Definable relations and first-order query languages over strings. *J. ACM* 50, 5, 694–751.

Jean Berstel. 1979. *Transductions and Context-Free Languages*. Teubner Studienbücher, Stuttgart.

Anthony J. Bonner and Giansalvatore Mecca. 1998. Sequences, datalog, and transducers. *J. Comput. Syst. Sci.* 57, 3, 234–259.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000a. Containment of conjunctive regular path queries with inverse. In *Proceedings of KR 2000*. 176–185.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000b. View-based query processing and constraint satisfaction. In *Proceedings of LICS*. 361–371.

Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. 2003. A formal study of practical regular expressions. *Int. J. Found. Comput. Sci.* 14, 6, 1007–1018.

Cezar Câmpeanu and Nicolae Santean. 2009. On the intersection of regex languages with regular languages. *Theoret. Comput. Sci.* 410, 24–25, 2336–2344.

Benjamin Carle and Paliath Narendran. 2009. On extended regular expressions. In *Proceedings of LATA 2009*. Lecture Notes in Computer Science, vol. 5457. 279–289.

Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. 2010. SystemT: An algebraic approach to declarative information extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computer Linguistics (ACL'10)*. 128–137.

Mariano P. Consens and Alberto O. Mendelzon. 1990. GraphLog: A visual formalism for real life recursion. In *Proceedings of PODS*. ACM, 404–416.

Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. 1987. A graphical query language supporting recursion. In *Proceedings of SIGMOD Conference*. ACM, 323–330.

Hamish Cunningham. 2002. GATE, A General Architecture for text engineering. *Comput. Human*. 36, 2, 223–254.

Alin Deutsch and Val Tannen. 2001. Optimization properties for classes of conjunctive regular path queries. In *Proceedings of DBPL*. 21–39.

Calvin C. Elgot and J. E. Mezei. 1965. On relations defined by generalized finite automata. *IBM J. Res. Devel.* 9, 47–68.

Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. 2013. Spanners: A formal framework for information extraction. In *Proceedings of PODS*. 37–48.

Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. 2014. Cleaning inconsistencies in information extraction via prioritized repairs. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*. ACM, 164–175.

Daniela Florescu, Alon Y. Levy, and Dan Suciu. 1998. Query containment for conjunctive queries with regular expressions. In *Proceedings of PODS*. 139–148.

Dayne Freitag. 1998. Toward general-purpose learning for information extraction. In *Proceedings of COLING-ACL*. 404–408.

Dominik D. Freydenberger. 2011. Extended regular expressions: Succinctness and decidability. In *Proceedings of STACS (LIPIcs)*. Vol. 9, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 507–518.

Jeffrey Friedl. 2006. *Mastering Regular Expressions*. O'Reilly Media.

Seymour Ginsburg and Xiaoyang Sean Wang. 1998. Regular sequence operations and their use in database queries. *J. Comput. Syst. Sci.* 56, 1, 1–26.

Gösta Grahne, Matti Nykänen, and Esko Ukkonen. 1999. Reasoning about strings in databases. *J. Comput. Syst. Sci.* 59, 1, 116–162.

Ralph Grishman and Beth Sundheim. 1996. Message understanding conference- 6: A brief history. In *Proceedings of COLING*. 466–471.

Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. 2010. Variable automata over infinite alphabets. In *Proceedings of LATA*. 561–572.

Donald E. Knuth. 1968. Semantics of context-free languages. *Math. Syst. Theory* 2, 2, 127–145.

Donald E. Knuth. 1971. Correction: Semantics of context-free languages. *Math. Syst. Theory* 5, 1, 95–96.

Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. 2008. SystemT: A system for declarative information extraction. *SIGMOD Record* 37, 4, 7–13.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*. Morgan Kaufmann, 282–289.

T. R. Leek. 1997. Information extraction using hidden Markov models. Master's thesis University of California, San Diego.

Peter Linz. 2001. *An Introduction to Formal Languages and Automata* 3rd Ed. Jones and Bartlett Publishers, Inc., Sudbury, M.A.

B. Liu, L. Chiticariu, V. Chu, H. V. Jagadish, and F. R. Reiss. 2010. Automatic rule refinement for information extraction. *Proc. VLDB Endow.* 3, 1–2, 588–597.

Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of ICML*. Morgan Kaufmann, 591–598.

Frank Neven and Thomas Schwentick. 2002. Query automata over finite trees. *Theoret. Comput. Sci.* 275, 2, 633–674.

Frank Neven and Jan Van den Bussche. 2002. Expressiveness of structured document query languages based on attribute grammars. *J. ACM* 49, 1, 56–100.

Maurice Nivat. 1968. Transduction des langages de Chomsky. *Ann. Inst. Fourier* 18, 339–455.

Frederick Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. 2008. An algebraic approach to rule-based information extraction. In *Proceedings of ICDE*. IEEE, 933–942.

Ellen Riloff. 1993. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of AAAI*. AAAI Press/The MIT Press, 811–816.

Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy G. Lehnert. 1995. CRYSTAL: Inducing a conceptual dictionary. In *Proceedings of IJCAI*. Morgan Kaufmann, 1314–1321.

Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. 2012. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.* 64, 2–3, 209–246.

Sheng Yu. 1997. Regular Languages. In *Handbook of Formal Languages*, Grzegorz Rozenberg and Arto Salomaa (Eds.), vol. 1, Springer, Chapter 2.

Received August 2013; revised October 2014; accepted October 2014