Optimization and Parallelization of RegEx Based Information Extraction

Doctoral Thesis
(Dissertation)

Accepted by the Bayreuth Graduate School of Natural and Mathematical Sciences (BayNAT), University of Bayreuth (Germany), to obtain the academic degree of
Doktor der Naturwissenschaften (Dr. rer. nat.)
and the transnational University Limburg, represented by Hasselt University (Belgium), to obtain the degree of
Doctor of Sciences: Information Technology
in agreement with a joint-degree-agreement

submitted by

Johannes Doleschal
born in Marktredwitz, Germany

2021

Optimization and Parallelization of RegEx Based Information Extraction

Doctoral Thesis
(Dissertation)

Accepted by the Bayreuth Graduate School of Natural and Mathematical Sciences (BayNAT), University of Bayreuth (Germany), to obtain the academic degree of
Doktor der Naturwissenschaften (Dr. rer. nat.)
and the transnational University Limburg, represented by Hasselt University (Belgium), to obtain the degree of
Doctor of Sciences: Information Technology
in agreement with a joint-degree-agreement

submitted by
Johannes Doleschal
born in Marktredwitz, Germany

2021

Supervisors:
Wim Martens, University of Bayreuth
Frank Neven, Hasselt University

D/2021/2451/64

Date of submission:	May 7th, 2021	
Date of defense:	August 13th, 2021	
Doctoral committee:	Prof. Dr. Wim Martens	(reviewer)
	Prof. Dr. Frank Neven	(reviewer)
	Prof. Dr. Rainer Gemulla	(reviewer)
	Prof. Dr. Christian Knauer	(chairman)
	Prof. Dr. Cristian Riveros	
	Prof. Dr. Jan Van den Bussche	

Abstract

The framework of *document spanners* abstracts the task of *information extraction* from text as a function that maps every document (a string) into a relation over the document's spans (intervals identified by their start and end indices). For instance, the regular document spanners are the regular expressions with capture variables, closed under the relational algebra. The expressive power of the regular spanners is precisely captured by the class of vset-automata—an extension of finite state automata, that mark the endpoints of selected spans. In this thesis, we embark the investigation of multiple different aspects of document spanners. Namely, parallel evaluation, weight annotation, and aggregation of document spanners.

Parallel Evaluation: Programs for extracting structured information from text, namely *information extractors*, often operate separately on document segments obtained from a generic splitting operation such as sentences, paragraphs, k-grams, HTTP requests, and so on. An automated detection of this behavior of extractors, which we refer to as *split-correctness*, would allow text analysis systems to devise query plans with parallel evaluation on segments for accelerating the processing of large documents. Other applications include the incremental evaluation on dynamic content, where re-evaluation of information extractors can be restricted to revised segments, and debugging, where developers of information extractors are informed about potential boundary crossing of different semantic components. We propose and study a new formal framework for split-correctness within the formalism of document spanners. Our analysis studies the complexity of split-correctness over regular spanners. We also discuss different variants of split-correctness, for instance, in the presence of black-box extractors with *split constraints.*

Weight Annotation: Concerning weight annotation, we embark on the investigation of document spanners that can annotate extractions with auxiliary information such as confidence, support, and confidentiality measures. To this end, we adopt the abstraction of provenance semirings by Green, Karvounarakis, and Tannen, where tuples of a relation are annotated with the elements of a commutative semiring, and where the annotation propagates through the positive relational Algebra operators via the semiring operators. Hence, the proposed spanner extension, referred to as an *annotator*, maps every document into an annotated relation over the spans. As a specific instantiation, we explore weighted vset-automata that, similarly to weighted automata and transducers, attach semiring elements to transitions. We investigate key aspects of expressiveness, such as the closure under the positive relational Algebra, and key aspects of computational complexity,

such as the enumeration of annotated answers and their ranked enumeration in the case of ordered semirings. For a number of these problems, fundamental properties of the underlying semiring, such as positivity, are crucial for establishing tractability.

Aggregation: Lastly we investigate the computational complexity of querying text by aggregate functions — such as sum, average, and quantile — on top of regular document spanners. To this end, we formally define aggregate functions over document spanners and analyze the computational complexity of exact and approximate computation. More precisely, we show that in a restricted case, all studied aggregate functions can be computed in polynomial time. In general, however, even though exact computation is intractable, some aggregates can still be approximated with fully polynomial-time randomized approximation schemes (FPRAS).

Zusammenfassung

Das System der *Document Spanner* (frei übersetzt: Abschnittsanfragen) abstrahiert die Aufgabe der *Informationsextraktion* aus Texten als eine Funktion, die Dokumente (Zeichenketten) in Relationen über Abschnitte des Dokuments (identifiziert durch die Start- und Endposition in der Zeichenkette) übersetzt. Reguläre Abschnittsanfragen zum Beispiel sind reguläre Ausdrücke mit Variablen, abgeschlossen unter der relationalen Algebra. Die Ausdrucksstärke von regulären Abschnittsanfragen umfasst exakt die Klasse der sogenannten vset-Automaten — eine Erweiterung von endlichen Automaten, welche die Endpunkte der selektierten Abschnitte markieren. In dieser Arbeit studieren wir mehrere verschiedene Aspekte von Abschnittsanfragen und befassen uns mit der parallelen Auswertung, Annotation von Gewichten und der Aggregation von Abschnittsanfragen.

Parallele Auswertung: Programme, die strukturierte Informationen aus Texten extrahieren, auch *informations Extraktoren* genannt, arbeiten oft unabhängig voneinander an mehreren Teilen eines Dokuments, die durch eine generische Aufteilung des Dokuments in Sätze, Absätze, HTTP Anfragen oder Ähnliches erzeugt werden. Ein automatisches Erkennen eines solchen Verhaltens, das wir als *Aufteilungskorrektheit* bezeichnen, würde es informations Extraktoren erlauben, Ausführungspläne zur parallelen Auswertung zu erzeugen und dadurch die Verarbeitung von großen Dokumenten zu beschleunigen. Andere Anwendungen sind die inkrementelle Auswertung von dynamischen Inhalten, für welche die erneute Auswertung von informations Extraktoren auf überarbeitete Bereiche beschränkt werden kann und die Suche von Fehlern, wobei Entwickler von informations Extraktoren über die mögliche Überschreitung von semantischen Inhaltsgrenzen informiert werden können. In dieser Arbeit definieren und studieren wir ein neues formales System für Aufteilung Korrektheit im Formalismus von Abschnittsanfragen. Unsere Analyse studiert die Komplexität der Aufteilungskorrektheit für reguläre Abschnittsanfragen und wir untersuchen verschiedene Varianten von Aufteilungskorrektheit, zum Beispiel im Zusammenhang mit Black Box Extraktoren unter *Aufteilungsbedingungen*.

Gewichts Annotation: Bezüglich der gewichteten Annotation untersuchen wir Abschnittsanfragen, welche die extrahierten Daten mit zusätzlichen Informationen, wie zum Beispiel der Irrtumswahrscheinlichkeit, dem Support oder Informationen zur Vertraulichkeit anreichern. Dazu passen wir die Abstraktion sogenannter “provenance semirings” (frei übersetzt: Ursprungs Semiringe) von Green, Karvounarakis und Tannen an. In diesen werden Tupel einer Relation mit Elementen eines kommutativen Semirings annotiert und Informationen zum Ursprung mittels der Semiring Operatoren durch positive relationale Algebra Operatoren propatiert. Die vorgeschlagene Erweiterung von Abschnittsanfragen,

welche wir als Annotatoren bezeichnen, bildet Dokumente auf annotierte Relationen über Abschnitte ab. Als eine spezifische Darstellungsmöglichkeit untersuchen wir gewichtete vset-Automaten welche ähnlich zu gewichteten endlichen Automaten und Transduktoren Semiring Elemente zu Zustandsübergängen hinzufügt. Wir untersuchen Schüsseleigenschaften der Ausdrucksstärke, wie zum Beispiel den Abschluss unter positiver relationaler Algebra und Schlüsseleigenschaften der Komplexität, wie zum Beispiel der Aufzählung von annotierten Antworten und deren geordnete Aufzählung im Fall von geordneten Semiringen. Für eine Vielzahl dieser Probleme sind fundamentale Eigenschaften des verwendeten Semirings, beispielsweise Positivität, für effiziente Algorithmen essenziell.

Aggregation: Zuletzt studieren wir die Komplexität der Berechnung von Aggregatfunktionen — wie zum Beispiel Summe, Durchschnitt oder Quantil — über Resultate von regulären Abschnittsanfragen. Dazu definieren wir Aggregatfunktionen über Abschnittsanfragen formal und analysieren die Komplexität der exakten und näherungsweisen Auswertung. Genauer zeigen wir das in einem eingeschränkten Fall, alle betrachteten Aggregatfunktionen in polynomieller Zeit ausgewertet werden können. Allgemein können manche Aggregatfunktionen noch immer durch “fully polynomial-time randomized approximation schemes” (frei übersetzt: randomisierte echt polynomielle Approximationsschemata) angenähert werden, auch wenn eine exakte Berechnung nach allgemein geglaubten Hypothesen nicht in polynomieller Laufzeit möglich ist.

Samenvatting

Het raamwerk van *document spanners* abstraheert het *extraheren van informatie* uit tekst als een functie die elk document (een tekst) omzet in een relatie bestaand uit overspanningen van het document (intervallen geïdentificeerd door hun start- en eindindices). Bijvoorbeeld, de reguliere document spanners zijn de reguliere expressies met variabelen die gesloten zijn onder de relationele algebra. De uitdrukkingskracht van de reguliere spanners valt precies samen met de klasse van vset-automata en deze zijn een uitbreiding van eindigetoestandsautomaten die de eindpunten van geselecteerde overspanningen markeren. In deze dissertatie beginnen we met het onderzoeken van verschillende aspecten van de document spanners zoals parallelle evaluatie, gewicht annotatie en aggregatie van de document spanners.

Parallele evaluatie: Programma's voor het extraheren van gestructureerde informatie uit tekst, genaamd *information extractors*, werken afzonderlijk op verschillende delen van het document, die verkregen zijn uit een generieke splitsingsoperatie. Voorbeelden van zulke delen zijn zinnen, paragrafen, k-grammen, HTTP verzoeken, enzovoort. Het geautomatiseerd detecteren van dit gedrag van de extractors, dat *split-correctness* heet, zou tekstanalyse systemen in staat stellen om queryplannen te bedenken met parallelle evaluatie op de verschillende delen zodat het verwerken van grote documenten versneld kan worden. Andere toepassingen zijn de incrementele evaluatie van dynamische inhoud, waarbij de herevaluatie van de informatie extractie kan beperkt worden tot de aangepaste delen, alsook foutopsporing, waarbij ontwikkelaars van informatie-extractors geïnformeerd worden over potentiële semantische componenten die over deelgrenzen heen gaan. We presenteren een nieuw formeel kader voor split-correctness en bestuderen dit binnen de theorie van document spanners. Onze analyse bestudeert de complexiteit van split-correctness voor reguliere spanners, bovendien bespreken we ook verschillende varianten van split-correctness, bijvoorbeeld bij de aanwezigheid van black-box extractors met *split constraints*.

Annotatie van gewichten: Met betrekking tot de gewichtsannotatie, beginnen we met het onderzoeken van document spanners die overspanningen kunnen annoteren met extra informatie zoals een maat een zekerheid, ondersteuning en vertrouwelijkheid. Hiervoor gebruiken we de abstractie van provenance semirings van Green, Karvounarakis, en Tannen, waarbij de tuples van een relatie geannoteerd worden met de elementen van een commutatieve semiring en waar de annotatie via de semiring operatoren doorgegeven wordt doorheen de positieve relationele Algebra operatoren. Vandaar dat de voorgestelde uitbreiding voor spanners, aangeduid als een *annotator*, elke document toekent aan een geannoteerde relatie die gaat over de overspanningen. Als een specifieke geval, verkennen we de gewogen vset-automata die, op dezelfde manier als gewogen automaten en transductoren, semiring elementen koppelen met overgangen. We onderzoeken belangrijke aspecten van expressiviteit, zoals het gesloten zijn onder de positieve relationele Algebra en belangrijke aspecten van computationele complexiteit, zoals de opsomming van de geannoteerde antwoorden of de gerangschikte opsomming in het geval dat geordende semirings gebruikt worden. Voor een aantal van deze problemen zijn de fundamentele eigenschappen van de onderliggende semiring, zoals positiviteit, cruciaal voor het nagaan van de uitvoerbaarheid ervan.

Aggregatie: Tenslotte onderzoeken we de computationele complexiteit van het bevragen van tekst door aggregatiefuncties zoals som, gemiddelde en kwantiel bovenop de reguliere document spanners. Hiervoor definiëren we formeel de aggregatiefuncties voor document spanners en analyseren we de computationele complexiteit van de exacte berekeningen alsook van de benaderingen. In meer detail, we tonen aan dat, met enkele beperkingen, alle bestudeerde aggregatiefuncties berekend kunnen worden in polynomiale tijd. Algemeen gezien kunnen aggregatie kunnen benaderd worden met fully polynomial-time randomized approximation schemes (FPRAS) ook al is de exacte berekening normaal lastig.

Acknowledgements

First of all, I want to thank my supervisors Wim Martens and Frank Neven, who never failed to support me during my journey as a PhD student. Thank you for enabling me to pursue this joint degree and continually motivating me to work on my research. I have to thank all colleagues for the fruitful discussions, the countless hours of coffee breaks, and their constant supply of cookies, cakes, and apples — I'm pretty sure I ate way more cake and cookies than I brought myself.

I want to thank everybody I worked with during the last years. Special thanks go to Benny Kimelfeld for inviting me multiple times to visit him and constantly bringing up engaging problems to study. I am grateful to Rainer Gemulla for acting as referee and Christian Knauer, Cristian Riveros, and Jan Van den Bussche for being part of my doctoral committee. Furthermore, I am very thankful to all members of my doctoral committee and Tina Popp for their helpful comments and suggestions regarding the final version of my thesis.

On a personal note, I want to thank all my friends and family who supported me over the past years. I am in deep debt to my parents Marita and Martin, and my sister Barbara, without whom I would not be the person I am. Last but certainly not least, I want to thank Verena for her constant support, her patience, and for enduring me during the period of writing this thesis. Thank you for constantly being there for me – even though I sometimes stopped talking mid-sentence to think about my research or kept you from sleeping at night as I started to speak when you wanted to go to sleep.

Contents

Introduction

Chapter 1

On Document Spanners and Information Extraction

A plethora of paradigms have been developed over the past decades towards the challenge of extracting structured information from text—a task generally referred to as Information Extraction (IE). Common textual sources include natural language from a variety of sources such as scientific publications, customer input and social media, as well as machine-generated activity logs. Instantiations of IE are central components in text analytics and include tasks such as segmentation, named-entity recognition, relation extraction, and coreference resolution [137]. Rules and rule systems have consistently been key components in such paradigms, yet their roles have varied and evolved over time. Systems such as Xlog [150] and SystemT [22] use IE rules for materializing relations inside *relational query languages*. Machine-learning classifiers and probabilistic graphical models (e.g., Conditional Random Fields) use rules for *feature generation* [93, 158]. Rules serve as *weak constraints* (later translated into probabilistic graphical models) in Markov Logic Networks [126] (abbrev. MLNs) and in the DeepDive system [151]. Rules are also used for generating *noisy training data* ("labeling functions") in the Snorkel system [130].

The framework of *document spanners* (*spanners* for short) provides a theoretical basis for investigating the principles of relational rule systems for IE [45]. Specifically, a spanner extracts from a document a relation over text intervals, called *spans*, using either atomic extractors or a relational query on top of the atomic extractors. More formally, by a *document* we refer to a string d over a finite alphabet, a *span* of d represents a substring of d by its start and end positions, and a *spanner* is a function that maps every document d into a relation over the spans of d. The most studied spanner language is that of the *regular* spanners: atomic extraction is via *regex formulas*, which are regular expressions with capture variables, and relational manipulation is via the relational algebra: projection, natural join, union, and difference.[1] Equivalently, the regular spanners are the ones expressible as *variable-set automata* (vset-automata for short), which are nondeterministic finite-state automata that can open and close variables (playing the role of the attributes of the extracted relation). Interestingly, there has been an independent recent effort to express artificial neural networks for natural language processing by means of finite-state automata [106, 108, 167].

[1] Adding string equality selection would result in *Core-Spanners*, which are more powerful.

In this thesis, we study multiple different aspects of document spanners. In Part I introduces the framework of *split-correctness*, where we study whether or not document spanners can be evaluated in a distributed setting. More specifically, we introduce and study the framework of split-correctness in Chapter 3 and study the computational complexity for the class of regular document spanners in Chapter 4. In Part II of the thesis, we study quantitative aspects of document spanners. Chapter 5 defines and studies an extension of document spanners, which adopt a quantitative approach. That is, each extracted tuple is associated with a level of confidence that the tuple coincides with the intent. The last Chapter, Chapter 6, studies the computational complexity of evaluating *aggregate functions* over regular document spanners. We begin by giving some motivation for the different aspects of the thesis.

1.1 Split-Correctness

When applied to a large document, an IE function may incur a high computational cost and, consequently, an impractical execution time. However, it is frequently the case that the program, or at least most of it, can be distributed by separately processing smaller chunks in parallel. For instance, Named Entity Recognition (NER) is often applied separately to different sentences [87, 89], and so are instances of *Relation Extraction* [99, 170]. Algorithms for *coreference resolution* (identification of places that refer to the same entity) are typically bounded to limited-size windows; for instance, Stanford's well known *sieve* algorithm [129] for coreference resolution processes separately intervals of three sentences [90]. Sentiment extractors typically process individual paragraphs or even sentences [119]. It is also common for extractors to operate on windows of a bounded number N of words (tokens), also known as N*-grams* or *local contexts* [21, 59]. Finally, machine logs often have a natural split into semantic chunks: query logs into queries, error logs into exceptions, web-server logs into HTTP messages, and so on.

Tokenization, N-gram extraction, paragraph segmentation (identifying paragraph breaks, whether or not marked explicitly [67]), sentence boundary detection, and machine-log itemization are all examples of what we call *splitters*. When IE is programmed in a development framework such as the aforementioned ones, we aspire to deliver the premise of being declarative—the developer specifies *what* end result is desired, and not *how* it is accomplished efficiently. In particular, we would like the system to automatically detect the ability to split and distribute. This ability may be crucial for the developer (e.g., data scientist) who often lacks the expertise in software and hardware engineering. In Part I of the thesis, we embark on a principled exploration of automated inference of *split-correctness* for information extractors. That is, we explore the ability of a system to detect whether an IE function can be applied separately to the individual segments of a given splitter, *without changing the semantics*.

The basic motivation comes from the scenario where a long document is pre-split by some conventional splitters (like the aforelisted ones), and developers provide different IE functions. If the system detects that the provided IE function is correctly splittable, then it can utilize its multi-processor or distributed hardware to parallelize the computation. Moreover, the system can detect that IE programs are frequently splittable, and recommend the system administrator to materialize splitters upfront. Even more, the split guarantee facilitates *incremental maintenance*: when a large document undergoes a minor edit, like in the Wikipedia model, only the relevant segments (e.g., sentences or paragraphs) need to be reprocessed.

1.2 Weight Annotations

To date, the research on spanners has exclusively adopted a Boolean approach: *a tuple is either extracted or not*. Nevertheless, when applied to noisy or fuzzy domains such as natural language, modern approaches in artificial intelligence adopt a quantitative approach where each extracted tuple is associated with a level of confidence that the tuple coincides with the intent. When used within an end-to-end IE system, such confidence can be used as a principled way of tuning the balance between precision and recall. For instance, in probabilistic IE models (e.g., Conditional Random Fields), each extraction has an associated probability. In systems of weak constraints (e.g., MLN), every rule has a numerical weight, and the confidence in an extraction is an aggregation of the weights of the invoked rules that lead to the extraction. IE via artificial neural networks typically involves thresholding over a produced score or confidence value [23, 121]. Numerical scores in the extraction process are also used for quantifying the similarity between associated substrings, as done with sequence alignment and edit distance in the analysis of biological sequences such as DNA and RNA [161, 166].

In Part II of this thesis we embark on the investigation of spanners that quantify the extracted tuples. We do so by adopting the concept of *annotated relations* from the framework of *provenance semirings* by Green et al. [62]. In essence, every tuple of the database is annotated with an element of a commutative semiring, and the positive relational algebra manipulates both the tuples and their annotations by translating relational operators into semiring operators (e.g., product for natural join and sum for union). An annotated relation is referred to as a $\mathbb{K}$-relation, where $\mathbb{K}$ is the domain of the semiring. The conceptual extension of the spanner model is as follows. Instead of a spanner, which is a function that maps every document d into a relation over the spans of d, we now consider a function that maps every document d into a $\mathbb{K}$-relation over the spans of d. We refer to such a function as a $\mathbb{K}$-*annotator*. Interestingly, as in the relational case, we can vary the meaning of the annotation by varying the semiring.

- We can model *confidence* via the *probability* (a.k.a. *inside*) semiring and the *Viterbi* (best derivation) semiring [61].
- We can model *support* (i.e., number of derivations) via the *counting* semiring [61].
- We can model *access control* via the semiring of the *confidentiality policies* [50] (e.g., does the extracted tuple require reading top-secret sections? which level suffices for the tuple?).
- We can model the traditional spanners via the *Boolean* semiring.

As a specific instantiation of $\mathbb{K}$-annotators, we study the class of $\mathbb{K}$-*weighted vset-automata*. Such automata generalize vset-automata in the same manner as weighted automata and weighted transducers (cf. the Handbook of Weighted Automata [39]): transitions are weighted by semiring elements, the cost of a run is the product of the weights along the run, and the weight (annotation) of a tuple is the sum of costs of all the runs that produce the tuple. (Again, there has been recent research that studies the connection between models of artificial neural networks in natural language processing and weighted automata [143].)

Our investigation answers several fundamental questions about the class of $\mathbb{K}$-weighted vset-automata:

1. Is this class closed under the positive relational algebra (according to the semantics of provenance semirings [62])?
2. What is the complexity of computing the annotation of a tuple?
3. Can we enumerate the annotated tuples as efficiently as we can do so for ordinary vset-automata (i.e., regular document spanners)?
4. In the case of ordered semirings, what is the complexity of enumerating the answers in ranked order by decreasing weight?

Our answers are mostly positive and show that $\mathbb{K}$-weighted vset-automata possess appropriate expressivity and tractability properties. As for the last question, we show that ranked enumeration is intractable and inapproximable for some of the aforementioned semirings (e.g., the probability and counting semirings), but tractable for positively ordered and bipotent semirings, such as the Viterbi semiring.

1.3 The Complexity of Aggregates

The last aspect of document spanners we study is the computational complexity of evaluating *aggregate functions* over regular spanners. These are queries that map a document d and a spanner S into a number $\alpha(S(d))$, where $S(d)$ is the relation obtained by applying S to d and α is a standard aggregate function: count, sum, average, min, max, or quantile. There are various scenarios where queries that involve aggregate functions over spanners can be useful. For example, such queries arise in the extraction of statistics from textual resources like medical publications [116] and news reports [142]. As another example, when applying advanced text search or protein/DNA motif matching using regular expressions [24, 112], the search engine typically provides the (exact or approximate) number of answers, and we would like to be able to compute this number without actually computing the answers, especially when the number of answers is prohibitively large. Finally, when programming feature generators or labeling functions in extractor development, the programmer is likely to be interested in aggregate statistics and summaries for the extractions (e.g., to get a holistic view of what is being extracted from the dataset, such as quantiles over extracted ages and so on), and again, we would like to be able to estimate these statistics faster than it takes to materialize the entire set of answers.

Our main objective is to understand when it is tractable to compute $\alpha(S(d))$. This question raises closely related questions that we also discuss, such as when the materialization of intermediate results (which can be exponentially large) can be avoided. Furthermore, when the exact computation of $\alpha(S(d))$ is intractable, we study whether it can be approximated.

At the technical level, each aggregate function (with the exception of count) requires a specification of how an extracted tuple of spans represents a number. For example, the number 21 can be represented by the span of the string "21", "21.0", "twenty one", "twenty first", "three packs of seven" and so on. To abstract away from specific textual representations of numbers, we consider several means of assigning weights to tuples. To this end, we assume that a (representation of a) *weight function* w, which maps every tuple of $S(d)$ into a number, is part of the input of the aggregate functions. Hence, the general form of the aggregate query we study is $\alpha(S, d, w)$. The direct approach to evaluating $\alpha(S, d, w)$ is to compute $S(d)$, apply w to each tuple, and apply α to the resulting sequence of numbers. This approach works well if the number of tuples in $S(d)$ is manageable (e.g., bounded by some polynomial). However, the number of tuples in $S(d)$ can be exponential in the number of variables of S, and so, the direct approach takes exponential time in the worst case. We will identify several cases in which $S(d)$ is exponential, yet $\alpha(S(d))$ can be computed in polynomial time.

Therefore, we focus on identifying when this exponential cost is unavoidable (lower bounds), when it can be avoided, and when approximatations allow to overcome intractability. Furthermore, we study how the choice of the weight function w impacts tractability.

It is not very surprising that, at the level of generality we adopt, each of the aggregate functions is intractable (#P-hard) in general. However, this hardness (and generally our lower bounds) applies to specific numerical representations w that have a relatively simple (or even a trivial) form. Hence, we focus on several assumptions that can potentially reduce the inherent hardness of evaluation:

- Restricting to positive numbers;
- Restricting to weight functions w that are determined by a single span or defined by (unambiguous) weighted vset-automata;
- Restricting to spanners that are represented by an unambiguous variant of vset-automata;
- Allowing for a randomized approximation (FPRAS, i.e., fully polynomial-time randomized approximation schemes).

Our analysis shows which of these assumptions brings the complexity down to polynomial time, and which is insufficient for tractability. Importantly, we derive an interesting and general tractable case for each of the aggregate functions we study.

1.4 Related Work

In this section we will discuss research which is closely related to the framework of document spanners. To this end, we begin by giving an overview over related research areas and conclude this section by giving an overview over the research on document spanners.

1.4.1 Related Research Areas

Many aspects of document spanners are also studied in other contexts, which we will discuss in this section.

Enumeration: Building upon Johnson et al. [74], there is plenty of research on enumerating the answers of queries expressed in different formalisms. One line of research studies enumeration of queries expressed in first order logic [40, 144] and monadic second order logic [4, 5, 6, 8, 94, 114, 115].

Another line of work [144, 147, 165] studies to how the complexity of enumerating is affected by assumptions on the underlying data, like the assumption that an input graph is nowhere dense [144]. In this thesis, we study enumeration and ranked enumeration of so called $\mathbb{K}$-annotators in Chapter 5.

Query Evaluation on Succinct Data Representations: Reminiscent yet different from our work on spanner aggregation, there is plenty of research on query answering on succinct representations of data. For instance in artificial intelligence, where knowledge compilation is used to answer reasoning tasks, based on succinct Boolean circuits (e.g. Darwiche and Marquis [29]) or factorized databases [79, 80, 118, 139], a succinct and lossless representation of relational data, which, for instance, can be used to speedup machine learning algorithms (cf. Olteanu and Schleich [118]). As we will discuss in Chapter 6, document spanners can also be seen as a succinct representation of their output on a given document. We study whether or not this representation can be used to compute aggregates over the output relation, without materializing the output relation.

Incremental View Maintenance and Query Evaluation under Updates Incremental maintenance of relational views dates back to Gupta et al. [65] and studies the question of updating query results on relational databases under updates to the underlying data. To only name a view lines of research, Griffin and Libkin [63] study maintenance of materialized views with duplicates and incremental XPath evaluation is studied by Björklund et al. [16]. More recently, Schwentick et al. [145] study maintainability of queries by rules expressed in first order logic and Keppeler [76] studies query evaluation under updates to the underlying database. The framework of split-correctness, which we introduce in Part I, can also be seen as a first step towards evaluation of document spanners under updates to the input document.

Parallel Query Evaluation: The framework of split-correctness is inspired by the parallel-correctness framework as proposed by Ameloot et al. [9, 10]. The parallel-correctness framework considers the parallel evaluation of relational queries and studies whether or not conjunctive queries can be evaluated distributively, in a setting where the data is distributed according to a distribution policy. Recent work studies parallel-correctness for conjunctive queries with union and negation [58], parallel-correctness for conjunctive queries under bag semantics [78], and distribution policies for datalog [77]. In Part I of this thesis, we will introduce the framework of split-correctness and study whether or not document spanners can be evaluated in a parallel fashion.

Weighted Automata and Transducers: There is extensive research on weighted automata and weighted transducers, studying various aspects thereof. To only name a few recent lines of research, weighted automata with storage are studied by Herrmann et al. [68], Mazowiecki and Riveros [107] study pumping lemmas for weighted automata, and the task of extracting weighted automata from Recurrent Neural Networks is studied by Okudono et al. [117]. We refer to the Handbook of Weighted Automata [39] for more background on weighted automata. In Part II of this thesis, we will introduce and study an extension of weighted automata, which can be used to enrich the output of a document spanner by provenance information.

Determinism and Unambiguity: Plenty of research studies different notions of determinism and unambiguity for regular (tree) languages. Brüggemann-Klein and Wood [19], define and study deterministic regular expressions, which loosely speaking are regular expressions that can be translated efficiently into deterministic finite automata. Based on this, there is plenty of work [28, 56, 64, 95, 96, 97] studying various aspects of deterministic regular expressions.

Similar to deterministic regular expressions, Brüggemann-Klein and Wood [20] and Brüggemann-Klein[18] define and study unambiguous regular languages. Concerning automata, Stearns and Hunt III [157] show that containment for unambiguous automata can be checked in polynomial time. As shown by Seidl [148], this result can also be extended to unambiguous tree automata.

Finding the correct notion of determinism and unambiguity is crucial for many applications. For instance, in the context of XML, Martens and Niehren [102] show that minimization of bottom-up deterministic unranked tree automata is NP-complete, whereas bottom-up deterministic stepwise tree automata allow for polynomial time minimization. We refer to Colcombet [25], for a survey on different notions of determinism and unambiguity. We study different notions of determinism for so called vset-automata in Chapter 4.1.

Sequential Pattern Mining: Introduced by Srikant and Agrawal [156], the goal of sequential pattern mining to find the most frequent patterns in a dataset. For instance, Beedkar et al. [15] define and study an unified framework for frequent pattern mining under subsequence constraints. Similar to regular document spanners, their algorithms build upon finite state transducers as their underlying computational model. We refer to Mabroukeh and Ezeife [98] for a comprehensive study of sequential pattern mining algorithms. In Chapter 5 we define and study so called $\mathbb{K}$-annotators, which can be used as a means of extracting frequent patterns.

Formal Language Theory: This work is also closely related to research in formal language theory [133] like pattern languages [11, 30, 152], extended regular expressions [56, 51, 131] and language decomposition [103, 136]. For instance we study the connection of this work to the classical problem of language primality in Chapter 4.5.

1.4.2 Research on Document Spanners

Since their introduction by Fagin, Kimelfeld, Reiss, and Vansummeren [44, 45, 46], the research on document spanners has focused on a variety of different aspects, which we will discuss now. We note that this discussion is by no means meant to be fully exhaustive.

Spanner Evaluation: Freydenberger, Kimelfeld, and Peterfreund [54] study the computational complexity of the evaluation of (unions of) conjunctive queries ((U)CQs) over regular document spanners. They show that, even though evaluation is NP-complete, UCQs can be evaluated with polynomial delay under the assumption that each involved CQ has a bounded number of atoms.

Enumeration: A series of articles [7, 8, 17, 48, 49, 141] study the computational complexity of enumerating the output relations of regular document spanners, leading to Amarilli et al. [8] who show that regular document spanners, represented by nondeterministic vset-automata can be enumerated with preprocessing linear in the document and polynomial in the spanner, and delay constant in the document and polynomial in the spanner. Schmid and Schweikardt [141] study evaluation and enumeration of document spanners over compressed documents.

Concerning ranked enumeration, Bourhis et al. [17] study the setting, where output tuples are ranked according to cost functions, expressed in monadic second order logic. To this end, they define cost transducers, which are quite similar to the unambiguous weighted vset-automata, which we study in Chapter 5.[2]

Counting and Uniform Sampling: Arenas et al. [12, 13] show that, given a regular document spanners and a document, there is a polynomial time algorithm for randomized uniform sampling of the output relation. Furthermore, there is a fully polynomial time approximation scheme which approximates the size of the output relation.

[2]To be precise, cost transducers are a bit more restrictive, as they require the multiplicative monoid of the semiring to be a group.

Core Spanners: Regular spanners closed under string equality selection, also called core spanners, are also studied in literature [45, 53, 123, 140]. Freydenberger and Holldack [53] study the expressive power of core spanners and compare them to similar formalisms like patterns, word equations and a subclass of extended regular expressions. Furthermore, they study the query evaluation and static analysis problems for core spanners and, for instance, show that universality and equivalence of core spanners are not semidecidable. Building upon this, Schmid and Schweikardt [140] define a fragment of core spanners, which incorporates features of core spanners directly into regular languages and for which typical static analysis questions are decidable.

Document Spanners and Logic: Another line of work [52, 55, 57], studies document spanners by the means of logic and finite model theory. That is, Freydenberger and Peterfreund [55] define and study the logic FC, which combines aspects of finite model theory and the theory of concatenation [127]. Based on this, Freydenberger and Thompson [57] study spanner evaluation under updates to the document.

Context Free Document Spanners: Another extension of regular document spanners is studied by Peterfreund [122], who defines context-free document spanners, by allowing context free languages in the spanner definitions.

Extracting Incomplete Information: Maturana, Riveros, Vrgoč [105] extend the classical setting, where all tuples in the output relation must assign the same set of variables. In their work, the authors provide some preliminary results on expressiveness and computational complexity of this extended setting of document spanners, which are further studied by Peterfreund et al. [123].

Datalog: Another line of work [14, 47, 111, 124, 150] studies the combination of document spanners and datalog. That is, Fagin et al. [47] define and study a framework for declarative cleaning of inconsistencies, using denial constraints over information extraction programs and prioritized repairs. Furthermore, Peterfreund et al. [124] show that recursive Datalog over regular document spanners exactly captures all spanners which can be evaluated in PTIME.

Ontology Mediated Information Extraction: Lembo et al. [91] and Scafoglieri and Lembo [138] study ontology mediated information extraction. In a nutshell, the proposed algorithms use semantic knowledge from ontologies to improve the results of queries expressed by document spanners.

1.5 Contributions by other Authors

This work is the result of many discussions with other researchers and is based on previously published research [31, 33, 34, 35, 36].

The work on split-correctness, presented in Part I, is based on joint work with Benny Kimelfeld, Yoav Nahshon, Frank Neven, Matthias Niewerth, and Wim Martens. A preliminary version of this work was presented at the 38th Symposium on Principles of Database Systems (PODS 2019) [33]. An extended version [34] of this work is currently under review. The author of this thesis is the main author of this work. Notable contributions from other authors are Sections 3.2, and 4.3, which is joint work with Matthias Niewerth.

The work on weight annotators, presented in Chapter 5, is based on joint work with Benny Kimelfeld, Wim Martens, and Liat Peterfreund. A preliminary version of this work was presented at the 23rd International Conference on Database Theory (ICDT 2020) [35]. An extended version [36] of this work is currently under review. The author of this thesis is the main author of this work.

The work on aggregates is based on joint work with Benny Kimelfeld, Wim Martens, and Noa Bratman. A preliminary version of this work was presented at the 24th International Conference on Database Theory (ICDT 2021) [31]. The revised version, as presented in Chapter 6, is by the author of this thesis.

The dutch summary was translated by Erik Bollen.

1.6 Structure of this Thesis

This thesis is organized as follows. In Chapter 2, we give the preliminaries on (regular) document spanners. Part I consists of two chapters. We define and study the framework of split-correctness in Chapter 3 and study its computational complexity for regular document spanners in Chapter 4. In Part II we study weight annotators (Chapter 5) and aggregation functions for document spanners (Chapter 6). We conclude and point out open problems and possible directions for future work in Chapter 7. We note that Parts I and II are of independent interest and therefore can be studied in arbitrary order.

Chapter 2

Preliminaries

The cardinality of a set A is denoted by $|A|$. A *multiset* over A is a function $M : A \to \mathbb{N}$. We call $M(a)$ the *multiplicity* of a in M and say that $a \in M$ if $M(a) > 0$. The *size* of M denoted $|M|$, is the sum of the multiplicities of all elements in A, that is, $\sum_{a \in A} M(a)$. Note that, the size may be infinite. We denote multisets in brackets $\{\!|\cdot|\!\}$ in the usual way. E.g., in the multiset $M = \{\!|1, 1, 3|\!\}$ we have that $M(1) = 2$, $M(3) = 1$, and $|M| = 3$. Furthermore, given a set X, we denote by 2^X the power set of X.

We assume countably infinite and disjoint sets D and Vars, containing *datavalues* (or simply *values*) and *variables*, respectively. We sometimes also call Vars the set of *span variables*. Let $V \subseteq \text{Vars}$ be a finite set of variables. A *V-tuple* is a total function $\mathrm{t} : V \to \mathrm{D}$ that assigns values to variables in V. We denote the set of all V-tuples by V-Tup. Let t be a V-tuple. We also denote V by Vars(t). The *arity* of t is the cardinality $|V|$ of V. For a subset $X \subseteq \text{Vars}$, we denote the restriction of t to the variables in X by $\pi_X(\mathrm{t})$. We sometimes leave V implicit when the precise set is not important. We say that a tuple t is *empty*, denoted by $\mathrm{t} = ()$, if $\text{Vars}(\mathrm{t}) = \emptyset$.

2.1 Document Spanners

This thesis is within the formalism of *document spanners* by Fagin et al. [45, 46]. We first revisit some definitions from this framework. Let Σ be a finite set, disjoint from D and Vars, of symbols called the *alphabet*. A document (over Σ) is a sequence $d = \sigma_1 \cdots \sigma_n$ of symbols where every symbol is from the alphabet, that is, $\sigma_i \in \Sigma$. If $n = 0$ we denote d by ε and call d *empty*. By Σ^* we denote the set of all documents over Σ. A *(k-ary) string relation R*, for some $k \in \mathbb{N}$, is a subset $R \subseteq (\Sigma^*)^k$ of the k-fold Cartesian product of Σ^*. We denote by $|d|$ the length n of a document $d \in \Sigma^*$.

A *span* of d is an expression of the form $[i, j\rangle$ with $1 \le i \le j \le n + 1$. For a span $[i, j\rangle$ of d, we denote by $d_{[i,j\rangle}$ the string $\sigma_i \cdots \sigma_{j-1}$. A span $[i, j\rangle$ is empty if $i = j$ which implies that $d_{[i,j\rangle} = \varepsilon$. For a document d, we denote by $\mathsf{Spans}(d)$ the set of all possible spans of d and by Spans the set of all possible spans of all possible documents. The framework focuses on functions that extract spans from documents and assigns them to variables. Since we will be working with relations over spans, also called *span relations*, we assume that D is such that $\mathsf{Spans} \subseteq \mathrm{D}$. A *$d$-tuple* t is a V-tuple which only assigns values from $\mathsf{Spans}(d)$, that is, $\mathrm{t}(x) \subseteq \mathsf{Spans}(d)$ for every $x \in \text{Vars}(\mathrm{t})$. If the document d

```
T h e r e ⊔ a r e ⊔ 7 ⊔ e v e n t s ⊔ i n ⊔ B e l g i u m , ⊔ 1 0 - 1 5 ⊔ i n ⊔
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 30 31 32 33 34 35 36 37 38 39 40

F r a n c e , ⊔ 4 ⊔ i n ⊔ L u x e m b o u r g , ⊔ t h r e e ⊔ i n ⊔ B e r l i n .
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
```

$d_{x_{\mathsf{loc}}}$	$d_{x_{\mathsf{events}}}$
Belgium	7
France	10-15
Luxembourg	4
Berlin	three

x_{loc}	x_{events}
$[23, 30\rangle$	$[11, 12\rangle$
$[41, 47\rangle$	$[32, 37\rangle$
$[54, 64\rangle$	$[49, 50\rangle$
$[75, 81\rangle$	$[66, 71\rangle$

Figure 2.1: A document d (top), a string relation (bottom left), and the corresponding span relation R (bottom right).

is clear from the context, we sometimes say simply *tuple* instead of d-tuple. Overloading notation, we denote by d_{t} the tuple $(d_{\mathrm{t}(x_1)}, \ldots, d_{\mathrm{t}(x_n)})$, where $\mathrm{Vars}(\mathrm{t}) = \{x_1, \ldots, x_n\}$.

Example 2.1.1. Consider the document in Figure 2.1. The string relation at the bottom left depicts a possible extraction of locations with the corresponding number of events. The table on the bottom right depicts the corresponding span relation. $\square$

Two spans $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ are *equal* if $i_1 = i_2$ and $j_1 = j_2$. In particular, we observe that two spans do not have to be equal if they select the same string. That is, $d_{[i_1,j_1\rangle} = d_{[i_2,j_2\rangle}$ does not imply that $[i_1, j_1\rangle = [i_2, j_2\rangle$. Two spans $[i, j\rangle$ and $[i', j'\rangle$ *overlap* if $i \leq i' < j$ or $i' \leq i < j'$, and are *disjoint* otherwise. Finally, $[i, j\rangle$ *covers* $[i', j'\rangle$ if $i \leq i' \leq j' \leq j$. Given a span $[i, j\rangle$ and a natural number n, we denote by $[i, j\rangle \gg n$ the span $[i + n, j + n\rangle$. Analogously, if $i > n$, we denote by $[i, j\rangle \ll n$ the span $[i - n, j - n\rangle$.

If s is a span of d and t is a d-tuple, we say that s covers t if s covers $\mathrm{t}(x)$ for every variable $x \in \mathrm{Vars}(\mathrm{t})$. Furthermore, let t be a non empty d-tuple for some document $d \in \Sigma^*$. We define the *minimal span that covers* t as the span $[i, j\rangle$, where

$$i \;:=\; \min\left\{i' \mid [i', j'\rangle = \mathrm{t}(v),\ \text{and } v \in \mathrm{Vars}(\mathrm{t})\right\} ,$$

and

$$j \;:=\; \max\left\{j' \mid [i', j'\rangle = \mathrm{t}(v),\ \text{and } v \in \mathrm{Vars}(\mathrm{t})\right\} .$$

If t is a d-tuple and n a natural number, we define the tuples $\mathrm{t} \gg n$ and $\mathrm{t} \ll n$ as the d-tuples that result from shifting each span in t by n. More formally, for all variables $x \in \mathrm{Vars}(\mathrm{t})$ we have:[1]

$$(\mathrm{t} \gg n)(x) \;:=\; \mathrm{t}(x) \gg n ,$$

and

$$(\mathrm{t} \ll n)(x) \;:=\; \mathrm{t}(x) \ll n .$$

[1] Notice that when n is too large, $\mathrm{t} \gg n$ or $\mathrm{t} \ll n$ could technically not be a d-tuple anymore. However, we only use the operator in situations where this does not happen.

A *document spanner* (also *spanner* for short) is a function S that maps every document d into a finite set $S(d)$ of d-tuples. By $\mathrm{Vars}(S) := \{v \in \mathrm{Vars}(\mathrm{t}) \mid d \in \Sigma^* \text{ and } \mathrm{t} \in S(d)\}$ we denote the variables of S. We note that, following Maturana et al. [105], we do not require that all tuples of a spanner S assign all variables in $\mathrm{Vars}(S)$, that is, given a document d and a tuple t, we require that $\mathrm{Vars}(\mathrm{t}) \subseteq \mathrm{Vars}(S)$. A spanner S is called *functional* if every tuple uses the same variables, i.e., $\mathrm{Vars}(\mathrm{t}) = \mathrm{Vars}(S)$, for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in S(d)$. By $S \subseteq S'$ we denote the fact that $S(d) \subseteq S'(d)$, for every document d. Furthermore, we denote by $S = S'$ the fact that the spanners S and S' define the same function.

In the following, we sometimes require that a spanner only selects tuples that use at least two different positions in d. More formally, a document spanner S is *proper* if for every document $d \in \Sigma^*$, the empty tuple is not selected by S, i.e., $() \notin S(d)$, and $\mathrm{t} \in S(d)$ implies that the minimal span that covers t is not empty.

2.1.1 Algebraic Operators on Document Spanners

We conclude this section by defining algebraic operations on spanners. Two d-tuples t_1 and t_2 are *compatible* if they agree on every common variable, i.e., $\mathrm{t}_1(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1) \cap \mathrm{Vars}(\mathrm{t}_2)$. In this case, define $\mathrm{t}_1 \cup \mathrm{t}_2$ as the tuple with $\mathrm{Vars}(\mathrm{t}_1 \cup \mathrm{t}_2) = \mathrm{Vars}(\mathrm{t}_1) \cup \mathrm{Vars}(\mathrm{t}_2)$ such that $(\mathrm{t}_1 \cup \mathrm{t}_2)(x) = \mathrm{t}_1(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1)$ and $(\mathrm{t}_1 \cup \mathrm{t}_2)(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_2)$.

Definition 2.1.2 (Algebraic Operations on Spanners)**.** Let S_1, S_2 be (document) spanners and let $d \in \Sigma^*$ be a document.

- *Variable enclosing.* The spanner $S = x\{S_1\}$ is defined, for all S_1 with $x \notin \mathrm{Vars}(S_1)$, by
$$S(d) \ := \ \big\{\mathrm{t} \cup \{x \mapsto [1, |d|+1\rangle\} \mid \mathrm{t} \in S_1(d)\big\} \ .$$
- *Concatenation.* The spanner $S = S_1 \cdot S_2$ is defined, for all S_1, S_2 with $\mathrm{Vars}(S_1) \cap \mathrm{Vars}(S_2) = \emptyset$, by
$$S(d) \ := \ \big\{\mathrm{t}_1 \cup \mathrm{t}_2 \mid d = d_1 \cdot d_2,\ \mathrm{t}_1 \in S_1(d_1),\ \text{and } \mathrm{t}_2 \ll |d_1| \in S_2(d_2)\big\} \ .$$
- *Union.* The union $S = S_1 \cup S_2$ is defined by $S(d) := S_1(d) \cup S_2(d)$.
- *Projection.* The projection $S = \pi_Y(S_1)$ is defined by $S(d) := \{\pi_Y(\mathrm{t}) \mid \mathrm{t} \in S_1(d)\}$. Recall that $\pi_Y(\mathrm{t})$ denotes the restriction of t to the variables in $\mathrm{Vars}(\mathrm{t}) \cap Y$.
- *Natural Join.* The (natural) join $S = S_1 \bowtie S_2$ is defined such that $S(d)$ consists of all tuples $\mathrm{t}_1 \cup \mathrm{t}_2$ such that $\mathrm{t}_1 \in S_1(d)$, $\mathrm{t}_2 \in S_2(d)$, and t_1 and t_2 are compatible, i.e., $\mathrm{t}_1(x) = \mathrm{t}_2(x)$ for all $x \in \mathrm{Vars}(\mathrm{t}_1) \cap \mathrm{Vars}(\mathrm{t}_2)$.

2.2 Representations of Regular Document Spanners

In this section, we recall the terminology and definition of regular languages and *regular* spanners [45]. We assume that the reader is familiar with (non)deterministic finite state automata (abbrev. NFA and DFA). By $\mathcal{L}(A)$ we denote the language accepted by a (non)deterministic finite state automaton A.

We use two main models for representing spanners: *regex-formulas* and *vset-automata*. Furthermore, following Freydenberger [52], we introduce so-called *ref-words*, which connect spanner representations with regular languages. We also introduce various classes of vset-automata, namely deterministic and unambiguous vset-automata, that have properties essential to the tractability of some problems we study. Figure 2.4 provides an overview of all representations of regular document spanners we use throughout this thesis. We also study a variation of (regular) document spanners, called $\mathbb{K}$-Annotators, which annotate tuples with an element from a commutative semiring. We refer to Chapter 5 for the formal definition.

2.2.1 Regex Formulas

A *regex-formula (over* Σ*)* is a regular expression that may include variables (called *capture variables*). Formally, we define the syntax with the recursive rule

$$\alpha \ := \ \emptyset \mid \varepsilon \mid \sigma \mid (\alpha \vee \alpha) \mid (\alpha \cdot \alpha) \mid \alpha^* \mid x\{\alpha\}\,,$$

where $\sigma \in \Sigma$ and $x \in \text{Vars}$. We use α^+ as a shorthand for $\alpha \cdot \alpha^*$ and Σ as a shorthand for $\bigvee_{\sigma\in\Sigma} \sigma$. The set of variables that occur in α is denoted by $\text{Vars}(\alpha)$ and the size $|\alpha|$ is defined as the number of symbols in α. The spanner $[\![\alpha]\!]$ represented by a regex formula α is given by the following inductive definition that uses the algebraic operations from Definition 2.1.2:

$$\begin{array}{llll} [\![\emptyset]\!] := \emptyset & [\![\varepsilon]\!] := \big\{\varepsilon \mapsto \{()\}\big\} & [\![(\alpha_1 \vee \alpha_2)]\!] := [\![\alpha_1]\!] \cup [\![\alpha_2]\!] & [\![\alpha^*]\!] := \cup_{i\geq 0}[\![\alpha^i]\!] \\ [\![x\{\alpha\}]\!] := x\big\{[\![\alpha]\!]\big\} & [\![\sigma]\!] := \big\{\sigma \mapsto \{()\}\big\} & [\![(\alpha_1 \cdot \alpha_2)]\!] := [\![\alpha_1]\!] \cdot [\![\alpha_2]\!] & \end{array}$$

We say that a regex formula α is *sequential* if

- no variable occurs under the Kleene star,
- for every subformula of the form $x\{\alpha_1\}$ it holds that x does not occur in α_1, and
- for every subformula of the form $\alpha_1 \cdot \alpha_2$ it holds that the sets of variables used in α_1 and α_2 are disjoint.

A regex formula α is *functional* if α is sequential and the spanner $[\![\alpha]\!]$ is functional.

The set of all regex formulas is denoted by RGX. Similarly, the sequential (or functional) regex formulas are denoted by sRGX (fRGX, respectively). It follows immediately from the definitions that every functional regex-formula is also sequential, but not vice versa. For instance, the regex-formula $\alpha = x_1\{a\} \vee x_2\{b\}$ is sequential, but not functional (and therefore, fRGX $\subsetneq$ sRGX).

Maturana et al. [105] showed that the class of spanners defined by regex-formulas is the same as the class of spanners defined by sequential regex-formulas. However, using the same technique as Freydenberger [52, Proposition 3.9], it can be shown that the smallest sequential regex-formula equivalent to a given regex formula α can be exponentially larger than α.

2.2.2 Ref-Words

For a finite set $V \subseteq \mathrm{Vars}$ of variables, ref-words are defined over the extended alphabet $\Sigma \cup \Gamma_V$, where $\Gamma_V := \{x\vdash, \dashv x \mid x \in V\}$. We assume that Γ_V is disjoint from Σ and Vars. Ref-words extend strings over Σ by encoding opening ($x\vdash$) and closing ($\dashv x$) of variables.

A ref-word $\mathrm{r} \in (\Sigma \cup \Gamma_V)^*$ is *valid* if every occurring variable is opened and closed exactly once. More formally, for each $x \in V$, the string r has precisely one occurrence of $x\vdash$ and precisely one occurrence of $\dashv x$, which is after the occurrence of $x\vdash$. For every valid ref-word r over $(\Sigma \cup \Gamma_V)$ we define Vars(r) as the set of variables $x \in V$ which occur in the ref-word. More formally,

$$\mathrm{Vars}(\mathrm{r}) \;:=\; \{x \in V \mid \exists \mathrm{r}_x^{\mathrm{pre}}, \mathrm{r}_x, \mathrm{r}_x^{\mathrm{post}} \in (\Sigma \cup \Gamma_V)^* \text{ such that } \mathrm{r} = \mathrm{r}_x^{\mathrm{pre}} \cdot x\vdash \cdot \mathrm{r}_x \cdot \dashv x \cdot \mathrm{r}_x^{\mathrm{post}}\}.$$

Intuitively, each valid ref-word r encodes a d-tuple for some document d, where the document is given by symbols from σ in r and the variable markers encode where the spans begin and end. Formally, we define functions doc and tup that, given a valid ref-word, output the corresponding document and tuple.[2] The morphism $\mathrm{doc}\colon (\Sigma \cup \Gamma_V)^* \to \Sigma^*$ is defined as:

$$\mathrm{doc}(\sigma) \;:=\; \begin{cases} \sigma & \text{if } \sigma \in \Sigma \\ \varepsilon & \text{if } \sigma \in \Gamma_V \end{cases}$$

By definition, every valid ref-word r over $(\Sigma \cup \Gamma_V)$ has an unique factorization

$$\mathrm{r} \;=\; \mathrm{r}_x^{\mathrm{pre}} \cdot x\vdash \cdot \mathrm{r}_x \cdot \dashv x \cdot \mathrm{r}_x^{\mathrm{post}}$$

for each $x \in \mathrm{Vars}(\mathrm{r})$. We are now ready to define the function tup as

$$\mathrm{tup}(\mathrm{r}) \;:=\; \{x \mapsto [i_x, j_x\rangle \mid x \in \mathrm{Vars}(\mathrm{r}), i_x = |\,\mathrm{doc}(\mathrm{r}_x^{\mathrm{pre}})|, j_x = i_x + |\,\mathrm{doc}(\mathrm{r}_x)|\}\;.$$

The usage of the doc morphism ensures that the indices i_x and j_x refer to positions in the document and do not consider other variable operations.

A *ref-word language* $\mathcal{R}$ is a language of ref-words. The spanner $[\![\mathcal{R}]\!]$ represented by a ref-word language $\mathcal{R}$ is given by

$$[\![\mathcal{R}]\!](d) \;:=\; \big\{\,\mathrm{tup}(\mathrm{r}) \mid \mathrm{r} \in \mathcal{R}, \mathrm{r} \text{ is valid, and } \mathrm{doc}(\mathrm{r}) = d\big\}\;.$$

A ref-word language $\mathcal{R}$ is *sequential* if every ref-word $\mathrm{r} \in \mathcal{R}$ is valid. It is *functional* if it is sequential and $[\![\mathcal{R}]\!]$ is functional.

2.2.3 Variable Order Condition

Let $\mathrm{r} = x_1\vdash x_2\vdash a \dashv x_1 \dashv x_2$ and $\mathrm{r}' = x_1\vdash x_2\vdash a \dashv x_2 \dashv x_1$ be ref-words. We observe that both ref-words encode the tuple which selects the span $[1,2\rangle$ in both variables x_1, x_2 on document a. Thus, the same spanner can be represented by multiple ref-word languages. We now introduce the *variable order condition*, in order to achieve a one-to-one mapping

[2]The function doc is sometimes also called clr in literature (cf. Freydenberger et al. [54]).

between ref-words (resp., ref-word languages) and tuples (resp., spanners). To this end, we fix a total, linear order $\prec$ on the set Γ_{Vars} of variable operations, such that $v\vdash \prec \dashv v$ for every variable $v \in \text{Vars}$. We say that a ref-word r satisfies the *variable order condition* if all adjacent variable operations in r are ordered according to the fixed linear order $\prec$. That is, the ref-word $\mathrm{r} = \sigma_1 \cdots \sigma_n$ satisfies the variable order condition if $\sigma_i \prec \sigma_{i+1}$ for every $1 \leq i < n$ with $\sigma_i, \sigma_{i+1} \in \Gamma_{\text{Vars}}$. We observe that, for every document d and every tuple t, there is exactly one ref-word r, with $d = \mathrm{doc}(\mathrm{r})$ and $\mathrm{t} = \mathrm{tup}(\mathrm{r})$, that satisfies the variable order condition. We define ref as the function that, given a document d and a d-tuple t, returns this unique ref-word that satisfies the variable order condition.

The following observation shows the connections between the functions doc, ref, and tup.

Observation 2.2.1. *Let* r *be a valid ref-word and let* $\mathrm{r}' := \mathrm{ref}(\mathrm{doc}(\mathrm{r}), \mathrm{tup}(\mathrm{r}))$*. Then* $\mathrm{tup}(\mathrm{r}) = \mathrm{tup}(\mathrm{r}')$*. Furthermore,* $\mathrm{r} = \mathrm{r}'$ *if and only if* r *satisfies the variable order condition.* □

Analogous to sequentiality, we say that a ref-word language $\mathcal{R}$ *satisfies the variable order condition* if every ref-word $\mathrm{r} \in \mathcal{R}$ satisfies the variable order condition. The following lemma connects spanners and sequential ref-word languages which satisfy the variable order condition.

Lemma 2.2.2. *Let* $\mathcal{R}_1, \mathcal{R}_2$ *be sequential ref-word languages which satisfies the variable order condition. Then* $\mathcal{R}_1 \subseteq \mathcal{R}_2$ *if and only if* $[\![\mathcal{R}_1]\!] \subseteq [\![\mathcal{R}_2]\!]$.

Proof. (If): Let $\mathrm{r} \in \mathcal{R}_1$. Thus, $\mathrm{tup}(\mathrm{r}) \in [\![\mathcal{R}_1]\!](\mathrm{doc}(\mathrm{r})) \subseteq [\![\mathcal{R}_2]\!](\mathrm{doc}(\mathrm{r}))$ and therefore, due to $\mathcal{R}_2$ satisfying the variable order condition, $\mathrm{r} \in \mathcal{R}_2$.

(Only if): Let $d \in \Sigma^*$ be a document and $\mathrm{t} \in [\![\mathcal{R}_1]\!](d)$. Thus, there must be a valid ref-word $\mathrm{r} \in \mathcal{R}_1$ with $\mathrm{doc}(\mathrm{r}) = d$ and $\mathrm{tup}(\mathrm{r}) = \mathrm{t}$. Due to $\mathcal{R}_1$ satisfying the variable order condition, r must satisfy the variable order condition and therefore, $\mathrm{ref}(d, \mathrm{t}) = \mathrm{r} \in \mathcal{R}_1 \subseteq \mathcal{R}_2$ and thus $\mathrm{t} \in [\![\mathcal{R}_2]\!](d)$, concluding the proof. □

Connection between ref-words and regex formulas

Every regex-formula can be interpreted as a generator of a (regular) ref-word language $\mathcal{R}(\alpha)$ over the extended alphabet $\Sigma \cup \Gamma_{\text{Vars}(\alpha)}$ using the usual semantics for regular expressions and interpreting every subformula of the form $x\{\beta\}$ as $x\vdash \cdot \beta \cdot \dashv x$.

A straightforward induction shows that $[\![\alpha]\!] = [\![\mathcal{R}(\alpha)]\!]$ for every regex-formula α. Furthermore α is sequential (functional) if and only if $\mathcal{R}(\alpha)$ is sequential (functional).

2.2.4 Variable Set-Automata

A *variable-set automaton (vset-automaton)* with variables from a finite set $V \subseteq \text{Vars}$ can be understood as an ε-NFA that is extended with edges that are labeled with variable operations Γ_V. Formally, a vset-automaton is a sextuple $A := (\Sigma, V, Q, q_0, Q_F, \delta)$, where Σ is a finite set of alphabet symbols, V is a finite set of variables, Q is a finite set of states, $q_0 \in Q$ is an initial state, $Q_F \subseteq Q$ is a set of final states, and $\delta\colon Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma_V) \to 2^Q$

is the transition function. The *size* of a vset-automaton A is defined by $|A| = |Q| + |Q_F| + |\delta| + 1$. By $\text{Vars}(A) := V$ we denote the variables of A. To define the semantics of A, we first interpret A as an ε-NFA over the terminal alphabet $\Sigma \cup \Gamma_V$, and define its *ref-word language* $\mathcal{R}(A)$ as the set of all ref-words $\mathrm{r} \in \mathcal{L}(A) \subseteq (\Sigma \cup \Gamma_V)^*$ that are accepted by the ε-NFA A.

Analogous to runs of ε-NFAs, we define a *run* ρ of A on a ref-word $\mathrm{r} = \sigma_1 \cdots \sigma_n$ as the sequence

$$\rho \ := \ q_0 \xrightarrow{\sigma_1} q_1 \ \cdots \ q_{n-1} \xrightarrow{\sigma_n} q_n \ ,$$

where $q_{i+1} \in \delta(q_i, \sigma_{i+1})$ for all $0 \leq i < n$, and $q_n \in Q_F$. We observe that all runs are accepting and that $\mathrm{r} \in \mathcal{R}(A)$ if and only if there is a run ρ of A on r. Furthermore, a run ρ of A on r accepts a d-tuple t if $\text{doc}(\mathrm{r}) = d$ and $\mathrm{t} = \text{tup}(\mathrm{r})$.

We define $[\![A]\!]$ as $[\![\mathcal{R}(A)]\!]$ and say that A is *sequential* if $\mathcal{R}(A)$ is sequential. Furthermore, we say that A is *functional* if $\mathcal{R}(A)$ is functional and $\text{Vars}([\![\mathcal{R}(A)]\!]) = V$. Two vset-automata A_1, A_2 are *equivalent* if they define the same spanner, i.e., if $[\![A_1]\!] = [\![A_2]\!]$. Furthermore, a vset-automaton A satisfies the *variable order condition*if $\mathcal{R}(A)$ satisfies the variable order condition.

We refer to the set of all vset-automata as VSA and to the set of all sequential (or functional) vset-automata as sVSA (or fVSA, respectively).

We observe that, given a vset-automaton A, ε-transitions can be removed in PTIME, using the classical ε-removal algorithm for ε-NFAs.

Observation 2.2.3. *Given a vset-automaton A an equivalent vset-automaton A' which does not use ε-transitions can be constructed in polynomial time.* ☐

Deterministic and Unambiguous vset-Automata

We use the notion of determinism as introduced by Maturana et al. [105], but refer to it as *weakly* deterministic because, as we will show in Theorem 4.1.4, weakly deterministic vset-automata still have sufficient nondeterminism to make the containment problem PSPACE-hard, which is as hard as for general vset-automata. We therefore define a stronger notion of determinism, which will lead to an NL-complete containment problem (Theorem 4.1.5). Furthermore, we define unambiguous vset-automata, which utilize a relaxed notion of determinism that preserves tractability of containment (Theorem 4.1.5).

Formally, a vset-automaton $A = (\Sigma, V, Q, q_0, Q_F, \delta)$ is *weakly deterministic*, if

1. $\delta(q, \varepsilon) = \emptyset$ for every $q \in Q$, i.e., it does not use ε-transitions, and

2. $|\delta(q, v)| \leq 1$ for every $q \in Q$ and every $v \in \Sigma \cup \Gamma_V$.

Finally we define deterministic and unambiguous vset-automata. To this end, we define the following three conditions:

(C1) A is weakly deterministic;

(C2) A satisfies the variable order condition;

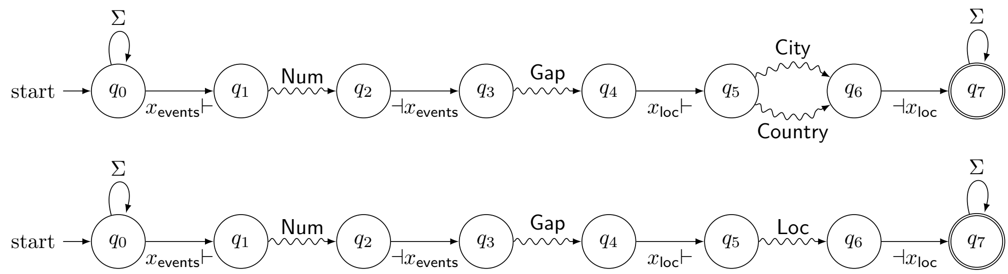

Figure 2.2: Two example vset-automata that extract the span relation R on input d as defined in Figure 2.1. For the sake of presentation, the automata are simplified as follows: Num is a sub-automaton matching anything representing a number (of events) or range, Gap is a sub-automaton matching sequences of at most three words, City and Country are sub-automata matching city and country names respectively. Loc is a sub-automaton for the union of City and Country. All these sub-automata are assumed to be deterministic.

(C3) there is exactly one run of A on every ref-word $\mathrm{r} \in \mathcal{R}(A)$.

We say that a vset-automaton A is *deterministic* if it satisfies conditions (C1) and (C2) and it is *unambiguous* if it satisfies conditions (C2) and (C3). The following observation is obvious, as (C1) clearly implies (C3).

Observation 2.2.4. *Every deterministic vset-automaton is also unambiguous.* □

We note that for Boolean spanners the definitions coincide with the classical unambiguity and determinism definitions of finite state automata. That is, a vset-automaton with $\mathrm{Vars}(A) = \emptyset$ is deterministic (unambiguous) if it is a deterministic (unambiguous) finite state automaton.

Example 2.2.5. The span relation on the bottom right of Figure 2.1 can be extracted from d by a spanner that matches textual representations of numbers (or ranges) in the variable x_{events}, followed by a city or country name, matched in x_{loc}. Figure 2.2 shows how two such vset-automata may look like. Note that some strings, like Luxembourg are the name of a city as well as a country. Thus, the upper automaton is ambiguous, because the tuple with Luxembourg is captured twice (thus, violating (C3)). The lower automaton is unambiguous, because the sub-automaton for Loc only matches such names once. □

We show in Proposition 2.2.6 that none of the conditions (C1), (C2), and (C3) restrict the expressiveness of regular spanners. We discuss complexity of deterministic vset-automata in Section 4.1.2. In the following, we denote by dVSA (resp., dfVSA and dsVSA) the class of deterministic (resp., deterministic and functional, deterministic and sequential) vset-automata and by uVSA (resp., ufVSA and usVSA) the class of unambiguous (resp., unambiguous and functional, unambiguous and sequential) vset-automata.

Deterministic vset-automata are similar to the *extended deterministic vset-automata* by Florenzano et al. [48], which allow multiple variable operations on a single transition and force each variable transition to be followed by a transition processing an alphabet symbol. However, deterministic vset-automata can be exponentially more succinct than extended deterministic vset-automata. An example class of automata where this blowup occurs is depicted in Figure 2.3.

As we will show next, deterministic vset-automata are equally expressive as vset-automata in general.

Proposition 2.2.6. *For every vset-automaton A there is an equivalent sequential deterministic vset-automaton A', i.e., $[\![A]\!] = [\![A']\!]$.*

Proof. We have to show that we can find a vset-automaton A', such that A' is equivalent to A and A' satisfies (C1) and (C2).

Maturana et al. [105, Proposition 5.6] show that, for every vset-automaton there is an equivalent sequential vset-automaton. Therefore, we can assume, w.l.o.g., that A is sequential. Florenzano et al. [48, Theorem 3.1, Proposition 3.2] show that every vset-automaton can be transformed into an equivalent extended vset-automaton and vice versa. The model of extended vset-automata allows to annotate a set of variable operations to a single edge. For the construction of a vset-automaton from a given extended vset-automaton they fix an order on the variables and replace each transition, containing multiple variable operations by a sequence of edges. Therefore, the variable order condition (C2) can be achieved by using $\prec$ as variable order for the transformation from extended to normal vset-automata.[3] We note that all involved constructions preserve sequentiality.

We can achieve (C1) by interpreting the vset-automaton as an ε-NFA that accepts ref-words and using the classical ε-NFA determinization construction. This construction also preserves sequentiality as it does not change the involved ref-word language. $\square$

Throughout this thesis, we will often assume that regular document spanners are given as sequential (or functional) vset-automata. The main reason is that, as we show next, problems like answering whether a vset-automaton produces a non-empty output on a given document become intractable if the vset-automaton is not sequential. We note that the following proposition is heavily based on Freydenberger [52, Lemma 3.1] who showed that given a vset-automaton A it is NP-hard to decide whether $[\![A]\!](\varepsilon) \neq \emptyset$. Based on the reduction by Freydenberger, we show that the problem remains NP-hard if the vset-automaton is deterministic.[4]

[3] Fagin et al. [45] already gave a similar construction on so called *lexicographic vset-automata*, i.e., vset-automata in which consecutive variable operations always follow a given linear order.

[4] Note that we require an non-empty input document d of linear size to cope with the determinism of A. The automaton constructed by Freydenberger is nondeterministic and therefore also does not require the input document to be of linear size.

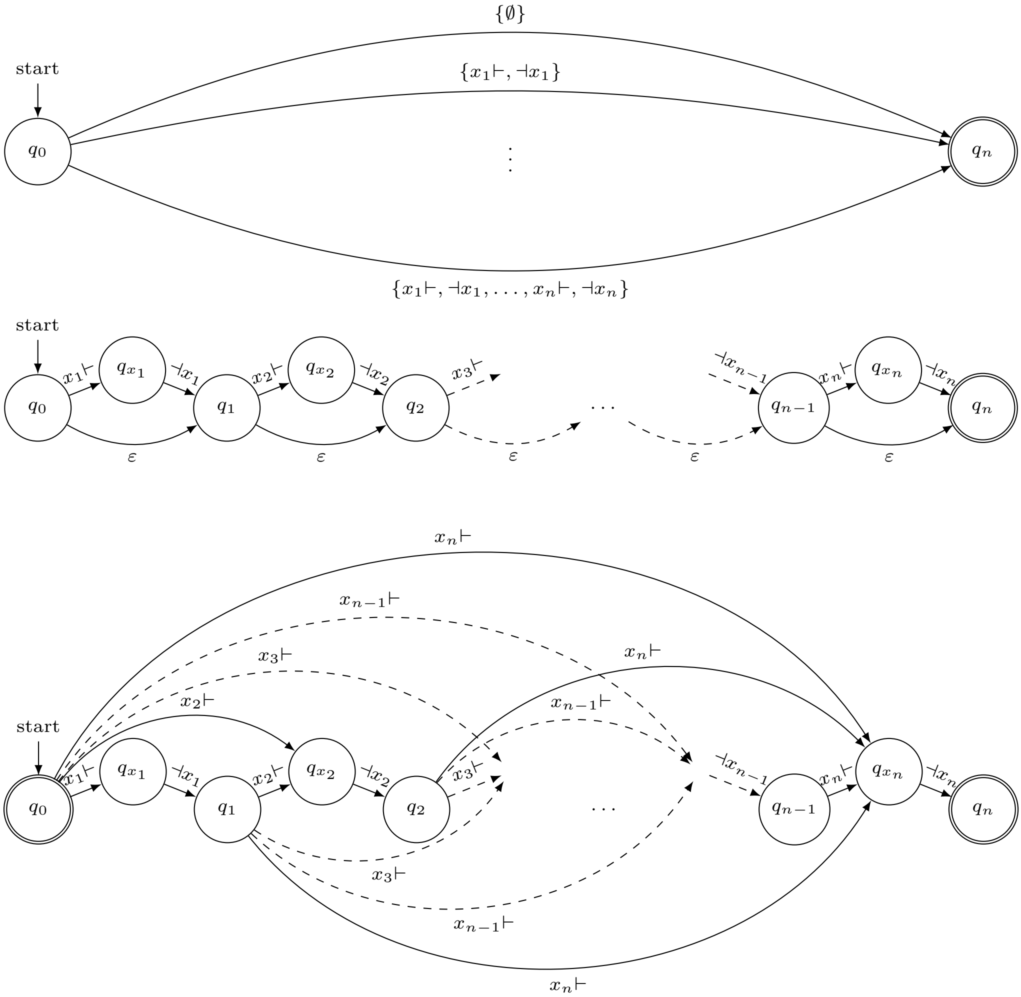

Figure 2.3: Class of example spanners where the smallest deterministic extended VSAs (top) are exponentially larger than the smallest deterministic vset-automata (bottom). The automaton on the top has a transition $\delta(q_0, \Gamma_V) = \{q_n\}$ for every $V \subseteq \{x_1, \ldots, x_n\}$, thus, it has 2^n transitions. Note that the automaton in the middle is not deterministic, as it contains ε-transitions. However, these ε-transitions can be removed via the classical ε removal algorithm for finite automata, resulting in the bottom automaton, which has $\frac{n(n+1)}{2} + n$ transitions in total.

Proposition 2.2.7. *Given a document d and a vset-automaton A, testing if $[\![A]\!](d) \neq \emptyset$ is* NP*-complete, even if A is deterministic.*

Proof. The upper bound is straightforward by guessing a d-tuple t and checking whether $\mathrm{t} \in [\![A]\!](d)$. We will now give a reduction from the Hamiltonian path problem. The reduction is heavily based on Freydenberger [52, Lemma 3.1] who shows that, given a vset-automaton A, it is NP-hard to decide whether $[\![A]\!](\varepsilon) \neq \emptyset$. We show that deciding whether $[\![A]\!](d) \neq \emptyset$ is NP-hard even if A is deterministic. Given a directed graph $G = (V, E)$, the Hamiltonian path problem asks whether there is a sequence $(i_1, \ldots, i_n)$ with $n = |V|$ and $(i_j, i_{j+1}) \in E$ for all $1 \leq j < n$ such that for every $v \in V$ there is exactly one $1 \leq j \leq n$ with $i_j = v$.

Given a directed graph G, we will construct $A \in$ dVSA over the alphabet $\Sigma = \{a\}$, such that each tuple $\mathrm{t} \in [\![A]\!](a^n)$ corresponds to a Hamiltonian path in G. We assume, w.l.o.g., that $V = \{1, \ldots, n\}$ for some $n \geq 1$ and $\dashv x_i \prec \dashv x_j$ if $i < j$. Then let $A := (\Sigma, V, Q, q_0, Q_F, \delta)$ with $\Sigma = \{a\}$, where the set of variables is exactly the set V of nodes of G, $Q = \{q_0\} \cup \{q_i, q_i^o, q_i^c | 1 \leq i \leq n\}$, $Q_F = \{q_n^c\}$, and δ is defined as follows:

$$
\begin{aligned}
\delta(q_0, x_i\vdash) &:= q_i^o \text{ for all } 1 \leq i \leq n,\\
\delta(q_i^o, a) &:= q_i \text{ for all } 1 \leq i \leq n,\\
\delta(q_i, x_j\vdash) &:= q_j^o \text{ for all } (i,j) \in E,\\
\delta(q_i, \dashv x_1) &:= q_1^c \text{ for all } 1 \leq i \leq n,\\
\delta(q_i^c, \dashv x_{i+1}) &:= q_{i+1}^c \text{ for all } 1 \leq i < n.
\end{aligned}
$$

Observe that S always reads an alphabet symbol a after opening a variable and closes all variables in a fixed order. Furthermore, we observe that $[\![A]\!](d) = \emptyset$ if $d \neq a^n$. We now show that A is deterministic. To this end, let $v, v' \in \Gamma_V$ be two variable operations such that there are states $q_1, q_2, q_3 \in Q$ with $q_2 \in \delta(q_1, v)$ and $q_3 \in \delta(q_2, v')$. Then, per construction of A, $v = \dashv x_i$ and $v' = \dashv x_{i+1}$ and $v \prec v'$, thus A obeys the variable order property. Furthermore, observe that per definition of δ, A is weakly deterministic and thus A is deterministic.

It remains to show that $[\![A]\!](a^n) \neq \emptyset$ if and only if G has a Hamiltonian path. We will show that there is a one-to-one correspondence between $\mathrm{t} \in [\![A]\!](a^n)$ and Hamiltonian paths in G. Let $\mathrm{t} \in [\![A]\!](a^n)$. Then, $\mathrm{ref}(d, \mathrm{t}) = x_{i_1}\vdash a\, x_{i_2}\vdash a \cdots x_{i_n}\vdash a \dashv x_1 \cdots \dashv x_n$. Furthermore, each variable x_{i_j} corresponds to a node $i_j \in V$ and for all $1 \leq j < n$, $(i_j, i_{j+1}) \in E$. As $\mathrm{ref}(d, \mathrm{t})$ is valid and each $\dashv x_i$ occurs in $\mathrm{ref}(d, \mathrm{t})$, each $x_i\vdash$ occurs exactly once. Thus, $(i_1, \ldots, i_n)$ is a Hamiltonian path in G. Vise versa, each Hamiltonian path in G corresponds to a valid ref-word r with $\mathrm{tup}(\mathrm{r}) \in [\![A]\!](a^n)$. $\square$

Finally, we recall that it is well known that the class of regex-formulas (RGX) is less expressive than the class of vset-automata (VSA) [45, 105]. In order to reach the expressiveness of vset-automata, RGX needs to be extended with projection, natural join, and union. Figure 2.4 gives an overview of the expressiveness and inclusions between the introduced classes of document spanners.

$$
\begin{array}{|c|c|ccccc|}
\hline
\mathrm{RGX} & & \mathrm{VSA} & \supseteq & \mathrm{uVSA} & \supseteq & \mathrm{dVSA} \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\mathrm{sRGX} & & \mathrm{sVSA} & \supseteq & \mathrm{usVSA} & \supseteq & \mathrm{dsVSA} \\
\hline
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\hline
\mathrm{fRGX} & & \mathrm{fVSA} & \supseteq & \mathrm{ufVSA} & \supseteq & \mathrm{dfVSA} \\
\hline
\end{array}
$$

Figure 2.4: Expressiveness and inclusion relations of classes of regular document spanners. All formalisms within the same box are equally expressive.

We denote the set of all representations depicted in Figure 2.4 by $\mathcal{S}_{\text{general}}$ and the unambiguous and sequential subset by $\mathcal{S}_{\text{tractable}}$, that is, $\mathcal{S}_{\text{tractable}} = \{\mathrm{usVSA}, \mathrm{ufVSA}, \mathrm{dsVSA}, \mathrm{dfVSA}\}$.

2.2.5 Computational Model

Throughout this thesis, we use the RAM model with uniform cost measure and logarithmic word size [2] for our complexity results. That is, we assume that addition and multiplication of numbers, represented by a logarithmic number of bits, take constant time.

Part I

Parallel Evaluation of Document Spanners

Chapter 3

Split-Correctness

We begin this chapter by defining the basic concepts of our framework. A *splitter* is a spanner P that outputs a *set of intervals* (e.g., sentences, paragraphs, N-grams, HTTP requests, etc.). A spanner S is *self-splittable* by a splitter P if for all documents d, evaluating S on d gives the same result as the union of the evaluations of S on each of the chunks produced by P. We also consider the more general case where we allow the spanner on the chunks produced by P to be some spanner S_P different from S. In this case, we say that *S is splittable by P via S_P*. If, for given S and P, such a spanner S_P exists, then we say that S is *splittable* by P. With these definitions, we formally define several computational problems, each parameterized by a class $\mathcal{S}$ of spanners. In the *Split-Correctness* problem, we are given S, P, and S_P, and the goal is to determine whether S is splittable by P via S_P. In the *Splittability* (resp., *Self-Splittability*) problem, we are given S and P and the goal is to determine whether S is splittable (resp., *self-splittable*) by P. We also consider other settings, which we will discuss in the later sections. In our analysis, we consider the classes of regex formulas and vset-automata, as well as vset-automata in known normal forms, namely *sequential, functional, unambiguous,* and *deterministic.*

We show several complexity results for the studied classes of spanners. For one, the problems Split-Correctness and Self-Splittability are PSPACE-complete for regex formulas and vset-automata. Furthermore, we also characterize a sufficient condition for the tractability of Split-Correctness and Self-Splittability for sequential and unambiguous vset-automata. This condition, which we will call the *highlander condition,*[1] also reduces to PSPACE-completeness the complexity of Splittability, which is solvable in EXPSPACE in general. One key property of splitters that, most of the time, is sufficient (but not necessary) for the highlander condition is the *disjointness* of the splitter. Disjointness is a natural property—it requires the splitter P to be such that for all input documents, the spans produced by P are pairwise disjoint (non-overlapping), such as in the case of tokenization, sentence boundary detection, paragraph splitting, and paragraph segmentation. Examples of *non-disjoint* splitters include N-grams and pairs of consecutive sentences.

Following our analysis of Split-Correctness and Splittability for regular spanners, we turn to discussing additional problems that arise in our framework. In Section 4.5,

[1] This is in acclimation to the tagline "There can be only one" of the Highlander movie. It will become clear why we choose this name later on.

we study the problem of *Split-Existence*: given a spanner S, is there a nontrivial splitter P such that S is splittable by P? Even though we do not solve this problem, we connect it to the problem of *language primality* [136, Problem 2.1], a classic problem in Formal Language Theory that is still not completely understood. More precisely, we prove that a special case of Split-Existence is equivalent to a variant of the language primality problem for which the complexity is still open. In Section 3.3, we study the splitter framework in the context of the relational algebra. We establish results on the associativity of composition, the transitivity of self-splittability, and the distributivity of composition and join.

In addition, we discuss problems that arise in natural extensions of the basic framework. One of these problems captures the case where some of the spanners in the query are treated as *black boxes* in a formalism that we do not understand well enough to analyze (as opposed to, e.g., regex formulas), and yet, are known to be splittable by the splitters at hand. For example, a coreference resolver may be implemented as a decision tree over a multitude of features [155] but still be splittable by sequences of three sentences, and a POS and a NER tagger may be implemented by a bidirectional LSTM-CNN (Long short-term memory convolutional neural network) [23] and a bidirectional dependency network [162], respectively, but still be splittable by sentences. Technically, our results heavily rely on the algebraic properties of the splitter framework (associativity, transitivity, distributivity) that we established earlier. Additional problems we discuss are split-correctness and splittability under the assumption that the document conforms to a regular language.

Our framework can be seen as an extension of the *parallel-correctness* framework as proposed by Ameloot et al. [9, 10]. That work considers the parallel evaluation of relational queries. In our terms, that work studies self-splittability where spanners are replaced by relational queries and splitters by *distribution policies*.

Further Motivation

Besides the obvious, there another, perhaps less straightforward, motivations comes from *debugging* in the development of IE programs. For illustration, suppose that the developer seeks HTTP requests to a specific host on a specific date, and for that she seeks Host and Date headers that are close to each other; the system can warn the developer that the program is not splittable by HTTP requests like other frequent programs over the log (i.e., it can extract the Host of one request along with the Date of another), which is indeed a bug in this case. In the general case, the system can provide the user with the different splitters (sentences, paragraphs, requests, etc.) that the program is split-correct for, in contrast with what the developer believes should hold true.

Organization

This chapter is organized as follows. In Section 3.1 we define the central concepts of the framework and define the main decision problems and the highlander and cover condition.

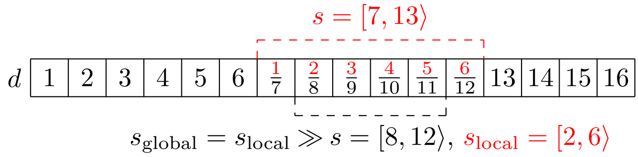

Figure 3.1: Visualization of the shift span operator, with $[8, 12\rangle = [2, 6\rangle \gg [7, 13\rangle$.

We study the framework in the context of the relational algebra in Section 3.2 and study extensions of the framework in Section 3.3.

3.1 General Framework and Main Problems

In this chapter, we are particularly interested in spanners that split documents into (possibly overlapping) segments. Formally, a *document splitter* (or *splitter* for short) is a functional unary document spanner S, that is, there is a single variable x such that, for every tuple $\mathrm{t} \in S(d)$ and $d \in \Sigma^*$, we have $\mathrm{Vars}(\mathrm{t}) = \{x\}$. So, a splitter can split the document into paragraphs, sentences, N-grams, HTTP messages, error messages, and so on.

In the sequel, unless mentioned otherwise, we denote a splitter by P and its unique variable by x_P. Furthermore, we assume, w.l.o.g., that $x_P \notin \mathrm{Vars}(S)$ for every spanner S that we do not call a splitter. Since a splitter outputs unary span relations, its output on a document d can be identified with the set of spans $\{t(x_P) \mid t \in P(d)\}$. We often use this simplified view on splitters and treat their output as a set of spans. A splitter P is *disjoint* if the spans extracted by P are always pairwise disjoint, that is, for all $d \in \Sigma^*$ and $s, s' \in P(d)$, the spans s and s' are disjoint. For instance, splitters that split documents into spans of the form $[1, k_1\rangle, [k_1, k_2\rangle, \ldots$ (such as paragraphs and sentences) are disjoint, but N-gram extractors are not disjoint for $N > 1$.

Next, we want to define when a spanner is *splittable* by a splitter, that is, when documents can be split into components such that the operation of a spanner can be distributed over the components. To this end, we first need some notation. Let d be a document, let $s := [i, j\rangle$ be a span of d, and let $s_{\text{local}} := [i', j'\rangle$ be a span of the document $d_{[i,j\rangle}$. Then, s_{local} also marks a span of the original document d, namely the one obtained from s_{local} by shifting it $i - 1$ characters to the right. We denote this shifted span by $s_{\text{global}} := s_{\text{local}} \gg s$, which abbreviates $s_{\text{local}} \gg (i-1)$ (cf. Figure 3.1). Hence, we have:

$$s_{\text{global}} \;=\; s_{\text{local}} \gg s \;=\; s_{\text{local}} \gg (i-1) \;=\; [i' + (i-1), j' + (i-1)\rangle\,.$$

Analogously, we denote by $s_{\text{global}} \ll s$ the span which is obtained from s_{global} by shifting it $i - 1$ characters to the left. We denote this shifted span by $s_{\text{local}} = s_{\text{global}} \ll s$, which abbreviates $s_{\text{global}} \ll (i-1)$. Hence, we have:

$$s_{\text{local}} \;=\; s_{\text{global}} \ll s \;=\; s_{\text{global}} \ll (i-1) \;=\; [i' - (i-1), j' - (i-1)\rangle\,.$$

Again, we overload the notation and write $\mathrm{t} \gg s$ (resp., $\mathrm{t} \ll s$) for the d-tuple that results from shifting each span in t by s to the right (resp., to the left).

Observation 3.1.1. *Let d be a document, s be a span of d, and t be a d_s-tuple. Then the d-tuple $\mathrm{t}' = \mathrm{t} \gg s$ is covered by s. Furthermore, given a d-tuple t, the tuple $\mathrm{t} \ll s$ is a well defined d_s-tuple if t is covered by s.* □

We now define the *composition* $S \circ P$ of a spanner S and splitter P. Intuitively, $S \circ P$ is the spanner that results from evaluating S on every part of the document extracted by P, with a proper shift of the indices. Recall that a splitter P is functional and has exactly one variable, thus, it always selects a set of unary tuples. In the following we abuse notation and simply write s rather than $s(x_P)$ when $s \in P(d)$ for some document d. We define on every document d,

$$(S \circ P)(d) \ := \bigcup_{s \in P(d)} \{\mathrm{t} \gg s \mid \mathrm{t} \in S(d_s)\} \ .$$

As an example, if S extracts person names and P is a sentence splitter, then $S \circ P$ is the spanner obtained by applying S to every sentence independently and taking the union of the results. Furthermore, if S extracts close mentions of email addresses and phone numbers, and P is the 5-gram splitter, then $S \circ P$ is obtained by applying S to each 5-gram individually. Since executing S on each individual output of P enables parallelization, it is interesting if there is a difference between the output of S and $S \circ P$ on some document d. This property clearly depends on the definitions of S and P. We define it formally in the following section under the name *self-splittability*.

3.1.1 Splittability and Split-correctness

We say that a spanner S is *splittable* by a splitter P via a spanner S_P if evaluating S on a document d always gives the same result as evaluating S_P on every string extracted by P (again with proper indentation of the indices). If such an S_P exists, we say that S is *splittable* by P; and if S_P is S itself, we say that S is *self-splittable* by P. We define these notions more formally.

Definition 3.1.2. Let S be a spanner and P a splitter. We say that:

1. S is *splittable by P via* a spanner S_P, if $S = S_P \circ P$;
2. S is *splittable by P* if there exists a spanner S_P such that $S = S_P \circ P$;
3. S is *self-splittable by P* if $S = S \circ P$.

We refer to S_P as the *split-spanner*.

As a simple example, suppose that we analyze a log of HTTP requests separated by blank lines and assume for simplicity that the log only consists of `GET` requests. Furthermore, assume that P splits the document into individual requests (without the blank lines) and that S extracts the request line, which is always the first line of the request. If S identifies the request line as the one following the blank line, then S is splittable by P via S_P, which is the same as S but replaces the requirement to follow a blank line with the requirement of being the first line. If, on the other hand, S identifies the request line as being the one starting with the word `GET`, then S is self-splittable by P, since we can apply S itself to every HTTP message independently.

Other examples are as follows. Many spanners S that extract person names do not look beyond the sentence level. This means that, if P splits to sentences, it is the case that S is self-splittable by P. Now suppose that S extracts mentions of email addresses and phone numbers based on the formats of the tokens, and moreover, it allows at most three tokens in between; if P is the N-gram splitter, then S is self-splittable by P for $N \geq 5$ but not for $N < 5$.

3.1.2 Main Decision Problems

The previous definitions and the motivating examples directly lead to the corresponding decision problems. We use $\mathcal{S}$ to denote a class of spanner representations (such as VSA or RGX).

SPLIT-CORRECTNESS[$\mathcal{S}$]	
Input:	Spanners $S, S_P \in \mathcal{S}$ and splitter $P \in \mathcal{S}$.
Question:	Is S splittable by P via S_P, that is, is $S = S_P \circ P$?

SPLITTABILITY[$\mathcal{S}$]	
Input:	Spanner $S \in \mathcal{S}$ and splitter $P \in \mathcal{S}$.
Question:	Is S splittable by P, that is, is there a spanner $S_P \in \mathcal{S}$, such that $S = S_P \circ P$?

SELF-SPLITTABILITY[$\mathcal{S}$]	
Input:	Spanner $S \in \mathcal{S}$ and splitter $P \in \mathcal{S}$.
Question:	Is S self-splittable by P, that is, is $S = S \circ P$?

Note that SELF-SPLITTABILITY[$\mathcal{S}$] is a special case of SPLIT-CORRECTNESS[$\mathcal{S}$] by choosing $S_P = S$. It can also be seen as a special case of SPLITTABILITY[$\mathcal{S}$] in the sense that SELF-SPLITTABILITY implies SPLITTABILITY.

One natural continuation of these three problems is the question where only S is given and it is asked whether P and S_P exist, such that S is splittable by P via S_P. In general, the answer to this question is yes, as every spanner is self-splittable by the splitter that only selects the whole document, i.e., $P = x\{\Sigma^*\}$. We therefore parameterize the decision problem with a class $\mathcal{P}$ of splitters.

SPLIT-EXISTENCE[$\mathcal{S}, \mathcal{P}$]	
Input:	Spanner $S \in \mathcal{S}$.
Question:	Is there a splitter $P \in \mathcal{P}$ such that S is splittable by P?

3.1.3 Cover and Highlander Condition

We now define two conditions on the interaction of spanners and splitters which will be useful to obtain upper bounds for Split-Correctness and Splittability. The first condition is the cover condition, which states that, for every tuple selected by a spanner, there is at least one span covering it.

Definition 3.1.3 (Cover Condition)**.** A splitter P *covers* a spanner S if for every document d and every non-empty tuple $\mathrm{t} \in S(d)$, there exists a span $s \in P(d)$ that covers the tuple t.

We show now that the cover condition is indeed necessary for Splittability.

Lemma 3.1.4. *Let S be a spanner which is splittable by a splitter P. Then P covers S.*

Proof. Let d be a document and $\mathrm{t} \in S(d)$ be a non-empty d-tuple. If S is splittable by P, there must be a spanner S_P such that $S = S_P \circ P$. By assumption, $\mathrm{t} \in S(d) = (S_P \circ P)(d)$, there is a span $s \in P(d)$, such that $\mathrm{t}' := \mathrm{t} \ll s \in S_P(d_s)$. Thus, $\mathrm{t} = \mathrm{t}' \gg s$ and therefore, by Observation 3.1.1, s covers t. $\square$

The second condition is the highlander condition which states that every tuple selected by the spanner is covered by at most one split.

Definition 3.1.5 (Highlander Condition)**.** A spanner S and a splitter P satisfy the *highlander condition* if, for every document d and every tuple $\mathrm{t} \in S(d)$, there exists at most one span $s \in P(d)$ that covers the tuple t.

Recall that disjointness is a natural property that splitters often satisfy in real life (e.g., tokenization, sentence boundary detection, paragraph splitting and segmentation). Given a disjoint splitter, it is easy to see that the highlander condition is almost guaranteed to be satisfied. The only case in which the highlander condition is not satisfied on a disjoint splitter is if the spanner selects a tuple which does not cover a non-empty part of the document, that is, S is not proper[2].

Lemma 3.1.6. *Let S be a proper spanner and let P be a disjoint splitter. Then S and P satisfy the highlander condition.*

Proof. For the sake of contradiction, assume that S is proper and P is disjoint but the highlander condition is not satisfied. Therefore there is a document $d \in \Sigma^*$ and a tuple $\mathrm{t} \in S(d)$, such that t is covered by $[i_1, j_1\rangle, [i_2, j_2\rangle \in P(d)$. We assume, w.l.o.g., that $i_1 \leq i_2$. The d-tuple t can not be empty, as S is proper. Therefore, let $[i, j\rangle$ be the minimal span covering t, which must be well defined as t is not empty. We observe that $[i, j\rangle$ is covered by $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$, that is $i_1 \leq i \leq j \leq j_1$ and $i_2 \leq i \leq j \leq j_2$. Due to disjointness of P, $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ must be disjoint, that is, $i_1 \leq j_1 \leq i_2 \leq j_2$. Thus, $[i, j\rangle$ can only be covered by both $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ if $i_1 = i = j = j_2$. Which implies that, the tuple $[i, j\rangle$ is empty, leading to the desired contradiction as $[i, j\rangle$ is the minimal span covering $\mathrm{t} \in S(d)$, which can not be empty if S is proper. $\square$

[2]Recall that a spanner S is proper if for every document $d \in \Sigma^*$, the empty tuple is not selected by S and $\mathrm{t} \in S(d)$ implies that the minimal span that covers t is not empty.

We conclude this section by defining the corresponding decision problems. As before, we use $\mathcal{S}$ to denote a class of spanner representations (such as VSA or RGX which we defined in Section 2.2).

	$\textsc{Disjoint}[\mathcal{S}]$
Input:	Splitter $P \in \mathcal{S}$.
Question:	Is P disjoint?

	$\textsc{Proper}[\mathcal{S}]$
Input:	Spanner $S \in \mathcal{S}$.
Question:	Is S proper?

	$\textsc{Cover}[\mathcal{S}]$
Input:	Spanner $S \in \mathcal{S}$ and splitter $P \in \mathcal{S}$.
Question:	Do S and P satisfy the cover condition?

	$\textsc{Highlander}[\mathcal{S}]$
Input:	Spanner $S \in \mathcal{S}$ and splitter $P \in \mathcal{S}$.
Question:	Do S and P satisfy the highlander condition?

3.2 The Framework in the Context of the Relational Algebra

In a complex pipeline that involves multiple spanners and splitters, it may be beneficial to reason about the manipulation or replacement of operators for the sake of query planning (in a similar way as we reason about query plans in a database system). In this section, we consider questions of this sort. As a basis for optimizing query plans, we show that the composition of spanners and splitters is associative (Section 3.2.3) and that splittability as well as self-splittability is transitive (Section 3.2.4). Furthermore, we give a sufficient condition which for distributivity of spanner composition over join (Section 3.2.5). Afterwards we study the problem of deciding on the splittability in the presence of *black-box* spanners that are known to follow *split constraints* (Section 3.3.1).

3.2.1 Characterization of Composition

The following lemma gives an algebraic characterization of $S \circ P$.

Lemma 3.2.1. *Let S be a spanner and P be a splitter. Then $S \circ P = \pi_{\mathrm{Vars}(S)}((\Sigma^* \cdot x_P\{S\} \cdot \Sigma^*) \bowtie P)$.*

Proof. Let S and P be as given and let $S' = \pi_{\mathrm{Vars}(S)}((\Sigma^* \cdot x_P\{S\} \cdot \Sigma^*) \bowtie P)$. We show both directions of the equation separately.

($S' \subseteq S \circ P$): Let $d \in \Sigma^*$ be a document and $\mathrm{t}' \in S'(d)$ be a d-tuple. Per definition of S', there is a tuple $\mathrm{t}_{x_P} \in \big((\Sigma^* \cdot x_P\{S\} \cdot \Sigma^*) \bowtie P\big)(d)$ with $\mathrm{t}' := \pi_{\mathrm{Vars}(S)}(\mathrm{t}_{x_P})$ and

$s := \mathrm{t}_{x_P}(x_P)$ covers t. Let $s := \mathrm{t}_{x_P}(x_P) \in P(d)$ and $\mathrm{t} = \mathrm{t}' \ll s$ be the d_s-tuple with $\mathrm{t} \in S(d_s)$. Thus, due to $s \in P(d)$ and $\mathrm{t} \in S(d_s)$, it must hold that $\mathrm{t}' = \mathrm{t} \gg s \in (S \circ P)(d)$.

($S \circ P \subseteq S'$): Let $d \in \Sigma^*$ be a document, $s \in P(d)$, and $\mathrm{t} \in S(d_s)$. Let $\mathrm{t}' = \mathrm{t} \gg s$, thus, by Observation 3.1.1, s covers t'. Let t_{x_P} be the d-tuple defined by

$$\mathrm{t}_{x_P}(v) \;:=\; \begin{cases} \mathrm{t}'(v) & \text{if } v \in \mathrm{Vars}(\mathrm{t}') \\ s & \text{if } v = x_P \;. \end{cases}$$

Therefore, $\mathrm{t}_{x_P} \in \big((\Sigma^* \cdot x_P\{S\} \cdot \Sigma^*) \bowtie P\big)(d)$ and $\mathrm{t}' := \pi_{\mathrm{Vars}(\mathrm{t}')}(\mathrm{t}_{x_P}) \in S'(d)$. □

3.2.2 Characterization of the Splittability Problem

We now give a characterization of the Splittability problem. To this end, we show that a spanner is splittable by a splitter if and only if it is splittable via a specific canonical split-spanner.

The following example illustrates that there can be different split-spanners witnessing splittability.

Example 3.2.2. Consider $S := ay\{b\}b$ and $P := x\{ab\}\,b \vee ax\{bb\}$. Then, both $S = S_P \circ P$ and $S = S'_P \circ P$ for $S_P := ay\{b\}$ and $S'_P := y\{b\}b$ but $S_P \neq S'_P$. The reason why this happens is that P selects two different spans $s = [1,3\rangle$ and $s' = [2,4\rangle$ that both cover the span $[2,3\rangle$ selected by S on abb. Since the selected spans are different, the split-spanners S_P and S'_P need to be different as well to be able to simulate S. Notice that P is not a disjoint splitter, as $[1,3\rangle$ and $[2,4\rangle$ are not disjoint. □

We show, that there is a canonical split-spanner S_P^{can} for every spanner S and splitter P such that S is splittable by P if and only if it is splittable via S_P^{can}:

$$S_P^{\mathrm{can}}(d) \;:=\; \big\{\mathrm{t} \mid \forall d' \in \Sigma^*, \forall s \in P(d') \text{ such that } d'_s = d, \text{ it holds that } (\mathrm{t} \gg s) \in S(d')\big\} \;.$$

Intuitively, a tuple is selected by S_P^{can} if and only if it is "safe" to be selected. A d-tuple t is not safe if there is a document d' and a split $s \in P(d')$ with $d'_s = d$ and $\mathrm{t} \gg s \notin S(d)$. As we will show in the following lemma, $S_P^{\mathrm{can}} \circ P \subseteq S$.

Note that the definition of S_P^{can} is not the same as in Doleschal et al. [33], where S_P^{can} is defined with an existential quantifier instead of the second universal quantifier in the present definition. The present canonical split-spanner can be used more generally.

Lemma 3.2.3. *Let S be a document spanner and P be a document splitter. Then $S_P^{\mathrm{can}} \circ P \subseteq S$.*

Proof. Let S and P be as stated. Recalling the definition of the $\circ$ operator, we have that

$$(S_P^{\mathrm{can}} \circ P)(d) \;:=\; \bigcup_{s \in P(d)} \{\mathrm{t} \gg s \mid \mathrm{t} \in S_P^{\mathrm{can}}(d_s)\}\,.$$

Let d be a document and $\mathrm{t} \in (S_P^{\mathrm{can}} \circ P)(d)$ be a d-tuple. Then, there is a span $s \in P(d)$, such that $\mathrm{t}' := \mathrm{t} \ll s \in S_P^{\mathrm{can}}(d_s)$. Per definition of S_P^{can} it must hold that $\mathrm{t} = \mathrm{t}' \gg s \in S(d)$, concluding the proof. □

Theorem 3.2.4. *Let S be a document spanner and P be a document splitter. Then S is splittable by P if and only if S is splittable by P via S_P^{can}.*

Proof. We only have to show the "only if" direction, since the other direction is trivial. Due to Lemma 3.2.3, it suffices to show that $S \subseteq S_P^{\mathrm{can}} \circ P$.

Assume that S is splittable by P via some spanner S_P. We begin by showing that $S_P \subseteq S_P^{\mathrm{can}}$. Let d be a document and $\mathrm{t} \in S(d)$ be a d-tuple. As $S = S_P \circ P$ there is a split $s \in P(d)$, such that $\mathrm{t}' := \mathrm{t} \ll s \in S_P(d_s)$. For the sake of contradiction, assume that $\mathrm{t}' \notin S_P^{\mathrm{can}}(d_s)$. By definition of S_P^{can}, there is a document $d' \in \Sigma^*$ and a span $s' \in P(d')$ with $d_s = d'_{s'}$ such that $\mathrm{t}' \gg s' \notin S(d')$. Therefore, $s' \in P(d')$ and $\mathrm{t}' \in S_P(d'_{s'})$ but $\mathrm{t}' \gg s' \notin S(d')$, leading to the desired contradiction as $S = S_P \circ P$. Therefore, $S_P \subseteq S_P^{\mathrm{can}}$. It remains to show that $S \subseteq S_P^{\mathrm{can}} \circ P$. Recalling the definition of $S \circ P$,

$$
\begin{aligned}
(S_P \circ P)(d) &= \bigcup_{s \in P(d)} \{\mathrm{t} \gg s \mid \mathrm{t} \in S_P(d_s)\} \\
&\subseteq \bigcup_{s \in P(d)} \{\mathrm{t} \gg s \mid \mathrm{t} \in S_P^{\mathrm{can}}(d_s)\} \\
&= (S_P^{\mathrm{can}} \circ P)(d) \ .
\end{aligned}
$$

Therefore, $S = S_P \circ P \subseteq S_P^{\mathrm{can}} \circ P$, concluding the proof. □

3.2.3 Associativity of Composition

Using the characterization of composition, we will now show that composition is *associative.*

Theorem 3.2.5. *Given a spanner S and two splitters P_1 and P_2, then it holds that $S \circ (P_1 \circ P_2) = (S \circ P_1) \circ P_2$.*

Proof. We use the algebraic characterization from Lemma 3.2.1 and denote the variables of the splitters P_1 and P_2 by x_1 and x_2, respectively.

We begin by showing that the following equality holds for every spanner S_1 and S_2 and every variable $x \notin \mathrm{Vars}(S_1) \cup \mathrm{Vars}(S_2)$.

$$
\Sigma^* \cdot x\{S_1 \bowtie S_2\} \cdot \Sigma^* \quad = \quad (\Sigma^* \cdot x\{S_1\} \cdot \Sigma^*) \bowtie (\Sigma^* \cdot x\{S_2\} \cdot \Sigma^*) \qquad (\dagger)
$$

Let d be a document and t be a tuple such that $\mathrm{t} \in (\Sigma^* \cdot x\{S_1 \bowtie S_2\} \cdot \Sigma^*)(d)$. Let $s = \mathrm{t}(x)$ be the span assigned to x and $\mathrm{t}' = \mathrm{t} \ll s$. By definition of concatenation and variable enclosing, it holds that

$$
\mathrm{t}' \ \in \ x\{S_1 \bowtie S_2\}(d_s) \quad \text{and} \quad \pi_{\mathrm{Vars}(S_1 \bowtie S_2)}(\mathrm{t}') \ \in \ (S_1 \bowtie S_2)(d_s)
$$

and therefore, for $i \in \{1, 2\}$, it holds that $\pi_{\mathrm{Vars}(S_i)}(\mathrm{t}') \in S_i(d_s)$ and $\pi_{\mathrm{Vars}(S_i) \cup \{x\}}(\mathrm{t}') \in x\{S_i(d_s)\}$. We can conclude that $\pi_{\mathrm{Vars}(S_i) \cup \{x\}}(\mathrm{t}) \in (\Sigma^* \cdot x\{S_i(d)\} \cdot \Sigma^*)(d)$, and finally

$$
\mathrm{t} \ \in \ \big((\Sigma^* \cdot x\{S_1\} \cdot \Sigma^*) \bowtie (\Sigma^* \cdot x\{S_2\} \cdot \Sigma^*)\big)(d) \ .
$$

The other direction can be shown symetrically. Let d be a document, $\mathrm{t} \in \big((\Sigma^* \cdot x\{S_1\} \cdot \Sigma^*) \bowtie (\Sigma^* \cdot x\{S_2\} \cdot \Sigma^*)\big)(d)$ be a tuple and $s = \mathrm{t}(x)$. Then, for all $i \in \{1,2\}$, $\pi_{\mathrm{Vars}(S_i)\cup\{x\}}(\mathrm{t}) \in (\Sigma^* \cdot x\{S_i\} \cdot \Sigma^*)(d)$ and therefore $\pi_{\mathrm{Vars}(S_i)\cup\{x\}}(\mathrm{t} \ll s) \in x\{S_i\}$. We can conclude that $\mathrm{t} \ll s \in x\{S_1 \bowtie S_2\}$ and therefore $\mathrm{t} \in \Sigma^* \cdot x\{S_1 \bowtie S_2\} \cdot \Sigma^*$, which concludes the proof of Equation (†).

We will now show the following equalities.

$$
\begin{aligned}
(S \circ P_1) \circ P_2 &\overset{(1)}{=} \pi_{\mathrm{Vars}(S)}\Big(\big(\Sigma^* \cdot x_2\{S \circ P_1\} \cdot \Sigma^*\big) \bowtie P_2\Big)\\
&\overset{(2)}{=} \pi_{\mathrm{Vars}(S)}\Big(\big(\Sigma^* \cdot x_2\big\{\pi_{\mathrm{Vars}(S)}\big((\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*) \bowtie P_1\big)\big\} \cdot \Sigma^*\big) \bowtie P_2\Big)\\
&\overset{(3)}{=} \pi_{\mathrm{Vars}(S)}\Big(\big(\Sigma^* \cdot x_2\big\{(\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*) \bowtie P_1\big\} \cdot \Sigma^*\big) \bowtie P_2\Big)\\
&\overset{(4)}{=} \pi_{\mathrm{Vars}(S)}\Big(\Big(\big(\Sigma^* \cdot x_2\big\{(\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*)\big\} \cdot \Sigma^*\big) \bowtie (\Sigma^* \cdot x_2\{P_1\} \cdot \Sigma^*)\Big) \bowtie P_2\Big)\\
&\overset{(5)}{=} \pi_{\mathrm{Vars}(S)}\Big(\big((\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*) \bowtie (\Sigma^* \cdot x_2\{P_1\} \cdot \Sigma^*)\big) \bowtie P_2\Big)\\
&\overset{(6)}{=} \pi_{\mathrm{Vars}(S)}\Big(\big(\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*\big) \bowtie \pi_{x_1}\big((\Sigma^* \cdot x_2\{P_1\} \cdot \Sigma^*) \bowtie P_2\big)\Big)\\
&\overset{(7)}{=} \pi_{\mathrm{Vars}(S)}\Big(\big(\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*\big) \bowtie (P_1 \circ P_2)\Big)\\
&\overset{(8)}{=} S \circ (P_1 \circ P_2)
\end{aligned}
$$

The equalities (1), (2), (7), and (8) hold by the algebraic characterization of Lemma 3.2.1. The equalities (3) and (6) hold by the definition of projection and join in the relational algebra, i.e., it is enough to project only once and the intermediate projections do not have an effect, as the variables removed by the projection are not part of the natural join. The Equality (4) follows from Equation (†) by using $S_1 := (\Sigma^* \cdot x_1\{S\} \cdot \Sigma^*)$, $S_2 := P_1$, and $x := x_2$.

The Equality (5) follows from the observation that in the left-hand side of the join, the only restriction of x_2 is that the span of x_2 has to cover the span of x_1. However, this restriction is already imposed by the right-hand side of the join, where x_2 has to cover the part of the document matched by P_1 and therefore the span of x_1. Therefore, removing x_2 on the lefthand side of the join does not alter the result. This concludes the proof. □

3.2.4 Transitivity of (Self-)Splittability

The fact that spanner composition is associative allows us to show that splittability and self-splittability are *transitive*.

Theorem 3.2.6. *Let S be a document spanner and P_1 and P_2 be document splitters such that S is splittable by P_1 and P_1 is splittable by P_2, then S is splittable by P_2. If furthermore S is self-splittable by P_1 and P_1 is self-splittable by P_2 then S is self-splittable by P_2.*

Proof. Assume that S is splittable by P_1 and P_1 is splittable by P_2, then there is a spanner S' such that $S = S' \circ P_1$. Furthermore, there is a splitter P' such that $P_1 = P' \circ P_2$. As the composition of document spanners is associative, we can conclude that

$$S \quad = \quad S' \circ \big(P' \circ P_2\big) \quad = \quad \big(S' \circ P'\big) \circ P_2 \ .$$

Therefore S is splittable by P_2 via $S' \circ P'$.

Let now S be self-splittable by P_1 and P_1 be self-splittable by P_2. Then we have $S' = S$ and $P' = P$ in the equation above and using $S = S \circ P_1 = S' \circ P'$ we can conclude that $S = S \circ P_2$, which shows that S is self-splittable by P_2. ☐

3.2.5 Distributivity of Composition and Join

Another important question is whether applying a splitter commutes with other operations of the algebra, especially the join operation. We now give a sufficient precondition for *distributivity*, which is defined as

$$(S_1 \bowtie S_2) \circ P \quad = \quad (S_1 \circ P) \bowtie (S_2 \circ P) \ .$$

The problem is that the two spans on the righthand side of the equation could be different. If they are, the equation needs not to be true, though it is still possible in some corner cases. An obvious idea is to require that $S_1 \circ P$ and $S_2 \circ P$ satisfy the highlander condition. However, as we show in Example 3.2.7, this might not be enough, as it is possible that there are two overlapping spans covering tuples from S_1 and S_2, respectively, such that x is in the intersection of both spans. Even requiring that the spanners are proper and the splitter is disjoint might not be enough if x is assigned the empty span. This explains the rather complicated precondition of the following theorem.

Example 3.2.7. Let $S_1 := \Sigma^* \cdot x_1\{a\} \cdot x_2\{b\} \cdot \Sigma^*$, $S_2 := \Sigma^* \cdot x_2\{b\} \cdot x_3\{a\} \cdot \Sigma^*$, and $P := \Sigma^* \cdot x\{\Sigma \cdot \Sigma\} \cdot \Sigma^*$. We observe that S_1 (resp., S_2) and P satisfy the highlander condition.

Let $S := S_1 \bowtie S_2$ be the join of both spanners and let $d = aba$. It follows that $P(d) = \{[1,3\rangle, [2,4\rangle\}$ and $S(d) = \{\mathrm{t}\}$, where $\mathrm{t}(x_1) = [1,2\rangle$, $\mathrm{t}(x_2) = [2,3\rangle$, and $\mathrm{t}(x_3) = [3,4\rangle$. As there is no span $s \in P(d)$ that covers $\mathrm{t} \in S(d)$ it follows directly from Lemma 3.1.4 that S is not splittable by P, and therefore $S \circ P \neq S$. However, both spanners, S_1 and S_2, are self-splittable by P which implies that $(S_1 \circ P) \bowtie (S_2 \circ P) = S_1 \bowtie S_2 = S$. It follows directly

$$(S_1 \bowtie S_2) \circ P \quad = \quad S \circ P \quad \neq \quad S \quad = \quad S_1 \bowtie S_2 \quad = \quad (S_1 \circ P) \bowtie (S_2 \circ P) \ ,$$

and therefore, spanner composition does not distribute over the join.

Theorem 3.2.8. *Let P be a disjoint document splitter and S_1 and S_2 be document spanners such that $X := \mathrm{Vars}(S_1) \cap \mathrm{Vars}(S_2) \neq \emptyset$ and the spanner $\pi_X(S_1) \bowtie \pi_X(S_2)$ is proper. Then, spanner composition distributes over the join, that is,*

$$(S_1 \bowtie S_2) \circ P \quad = \quad (S_1 \circ P) \bowtie (S_2 \circ P) \ .$$

Proof. Let d be a document and t be a tuple such that $\mathrm{t} \in \big((S_1 \bowtie S_2) \circ P\big)(d)$. Then there is a decomposition $d = d_1 \cdot d_2 \cdot d_3$ such that $s = [|d_1|+1, |d_1 \cdot d_2|+1\rangle \in P(d)$, and $\mathrm{t} \ll |d_1| \in (S_1 \bowtie S_2)(d_2)$. We can conclude that, for all $i \in \{1,2\}$, $\pi_{\mathrm{Vars}(S_i)}(\mathrm{t} \ll |d_1|) \in S_i(d_2)$, therefore $\pi_{\mathrm{Vars}(S_i)}(\mathrm{t}) \in (S_i \circ P)(d)$, and finally $\mathrm{t} \in \big((S_1 \circ P) \bowtie (S_2 \circ P)\big)(d)$.

For the other direction let d be a document and t be a tuple such that $\mathrm{t} \in \big((S_1 \circ P) \bowtie (S_2 \circ P)\big)(d)$. For $i \in \{1,2\}$, it must hold that $\pi_{\mathrm{Vars}(S_i)}(\mathrm{t}) \in (S_i \circ P)(d)$. Thus there are spans s_1 and s_2, such that $\pi_{\mathrm{Vars}(S_i)}(\mathrm{t}) \ll s_i \in S_i(d_{s_i})$. Due to $\pi_X(S_1) \bowtie \pi_X(S_2)$ being proper, the minimal span s that covers $\pi_X(\mathrm{t})$ is not empty. As furthermore s is covered by both s_1 and s_2 and P is disjoint, we can conclude that $s_1 = s_2$. Therefore, we have that $\mathrm{t} \ll s_1 \in (S_1 \bowtie S_2)(d_{s_1})$ and finally $\mathrm{t} \in \big((S_1 \bowtie S_2) \circ P\big)(d)$, concluding the proof. □

Note that in the previous theorem it is sufficient if either the spanner $\pi_X(S_1)$ or the spanner $\pi_X(S_2)$ is proper.

3.3 Extensions of the Framework

In this section, we will study extensions of the framework. We begin by studying the framework in the presence of document spanners which are represented by black boxes and conclude this section by studying the framework under schema constraints.

3.3.1 Split-Constrained Black Boxes

We begin with motivating examples.

Example 3.3.1. In this example and the next, we'll denote by $S(x,y)$ that spanner S uses the variables x and y. Consider the spanner S that seeks to extract adjectives for Galaxy phones from reports. We define this spanner by joining three spanners:

The spanner $S_1(x,y)$ is given by the regex formula

$$\Sigma^* \cdot x\{\mathsf{Galaxy\ [A\text{-}Z][0\text{-}9]}^*\} \cdot \Sigma^* \cdot y\{\Sigma^*\} \cdot \Sigma^*$$

that extracts mentions of Galaxy brands (e.g., Galaxy A6 and Galaxy S8) followed by substrings y.

The spanner $S_2(x,x')$ is a coreference resolver (e.g., the *sieve* algorithm [129]) that finds spans x' that coreference spans x. The spanner $S_3(x',y)$ finds pairs of noun phrases x' and attached adjectives y (e.g., based on a Recursive Neural Network [154]).

For example, consider the review *"I am happy with my Galaxy A6. It is stable."* Here, in one particular match, x will match (the span of) *Galaxy A6*, x' will match *it* (which is an anaphora for *Galaxy A6*), and y will match *stable*. (Other matches are possible too.)

How should a system find an efficient query plan to this join on a long report? Naively materializing each relation might be too costly: $S_1(x,y)$ may produce too many matches, and $S_2(x,x')$ and $S_3(x',y)$ may be computationally costly. Nevertheless, we may have the information that S_2 is splittable by paragraphs and that S_3 is splittable by sentences (hence, by paragraphs). This information suffices to determine that the entire join $S_1(x,y) \bowtie S_2(x,x') \bowtie S_3(x',y)$ is splittable, hence parallelizable, by paragraphs. □

Example 3.3.2. Now consider the spanner that joins two spanners: $S(x)$ extracts spans x followed by the phrase *"is kind"* (e.g., *"Barack Obama is kind"*). The spanner $S'(x)$ extracts all spans x that match person names. Clearly, the spanner $S(x)$ does not split by a natural splitter, since it includes, for instance, the entire prefix of the document before *"is kind"*. However, by knowing that $S'(x)$ splits by sentences, we know that the join $S(x) \bowtie S'(x)$ splits by sentences. Moreover, by knowing that $S'(x)$ splits by 3-grams, we can infer that $S(x) \bowtie S'(x)$ splits by 5-grams. Here, again, the holistic analysis of the join infers splittability in cases where intermediate spanners are not splittable. $\square$

We now formalize the splittability question that the examples give rise to. A *spanner signature* Λ is a collection $\{\lambda_1, \ldots, \lambda_k\}$ of *spanner symbols*, where each λ_i is associated with a set $\mathrm{Vars}(\lambda_i)$ of span variables. Furthermore, let $X_i := \mathrm{Vars}(\lambda_i) \cap \left(\bigcup_{i<j\leq k} \mathrm{Vars}(\lambda_j)\right)$. We assume that $X_i \neq \emptyset$, for all $1 \leq i \leq k$. An *instance* I of Λ associates with each spanner symbol λ_i an actual spanner S_i such that $\mathrm{Vars}(S_i) = \mathrm{Vars}(\lambda_i)$ and $\pi_{X_i}(S_i)$ is proper. In Example 3.3.1, λ_1 would correspond to the regex-formula S_1, with $\mathrm{Vars}(\lambda_1) = \{x, y\}$. Furthermore, λ_2 and λ_3 would correspond to the name of a coreference resolver S_2 and an adjective extractor S_3, respectively, with $\mathrm{Vars}(\lambda_1) = \{x, x'\}$ and $\mathrm{Vars}(\lambda_2) = \{x', y\}$.

Let Λ be a spanner signature and I an instance of Λ. We denote by $I_{\bowtie}$ the spanner that is given by

$$I_{\bowtie} \quad := \quad S_1 \bowtie \cdots \bowtie S_k \ .$$

We note that this is well-defined due to the associativity and commutativity of the join operator.

A *split constraint* over a spanner signature Λ is an expression of the form "λ_i is self-splittable by the splitter P," which we denote by $\lambda_i \sqsubseteq P$. An instance I of Λ *satisfies* a set C of split constraints, denoted $I \models C$, if for every constraint $\lambda_i \sqsubseteq P$ in C it is the case that P_i is self-splittable by P. The problem of *split-correctness with black boxes* is the following:

Black Box Splittability	
Input:	A spanner signature Λ, a set C of split constraints, and a splitter P.
Question:	Is $I_{\bowtie}$ self-splittable by P whenever I is an instance of Λ such that $I \models C$?

A natural question to ask is the following. Assume that all spanners are self-splittable by the same splitter P, that is, $\lambda \sqsubseteq P$, for every $\lambda \in \Lambda$. Does this imply that $I_{\bowtie}$ is self-splittable by P? In general, the answer to this question is no, as shown by the spanners and splitter defined in Example 3.2.7. The next result shows that in the presence of disjoint splitters the join operator preserves self-splittability.

Theorem 3.3.3. *Let P be a disjoint splitter, let Λ be a spanner signature, and let C be a set of split constraints, such that $\lambda_i \sqsubseteq P \in C$, for all $1 \leq i \leq k$. Then $I_{\bowtie}$ is self-splittable by P if $I \models C$.*

Proof. Let I be an instance of Λ, such that $I \models \Lambda$ and let P_i be the spanner interpreting λ_i. We have to show, that $I_{\bowtie} = I_{\bowtie} \circ P$.

Recall that per definition of Λ, $X_i = \text{Vars}(\lambda_i) \cap \big(\bigcup_{i<j\leq k} \text{Vars}(\lambda_j)\big)$, and $X_i \neq \emptyset$, for all $1 \leq i \leq k$. Furthermore, per definition of I, $\pi_{X_i}(S_i)$ is proper and S_i is self-splittable by P, for all $1 \leq i \leq k$. Thus, using associativity of $\bowtie$ and Theorem 3.2.8, it follows that

$$\begin{aligned}
I_{\bowtie} \circ P &= \big(S_1 \bowtie \ldots \bowtie S_k\big) \circ P \\
&= \Big(S_1 \bowtie \big(S_2 \bowtie (\cdots \bowtie S_k)\big)\Big) \circ P \\
&= (S_1 \circ P) \bowtie \Big(\big(S_2 \bowtie (S_3 \bowtie (\ldots \bowtie S_k))\big) \circ P\Big) \\
&\ \ \vdots \\
&= (S_1 \circ P) \bowtie \ldots \bowtie (S_k \circ P) \\
&= S_1 \bowtie \ldots \bowtie S_k \\
&= I_{\bowtie} \,.
\end{aligned}$$

This concludes the proof. $\square$

Observe that the requirement that $\pi_{X_i}(S_i)$ is proper is always satisfied if S_i does not assign the empty span to variables and it holds, for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in S_i(d)$, that $X_i \subseteq \text{Vars}(\mathrm{t})$.

3.3.2 Schema Constraints

Sometimes a spanner is not splittable by a given splitter, because of a reason that seems marginal. For instance, the spanner may first check that the document conforms to some standard format, such as Unicode, UTF-8, CSV, HTML, etc. This is no issue, if the document collection is verified to conform to the standard prior to splitting. In this section, we will introduce *schema constraints*, which extend the framework in order to embark this.

A *schema constraint* $\mathcal{L}$ is a—not necessary regular—language. We say that two spanners S, S' are equivalent under a schema constraint $\mathcal{L}$ if and only if for all documents $d \in \mathcal{L}$ it holds that $S(d) = S'(d)$. We denote this by $S \equiv_{\mathcal{L}} S'$. We say that S is splittable by P via S_P under the schema constraint $\mathcal{L}$ if and only if $S \equiv_{\mathcal{L}} S_P \circ P$. A schema constraint $\mathcal{L}$ is regular, if $\mathcal{L}$ is regular.

Note that every language $\mathcal{L}$ is also a Boolean spanner, extracting the empty tuple on every document $d \in \mathcal{L}$ and the empty set on every documents $d \notin \mathcal{L}$. Thus, the join of a spanner and a language, as used in the following lemma, is defined as the join of two spanners.

Lemma 3.3.4. *Let S, S_P be spanners, P be a splitter, and $\mathcal{L}$ be a schema constraint. Then $S \equiv_{\mathcal{L}} S_P \circ P$ if and only if $S \bowtie \mathcal{L} = \big(S_P \circ (P \bowtie \mathcal{L})\big)$.*

Proof. Per definition of $\equiv_{\mathcal{L}}$, it holds that $S \equiv_{\mathcal{L}} S_P \circ P$ if and only if $S \bowtie \mathcal{L} = (S_P \circ P) \bowtie \mathcal{L}$. Therefore, we have to show that $(S_P \circ P) \bowtie \mathcal{L} = S_P \circ (P \bowtie \mathcal{L})$.

$$\begin{aligned}
(S_P \circ P) \bowtie \mathcal{L} &\overset{(1)}{=} \pi_{\mathrm{Vars}(S_P)}\big((\Sigma^* \cdot x\{S_P\} \cdot \Sigma^*) \bowtie P\big) \bowtie \mathcal{L} \\
&\overset{(2)}{=} \pi_{\mathrm{Vars}(S_P)}\Big(\big((\Sigma^* \cdot x\{S_P\} \cdot \Sigma^*) \bowtie P\big) \bowtie \mathcal{L}\Big) \\
&\overset{(3)}{=} \pi_{\mathrm{Vars}(S_P)}\big((\Sigma^* \cdot x\{S_P\} \cdot \Sigma^*) \bowtie (P \bowtie \mathcal{L})\big) \\
&\overset{(4)}{=} S_P \circ (P \bowtie \mathcal{L})
\end{aligned}$$

The equalities (1) and (4) are by the algebraic characterization of Lemma 3.2.1. The Equality (2) is by the fact that $\mathcal{L}$ does not use variables and we are therefore allowed to change the order of projection and join. Finally, the Equality (3) holds because of the associativity of joins. $\square$

It follows directly from Lemma 3.3.4 that schema constraints do not extend the expressivity of the general framework.

Schema constraints also give rise to other problems that can be studied. For instance, it may be the case that we already have a spanner and splitter available that we do not want to change, but we want to know whether there exists a schema constraint $\mathcal{L}$ such that the spanner is splittable by the splitter under the schema constraint. In general, the answer to this is always positive, splittability holds for every combination of a spanner and a splitter under the schema constraint $\mathcal{L} = \emptyset$. Therefore, we say that *a schema constraint $\mathcal{L}$ covers S* if and only if the splitter $P_{\mathcal{L}} := x\{\mathcal{L}\}$ covers S.

Next we observe that, for each spanner S, there is a minimal schema constraint $\mathcal{L}_S := \{d \mid S(d) \neq \emptyset\}$ such that split-correctness holds under $\mathcal{L}_S$ if it holds for every schema constraint which covers the spanner. We first observe that $\mathcal{L}_S$ is indeed contained in every schema condition which covers S.

Observation 3.3.5. *Let S be a spanner and let $\mathcal{L}$ be a schema constraint which covers S. Then, $P_{\mathcal{L}} = x\{\mathcal{L}\}$ covers S and therefore, $\mathcal{L}_S \subseteq \mathcal{L}$.* $\square$

The following observation follows directly from Observation 3.3.5 and Lemma 3.3.4.

Observation 3.3.6. *Let S and S_P be spanners, P be a splitter, and $\mathcal{L}$ be a schema constraint which covers S. Then $S \equiv_{\mathcal{L}} S_P \circ P$ implies that $S \equiv_{\mathcal{L}_S} S_P \circ P$.* $\square$

Chapter 4

Complexity Results for Regular Document Spanners

We now give the main results for the decision problems we introduced in Chapter 3 in the case of regular spanners. The following two theorems summarize the main complexity results.

Recall that $\mathcal{S}_{\text{general}}$ is the set of all introduced representations of regular spanner, that is, all representations depicted in Figure 2.4, and $\mathcal{S}_{\text{tractable}}$ is the unambiguous and sequential subset thereof, that is, $\mathcal{S}_{\text{tractable}} = \{\text{usVSA}, \text{ufVSA}, \text{dsVSA}, \text{dfVSA}\}$.

Theorem 4.0.1. *Let $\mathcal{S} \in \mathcal{S}_{\text{general}}$ be a class of document spanners. Then the decision problems* Split-Correctness$[\mathcal{S}]$ *and* Self-Splittability$[\mathcal{S}]$ *are* PSPACE*-complete. Furthermore,* Split-Correctness$[\mathcal{S}]$ *and* Self-Splittability$[\mathcal{S}]$ *are in* PTIME *if*

- $\mathcal{S} \in \mathcal{S}_{\text{tractable}}$, *and*
- *the spanner is proper and the splitter is disjoint, or the highlander condition is satisfied by the spanner and splitter.*

Theorem 4.0.2. *Let $\mathcal{S} \in \mathcal{S}_{\text{general}}$ be a class of document spanners. Then deciding* Splittability$[\mathcal{S}]$ *is in* EXPSPACE *and* PSPACE*-hard. Furthermore, it is* PSPACE*-complete if one of the following two conditions is satisfied:*

- *the highlander condition is satisfied by spanner and splitter, or*
- *the spanner is proper and the splitter is disjoint.*

Organization

This chapter is organized as follows. In Section 4.1 we give some technical foundations. We study the upper bounds of Split-Correctness and Splittability in Section 4.2 and the upper bounds of Splittability in Section 4.3. The corresponding lower bounds are studied in Section 4.4. In Section 4.5 we study the connection of Split-Existence and language primality. We conclude this chapter by studying the complexity of schema constraints in Section 4.6.

4.1 Technical Foundations

In this section, we provide the technical foundation for our main results. In Section 4.1.1 we show that, given a spanner S and a splitter P, represented by vset-automata $A_S, A_P \in$ VSA, the spanner $S \circ P$ can be constructed as a vset-automaton $A_{S\circ P}$. Furthermore, if A_S and A_P are unambiguous and sequential, and $S \circ P$ and P satisfy the highlander condition, then the constructed vset-automaton for $A_{S\circ P}$ is also unambiguous and sequential. We study the complexity of containment in Section 4.1.2 and provide upper bounds for the complexity of Disjoint, Proper, Highlander and Cover in Section 4.1.3.

4.1.1 Spanner/Splitter Composition

We begin by showing that, given $A_S, A_P \in$ VSA, a vset-automaton that represents the spanner $[\![A_S]\!] \circ [\![A_P]\!]$ can be constructed. If A_S and A_P are unambiguous and sequential, and $[\![A_S]\!] \circ [\![A_P]\!]$ and P satisfy the highlander condition, then the constructed vset-automaton is also unambiguous and sequential.

Proposition 4.1.1. *Given vset-automata A_S and A_P representing a spanner and a splitter, respectively, a vset-automaton $A_{S\circ P}$ can be constructed in polynomial time, such that*

- $[\![A_{S\circ P}]\!] = [\![A_S]\!] \circ [\![A_P]\!]$;
- $A_{S\circ P} \in$ sVSA *if* $A_S \in$ sVSA; *and*
- $A_{S\circ P} \in$ usVSA *if* $A_S, A_P \in$ usVSA, *and $A_{S\circ P}$ and A_P satisfy the highlander condition.*

Peterfreund et al. [123] showed that the join of sequential vset-automata can be computed in polynomial time, if the number of shared variables is bounded by a constant. Furthermore, for sequential vset-automata, projection can be computed in polynomial time. The proof extends to arbitrary vset-automata, if the number of removed variables is bounded by a constant. This shows the first two bullet points. We show the last bullet point, using an explicit construction, that also proves the first two bullet points.

Proof. Let x_P be the variable of A_P and assume, w.l.o.g., that $x_P \notin \text{Vars}(A_S)$.[1] We use the algebraic characterization from Lemma 3.2.1 that states that

$$S \circ P \;=\; \pi_{\text{Vars}(S)}\big((\Sigma^* \cdot x_P\{S\} \cdot \Sigma^*) \bowtie P\big)$$

for a spanner S and a splitter P. Let $A_S = (\Sigma, V, Q_S, q_{0,S}, Q_{F,S}, \delta_S)$ and $A_P = (\Sigma, \{x_P\}, Q_P, q_{0,P}, Q_{F,P}, \delta_P)$ be vset-automata representing a spanner S and a splitter P. By Observation 2.2.3, we assume, w.l.o.g., that A_S and A_P do not use ε-transitions. We construct the vset-automaton

$$A_{S\circ P} \;:=\; \Big(\Sigma,\; V,\; Q_P \times (Q_S \cup \{\bot\}) \times \{1,2,3\},\; (q_{0,P}, \bot, 1),\; Q_{F,P} \times \{\bot\} \times \{3\},\; \delta\Big)\,.$$

[1]This is possible as the composition $S \circ P$ does not depend on the variable x_P. If $x_P \in \text{Vars}(A_S)$, we can therefore modify A_P to use a variable $x \notin \text{Vars}(A_S)$ instead. We observe that this obviously can be done in polynomial time.

The construction is similar to a product construction for the automata A_S, A_P, and a three state automaton that accepts the language $\Sigma^* \cdot x_P\{(\Sigma \cup \Gamma_V)^*\} \cdot \Sigma^*$. The main idea of the construction is simulation in three phases. In phase one, A_P runs. Whenever A_P can open its variable it is decided nondeterministically whether the simulation stays in phase one or continues with phase two. At the beginning of phase two, A_S is initialized with its start state and runs in parallel to the simulation of A_P. Whenever A_P allows to close its variable and A_S is in an accepting state, the simulation nondeterministically decides to stay in phase two or continue with phase three. In phase three, the simulation of A_P is finished. The simulation can end at every point in which A_P is in an accepting state. Thus, the transition function is defined by

$$\begin{aligned}
\delta := {} & \big\{\big((q,\bot,1),\sigma,(q',\bot,1)\big) \mid (q,\sigma,q') \in \delta_P, \sigma \in \Sigma\big\} \cup && A_P \text{ runs}\\
& \big\{\big((q,\bot,1),\varepsilon,(q',q_{0,S},2)\big) \mid (q,x_P\vdash,q') \in \delta_P\big\} \cup && A_S \text{ starts}\\
& \big\{\big((q,p,2),\sigma,(q',p',2)\big) \mid (q,\sigma,q') \in \delta_P, (p,\sigma,p') \in \delta_S, \sigma \in \Sigma\big\} \cup && A_P \text{ and } A_S \text{ run}\\
& \big\{\big((q,p,2),v,(q,p',2)\big) \mid (p,v,p') \in \delta_S, v \in \Gamma_V\big\} \cup && \text{variable operation of } A_S\\
& \big\{\big((q,p,2),\varepsilon,(q',\bot,3)\big) \mid (q,\dashv x_P,q') \in \delta_P, p \in Q_{F,S}\big\} \cup && A_S \text{ stops}\\
& \big\{\big((q,\bot,3),\sigma,(q',\bot,3)\big) \mid (q,\sigma,q') \in \delta_P, \sigma \in \Sigma\big\}. && A_S \text{ runs}
\end{aligned}$$

By construction, every run of $A_{S\circ P}$ on a valid ref-word $\mathrm{r} = \sigma_1 \cdots \sigma_n$ uses exactly two ε-transitions and is of the form

$$\begin{aligned}
&(q_0,\bot,1) \xrightarrow{\sigma_1} (q_1,\bot,1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{i-1}} (q_{i-1},\bot,1) \xrightarrow{\varepsilon} (q_i,p_0,2) \xrightarrow{\sigma_i}\\
&\quad \xrightarrow{\sigma_i} (q_{i+1},p_1,2) \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{j-1}} (q_j,p_{j-i},2) \xrightarrow{\varepsilon} (q_{j+1},\bot,3) \xrightarrow{\sigma_j} \cdots \xrightarrow{\sigma_n} (q_{n+2},\bot,3)
\end{aligned}$$

where

$$p_0 \xrightarrow{\sigma_i} p_1 \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{j-1}} p_{j-i}$$

is a run of A_S on $\sigma_i \cdots \sigma_{j-1}$ and

$$\begin{aligned}
&q_0 \xrightarrow{\sigma_1} q_1 \cdots q_{i-1} \xrightarrow{\sigma_{i-1}} q_{i-1} \xrightarrow{x_P\vdash} q_i \xrightarrow{\mathrm{doc}(\sigma_i)} \cdots\\
&\qquad \cdots \xrightarrow{\mathrm{doc}(\sigma_{j-1})} q_j \xrightarrow{\dashv x_P} q_{j+1} \xrightarrow{\sigma_j} q_{j+2} \cdots q_{n+1} \xrightarrow{\sigma_n} q_{n+2}
\end{aligned}$$

is a run of A_P on $d := \mathrm{doc}(\mathrm{r}) = \sigma_1 \cdots \sigma_{i-1} \cdot \mathrm{doc}(\sigma_i \cdots \sigma_{j-1}) \cdot \sigma_j \cdots \sigma_n$.[2] Furthermore, the span $[i,j'\rangle$ with $j' = i + |\,\mathrm{doc}(\sigma_i \cdots \sigma_{j-1})|$, which is defined by the positions of the ε-transitions in the run, is in $P(d)$ and covers $\mathrm{tup}(\mathrm{r})$.

We can therefore conclude that $[\![A_{S\circ P}]\!] = \pi_{\mathrm{Vars}(S)}((\Sigma^* \cdot x_P\{S\} \cdot \Sigma^*) \bowtie P)$ and $A_{S\circ P}$ is sequential if A_S is sequential.

It remains to show that $A_{S\circ P}$ is unambiguous if

- A_S and A_P are unambiguous, and
- $A_{S\circ P}$ and A_P satisfy the highlander condition.

[2] We note that by construction of $A_{S\circ P}$ the first component of the state does not change, when $\mathrm{doc}(\sigma_i) = \varepsilon$.

To this end, observe that:

1. a run of $A_{S \circ P}$ that witnesses the violation of the variable order condition (C2) of $A_{S \circ P}$ implies that there is a run of A_S that witnesses the violation of the condition for A_S;
2. two distinct runs of $A_{S \circ P}$ that violate unambiguity condition (C3) of $A_{S \circ P}$ must either
   - have ε-transitions at different positions and therefore witness the existence of two distinct spans in S that both cover tup(r), which violates the highlander condition, or
   - have ε-transitions at the same positions and therefore witness that either A_S has two distinct runs on r, violating the assumption that $A_S \in$ usVSA, or A_P has two distinct runs on the unique ref-word corresponding to the span indicated by the positions of the ε-transitions, violating the assumption that $A_P \in$ usVSA.

Altogether, this shows that $A_{S \circ P}$ being not unambiguous leads to a contradiction to the assumption that A_S and A_P are unambiguous and that $S \circ P$ and P satisfy the highlander condition. □

4.1.2 Containment of Regular Document Spanners

We now study the complexity of containment of regular document spanners. In particular, we show that containment of regex-formulas and vset-automata is PSPACE-complete (Corollary 4.1.3), even under some determinism assumptions introduced in past work [105] (Theorem 4.1.4), but it is solvable in PTIME for unambiguous and even in NL for deterministic vset-automata (Theorem 4.1.5).

Given two spanners $A_S, A_{S'} \in \mathcal{S}$ the containment problem asks whether $[\![A_S]\!](d) \subseteq [\![A'_S]\!](d)$ for every document d. As we will see later, deciding containment is essential for deciding many of the problems studied throughout this thesis.

	CONTAINMENT[$\mathcal{S}$]
Input:	Spanner $S, S' \in \mathcal{S}$.
Question:	Is $S \subseteq S'$?

The next theorem establishes the complexity of containment in the general case.

Theorem 4.1.2 (Maturana et al. [105, Theorem 6.4])**.** *Containment is* PSPACE*-hard for* fRGX *and* fVSA *and in* PSPACE *for* RGX *and* VSA.

Since we know from Figure 2.4 that fRGX $\subseteq$ sRGX $\subseteq$ RGX and fVSA $\subseteq$ sVSA $\subseteq$ VSA, we have the following corollary.

Corollary 4.1.3. *Containment of regex-formulas (*RGX, sRGX, fRGX*) and vset-automata (*VSA, fVSA, sVSA*) is* PSPACE*-complete.* □

We now consider containment of deterministic and weakly deterministic vset-automata. We first show that containment of weakly deterministic vset-automata is PSPACE-complete.[3] As we will see in the proof, the hardness of containment is due to the fact that multiple variable operations can occur without reading alphabet symbols and therefore, multiple different orderings of variable operations can be used to introduce nondeterministic choice.

Theorem 4.1.4. *Containment of weakly deterministic functional vset-automata is* PSPACE-*complete.*

Proof. The upper bound follows directly from Theorem 4.1.2. For the lower bound we reduce from the PSPACE complete problem of DFA union universality [84]. Given deterministic finite automata $A_1, \dots, A_n$ over the alphabet Σ, the union universality problem asks whether

$$\mathcal{L}(\Sigma^*) \subseteq \bigcup_{1 \le i \le n} \mathcal{L}(A_i) \,. \qquad (\dagger)$$

We construct vset-automata A, A' using the variable set $V = \{x_1, \dots, x_n\}$, such that $A(d) \subseteq A'(d)$ for all documents $d \in \Sigma^*$ if and only if ($\dagger$) holds. Let A accept the language defined by the regex-formula

$$\alpha_A \;:=\; x_1\Big\{x_2\{\cdots x_n\{\Sigma^*\}\cdots\}\Big\} \,,$$

selecting the whole document with every variable. Clearly, the regex-formula α_A can be represented by a weakly deterministic functional vset-automaton A. We now abuse notation and describe the language accepted by A' by a hybrid regex-formula

$$\alpha_{A'} \;:=\; x_1\{\alpha_1\} + \cdots + x_n\{\alpha_n\} \,,$$

where the DFAs A_i are plugged in. In particular,

$$\alpha_i \;:=\; x_1\Big\{\cdots x_{i-1}\{x_{i+1}\{\cdots\{x_n\{A_i\}\}\cdots\Big\} \,,$$

for $1 \le i \le n$. Term i in $\alpha_{A'}$ starts by first opening variable x_i, continues to open all other variables in increasing order, and finally selects the whole document d for every variable if $d \in \mathcal{L}(A_i)$. Clearly, as every term starts with a different variable symbol, this hybrid formula can be transformed into an equivalent weakly deterministic functional vset-automaton A' in linear time.

[3] We note that, assuming coNP $\neq$ PSPACE, this result contradicts Theorem 6.6 in Maturana et al. [105], where it is argued that containment for weakly deterministic sequential vset-automata is in coNP. There is an error in the upper bound of Maturana et al. [105], as can be seen in the version that includes the proofs [104]. The specific error is in the pumping argument for proving a polynomial size witness property for non-containment. The polynomial size witness property is not necessarily true, due to the nondeterminism entailed in the ability of the automaton to open variables in different orders. At every specific position in the string, the execution can be in $\Theta(n)$ possible states, where n is the number of states, implying that a minimal witness may require a length of $2^{\Theta(n)}$.

It remains to argue that $[\![A]\!](d) \subseteq [\![A']\!](d)$ for every document $d \in \Sigma^*$ if and only if (†) holds.

(if): Assume that $\mathcal{L}(\Sigma^*) \subseteq \bigcup_{1 \leq i \leq n} \mathcal{L}(A_i)$ holds. Let $d \in \Sigma^*$ be a document and $\mathrm{t} \in [\![A]\!](d)$ be a d-tuple. Per definition of A, we have $\mathrm{t}(v) = [1, |d| + 1\rangle$ for all variables $v \in V$. By assumption, there is an automaton A_i such that $d \in \mathcal{L}(A_i)$. Therefore, the tuple t is accepted by term i of A', thus $\mathrm{t} \in [\![A']\!](d)$.

(only if): Assume that $[\![A]\!](d) \subseteq [\![A']\!](d)$, for every document $d \in \Sigma^*$. Let $d \in \Sigma^*$ be an arbitrary document and $\mathrm{t} \in A(d)$. Per assumption, it follows that $\mathrm{t} \in [\![A']\!](d)$ and therefore there is a run of A' on d selecting t. Let x_i be the first variable which is opened in this run. Per construction of A' it follows, that $d \in \mathcal{L}(A_i)$. ☐

The question is now whether there exists a satisfactory notion of determinism for vset-automata that allows for efficient containment testing without loss of expressiveness. Our definitions of unambiguity and determinism resolves this complexity issue, without loss of expressiveness (cf. Proposition 2.2.6). Now, we can show that containment is tractable for deterministic and unambiguous vset-automata.

Theorem 4.1.5. *Containment for* usVSA *is in* PTIME *and containment for* dsVSA *is in* NL*.*

Proof. As we will see next, the NL upper bound for dsVSA follows from containment of deterministic finite state automata. The PTIME upper bound for usVSA follows from containment of unambiguous finite state automata. In the following, we only give the proof for dsVSA. The proof for usVSA is analogous (using the fact that containment for unambiguous finite automata is in PTIME [157, Corollary 4.7]).

To this end, let A_1, A_2 be deterministic sequential vset-automata (i.e., $A_1, A_2 \in$ dsVSA). By Lemma 2.2.2, $[\![A_1]\!] \subseteq [\![A_2]\!]$ if and only if $\mathcal{R}(A_1) \subseteq \mathcal{R}(A_2)$. Let $i \in \{1, 2\}$. Observe that due to A_i being weakly deterministic, it must hold that A_i, interpreted as ε-NFA, is deterministic. The result follows since containment for deterministic finite automata is well known to be in NL. ☐

4.1.3 Complexity of Checking Cover and Highlander Condition

Towards tractability results, we first show that the emptiness problem of sequential vset-automata is decidable in NL.

Proposition 4.1.6 (Maturana et al. [105, Theorem 6.2])**.** *Given a sequential vset-automaton A, it can be checked in* NL *whether $[\![A]\!](d) \neq \emptyset$ for some document d.*

Proof. Let $A \in$ sVSA. Due to A being sequential, all ref-words $\mathrm{r} \in \mathcal{R}(A)$ must be valid. Thus, $[\![A]\!](d) \neq \emptyset$ if and only if $\mathcal{R}(A) \neq \emptyset$. The result follows from the fact that emptiness of ε-NFAs can be checked in NL.[4] ☐

[4]Emptiness of ε-NFAs is the same problem as Reachability in graphs, which is well known to be NL-complete (cf. Papadimitriou [120, Theorem 16.2]).

The next proposition shows that deciding Proper, Disjoint, and Highlander are tractable if spanner and splitter are sequential.

Proposition 4.1.7. *Proper*[sVSA], *Disjoint*[sVSA], *and Highlander*[sVSA] *are in* NL*. Furthermore, Proper*[sRGX], *Disjoint*[sRGX], *and Highlander*[sRGX] *are in* PTIME*.*

Proof. For every regex-formula an equivalent vset-automaton can be constructed in polynomial time using the usual constructions that convert a regular expression into an NFA. Thus it suffices to show that the problems are in NL for sVSA.

Let $A_S, A_P \in$ sVSA be automata representing a spanner and a splitter, respectively. Let $S = [\![A_S]\!]$ and $P = [\![A_P]\!]$. We denote the variables of A_S by V, the single variable of A_P with x, and a fresh variable not used by A_S or A_P by y. We provide logspace constructions for sVSAs A_{prop}, A_{disjoint}, and $A_{\text{highlander}}$, such that the ref-word languages of the automata are empty if and only if S is proper, P is disjoint, and S and P satisfy the highlander condition, respectively. The result follows, as emptiness of sVSA can be checked in NL (cf. Proposition 4.1.6).

To ease readability, we abbreviate $\text{tup}(\text{r})(x)$ by $\text{tup}_\text{r}(x)$ in the remainder of this proof.

A_{proper}: The automaton A_{proper} is the intersection of A_S and an automaton A' such that $\mathcal{R}(A') = \Sigma^* \cdot (\Gamma_V)^* \cdot \Sigma^*$.

The automaton A_{proper} is sequential, since $\mathcal{R}(A_{\text{proper}}) \subseteq \mathcal{R}(A_S)$ and A_S is sequential. Thus, all ref-words $\text{r} \in \mathcal{R}(A_{\text{proper}})$ are valid. Assume that $\text{r} \in \mathcal{R}(A_{\text{proper}})$. Let $d := \text{doc}(\text{r})$. Observe that $\text{tup}(\text{r}) \in S(d)$, due to $\mathcal{R}(A_{\text{proper}}) \subseteq \mathcal{R}(A_S)$. Furthermore, due to $\mathcal{R}(A_{\text{proper}}) \subseteq \mathcal{R}(A') = \Sigma^* \cdot (\Gamma_V)^* \cdot \Sigma^*$ it must hold that $\text{tup}(\text{r})$ is either empty or the minimal span covering it is empty. In both cases $\text{tup}(\text{r})$ is a witness that A_S is not proper. For the other direction, assume there is a document d and a tuple $\text{t} \in S(d)$ such that t is empty or the minimal span covering t is empty, then the ref-word $\text{r} \in \mathcal{R}(A_S)$ with $\text{tup}(\text{r}) = \text{t}$ is in $\mathcal{R}(A')$ and therefore in $\mathcal{R}(A_{\text{proper}})$.

A_{disjoint}: We define the automaton A_{disjoint} as the intersection of the following four automata such that $\text{t} \in [\![A_{\text{disjoint}}]\!](d)$ is a tuple over variables x, y. The automaton A_P^x (resp., A_P^y) selects all (x, y) pairs such that $\text{t}(x) \in P(d)$ (resp., $\text{t}(y) \in P(d)$) for a document $d \in \Sigma^*$. The automaton A_{distinct} verifies whether $\text{t}(x) \neq \text{t}(y)$ and A_{overlap} verifies whether $\text{t}(x)$ and $\text{t}(y)$ overlap. More formally, we define the automata as follows:

- A_P^x is derived from A_P by adding self-loops for every label from $\Gamma_{\{y\}} = \{y\vdash, \dashv y\}$ to every state.
- A_P^y is derived from A_P by changing every label $x\vdash$ to $y\vdash$, every label $\dashv x$ to $\dashv y$, and afterwards adding self loops for every label from $\Gamma_{\{x\}} = \{x\vdash, \dashv x\}$ to every state.
- A_{distinct} ensures that $\text{tup}_\text{r}(x) \neq \text{tup}_\text{r}(y)$ for every ref-word $\text{r} \in \mathcal{R}(A_{\text{distinct}})$.[5] This automaton is depicted in Figure 4.1.

[5]Note that A_{distinct} does not select all ref-words with $\text{tup}_\text{r}(x) \neq \text{tup}_\text{r}(y)$, as it does not consider cases where one variables is opened and closed before the other variable is opened.

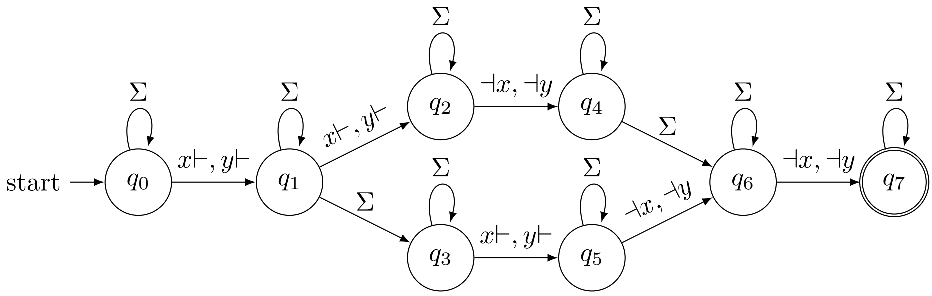

Figure 4.1: The automaton A_{distinct}.

- A_{overlap} is a three state automaton with $\mathcal{R}(A_{\text{overlap}}) := (\Sigma \cup \{x\vdash, y\vdash\})^* \cdot \Sigma \cdot (\Sigma \cup \{\dashv x, \dashv y\})^*$ that ensures that at least one symbol is read while both variables are open. We note that A_P^x and A_P^y already ensure that both variables are used.

The constructions of all automata can be carried out by an logarithmic space Turing Machine. Note that A_{disjoint} is sequential, since A_P^x, (resp., A_P^y) ensures that x (resp., y) is not opened or closed several times and that it is closed if and only if it is opened. A ref-word $\mathrm{r} \in \mathcal{R}(A_{\text{disjoint}})$ witnesses that P is not disjoint, since A_P^x verifies that $\text{tup}_\text{r}(x) \in P(\text{doc(r)})$, A_P^y verifies that $\text{tup}_\text{r}(y) \in P(\text{doc(r)})$, A_{distinct} verifies that $\text{tup}_\text{r}(x) \neq \text{tup}_\text{r}(y)$, and A_{overlap} verifies that $\text{tup}_\text{r}(x)$ and $\text{tup}_\text{r}(y)$ overlap. For the other direction, a document d where $P(d)$ has two overlapping spans s_1 and s_2 ensures that $\mathrm{r} \in \mathcal{R}(A_{\text{disjoint}})$, where r is derived from d by inserting opening and closing operations for x and y at the positions indicated by s_1 and s_2.

$A_{\text{highlander}}$: We define the automaton $A_{\text{highlander}}$ as the intersection of the following automata, such that $\text{Vars}(A_{\text{highlander}}) = V \cup \{x, y\}$. A_P^x (and A_P^y) again ensures that $\mathrm{t} \in [\![A_{\text{highlander}}]\!](d)$ implies that $\mathrm{t}(x) \in P(d)$ (resp., $\mathrm{t}(y) \in P(d)$). Following the same idea, A'_S ensures that $\pi_V(\mathrm{t}) \in S(d)$. The automaton A_{distinct} ensures that $\mathrm{t}(x) \neq \mathrm{t}(y)$ if $\mathrm{t} \in [\![A_{\text{highlander}}]\!](d)$. The last automaton, A_{enclosed} ensures that both, $\mathrm{t}(x)$ and $\mathrm{t}(y)$, contain $\pi_V(\mathrm{t})$. More formally:

- A_P^x, A_P^y, and A_{distinct} are as before but with additional self-loops for every symbol from Γ_V at each state.
- A'_S is derived from A_S by adding self-loops for every label from $\Gamma_{\{x,y\}}$ to every state.
- A_{enclosed} is an automaton with $\mathcal{R}(A_{\text{enclosed}}) = (\Sigma \cup \{x\vdash, y\vdash\})^* \cdot (\Sigma \cup \Gamma_V)^* \cdot (\Sigma \cup \{\dashv x, \dashv y\})^*$ that ensures that no variable operation for variables from V is used outside of the spans defined by x and y.

We compute $A_{\text{highlander}}$ as the intersection of A'_S, A_P^x, A_P^y, A_{distinct}, and A_{enclosed}. We note that even if the five automata are not sequential, the automaton $A_{\text{highlander}}$ is sequential. For every variable, one of the automata A'_S, A^x_P, and A^y_P ensures that it is not opened or closed several times and that it is closed if and only if it is opened.

We explain why a ref-word $\mathrm{r} \in \mathcal{R}(A_{\mathrm{highlander}})$ witnesses a violation of the highlander condition. By the construction of A_{enclosed}, the (different) spans $\mathrm{tup}_{\mathrm{r}}(x)$ and $\mathrm{tup}_{\mathrm{r}}(y)$ are both in $P(\mathrm{doc}(r))$ and both cover $\pi_V(\mathrm{tup}_{\mathrm{r}}) \in S(\mathrm{doc}(r))$. For the other direction, a document d, spans $s_1, s_2 \in P(d)$, and a tuple $\mathrm{t} \in S(d)$ witnessing the violation of the highlander condition ensure that

$$\mathrm{ref}(d, \mathrm{t} \cup \{x \mapsto s_1, y \mapsto s_2\}) \in \mathcal{R}(A_{\mathrm{highlander}}) .$$

Therefore, the language is not empty. □

We proceed by studying the complexity of testing the cover condition. Here, we only give an upper bound, a matching lower bound is established in Lemma 4.4.1.

Proposition 4.1.8. *Cover*[VSA] *is in* PSPACE.

Proof. Let S be a spanner and P be a splitter, given as $A_S, A_P \in \mathrm{VSA}$. We assume, w.l.o.g., that $x_S \notin V$. We define a spanner $A_V \in \mathrm{VSA}$ that selects every possible tuple. More formally, $A_V := (\Sigma, V, \{q_0\}, q_0, \{q_0\}, \delta)$ is the vset-automaton with a single state q_0, where $\delta := \{(q_0, c, q_0) \mid c \in \Sigma \cup \Gamma_V\}$. We argue next that P covers S if and only if $S \subseteq [\![A_V]\!] \circ P$.

(if): Assume that the cover condition does not hold. Then there is a document $d \in \Sigma^*$ and a non-empty tuple $\mathrm{t} \in S(d)$, such that there is no span $s \in P(d)$ which covers t. Even though A_V selects every possible tuple, we have $\mathrm{t} \notin ([\![A_V]\!] \circ P)(d)$.

(only if): Assume that the cover condition holds. Let $d \in \Sigma^*$ be a document and $\mathrm{t} \in S(d)$ be a non-empty d-tuple. Since P covers S, there is a span $s \in P(d)$ which covers t. Thus, per definition of A_V, it must hold that $\mathrm{t} \ll s \in [\![A_V]\!](d)$, and therefore $S \subseteq [\![A_V]\!] \circ P$ also holds.

The PSPACE upper bound follows from Proposition 4.1.1 (first bullet point), which shows that a vset-automaton $A \in \mathrm{VSA}$ with $[\![A]\!] = [\![A_V]\!] \circ P$ can be constructed in polynomial time, and Theorem 4.1.2 which states that containment of vset-automata is in PSPACE. □

4.2 Deciding Split-Correctness and Self-Splittability

In this section, we show that Split-Correctness and Self-Splittability are in PSPACE for regex-formulas and vset-automata, while both problems are in PTIME if S, S_P, and P are given as ufVSA and S and P satisfy the highlander condition.

It follows directly from Proposition 4.1.1 and Theorem 4.1.5 that split-correctness is decidable in PTIME when the highlander condition is satisfied and the vset-automata are unambiguous and sequential.

Lemma 4.2.1. *Deciding* Split-Correctness[VSA] *is in* PSPACE. *Furthermore, if S and P satisfy the highlander condition,* Split-Correctness[usVSA] *is in* PTIME.

Proof. Let $A_S, A_{S_P}, A_P \in$ VSA with $S = [\![A_S]\!], P = [\![A_P]\!]$, and $S_P = [\![A_{S_P}]\!]$. Furthermore, let $A_{S_P \circ P}$ be as constructed in Proposition 4.1.1, that is, $[\![A_{S_P \circ P}]\!] = S_P \circ P$. Thus, S is splittable by P via S_P if and only if $[\![A_S]\!] = [\![A_{S_P \circ P}]\!]$. It follows from Theorem 4.1.2 that this equivalence can be checked in PSPACE.

Assume that $A_S, A_{S_P}, A_P \in$ ufVSA and that S and P satisfy the highlander condition. By Proposition 4.1.1 (second bullet point), $A_{S_P \circ P} \in$ sVSA. We begin by checking whether $A_{S_P \circ P}$ and A_P satisfy the highlander condition, which can be done in PTIME due to Proposition 4.1.7. If this is the case, we can conclude that $A_{S_P \circ P} \in$ usVSA (Proposition 4.1.1, third bullet point) and therefore it can be checked in PTIME whether S is splittable by P via S_P as shown in Theorem 4.1.5. Otherwise, if $S_P \circ P$ and P do not satisfy the highlander condition, there must be a document $d \in \Sigma^*$ and a tuple $\mathrm{t} \in (S_P \circ P)(d)$ such that at least two splits $s, s' \in P(d)$ cover t. Therefore, due to S and P satisfying the highlander condition, it must hold that $\mathrm{t} \notin S(d)$, which implies that S is not splittable by P via S_P. □

The following corollary follows directly from Lemma 3.1.6 and Lemma 4.2.1.

Corollary 4.2.2. Split-Correctness[usVSA] *is in* PTIME *if S is proper and P is disjoint.* □

We observe that, by the definition of Self-Splittability, the following corollary follows directly.

Corollary 4.2.3. *Deciding* Self-Splittability[VSA] *is in* PSPACE. *Furthermore,* Self-Splittability[usVSA] *is in* PTIME *if either*

1. *S and P satisfy the highlander condition, or*
2. *S is proper and P is disjoint.* □

4.3 Deciding Splittability

We will now study the Splittability problem. Recall the definition of the canonical split-spanner:

$$S_P^{\mathrm{can}}(d) := \left\{\mathrm{t} \mid \forall d' \in \Sigma^*, \forall s \in P(d') \text{ such that } d'_s = d, \text{ it holds that } (\mathrm{t} \gg s) \in S(d')\right\} .$$

We begin by showing that S_P^{can} is regular if S and P are regular (Section 4.3.1). In Sections 4.3.2 and 4.3.3 we show the upper bounds for Splittability in the general case and in the presence of the highlander condition. We conclude this section by proving a key technical lemma in Section 4.3.4.

4.3.1 Constructing the Canonical Split-Spanner

In this section, we will show that S_P^{can} is regular if S and P are regular (Corollary 4.3.5). To this end, define a finite monoid M such that S_P^{can} is exactly the spanner represented by the language recognized by M.

A *monoid* is a triple $(M, \odot, e)$ consisting of a set M, an associative binary operation $\odot\colon M \times M \to M$, and a neutral element e. We say that a monoid M *recognizes* a language $\mathcal{L}$ over the alphabet Ξ if there is a homomorphism $h\colon \Xi^* \to M$ and a set $M^{\text{acc}} \subseteq M$ such that $w \in \mathcal{L}$ if and only if $h(w) \in M^{\text{acc}}$. A function h is a (string) homomorphism if and only if $h(\varepsilon) = e$ and $h(w_1 \cdot w_2) = h(w_1) \odot h(w_2)$ for all strings $w_1, w_2 \in \Xi^*$. It is well known that a language $\mathcal{L}$ is regular if and only if it is recognized by a finite monoid M. All monoids that we define will be finite.

Given a VSA $A = (\Sigma, V, Q, q_0, Q_F, \delta)$, the *transition monoid* M_A of A is $(2^{Q\times Q}, \odot, \mathrm{id}_Q)$, where $2^{Q\times Q}$ is the set of all possible binary relations over Q, the operation $\odot$ is the composition of relations, i.e.,

$$m_1 \odot m_2 \ := \ \big\{(x,z) \mid \exists y \in Q, \text{ such that } (x,y) \in m_1 \text{ and } (y,z) \in m_2\big\} \ ,$$

and $\mathrm{id}_q := \{(q,q) \mid q \in Q\}$ is the identity relation over Q. The canonical homomorphism h_A for the transition monoid is defined by

$$h_A(\mathrm{r}) \ := \ \big\{(p,q) \mid q \in \delta^*(p,\mathrm{r})\big\} \ .$$

For reasons that become apparent later, we define $h_A(\alpha) = \mathrm{id}_Q$ for every variable operation $\alpha \in \Gamma_{\mathrm{Vars} \setminus \mathrm{Vars}(A)}$ that does not belong to a variable used by A. This has the effect that h_A ignores all "foreign" variables, which is helpful when combining the transition monoids of different spanners.

Lemma 4.3.1. *Let $A \in \mathrm{VSA}$. The language $\mathcal{R}(A)$ is recognized by M_A.*

Proof. Let $\mathrm{r} \in (\Sigma \cup \Gamma_{\mathrm{Vars}(A)})^*$ be a ref-word over the alphabet $\Sigma \cup \Gamma_{\mathrm{Vars}(A)}$. Furthermore, let $M_A^{\text{acc}} := \{m \mid m \cap (\{q_0\} \times Q_F) \neq \emptyset\}$ be the set of accepting monoid elements. As $\mathrm{r} \in \mathcal{R}(A)$ if and only if there is an accepting run of A on r and by the definition of h_A, we get that $\mathrm{r} \in \mathcal{R}(A)$ if and only if $h_A(\mathrm{r}) \in M_A^{\text{acc}}$, concluding the proof. ☐

As we will show, given a spanner S, one can also construct a monoid that recognizes the language of all valid ref-words, satisfying the variable order condition, which correspond to a tuple selected by S. More formally, we define the language $\mathcal{R}^S$, where S is a document spanner:

$$\mathcal{R}^S \ := \ \big\{\mathrm{ref}(d,\mathrm{t}) \mid \exists d \in \Sigma^*, \text{ such that } \mathrm{t} \in S(d)\big\} \ .$$

Observe that $\mathcal{R}^S = \mathcal{R}(A)$ if S is given as a sequential vset-automaton A which satisfies the variable order condition. We generalize this and show that, for every document spanner S given by a vset-automaton A, there is a monoid M of size exponential in A which recognizes $\mathcal{R}^S$. Furthermore, as we will show this monoid can be constructed by a polynomial space Turing Machine.[6]

[6] We note that the polynomial space bound of the Turing Machine only refers to the working tapes, but not to the output tape, to which the exponential size monoid is written.

Lemma 4.3.2. *Let $A \in \mathrm{VSA}$. There is a monoid $M_A^{\prec}$ of exponential size that recognizes $\mathcal{R}^{[\![A]\!]}$. Furthermore, $M_A^{\prec}$ can be constructed by a polynomial space Turing Machine.*

We note that if A is sequential and satisfies the variable order condition, then $\mathcal{R}^{[\![A]\!]} = \mathcal{R}(A)$ and the transition monoid M_A of A can be used for $M_A^{\prec}$. In the general case, the construction of $M_A^{\prec}$ is quite involved. To meet the exponential size restriction it is not possible to compute an equivalent sequential vset-automaton that complies with the variable order condition. Instead, sequentiality and the variable order condition have to be dealt with in the monoid construction itself. We give a proof for Lemma 4.3.2 in Section 4.3.4.

Given a set V of variables, we define the monoid M_V that can test whether a ref-word (using variables from V) satisfies the variable order condition:

$$M_V \;:=\; \left(2^{\Gamma_V} \cup \{0\}, \odot_V, \emptyset\right) \;;$$
$$X \odot_V Y \;:=\; \begin{cases} X \cup Y & \text{if } X \cap Y = \emptyset \text{ and } x \prec y \text{ for all } x \in X, y \in Y \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 4.3.3. *For every finite set $V \subseteq \mathrm{Vars}$ of variables, M_V recognizes the set $\mathcal{R}^V$ of all valid ref-words over V which satisfy the variable order condition.*

Proof. Let $M_V^{\mathrm{acc}} = \{X \neq 0 \mid \forall v \in V, \text{it holds that}\, v\vdash \in X \Leftrightarrow \dashv v \in X\}$ and $h_V \colon (\Sigma \cup V)^* \to M_V$ be the homomorphism induced by

$$h_V(a) \;:=\; \begin{cases} a & \text{if } a \in \Gamma_V \\ \emptyset & \text{otherwise.} \end{cases}$$

It remains to show that $\mathrm{r} \in (\Sigma \cup \Gamma_V)^*$ is valid and satisfies the variable order condition if and only if $h_V(\mathrm{r}) \in M_V^{\mathrm{acc}}$. Let $h_V(\mathrm{r}) \in M_V^{\mathrm{acc}}$. Observe that, per definition of $\odot_V$, r must satisfy the variable order condition. Furthermore, per definition of $\prec$, it must hold that $v\vdash \prec \dashv v$ for all variables $v \in \mathrm{Vars}$. Thus, r must be valid, as all variables $v \in \mathrm{Vars}(\mathrm{r})$ must be opened and closed exactly once and opened before they are closed. For the other direction, assume that r is valid and satisfies the variable order condition. It is straightforward to verify that $h_V(\mathrm{r}) \neq 0$ and furthermore, $h_V(\mathrm{r}) \in M_V^{\mathrm{acc}}$. □

Let S be a regular document spanner, P be a regular document splitter, and $V = \mathrm{Vars}(S)$. We use Lemma 4.3.2 and show that the Cartesian product of the monoids M_V, $M_S^{\prec}$, and $M_P^{\prec}$ contains enough structure to recognize $\mathcal{R}^{S_P^{\mathrm{can}}}$. Therefore, S_P^{can} is indeed a regular document spanner.

Proposition 4.3.4. *For every regular document spanner S and every regular document splitter P, the monoid $M := M_V \times M_S^{\prec} \times M_P^{\prec}$ recognizes $\mathcal{R}^{S_P^{\mathrm{can}}}$.*

Proof. Let $h \colon (\Sigma \cup \Gamma_V)^* \to M$ be the homomorphism defined by

$$h(\mathrm{r}) := (h_V(\mathrm{r}), h_S(\mathrm{r}), h_P(\mathrm{r}))\;.$$

We define M^{acc} as

$$M^{\text{acc}} := \Big\{(m_V, m_S, m_P) \mid m_V \in M_V^{\text{acc}} \text{ and for all } d_1, d_2 \in \Sigma^* \text{ it holds that } m_P' \in M_P^{\text{acc}} \Rightarrow m_S' \in M_S^{\text{acc}}, \text{where } (m_V', m_S', m_P') = h(d_1) \odot h(x_P\vdash) \odot (m_V, m_S, m_P) \odot h(\dashv x_P) \odot h(d_2)\Big\}.$$

Recall that $M_V^{\text{acc}} = \{X \neq 0 \mid \forall v \in V, \text{it holds that}\, v\vdash \in X \Leftrightarrow \dashv v \in X\}$. Furthermore, for a ref-word r, it holds that $h(\mathrm{r}) \in M_S^{\text{acc}}$ (resp., $h(\mathrm{r}) \in M_P^{\text{acc}}$) if and only if $\mathrm{r} \in \mathcal{R}^S$ (resp., $\mathrm{r} \in \mathcal{R}^P$). We have to show that, for every ref-word r, it holds that $\mathrm{r} \in \mathcal{R}^{S_P^{\text{can}}}$ if and only if $h(\mathrm{r}) \in M^{\text{acc}}$.

(if): Let r be a ref-word and $d = \text{doc}(\mathrm{r})$. Assume that $h(\mathrm{r}) \in M^{\text{acc}}$. By definition of M_V and the fact that $h_V(r) \in M_V^{\text{acc}}$, we can conclude that r is valid and satisfies the variable order condition. It remains to show that $\text{tup}(\mathrm{r}) \in S_P^{\text{can}}(d)$, which implies that $\mathrm{r} \in \mathcal{R}^{S_P^{\text{can}}}$. To this end, let $d' \in \Sigma^*$ and $s \in P(d')$ such that $d'_s = d$. If no such d' and s exist, it follows that $\text{tup}(\mathrm{r}) \in S_P^{\text{can}}(d)$ and we are done. Otherwise, d' can be decomposed as $d' = d_1 \cdot d \cdot d_2$. Let $(m_V', m_S', m_P') := h(d_1) \odot h(x_P\vdash) \odot (m_V, m_S, m_P) \odot h(\dashv x_P) \odot h(d_2)$. By definition of h and M_P^{acc}, we have that $m_P' = h_P(d_1) \odot h_P(x_P\vdash) \odot h_P(\mathrm{r}) \odot h_P(\dashv x_P) \odot h_P(d_2) \in M_P^{\text{acc}}$. Let $\mathrm{r}' = d_1 \cdot \mathrm{r} \cdot d_2$. Thus, $\text{tup}(\mathrm{r}') = \text{tup}(\mathrm{r}) \gg s \in S(d')$ if and only if $h(\mathrm{r}') \in M_S^{\text{acc}}$. Furthermore, due to h_S ignoring x_P, $h_S(\mathrm{r}') \in M_S^{\text{acc}}$ if and only if $m_S' \in M_S^{\text{acc}}$. As $m_P' \in M_P^{\text{acc}}$, we have by the definition of M^{acc} that $m_S' \in M_S^{\text{acc}}$ and therefore $h_S(\mathrm{r}') \in M_S^{\text{acc}}$. This implies that $\text{tup}(\mathrm{r}') \in S(d')$ and therefore $\text{tup}(\mathrm{r}) \in S_P^{\text{can}}(d)$, concluding the if-part of the proof.

(only if): Let $\mathrm{r} \in \mathcal{R}^{S_P^{\text{can}}}$ and $d = \text{doc}(\mathrm{r})$. Thus, $\text{tup}(\mathrm{r}) \in S_P^{\text{can}}(d)$ and r is valid and satisfies the variable order condition. We show that $m = (m_V, m_S, m_P) = h(\mathrm{r}) \in M^{\text{acc}}$. As r is valid and satisfies the variable order condition, we have that $m_V \in M_V^{\text{acc}}$. It remains to show that for every $d_1, d_2 \in \Sigma^*$ it holds that $m_P' \in M_P^{\text{acc}}$ implies that $m_S' \in M_S^{\text{acc}}$, where $(m_V', m_S', m_P') = h(d_1) \odot h(x_P\vdash) \odot (m_V, m_S, m_P) \odot h(\dashv x_P) \odot h(d_2)$. To this end let $d_1, d_2 \in \Sigma^*$ be arbitrary documents and m_S' and m_P' be as above with $m_P' \in M_P^{\text{acc}}$. We have to show that $m_S' \in M_S^{\text{acc}}$. Let $\mathrm{r}' = d_1 \cdot x_P\vdash \cdot d \cdot \dashv x_P \cdot d_2$ and recall that $\mathrm{r} \in \mathcal{R}^P$ if and only if $h_P(\mathrm{r}) \in M_P^{\text{acc}}$. Observe that $h_P(\mathrm{r}') = m_P' \in M_P^{\text{acc}}$ and thus $\mathrm{r}' \in \mathcal{R}^P$. Let $d' = \text{doc}(\mathrm{r}')$. Thus, $s = [|d_1|, |d_1 \cdot d|\rangle \in P(d')$ and $d'_s = d$. This implies that $\text{tup}(\mathrm{r}) \gg s \in S(d)$ and $\text{ref}(d, \text{tup}(\mathrm{r}) \gg s) \in \mathcal{R}^S$. Observe that $\text{ref}(d, \text{tup}(\mathrm{r}) \gg s) = \mathrm{r}'$ and therefore it follows that $m_S' = h_S(\text{ref}(d, \mathrm{r}')) \in M_S^{\text{acc}}$, concluding the proof. ☐

Corollary 4.3.5. S_P^{can} *is a regular document spanner.* ☐

4.3.2 Complexity Upper Bound for Splittability in the General Case

The proof of the upper bound consists of two parts. We first show that testing whether an element $m \in M$ belongs to M^{acc} is in PSPACE (Proposition 4.3.6) and then give an EXPSPACE algorithm for testing splittability (Theorem 4.3.7).

Proposition 4.3.6. *Let $m \in M$ be a monoid element. It can be tested in* PSPACE *whether $m \in M^{\mathrm{acc}}$.*

Proof. Recall that

$$M^{\mathrm{acc}} := \Big\{ (m_V, m_S, m_P) \mid m_V \in M_V^{\mathrm{acc}} \text{ and for all } d_1, d_2 \in \Sigma^* \text{ it holds that } m'_P \in M_P^{\mathrm{acc}} \Rightarrow m'_S \in M_S^{\mathrm{acc}}, \text{where } (m'_V, m'_S, m'_P) = h(d_1) \odot h(x_P \vdash) \odot (m_V, m_S, m_P) \odot h(\dashv x_P) \odot h(d_2) \Big\}.$$

We give a PSPACE algorithm which decides whether $(m_V, m_S, m_P) \notin M^{\mathrm{acc}}$ by guessing a counterexample.[7] By definition of M^{acc}, $(m_V, m_S, m_P) \notin M^{\mathrm{acc}}$ if and only if

1. $m_V \notin M_V^{\mathrm{acc}}$; or
2. there are $d_1, d_2 \in \Sigma^*$ with
   $h(d_1) \odot h(x_P \vdash) \odot m \odot h(\dashv x_P) \odot h(d_2) \in M_V \times (M_S \setminus M_S^{\mathrm{acc}}) \times M_P^{\mathrm{acc}}$.

Recall that $M_V^{\mathrm{acc}} = \{X \neq 0 \mid \forall v \in V, v\vdash \in X \Leftrightarrow \dashv v \in X\}$. Thus, the first condition can be checked in PTIME. Due to h being a homomorphism, it must hold that $h(\sigma_1 \cdots \sigma_n) = h(\sigma_1) \odot \cdots \odot h(\sigma_n)$. Therefore, the second condition can be checked by guessing d_1 and d_2 symbol by symbol and computing $h(d_1)$ and $h(d_2)$ on the fly. More formally, the algorithm does not store the possibly large documents d_1 and d_2, but only stores the monoid elements $h(d_1)$ and $h(d_2)$, and the size of d_1 and d_2, encoded in binary. The algorithm rejects if no counterexample of size at most exponential in $|m|$ is found. Note that the existence of a counterexample d_1, d_2 of more than exponential length also implies the existance of an exponential length counter example, as the total number of monoid elements is exponential in $|m|$. Therefore, by the pigeonhole principle, the described algorithm would store the same pair of monoid elements $(h(d_1), h(d_2))$ at least once while guessing an counterexample of more than exponential length. ☐

We are now ready to given an upper bound for Splittability.

Theorem 4.3.7. *Splittability$[\mathcal{S}]$ is in* EXPSPACE*.*

Proof. Let $S \in \mathcal{S}$ and $P \in \mathcal{S}$ be a spanner and a splitter. By Theorem 3.2.4, S is splittable by P if and only if S is splittable by P by S_P^{can}. The high level idea of the proof is to compute a vset-automaton A for $S_P^{\mathrm{can}} \circ P$ and then test equivalence with S.

Recall that $|M|$ is exponential in the size of A_S and A_P (cf. Lemma 4.3.2). To exploit the construction of Proposition 4.1.1, we turn M into the vset-automaton $A_M = (\Sigma, V, M, h(\varepsilon), M^{\mathrm{acc}}, \delta)$, where the transition function is defined by $\delta(m, \sigma) = m \odot h(\sigma)$. We use the monoid elements as states of the automaton. From the construction and definition of M it is obvious that $[\![A]\!] = S_P^{\mathrm{can}}$ and that A_M is linear in the size of M. By Proposition 4.3.6, M^{acc} can be constructed in PSPACE. Now we apply Proposition 4.1.1 to obtain an automaton A for $S_P^{\mathrm{can}} \circ P$, which is of polynomial size in M and thus

[7] Recall that PSPACE is closed under complement.

exponential in the size of P and S. Testing equivalence of S and A can be done in space polynomial in S and A. As A is of exponential size, this yields the EXPSPACE bound claimed in the theorem statement. □

4.3.3 Complexity Upper Bound for Splittability under the Highlander Condition

In this section we will show that the upper bound of splittability can be improved to PSPACE if the spanner and the splitter satisfy the highlander condition. We begin by characterizing counterexamples to splittability under the highlander condition.

Lemma 4.3.8. *Let S and P be a spanner and a splitter such that the highlander and cover conditions are satisfied. Then S is splittable by P if and only if there is no ref-word $\mathrm{r} = d_1 \cdot x_P\vdash \cdot \mathrm{r}' \cdot \dashv x_P \cdot d_2 \in (\Sigma \cup \Gamma_{\mathrm{Vars}(S)} \cup \Gamma_{\mathrm{Vars}(P)})^*$ such that*

- $d_1, d_2 \in \Sigma^*$;
- $d_1 \cdot \mathrm{r}' \cdot d_2 \in \mathcal{R}^S$;
- $d_1 \cdot x_P\vdash \cdot \mathrm{doc}(\mathrm{r}') \cdot \dashv x_P \cdot d_2 \in \mathcal{R}^P$; *and*
- $\mathrm{r}' \notin \mathcal{R}^{S_P^{\mathrm{can}}}$.

Proof. Assume that S is not splittable by P. By Lemma 3.2.3 and Theorem 3.2.4, there must be a document d and a tuple $\mathrm{t} \in S(d) \setminus (S_P^{\mathrm{can}} \circ P)(d)$. Due to $\mathrm{t} \in S(d)$, it holds that $\mathrm{ref}(d, \mathrm{t}) \in \mathcal{R}^S$ and, due to the cover condition, there must be a span $[i, j\rangle \in P(d)$ which covers t. Let $d_1, d_2 \in \Sigma^*$ and r' be a ref-word, such that $d_1 \cdot \mathrm{r}' \cdot d_2 = \mathrm{ref}(d, \mathrm{t})$, $i = |d_1| + 1$, and $j = |d_1 \cdot \mathrm{doc}(\mathrm{r}')| + 1$. Furthermore, let $\mathrm{r} = d_1 \cdot x_P\vdash \cdot \mathrm{r}' \cdot \dashv x_P \cdot d_2$, thus $\mathrm{doc}(\mathrm{r}) = d$ and $d_{[i,j\rangle} = \mathrm{doc}(\mathrm{r}')$. Due to $i = |d_1| + 1$ and $j = |d_1 \cdot \mathrm{doc}(\mathrm{r}')| + 1$, it follows that $d_1 \cdot x_P\vdash \cdot \mathrm{doc}(\mathrm{r}') \cdot \dashv x_P \cdot d_2 \in \mathcal{R}^P$. Therefore r satisfies the first three conditions of the lemma statement. Assume, towards a contradiction, that $\mathrm{r}' \in \mathcal{R}^{S_P^{\mathrm{can}}}$. This implies that $\mathrm{t} \in (S_P^{\mathrm{can}} \circ P)(d)$, which is a contradiction to the assumption that $\mathrm{t} \in S(d) \setminus (S_P^{\mathrm{can}} \circ P)(d)$ showing that r also satisfies the last condition given in the lemma statement.

Conversely assume that there is a string $\mathrm{r} = d_1 \cdot x_P\vdash \cdot \mathrm{r}' \cdot \dashv x_P \cdot d_2 \in (\Sigma \cup \Gamma_{\mathrm{Vars}(S)} \cup \Gamma_{\mathrm{Vars}(P)})^*$ satisfying the conditions from the lemma statement. By $d_1 \cdot \mathrm{r}' \cdot d_2 \in \mathcal{R}^S$, we have that $\mathrm{t} = \mathrm{tup}(d_1 \cdot \mathrm{r}' \cdot d_2) \in S(\mathrm{doc}(\mathrm{r}))$. By $d_1 \cdot x_P\vdash \cdot \mathrm{doc}(\mathrm{r}') \cdot \dashv x_P \cdot d_2 \in \mathcal{R}^P$, we have that $[i, j\rangle \in P(\mathrm{doc}(\mathrm{r}))$ covers t, where $i = |d_1| + 1$ and $j = |d_1 \cdot \mathrm{doc}(\mathrm{r}')| + 1$. As S and P satisfy the highlander condition, there can be no other span in $P(\mathrm{doc}(\mathrm{r}))$ that covers t. Furthermore, as $\mathrm{r}' \notin \mathcal{R}^{S_P^{\mathrm{can}}}$, we can conclude that $t \notin S_P^{\mathrm{can}} \circ S(\mathrm{doc}(\mathrm{r}))$, contradicting that S is splittable by P using S_P^{can}. By Theorem 3.2.4, we can conclude that S is not splittable by P. □

Theorem 4.3.9. *Let S be a regular document spanner and P be a regular document splitter, both given as vset-automata, such that the highlander condition is satisfied. Then,* SPLITTABILITY[VSA] *is in* PSPACE.

Proof. We first verify whether P covers S. Note that the cover condition can be checked in PSPACE (Proposition 4.1.8) and is necessary for splittability (Lemma 3.1.4). Thus, for the remainder of this proof, we can assume that the cover condition is satisfied.

As S and P satisfy the highlander and cover condition, we can now use Lemma 4.3.8. We provide a nondeterministic algorithm that runs in polynomial space for the complement problem, i.e., checking whether S is not splittable by P. We exploit Lemma 4.3.8.

The algorithm guesses a string $\mathrm{r} = d_1 \cdot x_P{\vdash} \cdot \mathrm{r}' \cdot {\dashv}x_P \cdot d_2 \in (\Sigma \cup \Gamma_{\mathrm{Vars}(S)} \cup \Gamma_{\mathrm{Vars}(P)})^*$, letter by letter, and computes $h_S(\mathrm{r})$, $h_P(\mathrm{r})$, and $h(\mathrm{r}')$ on the fly. We note that $h_S(\mathrm{r})$ can be computed in polynomial space by starting with the monoid element $m_S = h_S(\varepsilon)$ and replacing m_S with $m_S \cdot h_S(\sigma)$ whenever a new letter σ is guessed. The elements $h_P(\mathrm{r})$ and $h(\mathrm{r}')$ can be computed analogously. If no counterexample is found within exponentially many steps, the algorithm rejects.[8]

Finally, by Lemma 4.3.8, the facts that S and P satisfy the highlander and cover condition, and the definition of the monoids M_S, M_P, and M, we have that S is not splittable by P if $h_S(\mathrm{r}) \in M_S^{\mathrm{acc}}$, $h_P(\mathrm{r}) \in M_P^{\mathrm{acc}}$, and $h(\mathrm{r}') \notin M^{\mathrm{acc}}$. We remind that h_S and h_P ignore "foreign" variables. By Proposition 4.3.6, the condition $h(\mathrm{r}') \notin M^{\mathrm{acc}}$ can be checked in polynomial space. As the other two conditions can be easily checked in polynomial space, this concludes the proof. $\square$

The following corollary is immediate by Lemma 3.1.6 and Theorem 4.3.9.

Corollary 4.3.10. *Deciding* SPLITTABILITY[VSA] *is in* PSPACE, *if the input spanner is proper and the splitter is disjoint.* $\square$

4.3.4 Proof of Lemma 4.3.2

We now give the proof of Lemma 4.3.2. To this end, we recall the lemma statement.

Lemma 4.3.2. *Let* $A \in$ VSA. *There is a monoid* $M_A^{\prec}$ *of exponential size that recognizes* $\mathcal{R}^{[\![A]\!]}$. *Furthermore,* $M_A^{\prec}$ *can be constructed by a polynomial space Turing Machine.*

We start by giving some intuition about the proof idea. Let $A = (\Sigma, V, Q, q_0, Q_F, \delta) \in$ VSA. We define the monoid M_V that can test whether a ref-word, using variables from V, satisfies the variable order condition:

$$M_V := \left(2^{\Gamma_V} \cup \{0\}, \odot_V, \emptyset\right)$$

$$X \odot_V Y := \begin{cases} X \cup Y & \text{if } X \cap Y = \emptyset \text{ and } x \prec y \text{ for all } x \in X, y \in Y \\ 0 & \text{otherwise.} \end{cases}$$

Building up on M_V, we define $M_A^{\prec}$ as

$$M_A^{\prec} := \left(M_V \cup (M_V \times M_A \times M_V),\ \odot_A^{\prec}, \emptyset\right).$$

[8]Again, as argued in the proof of Proposition 4.3.6, the algorithm only stores the monoid elements but not the ref-word r. Furthermore, there must be an counterexample of size at most exponential in the input, as otherwise, the algorithm would store the same combination of monoid elements at least once.

The intuitive idea behind our construction is that we use the monoid M_V to process substrings consisting entirely of variable operations. The monoid M_V conveniently already checks that the variable operations occur in the correct order and we can derive the whole set of processed variable operations from the monoid element obtained after processing a substring of variable operations. In fact, if the operations contain no duplicates and are in the correct order, the monoid element is the desired set. Otherwise it is 0 to denote that the processed ref-word is invalid.

Monoid elements m from $M_A^{\prec}$ that are from M_V correspond to substrings containing only variable operations. Monoid elements of the form $m = (m_{v_1}, m_a, m_{v_2})$ correspond to a substring containing variable operations and symbols. Here m_{v_1} and m_{v_2} correspond to the variable operations before the first and after the last symbol from Σ, respectively, while m_a corresponds to possible runs of the automaton for the substring r' from the first to the last Σ-symbol. However, we cannot simply compute $h_A(\mathrm{r}')$, as we also have to consider runs of the automaton that process the variable operations that occur inside r' in a different order.

At some point we need to connect monoid elements from M_V with monoid elements from M_A. We therefore define a function $f \colon M_V \to M_A$ that, given some $m_v \in M_V$, computes all possible runs in A that use exactly the variable operations encoded by m_v.

We give the formal proof now.

Proof. Let $A = (\Sigma, V, Q, q_0, Q_F, \delta) \in \mathrm{VSA}$. We define $M_A^{\prec}$ as

$$M_A^{\prec} \ := \ \Big(M_V \cup (M_V \times M_A \times M_V),\ \odot_A^{\prec},\ \emptyset\Big)\,.$$

It is obvious that $M_A^{\prec}$ can be constructed with polynomial space in $|A|$, as M_A and M_V can be constructed with polynomial space in $|A|$. Therefore, $M_A^{\prec}$ is of exponential size in $|A|$. First, we define for every subset Γ of Γ_V the language $R^{\Gamma} \subseteq \Gamma^{|\Gamma|}$ as the language containing all strings $v_1 \cdots v_{|\Gamma|}$ of variable operations such that each variable operation in V occurs exactly once and $i < j$ implies that for no variable x it holds that $v_i = \dashv x$ and $v_j = x\vdash$. With other words, $\mathcal{R}^{\Gamma}$ contains all strings of variable operations over Γ that can be completed to a valid ref-word by adding a prefix and a suffix. Both, the prefix and/or the suffix can be empty. We remind that $m_v \in M_V$ is a set of variable operations, except for the case $m_v = 0$.

Now we are ready to define the function $f \colon M_V \to M_A$.

$$f(m_v) \ := \ \begin{cases} \emptyset & \text{if } m_v = 0 \\ \big\{(q_1, q_2) \mid \text{there is a string } \mathrm{r} \in \mathcal{R}^{m_v}, \text{ such that } q_2 \in \delta^*(q_1, \mathrm{r})\big\} & \text{otherwise.} \end{cases}$$

We define the multiplication operation of $M_A^{\prec}$. There are four different cases depending on whether the operands are from M_V or from $M_V \times M_A \times M_V$.

$$\begin{aligned} m_{v_1} \odot_A^{\prec} m_{v_2} &:= m_{v_1} \odot_V m_{v_2} \\ m_{v_1} \odot_A^{\prec} \big(m_{v_2}, m_a, m_{v_3}\big) &:= \big(m_{v_1} \odot_V m_{v_2},\ m_a,\ m_{v_3}\big) \\ \big(m_{v_1}, m_a, m_{v_2}\big) \odot_A^{\prec} m_{v_3} &:= \big(m_{v_1},\ m_a,\ m_{v_2} \odot_V m_{v_3}\big) \\ \big(m_{v_1}, m_{a_1}, m_{v_2}\big) \odot_A^{\prec} \big(m_{v_3}, m_{a_2}, m_{v_4}\big) &:= \big(m_{v_1},\ m_{a_1} \odot_A f(m_{v_2} \odot_V m_{v_3}) \odot_A m_{a_2},\ m_{v_4}\big) \end{aligned}$$

We remind that $\odot_V$ denotes the multiplication of M_V and $\odot_A$ denotes the multiplication of M_A. It remains to show that $M_A^{\prec}$ accepts $\mathcal{R}^{[\![A]\!]}$. We use the homomorphism, induced by

$$h_A^{\prec}(a) := \begin{cases} h_V(a) & \text{if } a \in \Gamma_V \\ (\emptyset, h_A(a), \emptyset) & \text{if } a \in \Sigma\ , \end{cases}$$

that maps variable operations to the corresponding elements of m_V and symbols to the corresponding elements from m_A. We define $M_A^{\prec\text{acc}}$ as

$$\begin{aligned} M_A^{\prec\text{acc}} := &\ \big\{m \in M_V \ \mid\ f(m) \in M_A^{\text{acc}}\big\} \cup \\ &\ \big\{(m_{v_1}, m_a, m_{v_2}) \in M_V \times M_A \times M_V \ \mid\ f(m_{v_1}) \odot_A m_a \odot_A f(m_{v_2}) \in M_A^{\text{acc}}\big\}\ . \end{aligned}$$

The top row corresponds to the case that the document is empty, i.e., the ref-word consists only of variable operations, while the bottom row corresponds to non-empty documents. To determine whether a ref-word should be accepted, we have to incorporate the variable operations before the first and after the last symbol from Σ. Then, we can use M_A^{acc} to check whether we should accept.

It remains to show that $\{\mathrm{r} \mid h_A^{\prec}(\mathrm{r}) \in M_A^{\prec\text{acc}}\} = \mathcal{R}^{[\![A]\!]}$. Let $\mathrm{r}' \in \mathcal{R}^{[\![A]\!]}$, $\mathrm{t} := \mathrm{tup}(\mathrm{r}')$, and $d := \mathrm{doc}(\mathrm{r}')$. Thus, it must hold that $\mathrm{t} \in [\![A]\!](d)$ and there is a valid ref-word $\mathrm{r} \in \mathcal{R}(A)$ which is accepted by A, such that $\mathrm{tup}(\mathrm{r}) = \mathrm{t}$ and $\mathrm{doc}(\mathrm{r}) = d$. Per definition of $\mathcal{R}^{[\![A]\!]}$ it follows that $\mathrm{ref}(d, \mathrm{t}) = \mathrm{r}'$. We have to show that $h_A^{\prec}(\mathrm{r}') = h_A^{\prec}(\mathrm{ref}(d, \mathrm{t})) \in M_A^{\prec\text{acc}}$. We decompose r as

$$\mathrm{r} = V_0 \cdot d_1 \cdot V_1 \cdot d_2 \cdot V_2 \cdots V_{k-1} \cdot d_k \cdot V_k$$

and r′ as

$$\mathrm{r}' = V_0' \cdot d_1' \cdot V_1' \cdot d_2' \cdot V_2' \cdots V_{\ell-1}' \cdot d_k' \cdot V_\ell'$$

where $V_i, V_i' \in \Gamma_V^*$ and $d_j, d_j' \in \Sigma^*$. As both ref-words encode the same tuple for the same document, we have that $k = \ell$, $d_i = d_i'$, and V_j' is a permutation of the symbols in V_j for $0 \le i \le k$ and $1 \le j \le k$. By definition of M_A and $M_A^{\prec}$, we get that

$$\begin{aligned} h_A(\mathrm{r}) &= h_A(V_0) \ \odot_A\ h_A(d_1) \ \odot_A\ \cdots \ \odot_A\ h_A(d_k) \ \odot_A\ h_A(V_k) \ \in\ M_A^{\text{acc}} \\ h_A^{\prec}(\mathrm{r}') &= h_A^{\prec}(V_0') \ \odot_A^{\prec}\ h_A^{\prec}(d_1) \ \odot_A^{\prec}\ \cdots \ \odot_A^{\prec}\ h_A^{\prec}(d_k) \ \odot_A^{\prec}\ h_A^{\prec}(V_k') \\ &\overset{(1)}{=} h_V(V_0') \ \odot_A^{\prec}\ \big(\emptyset, h_A(d_1), \emptyset\big) \ \odot_A^{\prec} \cdots \odot_A^{\prec}\ \big(\emptyset, h_A(d_k), \emptyset\big) \ \odot_A^{\prec}\ h_V(V_k') \\ &\overset{(2)}{=} \Big(h_V(V_0'),\ h_A(d_1) \ \odot_A\ f(h_V(V_1')) \ \odot_A\ h_A(d_2) \ \odot_A\ \cdots \\ &\qquad \cdots \ \odot_A\ f(h_V(V_{k-1}')) \ \odot_A\ h_A(d_k),\ h_V(V_k')\Big) \end{aligned}$$

The equality (1) holds by the definition of $\odot_A^{\prec}$, which for substrings consisting only of variable operations just uses $\odot_V$ and for substrings containing only Σ-symbols uses basically m_A. We note that $f(\emptyset) = h_A(\varepsilon)$, as $R^{\emptyset} = \{\varepsilon\}$. The equality (2) can be derived by iteratively applying the definition of $\odot_A^{\prec}$ as often as possible.

By definition of $M_A^{\prec\text{acc}}$, we get that $h_A^{\prec}(\mathrm{r}') \in M_A^{\prec\text{acc}}$ if and only if m_a defined as

$$m_a := f(h_V(V_0')) \odot_A h_A(d_1) \odot_A f(h_V(V_1')) \odot_A h_A(d_2) \odot_A \cdots \\ \cdots \odot_A h_A(d_k) \odot_A f(h_V(V_k'))$$

is in M_A^{acc}. As V_i' respects the variable ordering, $h_V(V_i') \neq 0$ is the set containing all variable operations from V_i'. By definition of f and the fact that V_i' contains exactly the same variable operations as V_i, we can conclude that $h_A(V_i) \subseteq f(h_V(V_i'))$ for $0 \leq i \leq k$.[9] As the multiplication $\odot_A$ is monotone[10] and $h_A(V_i) \subseteq f(h_V(V_i'))$, we get that $h_A(\mathrm{r}) \subseteq m_a$. Furthermore, as A accepts r, it holds that $h_A(\mathrm{r}) \in M_A^{\text{acc}}$ and due to M_A^{acc} being upwards closed[11] we can conclude that $m_a \in M_A^{\text{acc}}$ and therefore $h_A^{\prec}(\mathrm{r}') \in M_A^{\prec\text{acc}}$. This concludes one direction of the proof.

Let now r be some ref-word, such that $h_A^{\prec}(\mathrm{r}) \in M_A^{\prec\text{acc}}$. We have to show that there exists a valid ref-word $\mathrm{r}' \in \mathcal{R}(A)$ such that $\mathrm{doc}(\mathrm{r}) = \mathrm{doc}(\mathrm{r}')$ and $\mathrm{tup}(\mathrm{r}) = \mathrm{tup}(\mathrm{r}')$.

We decompose r as

$$\mathrm{r} = V_0 \cdot d_1 \cdot V_1 \cdot d_2 \cdot V_2 \cdots V_{k-1} \cdot d_k \cdot V_k .$$

Observe that $k = 0$, if $h_A^{\prec}(\mathrm{r}) \in M_V$, and $k > 0$ otherwise. By the definition of $M_A^{\prec\text{acc}}$ we know that

$$m_{\mathrm{r}} := f(h_V(V_0)) \odot_A h_A(d_1) \odot_A f(h_V(V_1)) \odot_A \cdots \odot_A h_A(d_k) \odot_A f(h_V(V_k)) \in M_A^{\text{acc}}.$$

Let $q_0^V, q_1^d, q_1^V, q_2^d, \ldots, q_k^d, q_k^V, q_{k+1}^d$ be states such that q_0^V is the initial state and q_{k+1}^d is some final state of A and for $0 \leq i \leq k$ and $1 \leq j \leq k$ it holds that

- $(q_i^V, q_{i+1}^d) \in f(h_V(V_i))$; and
- $(q_j^d, q_j^V) \in h_A(d_j)$.

We note that, due to $m_{\mathrm{r}} \in M_A^{\text{acc}}$ and the definition of m_{r}, these states have to exist.

By the definition of f and the fact that $(q_i^V, q_{i+1}^d) \in f(h_V(V_i))$, for every $0 \leq i \leq k$, there must be a strings $V_0' \ldots V_k' \in \Gamma_V^*$ of variable operations, such that $V_i' \in \mathcal{R}^{h_V(V_i)}$ and $q_{i+1}^d \in \delta^*(q_i^V, V_i')$, for every $0 \leq i \leq k$. We define r′ as

$$\mathrm{r}' := V_0' \cdot d_1 \cdot V_1' \cdot \cdots \cdot V_{k-1}' \cdot d_k \cdot V_k' .$$

By the construction r′ is a valid ref-word, such that $\mathrm{doc}(\mathrm{r}) = \mathrm{doc}(\mathrm{r}')$ and $\mathrm{tup}(\mathrm{r}) = \mathrm{tup}(\mathrm{r}')$. Furthermore, we have that $\delta^*(q_0, \mathrm{r}') \cap Q_F \neq \emptyset$ and therefore r′ is accepted by A, concluding the proof. □

[9] We remind that elements of M_A are sets of pairs of states, which we can compare using $\subseteq$.

[10] That is $m_1 \subseteq m_1'$ and $m_2 \subseteq m_2'$ imply $m_1 \odot_A m_2 \subseteq m_1' \odot_A m_2'$ for all $m_1, m_2, m_1', m_2' \in M_A$.

[11] That is $m \subseteq m'$ and $m \in M_A^{\text{acc}}$ implies that $m' \in M_A^{\text{acc}}$.

4.4 Complexity Lower Bounds

In this section, we will give lower bounds for Split-Correctness, Splittability and other related decision problems. Recall that the problems Split-Correctness[$\mathcal{S}$] and Self-Splittability[$\mathcal{S}$] are in PTIME if $\mathcal{S} \in \mathcal{S}_{\text{tractable}}$ and the highlander condition is satisfied. Here, we show that neither $\mathcal{S} \in \mathcal{S}_{\text{tractable}}$ nor the highlander condition on its own are sufficient to achieve tractability.

We start by showing that the problems Split-Correctness, Splittability, and Self-Splittability are PSPACE-hard, even if the spanner is proper and all inputs are given as deterministic functional vset-automata. As we will see in the proof it is already PSPACE-hard to decide whether the cover condition is satisfied.

Lemma 4.4.1. *The problems Self-Splittability*[dfVSA]*, Splittability*[dfVSA]*, and Cover*[dfVSA] *are* PSPACE*-hard, even if all input spanner are proper.*

Proof. We give a reduction from the PSPACE-complete problem of DFA concatenation universality [73]. Given two DFAs A_1, A_2, DFA concatenation universality asks whether $\mathcal{L}(\Sigma^*) = \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$.

Let A_1, A_2 be regular languages, given as DFAs over the alphabet Σ. Furthermore, let $a \notin \Sigma$. Slightly abusing notation, we define the dfVSA by a hybrid regex-formula, where the automata A_i are plugged in. In particular, $A_S = \Sigma^* \cdot y\{a\}$ and $A_P = A_1 \cdot x\{A_2 \cdot a\}$. Let $S = [\![A_S]\!]$ and $P = [\![A_P]\!]$. Thus, $S(d) = \emptyset = P(d)$ if $d \notin \mathcal{L}(\Sigma^* \cdot a)$. Furthermore, if $d \in \mathcal{L}(\Sigma^* \cdot a)$, $S(d) = [|d|, |d|+1\rangle$ and for all $[i,j\rangle \in P(d)$ it holds that $i \leq |d|$ and $j = |d|+1$.

We show that the following statements are equivalent:

1. S is self-splittable by P,
2. S is splittable by P,
3. $\mathcal{L}(A_1) \cdot \mathcal{L}(A_2) = \mathcal{L}(\Sigma^*)$,
4. P covers S.

We observe that (1) implies (2). Thus, we only need to show that (2) implies (3), (3) implies (4), and (4) implies (1).

(2) implies (3): Assume that $\mathcal{L}(A_1) \cdot \mathcal{L}(A_2) \neq \mathcal{L}(\Sigma^*)$. Thus there is a document $d \in \Sigma^*$ such that $d \notin \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$. Therefore, $P(d \cdot a) = \emptyset$ but $S(d \cdot a) = \{[|d|+1, |d|+2\rangle\} \neq \emptyset$ and therefore S can not be splittable by P.

(3) implies (4): Assume that $\mathcal{L}(A_1) \cdot \mathcal{L}(A_2) = \mathcal{L}(\Sigma^*)$. Let $d' \in (\Sigma \cup \{a\})^*$ and $\mathrm{t} \in S(d')$. Thus, $d' = d \cdot a$, for some document $d \in \Sigma^*$ and $\mathrm{t}(y) = [|d|+1, |d|+2\rangle$. Per assumption, there is a decomposition $d = d_1 \cdot d_2$, such that $d_i \in A_i$, for $i \in \{1,2\}$. Therefore, $s := [|d_1|+1, |d|+2\rangle \in P(d \cdot a) = P(d')$ covers t.

(4) implies (1): We have to show that $S = S \circ P$. Let $\mathrm{t} \in S(d')$ be a tuple. Therefore, there is a document $d \in \Sigma^*$ such that $d' = d \cdot a$. As P covers S, there is a split $s \in P(d')$

which covers t. Observe that per definition of S, it holds that $\mathrm{t} \ll s \in S(d'_s)$ and therefore $\mathrm{t} \in (S \circ P)(d')$, implying that $S \subseteq S \circ P$. For the other direction, let $\mathrm{t} \in (S \circ P)(d')$. Therefore, there is a document $d \in \Sigma^*$ with $d' = d \cdot a$. Thus, there is a span $s \in P(d')$ which covers t. It follows per definition of S that $\mathrm{t} \ll s \in S(d'_s)$. Which implies that $\mathrm{t} \in S(d')$ and therefore $S \circ P \subseteq S$. $\square$

It follows directly that Split-Correctness[dfVSA] is PSPACE-hard.

Corollary 4.4.2. *Split-Correctness*[dfVSA] *is* PSPACE*-hard, even if all input spanner are proper.* $\square$

Smit [153, Proposition 3.3.7] shows that the problems Split-Correctness[dfVSA] and Self-Splittability[dfVSA] remain PSPACE-hard if S is a Boolean spanner (and therefore not proper) and P is disjoint. It is straightforward to extend the proof of Smit along the lines of Lemma 4.4.1 to show that Splittability[dfVSA] and Cover[dfVSA] are also PSPACE-hard for disjoint splitter. We refer to Appendix A for the proof.

Lemma 4.4.3. *The problems Self-Splittability*[dfVSA]*, Splittability*[dfVSA]*, and Cover*[dfVSA] *are* PSPACE*-hard, even if P is disjoint.*

Corollary 4.4.4. *Split-Correctness*[dfVSA] *is* PSPACE*-hard, even if P is disjoint.* $\square$

Recall that Self-Splittability[usVSA] is in PTIME if the spanner is proper and the splitter is disjoint (cf. Corollary 4.2.2). We will show now that tractability is also lost if the spanner and the splitter are not required to be unambiguous. That is, we show that Self-Splittability and Splittability remain PSPACE-hard even if the highlander condition is satisfied[12] and the spanner and splitter are given as functional regex-formulas or functional vset-automata.

Lemma 4.4.5. *Self-Splittability*[$\mathcal{S}$] *and Splittability*[$\mathcal{S}$]*, for $\mathcal{S} \in \{$*fRGX, fVSA*$\}$, are* PSPACE*-hard, even if the splitter P is disjoint and the spanner S is proper.*

Proof. The reductions are from the containment problem for regular expressions and NFAs which are both known to be PSPACE-complete.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be regular languages and let $S = y\{\mathcal{L}_1\}$ and $P = x\{\mathcal{L}_2\}$. We show the following statements are equivalent:

1. S is self-splittable by P,
2. S is splittable by P,
3. $\mathcal{L}_1 \subseteq \mathcal{L}_2$.

[12] Recall that the highlander condition is satisfied if S is proper and P is disjoint (cf. Lemma 3.1.6).

The lemma statement follows directly from the fact that containment of regular languages is PSPACE-complete for NFAs and regular expressions. It remains to show the equivalence of (1), (2), and (3). We observe that (1) implies (2) per definition.

(2) implies (3): Assume that S is splittable by P. Let $d \in \mathcal{L}_1$ be a document. By definition of S it follows that $[1, |d|+1\rangle \in S(d)$. Since S is splittable by P and $[1, |d|+1\rangle$ is only covered by itself, it follows that $[1, |d|+1\rangle \in P(d)$ and $[1, |d|+1\rangle \in S_P(d_{[1,|d|+1\rangle}) = S_P(d)$ for some spanner S_P. Therefore, by definition of P, we have that $d \in \mathcal{L}_2$.

(3) implies (1): Let $\mathcal{L}_1 \subseteq \mathcal{L}_2$. Observe, that S only selects the span $[1, |d| + 1\rangle$. Therefore, S is self-splittable by P:

$$\begin{aligned}
[1, |d| + 1\rangle \in S(d) \quad &\Leftrightarrow \quad d \in \mathcal{L}_1 \\
&\Leftrightarrow \quad d \in \mathcal{L}_1 \text{ and } d \in \mathcal{L}_2 \\
&\Leftrightarrow \quad [1, |d| + 1\rangle \in S(d) \text{ and } [1, |d| + 1\rangle \in P(d) \\
&\Leftrightarrow \quad [1, |d| + 1\rangle \in (S \circ P)(d)
\end{aligned}$$ □

Again, it follows directly that Split-Correctness[dfVSA] is PSPACE-hard.

Corollary 4.4.6. *Split-Correctness*[fRGX] *and Split-Correctness*[fVSA] *are hard for* PSPACE*, even if the splitter is disjoint and the spanner is proper.* □

4.5 Connection of Split-Existence and Language Primality

Recall the definition of the Split-Existence[$\mathcal{S}, \mathcal{P}$].

Split-Existence[$\mathcal{S}, \mathcal{P}$]	
Input:	Spanner $S \in \mathcal{S}$.
Question:	Is there a splitter $P \in \mathcal{P}$ such that S is splittable by P?

We now show that Split-Existence is strongly connected to a classical problem from Formal Language Theory, which is called Language Primality. To this end, we define *middle extractors*, which capture N-gram extractors or splitters extracting pairs of consecutive sentences. A splitter P is a *middle extractor*, if $P = \mathcal{L}_1 \cdot x\{\mathcal{L}_2\} \cdot \mathcal{L}_3$, where $\mathcal{L}_i \neq \{\varepsilon\}$, for $i \in \{1, 2, 3\}$, are regular languages. We denote the class of middle extractors by P_{middle}. Observe that the splitters used in the proofs of Lemma 4.4.1 and Lemma 4.4.5 are middle extractors and therefore, the problems are PSPACE-hard for middle extractors.

The Language-Primality problem asks, given a language $\mathcal{L}$, whether $\mathcal{L}$ is prime, i.e., whether it cannot be decomposed into two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L} = \mathcal{L}_1 \cdot \mathcal{L}_2$ and $\mathcal{L}_1 \neq \{\varepsilon\} \neq \mathcal{L}_2$. The complexity of Language-Primality has been considered an open problem since the late 90's (cf. Salomaa [136, Problem 2.1]). Martens, Niewerth and Schwentick [103] showed that Language-Primality is PSPACE-complete, if the language is given as a deterministic finite state automaton. However, to the best of our knowledge, the complexity of Language-Primality for other representations of the input remains open. Here, we define the complement of Language-Primality. Furthermore, we add an additional parameter k specifying into how many languages we want to decompose the language $\mathcal{L}$.

k-Decomposable	
Input:	A regular language $\mathcal{L}$.
Question:	Is there a decomposition of $\mathcal{L}$ into $\mathcal{L}_1, \ldots, \mathcal{L}_k$, such that $\mathcal{L} = \mathcal{L}_1 \cdots \mathcal{L}_k$, and $\mathcal{L}_i \neq \{\varepsilon\}$ for all $1 \leq i \leq k$?

Clearly 2-Decomposable is the complement of Language-Primality. There is a connection between Split-Existence and 3-Decomposable that is most easily seen in the case of Boolean spanners:

Observation 4.5.1. *Let $\mathcal{S} \in \mathcal{S}_{\text{general}}$ be a class of regular document spanners and $\mathcal{L} \in \mathcal{S}$ be a Boolean spanner (i.e., a regular language). Then $\mathcal{L}$ is* 3-Decomposable *if and only if $\mathcal{L} \in$ Split-Existence$[\mathcal{S}, P_{\text{middle}}]$.* □

Of course, we are not interested in studying Boolean spanners, but the observation above gives little hope to settle the complexity of Split-Existence$[\mathcal{S}, P_{\text{middle}}]$ without settling the complexity of 3-Decomposable. We note that the complexity of k-Decomposable is still open even for deterministic automata in the case $k > 2$.

4.6 Schema Constraints

In this section we study the complexity of deciding Split-Correctness, Splittability, and Self-Splittability in the presence of schema constraints, as defined in Section 3.3.2.

Let S, P be spanners, P be a splitter and $\mathcal{L}$ be a schema constraint. We begin by showing that the construction for $S \circ P$ in Proposition 4.1.1 can be extended to also embark schema constraints. By Lemma 3.3.4 it holds that $S \equiv_{\mathcal{L}} S_P \circ P$ if and only if $S \bowtie \mathcal{L} = \big(S_P \circ (P \bowtie \mathcal{L})\big)$. Therefore, it suffices to show the following lemma.

Lemma 4.6.1. *Given vset-automata A_S and $A_{\mathcal{L}}$ representing a spanner S and a regular schema constraint $\mathcal{L}$, respectively, a vset-automaton A can be constructed in polynomial time, such that*

1. $[\![A]\!] = [\![A_S]\!] \bowtie \mathcal{L}$;
2. $A \in$ sVSA *if* $A_S \in$ sVSA; *and*
3. $A \in$ uVSA *if* $A_S, A_{\mathcal{L}} \in$ uVSA.

Proof. Let $A_S = (\Sigma, V, Q_S, q_{0,S}, Q_{F,S}, \delta_S) \in \mathcal{S}$ and $A_{\mathcal{L}} = (\Sigma, \emptyset, Q_{\mathcal{L}}, q_{0,\mathcal{L}}, Q_{F,\mathcal{L}}, \delta_{\mathcal{L}}) \in \mathcal{S}$ be as given. We define the automaton $A := (\Sigma, V, Q, q_0, Q_F, \delta)$, where $Q := Q_S \times Q_{\mathcal{L}}$, $q_0 := (q_S, q_{\mathcal{L}})$, $Q_F := Q_{F,S} \times Q_{F,\mathcal{L}}$, and

$$\begin{aligned} \delta \quad := \quad & \big\{\big((q_S, q_{\mathcal{L}}), \sigma, (q'_S, q'_{\mathcal{L}})\big) \mid \sigma \in \Sigma \cup \{\varepsilon\}, (q_S, \sigma, q'_S) \in \delta_S, (q_{\mathcal{L}}, \sigma, q'_{\mathcal{L}}) \in \delta_{\mathcal{L}}\big\} \ \cup \\ & \big\{\big((q_S, q_{\mathcal{L}}), v, (q'_S, q_{\mathcal{L}})\big) \mid v \in \Gamma_V, (q_S, v, q'_S) \in \delta_S, q_{\mathcal{L}} \in Q_{\mathcal{L}}\big\} \ . \end{aligned}$$

The only difference to the usual product construction is, that transitions related to variable operations are only processed by A_S and ignored by $A_{\mathcal{L}}$. It is easy to see that $A \in \text{VSA}$ can be constructed in polynomial time. Furthermore, $\mathcal{R}(A) = \mathcal{R}(A_S) \cap \{\text{r} \mid \text{doc}(\text{r}) \in \mathcal{L}\}$. Therefore it must hold that

$$[\![A]\!] \;=\; [\![\mathcal{R}(A)]\!] \;=\; \mathcal{R}(A_S) \cap \{\text{r} \mid \text{doc}(\text{r}) \in \mathcal{L}\} \;=\; [\![A_S]\!] \bowtie \mathcal{L}\,,$$

concluding the proof of statements (1) and (2).

It only remains to show that $A \in \text{uVSA}$ if $A_S, A_{\mathcal{L}} \in \text{uVSA}$. To this end, assume that A is not unambiguous. As observed before, $\mathcal{R}(A) = \mathcal{R}(A_S) \cap \{\text{r} \mid \text{doc}(\text{r}) \in \mathcal{L}\}$ and therefore $\mathcal{R}(A) \subseteq \mathcal{R}(A_S)$. Thus, due to $A_S \in \text{uVSA}$, A must satisfy the variable order condition. Assume there are two distinct runs of A that violate unambiguity condition (C3). Due to $A_S \in \text{uVSA}$, both runs must coincide in the A_S component of A. However, by the same argument, both runs must also coincide in the $A_{\mathcal{L}}$ component of A, leading to the desired contradiction. This concludes the proof. $\square$

Due to Lemmas 3.3.4 and 4.6.1, the complexity results for Split-Correctness, Self-Splittability, and Splittability (cf. Theorems 4.0.1, 4.0.2) also hold in the presence of schema constraints. Note that this also includes the PTIME fragment if the schema constraint $\mathcal{L} \in \mathcal{S}$ is represented by a class of document spanners $\mathcal{S} \in \mathcal{S}_{\text{tractable}}$.

As we show next, given a sequential vset-automaton $A_S \in \text{sVSA}$, a vset-automaton A that represents the minimal schema constraint can be constructed in polynomial time.

Lemma 4.6.2. *Let $A_S \in \text{sVSA}$ be a sequential vset-automaton. Then an automaton $A \in \text{sVSA}$ with $[\![A]\!] = \pi_\emptyset [\![A_S]\!]$ can be constructed in polynomial time.*

Proof. Let $A_S = (\Sigma, V, Q_S, q_{0,S}, Q_{F,S}, \delta_S)$. We define $A := (\Sigma, \emptyset, Q_S, q_{0,S}, Q_{F,S}, \delta)$, where

$$\begin{aligned} \delta \quad := \quad & \big\{(p,\sigma,q) \mid \sigma \in \Sigma \cup \{\varepsilon\}, (p,\sigma,q) \in \delta_S\big\} \;\cup \\ & \big\{(p,\varepsilon,q) \mid (p,v,q) \in \delta_S, v \in \Gamma_V\big\}\,. \end{aligned}$$

Observe that $A \in \text{sVSA}$ can be constructed in polynomial time. Furthermore, due to the assumption that A_S is sequential, it follows that there is a tuple $\text{t} \in [\![A_S]\!](d)$ if and only if $() \in [\![A]\!]$. Therefore, it must hold that $[\![A]\!] = \pi_\emptyset [\![A_S]\!]$. $\square$

Due to Observation 3.3.5 and Observation 3.3.6, we can decide whether there exists a schema constraint $\mathcal{L}$ which covers S such that $S \equiv_{\mathcal{L}} S_P \circ P$ by checking whether $S \equiv_{\mathcal{R}(\pi_\emptyset(S))} S_P \circ P$. Furthermore, it follows directly from Lemma 4.6.2 that $\mathcal{R}(\pi_\emptyset(S))$ can indeed be constructed in polynomial time. However, given an unambiguous (resp., deterministic) and sequential vset-automaton, one can not guarantee that the automaton A, as constructed in Lemma 4.6.2, is unambiguous. Thus, all but the PTIME complexity result for Split-Correctness, Self-Splittability, and Splittability (cf. Theorems 4.0.1, 4.0.2) also hold if one asks whether there exists a schema constraints $\mathcal{L}$ which covers S, such that $S \equiv_{\mathcal{L}} S_P \circ P$.

Part II

Quantitative Aspects of Document Spanners

Chapter 5

Weight Annotators

In this chapter we introduce and study weight annotators. In contrast to classical document spanners, weight annotators quantify the extracted tuples. That is, each extracted tuple is associated with a weight from a semiring.

Organization

This chapter is organized as follows. We give some required algebraic background, preliminary definitions and notation in Section 5.1. In Sections 5.2 and 5.3 we define $\mathbb{K}$-Annotators and weighted vset-automata — a formalism to represent $\mathbb{K}$-Annotators. We discuss semiring encodings in Section 5.4. We conclude this chapter by studying their fundamental properties in Section 5.5 and various evaluation and enumeration problems of weighted vset-automata in Sections 5.6, and 5.7.

5.1 Annotated Relations

Weight annotators read documents and produce *annotated relations* [62], which are relations in which each tuple is annotated with an element from a commutative semiring. In this section, we revisit the basic definitions and properties of annotated relations.

5.1.1 Algebraic Foundations

We begin by giving some required background on algebraic structures like monoids and semirings [60].

A *commutative monoid* $(\mathbb{M}, *, \mathrm{id})$ is an algebraic structure consisting of a set $\mathbb{M}$, a binary operation $*$ and an element $\mathrm{id} \in \mathbb{M}$, such that:

1. $*$ is associative, i.e., $(a * b) * c = a * (b * c)$ for all $a, b, c \in \mathbb{M}$,

2. id is an identity, i.e., $\mathrm{id} * a = a * \mathrm{id} = a$ for all $a \in \mathbb{M}$, and

3. $*$ is commutative, i.e. $a * b = b * a$ for all $a, b \in \mathbb{M}$.

We say that a monoid $(\mathbb{M}, *, \mathrm{id})$ is *bipotent*, if $a * b \in \{a, b\}$, for every $a, b \in \mathbb{M}$.

A *commutative semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is an algebraic structure consisting of a set $\mathbb{K}$, containing two elements: the *zero* element $\overline{0}$ and the *one* element $\overline{1}$. Furthermore, it is equipped with two binary operations, namely *addition* $\oplus$ and *multiplication* $\otimes$ such that:

1. $(\mathbb{K}, \oplus, \overline{0})$ is a commutative monoid,
2. $(\mathbb{K}, \otimes, \overline{1})$ is a commutative monoid,
3. multiplication distributes over addition, that is, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$, for all $a, b, c \in \mathbb{K}$, and
4. $\overline{0}$ is absorbing for $\otimes$, that is, $\overline{0} \otimes a = \overline{0}$ for all $a \in \mathbb{K}$.

Furthermore, a semiring is *positive* if, for all $a, b \in \mathbb{K}$, the following conditions hold:

- $\overline{0} \neq \overline{1}$,
- if $a \oplus b = \overline{0}$, then $a = \overline{0} = b$, and
- if $a \otimes b = \overline{0}$, then $a = \overline{0}$ or $b = \overline{0}$.

We call a semiring *bipotent*, if its additative monoid is bipotent.

An element $a \in \mathbb{K}$ is a *zero divisor* if $a \neq \overline{0}$ and there is an element $b \in \mathbb{K}$ with $b \neq \overline{0}$ and $a \otimes b = \overline{0}$. Furthermore, an element $a \in \mathbb{K}$ has an *additive inverse*, if there is an element $b \in \mathbb{K}$ such that $a \oplus b = \overline{0}$. In the following, we will also identify a semiring by its domain $\mathbb{K}$ if the rest is clear from the context. When we do this for numeric semirings such as $\mathbb{Q}$ and $\mathbb{N}$, we always assume the usual addition and multiplication.

Given a semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ and a set $\mathbb{K}' \subseteq \mathbb{K}$ with $\overline{0}, \overline{1} \in \mathbb{K}'$ such that $\mathbb{K}'$ is closed under addition and multiplication (that is, for all $a, b \in \mathbb{K}'$ it holds that $a \oplus b \in \mathbb{K}'$ and $a \otimes b \in \mathbb{K}'$) then $(\mathbb{K}', \oplus, \otimes, \overline{0}, \overline{1})$ is a *subsemiring* of $\mathbb{K}$.

Example 5.1.1. The following are examples of commutative semirings. It is easy to verify that all but the numeric semirings and the Łukasiewicz semiring are positive.

1. The *numeric semirings* are $(\mathbb{Q}, +, \cdot, 0, 1)$, and $(\mathbb{Z}, +, \cdot, 0, 1)$.
2. The *counting semiring* $(\mathbb{N}, +, \cdot, 0, 1)$.
3. The *Boolean semiring* $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ where $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$.
4. The *probability semiring* $(\mathbb{Q}^+, +, \cdot, 0, 1)$.[1] Rabin [128] and Segala [146] define probabilistic automata over this semiring, where all transition weights must be between 0 and 1 and the sum of all transition weights starting some state, labeled by the same label must be 1.
5. The *Viterbi semiring* $([0, 1], \max, \cdot, 0, 1)$ is used in probabilistic parsing [38].

[1] One may expect the domain to be $[0, 1]$, but this is difficult to obtain while maintaining the semiring properties. For instance, defining $a \oplus b$ as $\min\{a + b, 1\}$ would violate distributivity.

6. The *access control semiring* $\mathbb{A} = (\{P < C < S < T < 0\}, \min, \max, 0, P)$, where P is "public", C is "confidential", S is "secret", T is "top secret", and 0 is "so secret that nobody can access it" [50].

7. The *tropical semiring* $(\mathbb{Q} \cup \{\infty\}, \min, +, \infty, 0)$ where min stands for the binary minimum function. This semiring is used in optimization problems of networks [38].[2]

8. The *Łukasiewicz semiring*, whose domain is $[0, 1]$, with addition given by $x \oplus y = \max(x, y)$, with multiplication $x \otimes y = \max(0, x + y - 1)$, zero element 0, and one element 1. This semiring is used in multivalued logics [38].

Complexity-wise, we assume that semiring elements are encoded in binary. That is, the encoding of a semiring $\mathbb{K}$, is a function $\mathrm{enc} : \mathbb{K} \to \{0, 1\}^*$, which assigns a binary encoding to every semiring element. Furthermore, we denote the length of the encoding of an element $a \in \mathbb{K}$ by $\|a\|$.[3] We discuss semiring encodings into more detail in Section 5.4.

5.1.2 Annotated Relations

For the rest of this thesis, we assume that $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a commutative semiring. Let $V \subseteq \mathrm{Vars}$ be a finite set of variables. Recall that a V-*tuple* is a function $\mathrm{t} : V \to \mathrm{D}$ that assigns values to variables in V and that we denote the set of all the V-tuples by V-Tup. A $\mathbb{K}$-*relation R over V* is a function $R : V\text{-}\mathsf{Tup} \to \mathbb{K}$ such that its *support*, defined by $\mathsf{supp}(R) := \{\mathrm{t} \mid R(\mathrm{t}) \neq \overline{0}\}$, is finite. We will also write $\mathrm{t} \in R$ to abbreviate $\mathrm{t} \in \mathsf{supp}(R)$. Furthermore, we say that two $\mathbb{K}$-relations R_1 and R_2 are *disjoint* if $\mathsf{supp}(R_1) \cap \mathsf{supp}(R_2) = \emptyset$. The size of a $\mathbb{K}$-relation R is the *size* of its support, that is, $|R| := |\mathsf{supp}(R)|$. The *arity* of a $\mathbb{K}$-relation over V is $|V|$.

Example 5.1.2. The bottom left table in Figure 5.1 shows an example $\mathbb{K}$-relation, where $\mathbb{K}$ is the Viterbi semiring. The variables are x_{pers} and x_{loc}, so the V-tuples are described in the first two columns. The third column contains the element in $\mathbb{K}$ associated to each tuple. □

Green et al. [62] defined a set of operators on $\mathbb{K}$-relations that naturally correspond to relational algebra operators and map $\mathbb{K}$-relations to $\mathbb{K}$-relations. They define the algebraic operators[4] *union*, *projection*, *natural join*, and *selection* for all finite sets $V_1, V_2 \subseteq \mathrm{Vars}$ and for all $\mathbb{K}$-relations R_1 over V_1 and R_2 over V_2, as follows.

- **Union**: If $V_1 = V_2$ then the union $R := R_1 \cup R_2$ is a function $R : V_1\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by $R(\mathrm{t}) := R_1(\mathrm{t}) \oplus R_2(\mathrm{t})$. (Otherwise, the union is not defined.)

[2] In literature there are actually multiple different definitions for the tropical semiring, e.g., $(\mathbb{Q} \cup \{-\infty\}, \max, +, -\infty, 0)$ and $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$. If not mentioned otherwise, we use the tropical semiring as defined here.

[3] Note that we do not denote the encoding length of semiring elements by $|a|$ to obviate confusions with the absolute value function for numbers.

[4] As in much of the work on semirings in provenance, e.g., Green et al. [62], we do not consider the *difference* operator (which would require additive inverses).

`Carter␣from␣Plains,␣Georgia,␣Washington␣from␣Westmoreland,␣Virginia`

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 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67

x_{pers}	x_{loc}	annotation
Carter	Plains,␣Georgia	0.9
Washington	Westmoreland,␣Virginia	0.9
Carter	Georgia,␣Washington	0.81
Carter	Westmoreland,␣Virginia	0.59049

x_{pers}	x_{loc}	annotation
$[1,7\rangle$	$[13,28\rangle$	0.9
$[30,40\rangle$	$[46,68\rangle$	0.9
$[1,7\rangle$	$[21,40\rangle$	0.81
$[1,7\rangle$	$[46,68\rangle$	0.59049

Figure 5.1: A document (top), a $\mathbb{K}$-relation (bottom left), and the corresponding $(\mathbb{K}, d)$-relation (bottom right).

- **Projection**: For $X \subseteq V_1$, the projection $R := \pi_X R_1$ is a function $R : X\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathrm{t}) := \bigoplus_{\mathrm{t}=\pi_X(\mathrm{t}') \text{ and } R_1(\mathrm{t}')\neq\overline{0}} R_1(\mathrm{t}') \ .$$
- **Natural Join:** The natural join $R := R_1 \bowtie R_2$ is a function $R : (V_1 \cup V_2)\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathrm{t}) := R_1(\pi_{V_1}(\mathrm{t})) \otimes R_2(\pi_{V_2}(\mathrm{t})) \ .$$
- **Selection:** If $\mathrm{P} : V_1\text{-}\mathsf{Tup} \to \{\overline{0}, \overline{1}\}$ is a selection predicate that maps each V_1-Tup t to either $\overline{0}$ or $\overline{1}$. Then $R := \sigma_{\mathrm{P}}(R_1)$ is a function $R : V_1\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathrm{t}) := R_1(\mathrm{t}) \otimes \mathrm{P}(\mathrm{t}) \ .$$

Proposition 5.1.3 (Green et al. [62])**.** *The above operators preserve the finiteness of the supports and therefore they map $\mathbb{K}$-relations into $\mathbb{K}$-relations.*

Hence, we obtain an algebra on $\mathbb{K}$-relations.

5.2 K-Annotators

Recall that a d-tuple t is a V-tuple which only assigns values from $\mathsf{Spans}(d)$. A $(\mathbb{K}, d)$-*relation* over $V \subseteq \mathrm{Vars}$ is defined analogously to a $\mathbb{K}$-relation over V but only uses d-tuples t with $V = \mathrm{Vars}(\mathrm{t})$.

Definition 5.2.1. A $\mathbb{K}$-*annotator* (or *annotator* for short), is a function S that is associated with a finite set $V \subseteq \mathrm{Vars}$ of variables and maps documents d into $(\mathbb{K}, d)$-relations over V. We denote V by $\mathrm{Vars}(S)$. We sometimes also refer to a $\mathbb{K}$-annotator as an *annotator over* $\mathbb{K}$ when we want to emphasize the semiring.

Notice that $\mathbb{B}$-annotators, i.e., annotators over the Boolean semiring $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ are simply the *functional document spanners*.[5] Furthermore, we say that two $\mathbb{K}$-annotators S_1 and S_2 are *disjoint* if, for every document $d \in \Sigma^*$, the $\mathbb{K}$-relations $S_1(d)$ and $S_2(d)$ are disjoint.

Example 5.2.2. We provide an example document d in Figure 5.1 (top). The table at the bottom right depicts a possible $(\mathbb{K}, d)$-relation obtained by a spanner that extracts (person, hometown) pairs from d. Notice that for each span $[i, j\rangle$ occurring in this table, the string $d_{[i,j\rangle}$ can be found in the table to the left.

In this naïve example, which is just to illustrate the definitions, we used the Viterbi semiring and annotated each tuple with $(0.9)^k$, where k is the number of words between the spans associated to x_{pers} and x_{loc}. The annotations can therefore be interpreted as confidence scores. □

We now lift the relational algebra operators on $\mathbb{K}$-relations to the level of $\mathbb{K}$-annotators. For all documents d and for all annotators S_1 and S_2 associated with V_1 and V_2, respectively, we define the following:

- **Union:** If $V_1 = V_2$ then the union $S := S_1 \cup S_2$ is defined by $S(d) := S_1(d) \cup S_2(d)$.[6]
- **Projection:** For $X \subseteq V_1$, the projection $S := \pi_X S_1$ is defined by $S(d) := \pi_X S_1(d)$.
- **Natural Join:** The natural join $S := S_1 \bowtie S_2$ is defined by $S(d) := S_1(d) \bowtie S_2(d)$.
- **String selection:** Let R be a k-ary string relation.[7] The string-selection operator σ^R is parameterized by k variables $x_1, \dots, x_k$ in V_1 and can be written as $\sigma^R_{x_1,\dots,x_k}$. Then the annotator $S := \sigma^R_{x_1,\dots,x_k} S_1$ is defined as $S(d) := \sigma_{\mathrm{P}}(S_1(d))$ where P is a selection predicate with $\mathrm{P}(\mathrm{t}) = \overline{1}$ if $(d_{\mathrm{t}(x_1)}, \dots, d_{\mathrm{t}(x_k)}) \in R$; and $\mathrm{P}(\mathrm{t}) = \overline{0}$ otherwise.

Due to Proposition 5.1.3, it follows that the above operators form an algebra on $\mathbb{K}$-annotators.

5.3 Weighted Variable Set-Automata

In this section, we define the concept of a *weighted vset-automaton* as a formalism to represent $\mathbb{K}$-annotators. This formalism is the natural generalization of vset-automata and weighted automata [39]. Later in this section, we present another formalism, which is based on parametric factors and can be translated into weighted vset-automata (Section 5.3.1).

Let $V \subseteq \mathrm{Vars}$ be a finite set of variables. A *weighted variable-set automaton over semiring* $\mathbb{K}$ (alternatively, a *weighted vset-automaton* or a $\mathbb{K}$-*weighted vset-automaton*)

[5] Recall that a document spanner S is functional, if every tuple uses the same variables, that is, $\mathrm{Vars}(\mathrm{t}) = \mathrm{Vars}(S)$ for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in S(d)$.

[6] Here, $\cup$ stands for the union of two K-relations as was defined previously. The same is valid also for the other operators.

[7] Recall that a *(k-ary) string relation* is the Cartesian product of k languages, that is, $\mathcal{L}_1 \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_k$, with $\mathcal{L}_i \subseteq \Sigma^*$ for all $1 \leq i \leq k$.

is a tuple $A := (\Sigma, V, Q, I, F, \delta)$ where Σ is a finite alphabet; $V \subseteq \text{Vars}$ is a finite set of variables; Q is a finite set of *states*; $I : Q \to \mathbb{K}$ is the *initial weight function*; $F : Q \to \mathbb{K}$ is the *final weight function*; and $\delta : Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma_V) \times Q \to \mathbb{K}$ is a *($\mathbb{K}$-weighted) transition* function.

We define the *transitions* of A as the set of triples (p, o, q) with $\delta(p, o, q) \neq \overline{0}$. Likewise, the *initial* (resp., *accepting*) states are those states q with $I(q) \neq \overline{0}$ (resp., $F(q) \neq \overline{0}$). A *run* ρ of A on ref-word $\mathrm{r} := \sigma_1 \dots \sigma_m$ is a sequence

$$q_0 \overset{\sigma_1}{\to} \cdots \overset{\sigma_{m-1}}{\to} q_{m-1} \overset{\sigma_m}{\to} q_m \ ,$$

where

- $I(q_0) \neq \overline{0}$ and $F(q_m) \neq \overline{0}$;
- $\delta(q_i, \sigma_{i+1}, q_{i+1}) \neq \overline{0}$ for all $0 \leq i < m$.

Recall that, for every semiring element $a \in \mathbb{K}$, we denote the length of the encoding of a by $\|a\|$. The *size* of a weighted vset-automaton A is defined by

$$|A| := |Q| + \sum_{q \in Q} \|I(q)\| + \sum_{q \in Q} \|F(q)\| + \sum_{p,q \in Q,\ a \in (\Sigma \cup \{\varepsilon\} \cup \Gamma_V)} \|\delta(p, a, q)\| \ .$$

Slightly overloading notation, we say that a run ρ is on a document d if ρ is a run on r and $\text{doc}(\mathrm{r}) = d$. Furthermore, again overloading notation, given a run ρ of A on r, we denote r by $\text{ref}(\rho)$. We define the *ref-word language* $\mathcal{R}(A)$ as the set of all ref-words r such that A has a run ρ on r.

The *weight* of a run is obtained by $\otimes$-multiplying the weights of its constituent transitions. Formally, the weight w_ρ of ρ is an element in $\mathbb{K}$ given by the expression

$$I(q_0) \otimes \delta(q_0, \sigma_1, q_1) \otimes \cdots \otimes \delta(q_m, \sigma_m, q_{m+1}) \otimes F(q_{m+1}) \ .$$

We call ρ *nonzero* if $\mathsf{w}_{\text{ref}(\rho)} \neq \overline{0}$. Furthermore, ρ is called *valid* if $\text{ref}(\rho)$ is valid and $\text{Vars}(\text{tup}(\text{ref}(\rho))) = V$.[8] If ρ is valid we denote the tuple $\text{tup}(\text{ref}(\rho))$ by $\text{tup}(\rho)$.

We say that a weighted vset-automaton A is *functional* if $\mathcal{R}(A)$ is functional and $\text{Vars}(\text{tup}(\mathrm{r})) = V$, for every $\mathrm{r} \in \mathcal{R}(A)$. Furthermore, a vset-automaton A satisfies the *variable order condition* if $\mathcal{R}(A)$ satisfies the variable order condition. We denote the set of all valid and nonzero runs of A on d by

$$P(A, d) := \{\rho \mid \text{ref}(\rho) \in \mathcal{R}(A) \text{ and } d = \text{doc}(\text{ref}(\rho))\} \ .$$

Notice that there may be infinitely many valid and nonzero runs of a weighted vset-automaton on a given document, due to *ε-cycles* , which are states $q_1, \dots, q_k$ such that $(q_i, \varepsilon, q_{i+1})$ is a transition for every $i \in \{1, \dots, k-1\}$ and $q_1 = q_k$. Similar to much of the standard literature on weighted automata (see, e.g., [43]) we assume that weighted

[8] Note that the second condition ensures that all runs are over the same set of variables. This is required as $\mathbb{K}$-annotators map documents to annotated relations.

vset-automata do not have ε-cycles, unless mentioned otherwise. The reason for this restriction is that automata with such cycles need $\mathbb{K}$ to be closed under infinite sums for their semantics to be well-defined.[9]

As such, if A does not have ε-cycles, then the result of applying A on a document d, denoted $[\![A]\!]_{\mathbb{K}}(d)$, is the $(\mathbb{K}, d)$-relation R for which

$$R(\mathrm{t}) := \bigoplus_{\rho \in P(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \; .$$

Note that $P(A, d)$ only contains runs ρ that are valid and nonzero. If t is a V'-tuple with $V' \neq V$ then $R(\mathrm{t}) = \overline{0}$, because we only consider valid runs. In addition, $[\![A]\!]_{\mathbb{K}}$ is well defined since every V-tuple in the support of $[\![A]\!]_{\mathbb{K}}(d)$ is a V-tuple over $\mathsf{Spans}(d)$. To simplify notation, we sometimes denote $[\![A]\!]_{\mathbb{K}}(d)(\mathrm{t})$ — the weight assigned to the d-tuple t by A — by $[\![A]\!]_{\mathbb{K}}(d, \mathrm{t})$. We say that two $\mathbb{K}$-weighted vset-automata A_1 and A_2 are *disjoint* if $\mathcal{R}(A_1) \cap \mathcal{R}(A_2) = \emptyset$. This also implies that the corresponding $\mathbb{K}$-annotators $[\![A_1]\!]_{\mathbb{K}}$ and $[\![A_2]\!]_{\mathbb{K}}$ are disjoint. We observe that if the semiring is not positive, there can be ref-words $\mathrm{r} \in \mathcal{R}(A)$ but $R(\mathrm{tup}(\mathrm{r})) = \overline{0}$. This happens, if there are multiple runs encoding the same tuple, which have a total weight of $\overline{0}$.

We say that a $\mathbb{K}$-annotator S is *regular* if there exists a weighted vset-automaton A such that $S = [\![A]\!]_{\mathbb{K}}$. Note that this is an equality between functions. Furthermore, we say that two weighted vset-automata A and A' are equivalent if they define the same $\mathbb{K}$-annotator, that is, $[\![A]\!]_{\mathbb{K}} = [\![A']\!]_{\mathbb{K}}$, which is the case if $[\![A]\!]_{\mathbb{K}}(d) = [\![A']\!]_{\mathbb{K}}(d)$ for every $d \in \Sigma^*$. We denote the set of all *regular* $\mathbb{K}$*-annotators* as $\textsc{Reg}_{\mathbb{K}}$. Similar to our terminology on $\mathbb{B}$-annotators, we use the term $\mathbb{B}$*-weighted vset-automata* to refer to the "classical" vset-automata. Indeed, we observe that the class of functional document spanners is exactly the class of weighted $\mathbb{B}$-annotators. Furthermore, we observe that not every regular spanner, represented by a vset-automaton A, can also be represented by a $\mathbb{B}$-weighted vset-automaton, as the spanner $[\![A]\!]$ might not be functional.

We say that a $\mathbb{K}$-weighted vset-automaton A is *unambiguous* if A satisfies unambiguity conditions (C2) and (C3). That is, A satisfies the following two conditions:

(C2) A satisfies the variable order condition;

(C3) there is exactly one run of A on every ref-word $\mathrm{r} \in \mathcal{R}(A)$.

We note that, over some semirings, the class of unambiguous weighted vset-automata is strictly contained in the class of weighted vset-automata, as shown in the following proposition. However, over the Boolean semiring, every $\mathbb{B}$-weighted automaton can be determinized (Proposition 2.2.6). Therefore there is also an unambiguous $\mathbb{B}$-weighted automaton A_u which is equivalent to A, as every deterministic $\mathbb{B}$-weighted automaton is also unambiguous. We denote the set of all regular $\mathbb{K}$-annotators which can be represented by an unambiguous $\mathbb{K}$-weighted vset-automaton by $\textsc{UReg}_{\mathbb{K}}$.

[9]The semirings need to fulfill additional properties as well such as distributivity, commutativity and associativity must also hold for infinite sums. Such semirings are called *complete* [110].

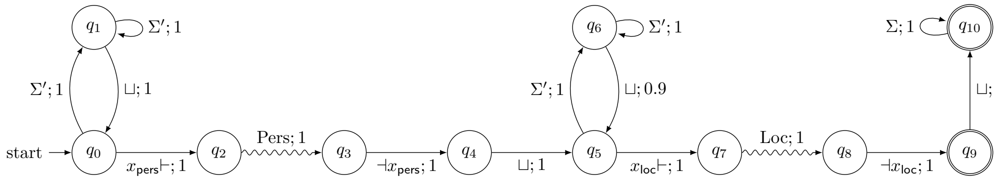

Figure 5.2: An example weighted vset-automaton over the Viterbi semiring with initial state q_0 (with weight 1), two final states q_9, q_{10} (both with weight 1), and alphabet $\Sigma' = \Sigma \setminus \{\sqcup\}$. Pers and Loc are sub-automata matching person and location names respectively. All edges, including the edges of the sub-automata, have the weight 1 besides the transition from q_6 to q_5 with weight 0.9.

Proposition 5.3.1. *$\text{UREG}_{\mathbb{K}} \subsetneq \text{REG}_{\mathbb{K}}$, where $\mathbb{K} = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$ is the tropical semiring.*

Proof. We have to show that there is a $\mathbb{K}$-weighted vset-automaton A such that there is no $\mathbb{K}$-weighted unambiguous vset-automaton A' which is equivalent to A.

Weighted automata can be seen as weighted vset-automata over the empty set of variables. Thus, the statement follows directly from Kirsten [82, Proposition 3.2] who showed that there is a $\mathbb{K}$-weighted automaton A such that there is no equivalent unambiguous $\mathbb{K}$-weighted automaton A'.[10] □

Example 5.3.2. Figure 5.2 shows an example weighted vset-automaton over the Viterbi semiring, which is intended to extract (person, hometown)-tuples from a document. Here, "Pers" and "Loc" should be interpreted as sub-automata that test whether a string could be a person name or a location. (Such automata can be compiled from publicly available regular expressions[11] and from deterministic rules and dictionaries as illustrated in SystemT [22].)

The relation extracted by this automaton from the document in Figure 5.1 is exactly the annotated span relation of the same figure. The weight of a tuple t depends on the number of spaces occurring between the span captured by x_{pers} and the span captured by x_{loc}. More specifically the automaton assigns the weight $(0.9)^k$ to each tuple, where k is the number of words between the two variables. □

As we see next, checking equivalence of weighted vset-automata is undecidable in general.

Proposition 5.3.3. *Given two weighted vset-automata A_1 and A_2 over the tropical semiring, it is undecidable to test whether $[\![A]\!]_{\mathbb{K}} = [\![A']\!]_{\mathbb{K}}$.*

[10] Actually, Kirsten [82, Proposition 3.2] showed an even stronger result. That is, he showed that, the result still holds if A is a polynomially ambiguous weighed automaton, i.e., a weighted automaton for which the number of accepting runs of a word of length n is bound by a fixed polynomial $p(n)$.

[11] For example, `http://regexlib.com/`.

Proof. Follows directly from undecidability of the containment problem of weighted automata over the tropical semiring (cf. Krob [86, Corollary 4.3][12]). ☐

5.3.1 Annotators via Parametric Factors

We now describe another way of introducing weights (or *softness*) in document spanners. In fact, this section can be seen as an additional motivation for $\mathbb{K}$-annotators. Indeed, we will show that, if softness is introduced in document spanners (i.e., $\mathbb{B}$-annotators) in the standard manner that we recall here, the resulting annotators can be captured in our framework.

We introduce softness via the concept of *parametric factors*, which is a very common concept that is used in a wide range of contexts. Examples are the *soft keys* of Jha et al. [72], the *PrDB* model of Sen et al. [149], the *probabilistic unclean databases* of De Sa et al. [134] which can be viewed as a special case of the *Markov Logic Network* (MLN) [132]. Intuitively, a parametric factor is a succinct expression of numerical factors of a probability via weighted rules: whenever the rule *fires*, a corresponding factor (determined by the weight) is added to the product that constitutes the probability. What we want to show in this section is that, if one has rules that involve $\mathbb{B}$-annotators, and one adds uncertainty or softness to these rules in this standard way — using parametric factors — then the obtained formalism naturally leads to $\mathbb{K}$-annotators.

Next, we give the precise definition of a soft spanner and show that, when the factors are regular, a soft spanner can be translated into a weighted vset-automaton.

Formally, a *soft spanner* is a triple $Q = (P, \mathfrak{S}, w)$, where:

- P is a functional document spanner, i.e., a $\mathbb{B}$-annotator,
- $\mathfrak{S}$ is a finite set of functional document spanners referred to as the *factor spanners*, and
- $w : \mathfrak{S} \to \mathbb{Z}$ assigns a (positive or negative) numerical value to each factor spanner.

Given a document d, the soft spanner Q assigns to each $\mathrm{t} \in P(d)$ a probability as follows:

$$\hat{Q}(d, \mathrm{t}) := \exp\left(\sum_{S\in\mathfrak{S}} \sum_{\mathrm{t}'\in\{\mathrm{t}\}\bowtie S(d)} w(S)\right) = \prod_{S\in\mathfrak{S}} e^{w(S)\cdot|\{\mathrm{t}\}\bowtie S(d)|} ,$$
$$Q(d, \mathrm{t}) := \hat{Q}(d, \mathrm{t})/Z(d) ,$$

where $Z(d)$ is a normalization factor (or the *partition function*) defined in the usual way:

$$Z(d) = \sum_{\mathrm{t}\in P(d)} \hat{Q}(d, \mathrm{t}) .$$

Note that $\{\mathrm{t}\} \bowtie S(d)$ is the join of the relation $S(d)$ with the relation that consists of the single tuple t. Hence, $|\{\mathrm{t}\} \bowtie S(d)|$ is the number of tuples $\mathrm{t}' \in S(d)$ that are *compatible* (*joinable*) with t, that is, $\mathrm{t}(x) = \mathrm{t}'(x)$ whenever x is in the domain of both t and t'.

[12] The proof by Krob is quite algebraic. See Almagor et al. [3, Theorem 4] for an automata theoretic proof.

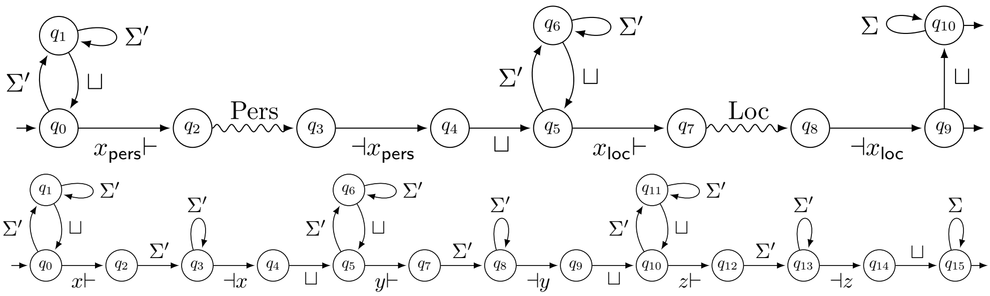

Figure 5.3: Example soft spanner $Q = (P, \{S\}, w)$; where P is represented by the automaton on top (as in Figure 5.2, Pers and Loc are sub-automata matching person and location names respectively), S is defined by the automaton below, renaming x (resp. z) into x_{pers} (resp. x_{loc}), and $w(S) = -1$.

Example 5.3.4. A relation along the lines of the relation as discussed in Example 5.3.2 can be extracted using a soft spanner $Q = (P, \{S\}, w)$. To this end, P is a Boolean spanner extracting (person, hometown)-tuples; S is the spanner, extracting $(x_{\mathsf{pers}}, y, x_{\mathsf{loc}})$-triples of words, where y matches a word between x_{pers} and x_{loc}; and the weight function w is the function assigning $w(S) = -1$. Note that S simply extracts words and does not test whether the words matched by x_{pers} or x_{loc} correspond to a person or location. $\square$

We therefore see that $\mathbb{K}$-annotators can also be defined by applying the standard technique of parametric factors to document spanners. In fact, as we will see next, soft spanners can be compiled into weighted vset-automata, which serves as an additional motivation for weighted vset-automata. To prove this result, we use closure properties of weighted vset-automata that we will obtain further in this chapter (so the proof can be seen as a motivation for the closure and computational properties of weighted vset-automata as well).

Theorem 5.3.5. *Let $Q = (P, \mathfrak{S}, w)$ be a soft spanner such that P and every $S \in \mathfrak{S}$ is regular. There exists an $\mathbb{Z}$-weighted vset-automaton A such that $[\![A]\!]_{\mathbb{Z}}(d, \mathrm{t}) = \log(\hat{Q}(d, \mathrm{t}))$ for all documents d and tuples t. Moreover, A can be constructed in polynomial time in the size of Q if the spanners of Q are represented as unambiguous functional vset-automata.*

Proof. In this proof we will use two results which are shown later in this chapter. That is,

1. Given two unambiguous functional vset-automata A_1, A_2 over the Boolean semiring, one can construct an unambiguous functional vset-automaton A with $[\![A]\!]_{\mathbb{B}} = [\![A_1]\!]_{\mathbb{B}} \bowtie [\![A_2]\!]_{\mathbb{B}}$ in polynomial time (cf. Corollary 5.5.10).
2. Regular annotators are closed under finite union, projection, and finite natural join. Furthermore, the constructions preserve functionality and, if the annotators are given as functional weighted vset-automata, the construction for a single union, projection, and join can be done in polynomial time. (cf. Theorem 5.5.4).

Recall that every regular document spanner can be represented by an unambiguous $\mathbb{B}$-weighted vset-automaton (cf. Observation 2.2.4 and Proposition 2.2.6). Let A_P be an unambiguous $\mathbb{B}$-weighted vset-automaton with $P = [\![A_P]\!]_{\mathbb{B}}$ and, for every $S \in \mathfrak{S}$, let A_S be an unambiguous $\mathbb{B}$-weighted vset-automaton with $S = [\![A_S]\!]_{\mathbb{B}}$. By result (1), there is an unambiguous $\mathbb{B}$-weighted vset-automaton $[\![A_{P\bowtie S}]\!]_{\mathbb{B}} = [\![A_P]\!]_{\mathbb{B}} \bowtie [\![A_S]\!]_{\mathbb{B}}$. Thus, for every document $d \in \Sigma^*$, there is exactly one run for every tuple $\mathrm{t} \in [\![A_{P\bowtie S}]\!]_{\mathbb{B}}(d)$. From $A_{P\bowtie S}$ we compute a weighted vset-automaton $A_{S'}$ by interpreting it as an $\mathbb{Z}$-weighted vset-automaton, such that $A_{S'}$ has a transition $\delta_{S'}(p,o,q) = \overline{1}$ if and only if $\delta(p,o,q) = \mathsf{true}$ is a transition in $A_{P\bowtie S}$. Furthermore, we assign to each accepting state q of $A_{P\bowtie S}$ the weight $F(q) = w(S)$. Therefore, $A_{S'}$ is unambiguous and has exactly one run with weight $w(S)$ for every tuple $\mathrm{t} \in [\![A_{S'}]\!]_{\mathbb{Z}}(d)$. Then the automaton we need for computing $\log(\hat{Q}(d,\mathrm{t}))$ is

$$A = \bigcup_{S\in\mathfrak{S}} \pi_{V_P} [\![A_{S'}]\!]_{\mathbb{Z}} \ .$$

Note that, due to result (2), A actually exists. Recall that every $A_{S'}$ is unambiguous and has exactly one run with weight $w(S)$ for every tuple $\mathrm{t} \in [\![A_{S'}]\!]_{\mathbb{Z}}(d)$. Per definition of union and projection, it follows that $[\![A]\!]_{\mathbb{Z}}(d,\mathrm{t}) = \sum_{S\in\mathfrak{S}} \sum_{\mathrm{t}'\in\{\mathrm{t}\}\bowtie[\![A_{S'}]\!]_{\mathbb{Z}}(d)} w(S) = \log(\hat{Q}(d,\mathrm{t}))$. Observe that, due to (1) and (2), A can be constructed in polynomial time in the size of Q if the spanners of Q are represented as unambiguous functional vset-automata, concluding the proof. □

5.4 Semiring-Encodings

In this section we discuss the encodings of semirings. In order to state complexity results, we need to make some assumptions about the representation and computation of the semiring operations. That is, as mentioned in Section 5.1.1, we assume that semiring elements are encoded in binary, i.e. there is a function $\mathrm{enc} : \mathbb{K} \to \{0,1\}^*$, which assigns a binary encoding to every semiring element. Furthermore, the length of the encoding of an element $a \in \mathbb{K}$ is denoted by $\|a\|$.

Throughout this chapter, we often encode computations into matrix multiplications. To this end, we define *matrix multiplication systems* $\mathit{MMS}_{\mathbb{K}}$ *of dimension* $n \in \mathbb{N}$ as triples $\mathrm{MMS}_{\mathbb{K}} := (I, M, F)$, where $I, F \in \mathbb{K}^n$ are n-dimensional vectors over $\mathbb{K}$ and $M \in \mathbb{K}^{n\times n}$ is a $n \times n$ matrix. We define the *size* of a matrix multiplication system as its dimension plus the sum of the encoding lengths of all semiring elements in the system. That is,

$$|\mathrm{MMS}_{\mathbb{K}}| = n + \sum_{a\in I} \|a\| + \sum_{a\in M} \|a\| + \sum_{a\in F} \|a\| \ .$$

For an $n \times n$ matrix $X \in \mathbb{K}^{n\times n}$ (resp., a vector $X \in \mathbb{K}^n$), we define $\max(X)$ as the maximum of the dimension of X and the biggest representation length of a semiring element in X. More formally,

$$\max(X) := \max(n, \max_{x\in X} \|x\|) \ .$$

Furthermore, for a matrix multiplication system $\mathrm{MMS}_{\mathbb{K}} = (I, M, F)$, we define

$$\max(\mathrm{MMS}_{\mathbb{K}}) = \max(\max(I), \max(M), \max(F)) \ .$$

Let F^T be the transpose of vector F. By $I \times M$ we denote the matrix multiplication of I and M. We define *efficient semiring encodings* as follows.

Definition 5.4.1. Let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring. The encoding of $\mathbb{K}$ is *efficient* if, given a matrix multiplication system $\mathrm{MMS}_{\mathbb{K}}$ and a natural number k, the semiring elements

$$w_i := I \times M^i \times F^T \ ,$$

for all $0 \leq i \leq k$, and

$$w := \bigoplus_{1 \leq i \leq k} w_i$$

can be computed in time polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$.

Throughout this chapter, whenever we give complexity bounds, we assume that an efficient encoding of the semiring is used. As we show now, the standard encodings of most of the semirings in Example 5.1.1 are efficient.

Proposition 5.4.2. *Let* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be a semiring. Then the encoding of* $\mathbb{K}$ *is efficient if for all semiring elements* $a, b \in \mathbb{K}$*,* $a \oplus b$ *and* $a \otimes b$ *can be computed in time polynomial in* $\|a\| + \|b\|$ *and*

$$\|a \oplus b\| \leq \max(\|a\|, \|b\|) + 1, \textit{and}$$
$$\|a \otimes b\| \leq \|a\| + \|b\| \ .$$

Proof. Let $\mathrm{MMS}_{\mathbb{K}} = (I, M, F)$ be a matrix multiplication system of dimension n and $k \in \mathbb{N}$ be a natural number. Let $w_1, \ldots, w_k$ and w be as defined in the definition of efficient semiring encodings (Definition 5.4.1).

We observe that the computation of w requires a polynomial number of additions and multiplications. However, as the encoding of the semiring elements, which are used for the computation, might become large, this does not immediately imply that w can be computed in time polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$.

We therefore show, for every $1 \leq i \leq k$, that the semiring elements, which are required for the computation of w_i have an encoding of size at most polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$. Due to the assumption that $\|a \oplus b\| \leq \max(\|a\|, \|b\|) + 1$, this immediately implies that $\|w\|$ is polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$, which concludes the proof.

Recall that $\max(M)$ is the maximum of the dimension n of M and the representation size of the element in M with the largest representation. We begin by showing that, $\max(I \times M) \leq \max(I) + \max(M) + n$ for all vectors $I \in \mathbb{K}^n$ and matrices $M \in \mathbb{K}^{n \times n}$. Let x be an element of $I \times M$. Per definition of matrix multiplication, x is the sum of n elements $x_1, \ldots, x_n$, each of which is the product of an element from I and an element from M. Thus

$$\|x_i\| \leq \max(I) + \max(M)$$

and therefore,
$$\|x\| \leq \max(I) + \max(M) + n \ .$$
We can conclude that $\max(I \times M) \leq \max(I) + \max(M) + n$ for all vectors $I \in \mathbb{K}^n$ and matrices $M \in \mathbb{K}^{n \times n}$.

We now show by induction that, for all $i \in \mathbb{N}$, it holds that

$$\max(I \times M^i \times F^T) \leq \max(I) + i \cdot (\max(M) + n) + \max(F) + n \ ,$$

for all vectors $I, F \in \mathbb{K}^n$ and all matrices $M \in \mathbb{K}^{n \times n}$. We observe that, due to $\max(I) + \max(M) + \max(F) + n \leq |\mathrm{MMS}_{\mathbb{K}}|$, this implies that $\|w_i\|$ is polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$, for all $0 \leq i \leq k$.

For the base case, we observe that $w_0 = I \times F$ is the sum of n elements, each of which has size at most $\max(I) + \max(F)$. As desired, we therefore have that

$$\max(I \times F) = \|w_0\| \leq \max(I) + \max(F) + n \ .$$

For the inductive step, assume there is an $i \in \mathbb{N}$ such that,

$$\max(I \times M^i \times F^T) \leq \max(I) + i \cdot (\max(M) + n) + \max(F) + n \ ,$$

for all vectors $I, F \in \mathbb{K}^n$ and all matrices $M \in \mathbb{K}^{n \times n}$. With $I' := I \times M$ we have that

$$\max(I') = \max(I \times M) \leq \max(I) + \max(M) + n$$

and therefore,

$$\begin{aligned}\max(I \times M^{i+1} \times F^T) &= \max(I' \times M^i \times F^T)\\ &\leq \max(I') + i \cdot (\max(M) + n) + \max(F) + n\\ &\leq \max(I) + \max(M) + n + i \cdot (\max(M) + n) + \max(F) + n\\ &= \max(I) + (i+1) \cdot (\max(M) + n) + \max(F) + n \ .\end{aligned}$$

This concludes the proof. □

Note that all semirings over a finite domain have an efficient encoding, as each semiring element can be encoded with constant size and all operations can be carried out in constant time via a constant size lookup table.

Corollary 5.4.3. *A semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *has an efficient encoding, if its domain is finite.* □

We observe that for many semirings, the standard encodings satisfy the conditions of Proposition 5.4.2. For instance the numeric semiring $(\mathbb{Z}, +, \cdot, 0, 1)$, the counting semiring, Boolean semiring, the Viterbi semiring (over the rationals $\mathbb{Q}$), the access control semiring, and the tropical semirings. However, for some semirings standard encodings of the semiring elements do not satisfy the conditions of Proposition 5.4.2. For example, consider the numeric semiring $(\mathbb{Q}, +, \cdot, 0, 1)$ and the encoding, where every semiring element $a = \frac{n}{d}$ is encoded by its numerator $n \in \mathbb{Z}$ and its denominator $d \in \mathbb{N}$. The problem is that the sum of two rational numbers $\frac{a}{b}, \frac{c}{d}$ is given by $x = \frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + b \cdot c}{b \cdot d}$ and therefore the size of the encoding of x is $\|x\| \leq \|a \cdot d\| + \|b \cdot c\| + \|c \cdot d\|$ which, in general, does not satisfy the condition that $\|\frac{a}{b} + \frac{c}{d}\| \leq \max(\|\frac{a}{b}\|, \|\frac{c}{d}\|) + 1$. Even though this only increases the size of the representation by a small margin, it forces us to study the complexity in more depth to conclude that this encoding is efficient.

Proposition 5.4.4. *The numeric semiring* $(\mathbb{Q}, +, \cdot, 0, 1)$ *has an efficient encoding.*

Proof. Let $(\mathbb{Q}, +, \cdot, 0, 1)$ be the numeric semiring. We assume that every semiring element $x = \frac{a}{b}$ is encoded by its numerator $a \in \mathbb{Z}$ and its denominator $b \in \mathbb{N}$. Let all numerators and denominators be encoded in binary, where two's complement encoding is used for the numerators. We observe that Proposition 5.4.2 holds for both encodings. Furthermore, the encoding of the denominators is monotone, that is, for every $x, y \in \mathbb{N}$ it holds that $\|x\| \leq \|y\|$ if $x \leq y$.

Given a matrix multiplication system $\mathrm{MMS}_{\mathbb{K}} = (I, M, F)$, let D be the set of all denominators of the rationals in I, F and M. We will compute the least common multiple d_{lcm} of all denominators in D and expand the representations of all numbers to the denominator d_{lcm}. Observe that all denominators $d \in D$ are natural numbers. Therefore,

$$\|d_{\mathrm{lcm}}\| \overset{(1)}{\leq} \Big\| \prod_{d \in D} d \Big\| \overset{(2)}{\leq} \sum_{d \in D} \|d\| \overset{(3)}{\leq} |\mathrm{MMS}_{\mathbb{K}}| \ ,$$

where (1) follows from $d_{\mathrm{lcm}} \leq \prod\limits_{d \in D} d$ and the monotonicity of the encoding of the denominators, (2) follows from $\|x \otimes y\| \leq \|x\| + \|y\|$, and (3) follows from the definition of $|\mathrm{MMS}_{\mathbb{K}}|$. Therefore, $\|d_{\mathrm{lcm}}\| \leq |\mathrm{MMS}_{\mathbb{K}}|$ is polynomial in $|\mathrm{MMS}_{\mathbb{K}}|$. Furthermore, the computation of d_{lcm} as well as the expansion can be done in polynomial time.[13] We therefore assume, w.l.o.g., that all rationals in I, F, and M have the denominator d_{lcm}.

Let $I_{\mathbb{Z}}, F_{\mathbb{Z}} \in \mathbb{Z}^n$ and $M_{\mathbb{Z}} \in \mathbb{Z}^{n \times n}$ be the vectors I, F and the matrix M where all numbers are replaced by it's numerator. For all $1 \leq i \leq k$, we define

$$w_{\mathbb{Z},i} := I_{\mathbb{Z}} \times M_{\mathbb{Z}}^i \times F_{\mathbb{Z}}^T \ .$$

We recall that, due to Proposition 5.4.2, $w_{\mathbb{Z},i}$ can be computed in time polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$. Per assumption that all rationals in I, F and M have the denominator d_{lcm}, we have that $w_i = \frac{w_{\mathbb{Z},i}}{d_{\mathrm{lcm}}^{i+2}}$. Furthermore, the denominator can also be computed in time polynomial in $|\mathrm{MMS}_{\mathbb{K}}|$, as $\|d_{\mathrm{lcm}}\|$ is polynomial in $|\mathrm{MMS}_{\mathbb{K}}|$ and $\|x \otimes y\| \leq \|x\| + \|y\|$ for the encoding of natural numbers. Thus, for all $i \leq k$, w_i can be computed in time polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$. Furthermore, w can be computed in time polynomial in $|\mathrm{MMS}_{\mathbb{K}}| \cdot k$ by first expanding all w_i to the denominator d_{lcm}^{k+2} and summing up the expanded fractions. This concludes the proof. $\square$

[13] The least common multiple can be computed using the Eucledian algorithm and the expansion of $x = \frac{a}{b}$ by multiplying the numerator a by $\frac{b}{d_{\mathrm{lcm}}}$.

5.5 Fundamental Properties

We now study fundamental properties of annotators. Specifically, we show that regular annotators are closed under union, projection, and join. Furthermore, annotators over a semiring $\mathbb{K}$ behave the same as document spanners with respect to string selection if $\mathbb{K}$ is positive or $\oplus$ is bipotent[14] and for every $a, b \in \mathbb{K}$, $a \otimes b = \overline{1}$ implies that $a = b = \overline{1}$.

5.5.1 Epsilon Elimination

We begin the section by showing that every regular $\mathbb{K}$-annotator can be transformed into an equivalent functional regular $\mathbb{K}$-annotator without ε-transitions.

Proposition 5.5.1. *For every weighted vset-automaton A there is an equivalent weighted vset-automaton A' that has no ε-transitions. This automaton A' can be constructed from A in polynomial time. Furthermore, A is functional if and only if A' is functional.*

Proof. We use a result by Mohri [110, Theorem 7.1] who showed that, given a weighted automaton, one can construct an equivalent weighted automaton without epsilon transitions.

More precisely, let $A = (\Sigma, V, Q, I, F, \delta)$ be a weighted vset-automaton. Notice that A can also be seen as an ordinary weighted finite state automaton $B = (\Sigma \cup \Gamma_V, Q, I, F, \delta)$. In this automaton, one can remove epsilon transitions by using Mohri's epsilon removal algorithm [110, Theorem 7.1]. The resulting ε-transition free automaton $B' = (\Sigma \cup \Gamma_V, Q', I', F', \delta')$ accepts the same strings as B. Therefore, interpreting B' as an weighted vset-automaton $A' = (\Sigma, V, Q', I', F', \delta')$ we have that $[\![A]\!]_{\mathbb{K}} = [\![A']\!]_{\mathbb{K}}$ and A' is functional if and only if A is functional.

Concerning complexity, Mohri shows that this algorithm runs in polynomial time, assuming that weighted-ε-closures can be computed in polynomial time. However, in our setting this is obvious as we do not allow ε-cycles. Therefore, the weight of an element of an ε-closure can be computed by at most n matrix multiplications, where n is the number of states in A. Per assumption that $\mathbb{K}$ has an efficient encoding, these matrix multiplications can be computed in polynomial time. $\square$

5.5.2 Functionality

Non-functional vset-automata are inconvenient to work with, since some of their nonzero runs are not valid and therefore do not contribute to the weight of a tuple. It is therefore desirable to be able to automatically convert weighted vset-automata into functional weighted vset-automata.

Proposition 5.5.2. *Let A be a weighted vset-automaton. Then there is a functional weighted vset-automaton A_{fun} that is equivalent to A. If A has n states and uses k variables, then A_{fun} can be constructed in time polynomial in n and exponential in k. Furthermore, the construction preserves unambiguity.*

[14] Recall, $\oplus$ is bipotent, if $a \oplus b \in \{a, b\}$, for every $a, b \in \mathbb{K}$.

Proof. The proof follows the idea of a similar result by Freydenberger [52, Proposition 3.9] for unweighted vset-automata. Like Freydenberger, we associate each state in A_{fun} with a function $s : V \to \{w, o, c\}$, where $s(x)$ represents the following:

- w stands for "waiting", meaning $x\vdash$ has not been read,
- o stands for "open", meaning $x\vdash$ has been read, but not $\dashv x$,
- c stands for "closed", meaning $x\vdash$ and $\dashv x$ have been read.

Let S be the set of all such functions. Observe that $|S| = 3^{|V|}$. We now define $A_{\text{fun}} := (\Sigma, V, Q_{\text{fun}}, I_{\text{fun}}, F_{\text{fun}}, \delta_{\text{fun}})$ as follows:

$$Q_{\text{fun}} := Q \times S\ ;$$
$$I_{\text{fun}}(p, s) := \begin{cases} I(p) & \text{where } s(x) = w \text{ for all } x \in V \\ \overline{0} & \text{otherwise;} \end{cases}$$
$$F_{\text{fun}}(p, s) := \begin{cases} F(p) & \text{where } s(x) = c \text{ for all } x \in V \\ \overline{0} & \text{otherwise.} \end{cases}$$

Furthermore, for all $(p, s) \in Q_{\text{fun}}$ and $x \in V$ we define s_o^x (resp., s_c^x) as $s_o^x(x) := o$ (resp., $s_c^x := c$) and $s_o^x(y) := s(y)$ (resp., $s_c^x(y) = s^x(y)$) for all $x \neq y$. Using these, we define

$$\begin{aligned} \delta_{\text{fun}}((p, s), a, (q, s)) &= \delta(p, a, q) && \text{for } a \in (\Sigma \cup \{\varepsilon\}), \\ \delta_{\text{fun}}((p, s), x\vdash, (q, s_o^x)) &= \delta(p, x\vdash, q) && \text{if } s(x) = w, \\ \delta_{\text{fun}}((p, s), \dashv x, (q, s_c^x)) &= \delta(p, \dashv x, q) && \text{if } s(x) = o, \\ \delta_{\text{fun}}((p, s), a, (q, t)) &= \overline{0} && \text{otherwise.} \end{aligned}$$

Functionality follows analogously to Freydenberger [52, Proposition 3.9]. It remains to show equivalence, i.e., that for every document $d \in \Sigma^*$ it holds that $[\![A]\!]_{\mathbb{K}}(d) = [\![A_{\text{fun}}]\!]_{\mathbb{K}}(d)$. Observe that there is a one-to-one correspondence between valid nonzero runs $\rho \in P(A, d)$ and valid nonzero runs $\rho_{\text{fun}} \in P(A_{\text{fun}}, d)$ with $\mathsf{w}_\rho = \mathsf{w}_{\rho_{\text{fun}}}$. Therefore, $[\![A]\!]_{\mathbb{K}}(d) = [\![A_{\text{fun}}]\!]_{\mathbb{K}}(d)$ must also hold and the construction preserves unambiguity. □

The exponential blow-up in Proposition 5.5.2 cannot be avoided, since it already occurs for weighted vset-automata over the Boolean semiring.[15] Functionality of vset-automata can be checked efficiently, as we have the following result.

Proposition 5.5.3. *Given a $\mathbb{K}$-weighted vset-automaton A with m transitions and k variables, it can be decided whether A is functional in time $O(km)$. Furthermore, A is functional if and only if it is functional when interpreted as $\mathbb{B}$-weighted document spanner.*

Proof. Per definition, a weighted vset-automaton is functional if all runs are valid. Furthermore, a run on a ref-word r is valid if $\text{Vars}(\text{tup}(\text{r})) = V$, where $V \subseteq \text{Vars}$ is the set

[15] Freydenberger [52, Proposition 3.9] showed that there is a class of vset-automata $\{A_k \mid k \in \mathbb{N}\}$, each with one state and k, such that every functional vset-automaton equivalent to A_k has at least 3^k states.

of variables of V. Observe that this definition only depends on the ref-word's and not on the semiring of the automaton. Therefore, a $\mathbb{K}$-weighted vset-automaton A is functional if and only if A is functional when interpreted as an $\mathbb{B}$-weighted vset-automaton $A^{\mathbb{B}}$. More formally, let $A^{\mathbb{B}}$ be the $\mathbb{B}$-weighted vset-automaton obtained by replacing nonzero weights with true, sum by $\vee$ and multiplication by $\wedge$. The result now follows directly from Freydenberger [52, Lemma 3.5], who showed that it can be verified in $O(km)$ whether a vset-automaton is functional. $\square$

5.5.3 Closure Under Join, Union, and Projection

We will obtain the following result.

Theorem 5.5.4. *Regular annotators are closed under finite union, projection, and finite natural join. Furthermore, if the annotators are given as functional weighted vset-automata, the construction for a single union, projection, and join can be done in polynomial time. All constructions preserve functionality.*

The theorem follows immediately from Lemmas 5.5.5, 5.5.6, and 5.5.9. Whereas the constructions for union and projection are fairly standard, the case of join needs some care in the case that the two automata A_1 and A_2 process variable operations in different orders.[16]

Lemma 5.5.5. *Given two $\mathbb{K}$-weighted vset-automata A_1 and A_2 with $V_1 = V_2$, one can construct a weighted vset-automaton A in linear time, such that $[\![A]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \cup [\![A_2]\!]_{\mathbb{K}}$. Furthermore, A is unambiguous if A_1 and A_2 are unambiguous and disjoint.*

Proof. Let $A_1 := (\Sigma, V, Q_1, I_1, F_1, \delta_1)$ and $A_2 := (\Sigma, V, Q_2, I_2, F_2, \delta_2)$, such that $Q_1 \cap Q_2 = \emptyset$. We construct an automaton $A := (\Sigma, V, Q, I, F, \delta)$, such that $[\![A]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \cup [\![A_2]\!]_{\mathbb{K}}$. To this end, let $Q = Q_1 \cup Q_2$, be the set of states, $I, F : Q \to \mathbb{K}$ with $I(q) = I_i(q)$ and $F(q) = F_i(q)$ for $q \in Q_i$ and $i \in \{1, 2\}$. Let $\delta(p, a, q) = \delta_i(p, a, q)$ if $p, q \in Q_i$, for $i \in \{1, 2\}$, and $\delta(p, a, q) = \overline{0}$ if p, q are not from the state set of the same automaton. Observe that this construction can be carried out in linear time. It remains to show the correctness of the construction. We observe that there are no nonzero transitions between states in Q_1 and Q_2, thus no nonzero run ρ of A can have states p, q such that $p \in Q_1$ and $q \in Q_2$. Let $d \in \Sigma^*$ be an arbitrary document. The set $P(A, d)$ of all valid and nonzero runs of A on d is the union of two sets $P_1(A, d), P_2(A, d)$, where a run ρ is in $P_i(A, d)$ if it consists of states in Q_i. Furthermore, for $i \in \{1, 2\}$ it holds that $\rho \in P_i(A, d)$

[16] More formally, if A_1 processes $x{\vdash}\, y{\vdash}\, a \dashv y \dashv x$ and A_2 processes $y{\vdash}\, x{\vdash}\, a \dashv x \dashv y$, then these two different sequences produce different encodings of the same tuple. This has to be considered by the automata construction.

if and only if $\rho \in P(A_i, d)$ and therefore,

$$\begin{aligned}
[\![A]\!]_{\mathbb{K}}(d, \mathrm{t}) = & \bigoplus_{\rho \in P(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \\
= & \left(\bigoplus_{\rho \in P_1(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \right) \oplus \left(\bigoplus_{\rho \in P_2(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \right) \\
= & \left(\bigoplus_{\rho \in P(A_1,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \right) \oplus \left(\bigoplus_{\rho \in P(A_2,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \right) \\
= & [\![A_1]\!]_{\mathbb{K}}(d, \mathrm{t}) \oplus [\![A_2]\!]_{\mathbb{K}}(d, \mathrm{t}) \; .
\end{aligned}$$

This concludes the proof that $[\![A]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \cup [\![A_2]\!]_{\mathbb{K}}$.

It remains to show that A is unambiguous if A_1 and A_2 are unambiguous and disjoint. We observe that $\mathcal{R}(A) = \mathcal{R}(A_1) \cup \mathcal{R}(A_2)$. Thus, the variable order condition condition (C2) must be satisfied as both A_1 and A_2 satisfy the variable order condition. Furthermore, due to the disjointness of A_1 and A_2 it must hold that $\mathcal{R}(A_1) \cap \mathcal{R}(A_2) = \emptyset$ and therefore the unambiguity condition (C3) must also be satisfied. □

Lemma 5.5.6. *Given a $\mathbb{K}$-weighted vset-automaton A and a subset $X \subseteq V$ of the variables V of A, there exists a weighted vset-automaton A' with $[\![A']\!]_{\mathbb{K}} = \pi_X [\![A]\!]_{\mathbb{K}}$. Furthermore, if A is functional, then A' can be constructed in polynomial time.*

Proof. Let $A := (\Sigma, V, Q, I, F, \delta)$ and $V^- = V \setminus X$. If A is not yet functional, we can assume by Proposition 5.5.2 that it is, at exponential cost in the number of variables of A. Furthermore, assume that, for every nonzero transition, there is a run ρ which uses the transition. Due to A being functional, we will be able to construct A' by replacing all transitions labeled with a variable operation $o \in \Gamma_{V^-}$ with an ε-transition of the same weight. More formally, let $A' := (\Sigma, X, Q, I, F, \delta')$, such that

- $\delta'(p, o, q) = \delta(p, o, q)$ for all $p, q \in Q$ and $o \in \Sigma \cup \{\varepsilon\} \cup \Gamma_X$, and
- $\delta'(p, \varepsilon, q) = \delta(p, o, q)$ for all $p, q \in Q$ and $o \in \Gamma_{V^-}$.

Observe that A' can be constructed in polynomial time if A is functional. We argue why δ' is well defined. Towards a contradiction, assume that δ' is not well-defined. This can only happen if A has two transitions $\delta(p, o_1, q)$ and $\delta(p, o_2, q)$ with $o_1, o_2 \in \Gamma_{V^-} \cup \{\varepsilon\}$ and $o_1 \neq o_2$. Therefore, there are two runs ρ_1, ρ_2, which use $\delta(p, o_1, q)$ and $\delta(p, o_2, q)$ respectively. Let ρ_1' be the same as ρ_1, however, using $\delta(p, o_2, q)$ instead of $\delta(p, o_1, q)$. Therefore, since $o_1 \neq o_2$ and $o_1, o_2 \in \Gamma_{V^-} \cup \{\varepsilon\}$, either ρ_1 or ρ_1' are not valid, contradicting functionality of A.

It remains to show that $[\![A']\!]_{\mathbb{K}} = \pi_X [\![A]\!]_{\mathbb{K}}$. Let $d \in \Sigma^*$ be an arbitrary document. Every run ρ of A selecting t on d corresponds to exactly one run ρ' of A' selecting t′ on d such that $\mathrm{t}' = \pi_X \mathrm{t}$ and $\mathsf{w}_\rho = \mathsf{w}_{\rho'}$. Therefore,

$$\begin{aligned}
\pi_X [\![A]\!]_{\mathbb{K}}(d)(\mathrm{t}') &= \bigoplus_{\mathrm{t}'=\pi_X \mathrm{t} \text{ and } [\![A]\!]_{\mathbb{K}}(d,\mathrm{t}) \neq \overline{0}} [\![A]\!]_{\mathbb{K}}(d,\mathrm{t}) \\
&= \bigoplus_{\mathrm{t}'=\pi_X \mathrm{t} \text{ and } [\![A]\!]_{\mathbb{K}}(d,\mathrm{t}) \neq \overline{0}} \; \bigoplus_{\rho \in P(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \\
&= \bigoplus_{\rho' \in P(A',d) \text{ and } \mathrm{t}'=\mathrm{tup}(\rho')} \mathsf{w}_{\rho'} \\
&= [\![A']\!]_{\mathbb{K}}(d)(\mathrm{t}') \;.
\end{aligned}$$

Therefore, $[\![A']\!]_{\mathbb{K}} = \pi_X [\![A]\!]_{\mathbb{K}}$. $\square$

We will now show that regular annotators are closed under join. Freydenberger et al. [54, Lemma 3.10], showed that, given two functional $\mathbb{B}$-weighted vset-automata A_1 and A_2, one can construct a functional vset-automaton A with $[\![A]\!]_{\mathbb{B}} = [\![A_1]\!]_{\mathbb{B}} \bowtie [\![A_2]\!]_{\mathbb{B}}$ in polynomial time. The construction is based on the classical product construction for the intersection of NFAs. However, A_1 and A_2 can process consecutive variable operations in different orders which must be considered during the construction. To deal with this issue, we adapt and combine multiple constructions from literature.

To be precise, we adopt so called *extended vset-automata* as defined by Amarilli et al. [6][17]. An extended $\mathbb{K}$-weighted vset-automaton on alphabet Σ and variable set V is an automaton $A_E = (\Sigma, V, Q, I, F, \delta)$, where $Q = Q_v \uplus Q_\ell$ is a disjoint union of *variable states* Q_v and *letter states* Q_ℓ. Furthermore, $I : Q \to \mathbb{K}$ is the initial weight function, $F : Q \to \mathbb{K}$ the final weight function, and $\delta : Q \times (\Sigma \cup 2^{\Gamma_V}) \times Q \to \mathbb{K}$ is a transition function, such that transitions labeled by $\sigma \in \Sigma$ originate in letter states and terminate in variable states and $T \subseteq \Gamma_V$ transitions originate in variable states and terminate in letter states. More formally, for every $\sigma \in \Sigma$, it holds that $\delta(p, \sigma, q) = \overline{0}$ if $p \in Q_v$ and $q \in Q_\ell$. Furthermore, for every $T \subseteq \Gamma_V$, $\delta(p, T, q) = \overline{0}$ if $p \in Q_\ell$ and $q \in Q_v$. The weight w_ρ of a run ρ on a weighted extended vset-automaton, $[\![A_E]\!]_{\mathbb{K}}$, and functionality are defined analogously to the weighted vset-automata. Furthermore, an extended weighted vset-automaton A_E is unambiguous, if all runs encode a different tuple, that is, for every two runs $\rho_1 \neq \rho_2$ of A_E it holds that $\mathrm{tup}(\rho_1) \neq \mathrm{tup}(\rho_2)$. We observe that we do not need to enforce the variable order condition for unambiguous extended weighted vset-automata, as consecutive variable operations are encoded into a single transition.

Proposition 5.5.7. *For every functional weighted vset-automaton A, there exists an equivalent functional extended weighted vset-automaton A_E and vice versa. Given an automaton in one model, one can construct an automaton in the other model in polynomial time. Furthermore, the conversion preserves unambiguity.*

Proof. Let $A := (\Sigma, V, Q, I, F, \delta)$ be a weighted functional vset-automaton.

[17] Extended vset-automata were first introduced by Florenzano et al. [48], but the definition of Amarilli et al. [6] is more convenient for us.

Due to Proposition 5.5.3 a weighted vset-automaton is functional if and only if the automaton A interpreted as $\mathbb{B}$-weighted vset-automaton is functional. For functional vset-automata it is well known[18] that there is a function $s : Q \times V \to \{w, o, c\}$, where

- $s(q, v) = w$ stands for "waiting", meaning that no run ρ of A such that $v\vdash$ is read before reaching state q.
- $s(q, v) = o$ stands for "open", meaning that all runs ρ of A read $v\vdash$ but not $\dashv v$ before reaching state q.
- $s(q, v) = c$ stands for "closed", meaning that all runs ρ of A read $v\vdash$ and $\dashv v$ before reaching state q.

Furthermore, based on s, we define the function $S_T : Q \times Q \to \Gamma_V$, such that $S_T(q, q')$, if on every run ρ of A which visits q' after q, the variable operations $S_T(q, q')$ must be read between q and q'. More formally, $x\vdash \in S_T(p, q)$ if and only if $s(p, x) = w$ and $s(q, x) \neq w$ and $\dashv x \in S_T(p, q)$ if and only if $s(p, x) \neq c$ and $s(q, x) = c$.

We assume, w.l.o.g., that the states of A are $\{1, \ldots, n\}$ for some $n \in \mathbb{N}$. For every state $i \in Q$, we define the vector V_i where

$$V_i(j) = \begin{cases} \overline{0} & \text{if } i \neq j \\ \overline{1} & \text{if } i = j \;. \end{cases}$$

Furthermore, we define the $n \times n$ matrix $M_{p,q}$ where

$$M_{p,q}(i, j) = \begin{cases} \delta(i, o, j) & \text{if } o \in S_T(p, q) \\ \overline{1} & \text{otherwise} \;. \end{cases}$$

We construct the weighted extended functional vset-automaton $A_E := (\Sigma, V, Q_\ell \cup Q_v, I_E, F_E, \delta_E)$ as follows. Let $Q_\ell := \{q_\ell \mid q \in Q\}$ and $Q_v := \{q_v \mid q \in Q\}$ be two disjoint copies of the states of A. Furthermore, let

$$I_E(q) := \begin{cases} I(q) & \text{if } q \in Q_v \\ \overline{0} & \text{if } q \in Q_\ell \;; \end{cases}$$

$$F_E(q) := \begin{cases} \overline{0} & \text{if } q \in Q_v \\ F(q) & \text{if } q \in Q_\ell \;. \end{cases}$$

We define δ_E as follows

$$\begin{aligned} \delta_E(p_l, \sigma, q_v) &:= \delta(p, \sigma, q) && \text{for all } \sigma \in \Sigma, \\ \delta_E(p_v, O, q_\ell) &:= V_{p_v} \times (M_{p_v,q_\ell})^{|O|} \times V_{q_\ell}^T && \text{for } O = S_T(p_v, q_\ell) \\ \delta_E(p_v, \emptyset, p_\ell) &:= \overline{1} \end{aligned}$$

[18] For example, compare Freydenberger [52], Freydenberger et al. [54].

We observe that per assumption that $\mathbb{K}$ has an efficient encoding, it follows that A_E can be constructed in polynomial time. It remains to show that $[\![A]\!]_{\mathbb{K}} = [\![A_E]\!]_{\mathbb{K}}$. To this end, we define a function, which maps valid runs of A to runs of A_E. More formally, let

$$\rho = q_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} q_{m-1} \xrightarrow{\sigma_m} q_m$$

be a run of A on $\mathrm{r} = \sigma_1 \cdots \sigma_m$. Furthermore, let $d = d_1 \cdots d_n = \mathrm{doc}(\mathrm{r})$, $\mathrm{t} := \mathrm{tup}(\mathrm{r})$, and, for $1 \leq i \leq n+1$, let

$$T_i = \{x\vdash \mid \mathrm{t}(x) = [i,j\rangle \text{ for some } i \leq j \leq n+1\} \cup \{\dashv x \mid \mathrm{t}(x) = [j,i\rangle \text{ for some } 1 \leq j \leq i\}\ .$$

Let $q_v^0 \in Q_v$ (resp., $q_\ell^{n+1} \in Q_\ell$) be the variable state (resp., letter state) corresponding to q_0 (resp., q_m). Furthermore, for $1 \leq k \leq n$, let q_ℓ^{k-1}, q_v^k be the states corresponding to the states visited by ρ while reading the symbol d_k. That is, for $q_j \xrightarrow{d_{a+1}} q_{j+1}$ in ρ , q_ℓ^{k-1} corresponds to q_j and q_v^k to q_{j+1}. We observe that, due to A being functional, it must hold that $T_i = S_T(q_v^{i-1}, q_\ell^{i-1})$.

We define $f(\rho)$ as the run $\rho_E \in P(A_E, d)$ such that

$$\rho_E = q_v^0 \xrightarrow{T_1} q_\ell^0 \xrightarrow{d_1} q_v^1 \xrightarrow{T_2} \cdots \xrightarrow{d_n} q_v^n \xrightarrow{T_{n+1}} q_\ell^n\ .$$

For every valid run $\rho_E \in P(A_E, d)$, it holds that $\mathsf{w}_{\rho_E} = \bigoplus_{\rho \in P(A,d) \text{ with } f(\rho)=\rho_E} \mathsf{w}_\rho$. Therefore, it follows that

$$\begin{aligned}
[\![A_E]\!]_{\mathbb{K}}(d,\mathrm{t}) &= \bigoplus_{\rho_E \in P(A_E,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho_E)} \mathsf{w}_{\rho_E} \\
&= \bigoplus_{\rho_E \in P(A_E,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho_E)} \ \bigoplus_{\rho \in P(A,d) \text{ with } f(\rho)=\rho_E} \mathsf{w}_\rho \\
&= \bigoplus_{\rho \in P(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \\
&= [\![A]\!]_{\mathbb{K}}(d,\mathrm{t})\ .
\end{aligned}$$

It remains to show that A_E is unambiguous if A is unambiguous. To this end, assume that A_E is not unambiguous. Thus, there must be two runs $\rho_E^1 \neq \rho_E^2$ on A_E, encoding the same tuple. By construction of A_E, there must be two runs $\rho_1 \neq \rho_2$ of A which encode the same tuple. Due to the variable order condition (C2), $\mathrm{ref}(\rho_1) = \mathrm{ref}(\rho_2)$, however this contradicts the unambiguity condition (C3). Therefore A_E must be unambiguous.

For the other direction, one can construct a weighted vset-automaton A with ε-transitions[19], by replacing every edge $\delta(p, O, q) = w$ by a sequence of transitions $\delta(p, v_1, q_1) = w$, $\delta(q_1, v_2, q_2) = \overline{1}, \ldots, \delta(q_{n-1}, v_n, q) = \overline{1}$, where $O = \{v_1, \ldots, v_n\}$, with $v_1 \prec v_2 \prec \cdots \prec v_n$, and $q_1, \ldots, q_{n-1}$ are new states. We observe that only the first transition has weight w, whereas all other transitions have weight $\overline{1}$. This construction also runs in polynomial time and it is straightforward to verify that $[\![A]\!]_{\mathbb{K}} = [\![A_E]\!]_{\mathbb{K}}$ and that A is unambiguous if A_E is unambiguous. $\square$

[19]By Proposition 5.5.1, the ε-transitions can be removed in polynomial time.

Proposition 5.5.8. *Let A_1, A_2 be two functional extended $\mathbb{K}$-weighted vset-automata. One can construct a functional extended $\mathbb{K}$-weighted vset-automaton A in polynomial time, such that $[\![A]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \bowtie [\![A_2]\!]_{\mathbb{K}}$. Furthermore, A is unambiguous if A_1 and A_2 are unambiguous.*

Proof. Let $A_1 = (\Sigma, V_1, Q_1, I_1, F_1, \delta_1)$ and $A_2 = (\Sigma, V_2, Q_2, I_2, F_2, \delta_2)$ be two $\mathbb{K}$-weighted extended functional vset-automata. We construct a $\mathbb{K}$-weighted extended functional vset-automaton $A = (\Sigma, V_1 \cup V_2, Q_1 \times Q_2, I, F, \delta)$ such that $[\![A]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \bowtie [\![A_2]\!]_{\mathbb{K}}$. To this end, let $I(q_1, q_2) = I_1(q_1) \otimes I_2(q_2)$ and $F(q_1, q_2) = F_1(q_1) \otimes F_2(q_2)$. Furthermore, let

$$\delta((p_1, p_2), \sigma, (q_1, q_2)) = \delta_1(p_1, \sigma, q_1) \otimes \delta_2(p_2, \sigma, q_2) \ ,$$

if $\sigma \in \Sigma$, and otherwise, if $T \subseteq \Gamma_V$,

$$\delta\big((p_1, p_2), T, (q_1, q_2)\big) = \delta_1\big(p_1, T \cap \Gamma_{V_1}, q_1\big) \otimes \delta_2(p_2, T \cap \Gamma_{V_2}, q_2) \ .$$

We observe that A can be constructed in polynomial time. We have to show that $[\![A]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \bowtie [\![A_2]\!]_{\mathbb{K}}$. Let $d \in \Sigma^*$ be a document and t be a tuple. Every run $\rho \in P(A, d)$ with $\mathrm{tup}(\rho) = \mathrm{t}$ originates from of a set of runs $\rho_1 \in P(A_1, d)$ selecting $\pi_{V_1}\mathrm{t}$ and a set of runs $\rho_2 \in P(A_2, d)$ selecting $\pi_{V_2}\mathrm{t}$. Due to distributivity of $\otimes$ over $\oplus$, it holds that $\mathsf{w}_\rho = \mathsf{w}_{\rho_1} \otimes \mathsf{w}_{\rho_2}$. Furthermore, every run in A corresponds to exactly one run in A_1 and one run in A_2. It follows directly that $[\![A]\!]_{\mathbb{K}}(d, \mathrm{t}) = [\![A_1]\!]_{\mathbb{K}}(d)(\pi_{V_1}\mathrm{t}) \otimes [\![A_2]\!]_{\mathbb{K}}(d)(\pi_{V_2}\mathrm{t})$ and that the construction preserves unambiguity. $\square$

We now show that regular annotators are closed under join.

Lemma 5.5.9. *Given two $\mathbb{K}$-weighted vset-automata A^1 and A^2, one can construct a weighted functional vset-automaton A with $[\![A]\!]_{\mathbb{K}} = [\![A^1]\!]_{\mathbb{K}} \bowtie [\![A^2]\!]_{\mathbb{K}}$. Furthermore, A can be constructed in polynomial time if A^1 and A^2 are functional and A is unambiguous if A_1 and A_2 are unambiguous.*

Proof. If A^1 and A^2 are not yet functional, we can assume that they are at an exponential cost in their number of variables (cf. Proposition 5.5.2). By Proposition 5.5.7, one can construct functional extended weighted vset-automata A_E^1, A_E^2 with $[\![A^i]\!]_{\mathbb{K}} = [\![A_E^i]\!]_{\mathbb{K}}$. Furthermore, due to Proposition 5.5.8, one can construct a functional extended weighted vset-automaton A_E with $[\![A_E]\!]_{\mathbb{K}} = [\![A_E^1]\!]_{\mathbb{K}} \bowtie [\![A_E^2]\!]_{\mathbb{K}}$. Thus, again applying Proposition 5.5.7, one can construct a functional weighted vset-automaton A_E with $[\![A_E]\!]_{\mathbb{K}} = [\![A_E^1]\!]_{\mathbb{K}} \bowtie [\![A_E^2]\!]_{\mathbb{K}} = [\![A^1]\!]_{\mathbb{K}} \bowtie [\![A^2]\!]_{\mathbb{K}}$. Note that all constructions are in polynomial time if A^1 and A^2 are functional and preserve unambiguity. Thus, concluding the proof with $A := A_E$. $\square$

The previous lemma also has applications to unambiguous functional vset-automata over the Boolean semiring.

Corollary 5.5.10. *Given two unambiguous functional vset-automata A_1, A_2 over the Boolean semiring, one can construct an unambiguous functional vset-automaton A with $[\![A]\!]_{\mathbb{B}} = [\![A_1]\!]_{\mathbb{B}} \bowtie [\![A_2]\!]_{\mathbb{B}}$ in polynomial time.* $\square$

5.5.4 Closure Under String Selection

A k-ary string relation is *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 [135]. Recall that $\textsc{Reg}_{\mathbb{K}}$ is the set of all regular $\mathbb{K}$-annotators. We say that a k-ary string relation[20] R is *selectable by regular* $\mathbb{K}$*-annotators* if the class of $\mathbb{K}$-annotators is closed under the string selection σ^R. More formally:

$$\{\sigma^R_{x_1,\ldots,x_k}(S) \mid S \in \textsc{Reg}_{\mathbb{K}} \text{ and } x_i \in \text{Vars}(S) \text{ for all } 1 \leq i \leq k\} \subseteq \textsc{Reg}_{\mathbb{K}}\ ,$$

If $\mathbb{K} = \mathbb{B}$, we say that R is *selectable by document spanners.* Fagin et al. [45, Theorem 4.16] proved that a string relation is recognizable *if and only if* it is selectable by document spanners. Here, we generalize this result in the context of weights and annotation. Indeed, it turns out that the equivalence is maintained for all positive semirings.

Theorem 5.5.11. *Let* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be a positive semiring and* R *be a string relation. The following are equivalent:*

1. *R is recognizable.*
2. *R is selectable by document spanners.*
3. *R is selectable by $\mathbb{K}$-annotators.*

We note that the equivalence between (1) and (2) is known [45, Theorem 4.16]. The implication (2) $\Rightarrow$ (3) is heavily based on the closure properties from Theorem 5.5.4 and holds beyond positive semirings. For the proof of the implication (3) $\Rightarrow$ (2), we use semiring morphisms to turn $\mathbb{K}$-weighted vset-automata into $\mathbb{B}$-weighted vset-automata and need positivity of the semiring. We need some preliminary results in order to give the proof.

Definition 5.5.12. Let R be a k-ary string relation. A $\mathbb{K}$-weighted vset-automaton $A^{\mathbb{K}}_R$ with variables $\{x_1, \ldots, x_k\}$ selects R over $\mathbb{K}$ if for every document $d \in \Sigma^*$ and every tuple t it holds that $[\![A^{\mathbb{K}}_R]\!]_{\mathbb{K}}(d, \mathrm{t}) = \overline{1}$ if $(d_{\mathrm{t}(x_1)}, \ldots, d_{\mathrm{t}(x_k)}) \in R$, and $\overline{0}$, otherwise, that is, $(d_{\mathrm{t}(x_1)}, \ldots, d_{\mathrm{t}(x_k)}) \notin R$.

Lemma 5.5.13. *Let R be a k-ary string relation. Then R is selectable by $\mathbb{K}$-annotators if and only if there is a vset-automaton $A^{\mathbb{K}}_R$ that selects R over $\mathbb{K}$.*

Proof. Assume that R is selectable by $\mathbb{K}$-annotators. Let A be the $\mathbb{K}$-weighted vset-automaton that assigns weight $\overline{1}$ to all possible tuples for all documents. As R is selectable by $\mathbb{K}$-annotators, $\sigma^R_{x_1,\ldots,x_k}([\![A]\!]_{\mathbb{K}})$ must be a regular $\mathbb{K}$-annotator. Thus, the $\mathbb{K}$-weighted vset-automaton $A^{\mathbb{K}}_R$ representing $\sigma^R_{x_1,\ldots,x_k}([\![A]\!]_{\mathbb{K}})$ selects R over $\mathbb{K}$.

[20] Recall that a *(k-ary) string relation* is the Cartesian product of k languages, that is, $\mathcal{L}_1 \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_k$, with $\mathcal{L}_i \subseteq \Sigma^*$, for all $1 \leq i \leq k$.

For the other direction, let $A^{\mathbb{K}}_R$ be as defined. Let A be a $\mathbb{K}$-weighted vset-automaton. Per definition of string selection,

$$\sigma^R_{x_1,\ldots,x_k}(\llbracket A \rrbracket_{\mathbb{K}})(d,\mathrm{t}) = \begin{cases} \llbracket A \rrbracket_{\mathbb{K}}(d,\mathrm{t}) \otimes \overline{0} = \overline{0} & \text{if } (d_{\mathrm{t}(x_1)},\ldots,d_{\mathrm{t}(x_k)}) \notin R \\ \llbracket A \rrbracket_{\mathbb{K}}(d,\mathrm{t}) \otimes \overline{1} = \llbracket A \rrbracket_{\mathbb{K}}(d,\mathrm{t}) & \text{otherwise.} \end{cases}$$

Therefore, $\sigma^R_{x_1,\ldots,x_k}(\llbracket A \rrbracket_{\mathbb{K}}) = \llbracket A \rrbracket_{\mathbb{K}} \bowtie \llbracket A^{\mathbb{K}}_R \rrbracket_{\mathbb{K}}$, which proves that R is selectable by $\mathbb{K}$-annotators, as $\mathbb{K}$-annotators are closed under join (cf. Theorem 5.5.4). □

We will now define means of transferring the structure of weighted automata between different semirings, that is, we define $\mathbb{B}$-*projections* and $\mathbb{K}$-*extensions* of weighted vset-automata.

Definition 5.5.14. Let A be a weighted vset-automaton over $\mathbb{K}$. A $\mathbb{B}$-weighted vset-automaton $A^{\mathbb{B}}$ is a $\mathbb{B}$-*projection of* A if, for every document $d \in \Sigma^*$, it holds that $\mathrm{t} \in \llbracket A^{\mathbb{B}} \rrbracket_{\mathbb{B}}(d) \Leftrightarrow \mathrm{t} \in \llbracket A \rrbracket_{\mathbb{K}}(d)$.

Definition 5.5.15. Let A be a $\mathbb{B}$-weighted vset-automaton. Then a $\mathbb{K}$-weighted vset-automaton $A^{\mathbb{K}}$ is called a $\mathbb{K}$-*extension of* A if, for every document $d \in \Sigma^*$ and every tuple t, the following are equivalent:

1. $\mathrm{t} \in \llbracket A \rrbracket_{\mathbb{B}}(d)$
2. $\mathrm{t} \in \llbracket A^{\mathbb{K}} \rrbracket_{\mathbb{K}}(d)$ and $\llbracket A^{\mathbb{K}} \rrbracket_{\mathbb{K}}(d,\mathrm{t}) = \overline{1}$

Furthermore, $A^{\mathbb{K}}$ has exactly one run for every tuple in $\llbracket A^{\mathbb{K}} \rrbracket_{\mathbb{K}}(d)$.

We now show that a $\mathbb{B}$-projections of a $\mathbb{K}$-weighted vset-automaton A exists if $\mathbb{K}$ is positive. Furthermore, a $\mathbb{K}$-extensions of a $\mathbb{B}$-weighted vset-automaton always exists. To this end, let $(\mathbb{K},\oplus,\otimes,\overline{0},\overline{1})$ and $(\mathbb{K}',\oplus',\otimes',\overline{0'},\overline{1'})$ be semirings. For a function $f : \mathbb{K} \to \mathbb{K}'$ and a weighted vset-automaton $A := (\Sigma, V, Q, I, F, \delta)$ over $\mathbb{K}$, we define the weighted vset-automaton $A_f := (\Sigma, V, Q, I_f, F_f, \delta_f)$ over $\mathbb{K}'$, where $I_f := f \circ I, F_f := f \circ F$, and $\delta_f := f \circ \delta$.

Lemma 5.5.16. *Let $\mathbb{K}$ be a positive semiring. Then there exists a $\mathbb{B}$-projection $A^{\mathbb{B}}$ of A for every $\mathbb{K}$-weighted vset-automaton A.*

Proof. Let $f : \mathbb{K} \to \mathbb{B}$ be the function

$$f(x) = \begin{cases} \mathsf{true} & \text{if } x \neq \overline{0}\ , \\ \mathsf{false} & \text{if } x = \overline{0}\ . \end{cases}$$

Eilenberg [42, Chapter VI.2] shows that, due to $\mathbb{K}$ being positive[21], the function f is a semiring morphism, that is,

$$\begin{aligned} f(x \oplus^{\mathbb{K}} y) &= f(x) \oplus^{\mathbb{B}} f(y), \quad f(\overline{0}) = \mathsf{false}\ , \\ f(x \otimes^{\mathbb{K}} y) &= f(x) \otimes^{\mathbb{B}} f(y), \quad f(\overline{1}) = \mathsf{true}\ . \end{aligned}$$

[21] Eilenberg [42, Chapter VI.2] actually showed that f is a semiring morphism if and only if $\mathbb{K}$ is positive.

Observe that these properties ensure that, for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in [\![A]\!]_{\mathbb{K}}$, it holds that

$$f\left([\![A]\!]_{\mathbb{K}}(d, \mathrm{t})\right) = [\![A_f]\!]_{\mathbb{K}'}(d, \mathrm{t}).$$

Therefore, A_f is a $\mathbb{B}$-projection of A. □

Lemma 5.5.17. *Every $\mathbb{B}$-weighted vset-automaton A has a $\mathbb{K}$-extension.*

Proof. Let $A := (V, Q, I, F, \delta)$ be a $\mathbb{B}$-weighted vset-automaton. By Proposition 2.2.6 there is an equivalent deterministic vset-automaton $A_{\mathtt{det}}$, for every vset-automaton A. Therefore, we can assume, w.l.o.g., that A is deterministic and has exactly one run ρ for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in [\![A_{\mathtt{det}}]\!]_{\mathbb{B}}(d)$, with $\mathrm{ref}(\rho) = \mathrm{ref}(d, \mathrm{t})$. Let $g : \mathbb{B} \to \mathbb{K}$ be the function[22]

$$g(x) = \begin{cases} \overline{1} & \text{if } x = \mathsf{true}\ , \\ \overline{0} & \text{if } x = \mathsf{false}\ . \end{cases}$$

Observe that the automaton A_g must also have exactly one run ρ for every document $d \in \Sigma^*$ and every tuple $\mathrm{t} \in [\![A_{\mathtt{det}}]\!]_{\mathbb{K}}(d)$. It remains to show that A_g is indeed a $\mathbb{K}$-extension of A. To this end, let $d \in \Sigma^*$ be a document. We have to show that the following are equivalent:

1. $\mathrm{t} \in [\![A]\!]_{\mathbb{B}}(d)$
2. $\mathrm{t} \in [\![A_g]\!]_{\mathbb{K}}(d)$ and $[\![A_g]\!]_{\mathbb{K}}(d, \mathrm{t}) = \overline{1}$

(1) implies (2): Let $\mathrm{t} \in [\![A]\!]_{\mathbb{B}}(d)$ and let $\mathrm{r} = \mathrm{ref}(d, \mathrm{t})$. Per assumption that A is deterministic, A must have a run on r. Thus, A_g must also have a run, accepting r. Furthermore, as A is deterministic, A_g can not have a run $\rho' \neq \rho$ with $\mathrm{tup}(\rho) = \mathrm{tup}(\rho')$, as otherwise, A would also have a run ρ' and thus not be deterministic. Per construction, all transitions of A_g have weight $\overline{0}$ or $\overline{1}$. Thus, (2) must hold.

(2) implies (1): Let $\mathrm{t} \in [\![A_g]\!]_{\mathbb{K}}(d)$ and $[\![A_g]\!]_{\mathbb{K}}(d, \mathrm{t}) = \overline{1}$. Thus, there is a run ρ_g of A_g on d accepting t. Therefore, there must also be a run ρ of A on d, accepting t, concluding the proof. □

We are now ready to prove Theorem 5.5.11.

Proof of Theorem 5.5.11. The equivalence between (1) and (2) is shown in [45, Theorem 4.16].

We show (2) ⇒ (3). Let A be a $\mathbb{K}$-weighted vset-automaton and R be a relation that is selectable by regular $\mathbb{B}$-annotators. We have to show that every string selection $\sigma^R_{x_1,\ldots,x_k}[\![A]\!]_{\mathbb{K}}$ is definable by a $\mathbb{K}$-weighted vset-automaton. By assumption R is selectable by regular $\mathbb{B}$-annotators. Let $A^R_{\mathbb{B}}$ be the vset-automaton that selects R over $\mathbb{B}$, which exists by Lemma 5.5.13. Let $A^R_{\mathbb{K}}$ be a $\mathbb{K}$-extension of $A^R_{\mathbb{B}}$ vset-automaton, which exists

[22] Notice that g is not necessarily a semiring morphism. Depending on $\mathbb{K}$, it may be the case that $\overline{1} \oplus \overline{1} = \overline{0}$, contradicting the properties of semiring morphisms. Take $\mathbb{K} = \mathbb{Z}/2\mathbb{Z}$, for instance.

by Lemma 5.5.17. Thus, $A_{\mathbb{K}}^{R}$ selects R over $\mathbb{K}$ and therefore, (3) follows directly from Lemma 5.5.13.

We now prove the implication (3) $\Rightarrow$ (2). Let R be a string relation selectable by $\mathbb{K}$-annotators and let A be a $\mathbb{B}$-weighted vset-automaton. We have to show that R is also selectable over $\mathbb{B}$, i.e., there is a $\mathbb{B}$-weighted vset-automaton $A_{\mathbb{B}}^{R}$ such that $[\![A_{\mathbb{B}}^{R}]\!]_{\mathbb{B}} = \sigma_{x_1,\ldots,x_k}^{R}[\![A]\!]_{\mathbb{B}}$. Let $A_{\mathbb{K}}$ be a $\mathbb{K}$-extension of A, which exists by Lemma 5.5.17. Per assumption R is selectable over $\mathbb{K}$, therefore, due to Lemma 5.5.13, there exists a $\mathbb{K}$-weighted vset-automaton $A_{\mathbb{K}}^{R}$ which selects R over $\mathbb{K}$. Thus, $\sigma_{x_1,\ldots,x_k}^{R}[\![A_{\mathbb{K}}]\!]_{\mathbb{K}} = [\![A_{\mathbb{K}}^{R}]\!]_{\mathbb{K}}$. Let $A_{\mathbb{B}}^{R}$ be a $\mathbb{B}$-projection of $A_{\mathbb{K}}^{R}$, which exists by Lemma 5.5.16. It remains to show that $\sigma_{x_1,\ldots,x_k}^{R}[\![A]\!]_{\mathbb{B}} = [\![A_{\mathbb{B}}^{R}]\!]_{\mathbb{B}}$. Let $d \in \Sigma^*$ and $\mathrm{t} \in [\![A_{\mathbb{B}}^{R}]\!]_{\mathbb{B}}(d)$. By Lemma 5.5.16, $\mathrm{t} \in [\![A_{\mathbb{K}}^{R}]\!]_{\mathbb{K}}(d)$ and therefore, $\mathrm{t} \in \sigma_{x_1,\ldots,x_k}^{R}[\![A_{\mathbb{K}}]\!]_{\mathbb{K}}(d)$. Per definition of string selection, it follows that $(d_{\mathrm{t}(x_1)},\ldots,d_{\mathrm{t}(x_k)}) \in R$ and $\mathrm{t} \in [\![A_{\mathbb{K}}]\!]_{\mathbb{K}}(d)$. By Lemma 5.5.17, it follows that $(d_{\mathrm{t}(x_1)},\ldots,d_{\mathrm{t}(x_k)}) \in R$ and $\mathrm{t} \in [\![A]\!]_{\mathbb{B}}(d)$, and therefore $\mathrm{t} \in \sigma_{x_1,\ldots,x_k}^{R}[\![A]\!]_{\mathbb{B}}(d)$. Observe that all implications in the previous argument where actually equivalences. Therefore, the inclusion $\sigma^{R}[\![A]\!]_{\mathbb{B}}(d) \subseteq [\![A_{\mathbb{B}}^{R}]\!]_{\mathbb{B}}(d)$ also holds. ☐

Since the implication from (2) to (3) does not assume positivity of the semiring, it raises the question whether the equivalence can be generalized even further. We show next that this is indeed the case, for instance the equivalence also holds for the Łukasiewicz semiring, which is not positive.

Beyond Positive Semirings

We provide some insights about the cases where $\mathbb{K}$ is not positive. First of all, one implication always holds.

Lemma 5.5.18. *Let* $(\mathbb{K},\oplus,\otimes,\overline{0},\overline{1})$ *be an arbitrary semiring and* R *be a recognizable string relation. Then* R *is also selectable by* $\mathbb{K}$*-annotators.*

Proof. This is an immediate consequence of the proofs of the implications (1) $\Rightarrow$ (2) $\Rightarrow$ (3) of Theorem 5.5.11. ☐

The question is: For which semirings $\mathbb{K}$ does selectability by $\mathbb{K}$-annotators imply selectability by ordinary document spanners? It turns out that this is indeed possible for some non-positive semirings, such as the Łukasiewicz semiring Ł.

Let $(\mathbb{K}',\oplus',\otimes',\overline{0'},\overline{1'})$ be a subsemiring of a semiring $\mathbb{K}$.[23] The semiring $(\mathbb{K}',\oplus',\otimes',\overline{0'},\overline{1'})$ is *minimal* if there is no subsemiring of $(\mathbb{K},\oplus,\otimes,\overline{0},\overline{1})$ with fewer elements. Recall that a semiring $\mathbb{K}$ is bipotent, if $a \oplus b \in \{a,b\}$, for every $a,b \in \mathbb{K}$. We begin with some intermediate results.

Lemma 5.5.19. *Let* $(\mathbb{K},\oplus,\otimes,\overline{0},\overline{1})$ *be a bipotent semiring. Then* $\mathbb{K}_{\min} := \{\overline{0},\overline{1}\}$ *is the unique minimal subsemiring of* $\mathbb{K}$*. Furthermore,* $\mathbb{K}_{\min}$ *is isomorphic to the Boolean semiring.*

[23]Recall that a subsemiring of $\mathbb{K}$ is a set $\mathbb{K}'$, closed under addition and multiplication.

Proof. For every semiring it holds that $\overline{0} \otimes \overline{1} = \overline{0}$, $\overline{1} \otimes \overline{0} = \overline{0}$, and $\overline{0} \otimes \overline{0} = \overline{0}$. Furthermore, $\overline{1} \oplus \overline{0} = \overline{1}$ and $\overline{0} \oplus \overline{0} = \overline{0}$. As $\mathbb{K}$ is bipotent, it also holds that $\overline{1} \oplus \overline{1} = \overline{1}$. Let $\mathbb{K}' = \{\overline{0}, \overline{1}\}$. Thus, $\mathbb{K}_{\min} = \{\overline{0}, \overline{1}\}$ is a subsemiring of $\mathbb{K}$, as $\{\overline{0}, \overline{1}\}$ is closed under addition and multiplication. Observe that $\mathbb{K}_{\min}$ must be unique and minimal, as every subsemiring must at least contain $\overline{0}$ and $\overline{1}$.

It remains to show that $\mathbb{K}_{\min}$ is isomorphic to $\mathbb{B}$. To this end, let $f : \mathbb{K}_{\min} \to \mathbb{B}$ be the bijection

$$f(x) = \begin{cases} \mathsf{true} & \text{if } x = \overline{1}\ , \\ \mathsf{false} & \text{if } x = \overline{0}\ . \end{cases}$$

It is straightforward to verify that f is indeed a semiring isomorphism. □

It follows directly that $\mathbb{K}_{\min}$ is a positive semiring.

Lemma 5.5.20. *Let $\mathbb{K}$ be a bipotent semiring such that $a \otimes b = \overline{1}$ implies that $a = b = \overline{1}$. Then a string relation R is selectable by $\mathbb{K}$-annotators if and only if it is selectable by $\mathbb{K}_{\min}$-annotators.*

Proof. Every $\mathbb{K}_{\min}$-annotator is also a $\mathbb{K}$-annotator. Therefore, we only have to show that every string relation selectable by $\mathbb{K}$-annotators is also selectable by $\mathbb{K}_{\min}$-annotators.

Let A be a $\mathbb{K}_{\min}$-weighted vset-automaton and R be selectable by $\mathbb{K}$-annotators. We have to show that $\sigma^R_{x_1,\ldots,x_k}[\![A]\!]_{\mathbb{K}_{\min}}$ is definable by a $\mathbb{K}_{\min}$-weighted vset-automaton. Let $d \in \Sigma^*$ be a document.

Per assumption R is selectable over $\mathbb{K}$. Let $A^R_{\mathbb{K}}$ be a $\mathbb{K}$-weighted vset-automaton, guaranteed by Lemma 5.5.13. We argue that we can assume, w.l.o.g., that all edges in $A^R_{\mathbb{K}}$ have either weight $\overline{1}$ or $\overline{0}$ and therefore $A^R_{\mathbb{K}}$ is a $\mathbb{K}_{\min}$-annotator, concluding the proof.

We observe that $A^R_{\mathbb{K}}$ only assigns weight $\overline{1}$ and $\overline{0}$. Therefore, $\mathrm{t} \in [\![A^R_{\mathbb{K}}]\!]_{\mathbb{K}}(d)$ if and only if $[\![A^R_{\mathbb{K}}]\!]_{\mathbb{K}}(d, \mathrm{t}) = 1$. Recall that $\mathbb{K}$ is bipotent, that is, for every $a, b \in \mathbb{K}$, $a \oplus b \in \{a, b\}$. Therefore, for every $\mathrm{t} \in [\![A^R_{\mathbb{K}}]\!]_{\mathbb{K}}(d)$ there must be a run ρ of $A^R_{\mathbb{K}}$ on d with $\mathsf{w}_\rho = \overline{1}$. Furthermore, as $a \otimes b = \overline{1}$ implies that $a = b = \overline{1}$, this run must not have an edge with weight $a \neq \overline{1}$. On the other hand, assume that there is a run ρ of $A^R_{\mathbb{K}}$ on d with weight $\mathsf{w}_\rho = \overline{1}$. Again, this run must not have an edge with weight $a \neq \overline{1}$. Furthermore, due to $a \oplus b \neq \overline{0}$, unless $a = \overline{0}$ and $b = \overline{0}$[24], this implies that $\mathrm{tup}(\rho) \in [\![A^R_{\mathbb{K}}]\!]_{\mathbb{K}}(d)$. Thus, there is a run ρ of $A^R_{\mathbb{K}}$ on d consisting only of edges with weight $\overline{1}$ if and only if $\mathrm{tup}(\rho) \in [\![A^R_{\mathbb{K}}]\!]_{\mathbb{K}}$. Therefore, all edges in $A^R_{\mathbb{K}}$ with weight $w \neq \overline{1}$ can be removed without changing the $\mathbb{K}$-annotator. Thus, we can assume, w.l.o.g., that all edges in $A^R_{\mathbb{K}}$ have either weight $\overline{0}$ or $\overline{1}$. It follows that $A^R_{\mathbb{K}}$ is a $\mathbb{K}_{\min}$-annotator and therefore, by Lemma 5.5.13, it follows that R is selectable over $\mathbb{K}_{\min}$. □

The following corollary follows directly from Theorem 5.5.11, Lemma 5.5.20, and Lemma 5.5.19.

Corollary 5.5.21. *Let $\mathbb{K}$ be a bipotent semiring, such that $a \otimes b = \overline{1}$ implies that $a = b = \overline{1}$. A string relation R is recognizable if and only if it is selectable by $\mathbb{K}$-annotators.* □

[24]This follows from $\mathbb{K}$ being bipotent and $\oplus$ being the additive identity.

Recall the Łukasiewicz semiring, whose domain is $[0,1]$, with addition given by $x \oplus y = \max(x,y)$, multiplication $x \otimes y = \max(0, x+y-1)$, zero element 0, and one element 1. Thus, for every $a,b \in [0,1]$, $a \oplus b \in \{a,b\}$ and $a \otimes b = 1$ if and only if $a = b = 1$. Therefore, the Łukasiewicz semiring satisfies the conditions of Corollary 5.5.21.

Corollary 5.5.22. *A string relation R is recognizable if and only if it is selectable by Ł-annotators.* □

5.6 Evaluation Problems

We consider two types of evaluation problems in this section: *answer testing* and *best weight evaluation.* The former is given an annotator, a document d, and a tuple t, and computes the annotation of t in d according to the annotator. The latter does not receive the tuple as input, but receives a weight threshold and is asked whether there exists a tuple to which a weight greater than or equal to the threshold is assigned.

5.6.1 Answer Testing

Recalling Proposition 2.2.7, it follows that answer testing is NP-complete for $\mathbb{B}$-weighted vset-automata in general. However, the proof makes extensive use of non-functionality of the automaton. As we show next, answer testing is tractable for functional weighted vset-automata.

Theorem 5.6.1. *Given a functional weighted vset-automaton A, a document d, and a tuple t, the weight $[\![A]\!]_{\mathbb{K}}(d,\mathrm{t})$ assigned to t by A on d can be computed in polynomial time.*

Proof. Let A, d, and t be as stated. Per definition, the weight assigned to t by A is

$$[\![A]\!]_{\mathbb{K}}(d,\mathrm{t}) := \bigoplus_{\rho \in P(A,d) \text{ and } \mathrm{t}=\mathrm{tup}(\rho)} \mathsf{w}_\rho \ .$$

Therefore, in order to compute the weight $[\![A]\!]_{\mathbb{K}}(d,\mathrm{t})$, we need to consider the weights of all runs ρ for which $\mathrm{t} = \mathrm{tup}(\rho)$. Furthermore, multiple runs can select the same tuple t but assign variables in a different order.[25]

We first define an automaton A_t, such that $[\![A_\mathrm{t}]\!]_{\mathbb{K}}(d,\mathrm{t}) = \overline{1}$ and $[\![A_\mathrm{t}]\!]_{\mathbb{K}}(d)(\mathrm{t}') = \overline{0}$ for all $\mathrm{t}' \neq \mathrm{t}$. Such an automaton A_t can be defined using a chain of $|d| + 2|V| + 1$ states, which checks that the input document is d and which has exactly one nonzero run ρ, with $\mathsf{w}_\rho = \overline{1}$ and $\mathrm{tup}(\rho) = \mathrm{t}$.

By Theorem 5.5.4, there is a weighted vset-automaton A' with $[\![A']\!]_{\mathbb{K}} = [\![A]\!]_{\mathbb{K}} \bowtie [\![A_\mathrm{t}]\!]_{\mathbb{K}}$. It follows directly from the definition of A' that $[\![A']\!]_{\mathbb{K}}(d',\mathrm{t}') = \overline{0}$ if $d' \neq d$ or $\mathrm{t}' \neq \mathrm{t}$ and $[\![A']\!]_{\mathbb{K}}(d,\mathrm{t}) = [\![A]\!]_{\mathbb{K}}(d,\mathrm{t})$, otherwise. Furthermore, all runs $\rho \in P(A',d)$ have length $|d| + 2|V|$. Therefore, the weight $[\![A']\!]_{\mathbb{K}}(d,\mathrm{t})$ can be obtained by taking the sum of the

[25] This may happen when variable operations occur consecutively, that is, without reading an alphabet symbol in between.

weights of all runs of length $|d| + 2|V|$ of A'. We assume, w.l.o.g., that the states of A' are $\{1, \dots, n\}$ for some $n \in \mathbb{N}$. Due to distributivity of $\oplus$ over $\otimes$, this sum can be computed as

$$[\![A']\!]_{\mathbb{K}}(d, \mathrm{t}) = v_I \times (M_\delta)^{|d|+2|V|} \times (v_F)^T ,$$

where

- v_I is the vector $(I(1), \dots, I(n))$,
- M_δ is the $n \times n$ matrix with $M_\delta(i, j) = \bigoplus_{a \in \Sigma \cup \Gamma_V} \delta(i, a, j)$, and
- (v_F) is the vector $v_F = (F(1), \dots, F(n))$.

Therefore, by the assumption that $\mathbb{K}$ has an efficient encoding (Definition 5.4.1), the weight can be computed in polynomial time. □

5.6.2 Best Weight Evaluation

In many semirings, the domain is naturally ordered by some relation. For instance, the domain of the probability semiring is $\mathbb{Z}^+$, which is ordered by the $\leq$-relation. This motivates evaluation problems, where one is interested in some kind of optimization of the weight. We start by giving the definition of an ordered semiring.[26]

Definition 5.6.2 (similar to Droste and Kuich [38]). A commutative monoid $(\mathbb{K}, \oplus, \overline{0})$ is *ordered* if it is equipped with a linear order $\preccurlyeq$ preserved by the $\oplus$ operation. An ordered monoid is *positively ordered* if $\overline{0} \preccurlyeq a$ for all $a \in \mathbb{K}$. A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is *(positively) ordered* if the additive monoid is (positively) ordered and multiplication with elements $\overline{0} \preccurlyeq a$ preserves the order.

We consider the following two problems.

	THRESHOLD
Input:	Regular annotator A over an ordered semiring, document $d \in \Sigma^*$, and a weight $w \in \mathbb{K}$.
Question:	Is there a tuple t with $w \preccurlyeq [\![A]\!]_{\mathbb{K}}(d, \mathrm{t})$?

	MAXTUPLE
Input:	Regular annotator A over an ordered semiring and a document $d \in \Sigma^*$.
Task:	Compute a tuple with maximal weight, if it exists.

Notice that, if MAXTUPLE is efficiently solvable, then so is THRESHOLD. We therefore prove upper bounds for MAXTUPLE and lower bounds for THRESHOLD. The THRESHOLD problem is sometimes also called the *emptiness problem* in the weighted automata literature. It turns out that both problems are tractable for positively ordered semirings that are bipotent.

[26] Note that the following definition slightly deviates from the definition by Droste and Kuich [38]. We require the order to be linear, as the maximal weight would otherwise not be well defined.

Theorem 5.6.3. *Let* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be a positively ordered, bipotent semiring. Furthermore, let* A *be a functional* $\mathbb{K}$*-weighted vset-automaton, and let* $d \in \Sigma^*$ *be a document. Then* MAXTUPLE *for* A *and* d *can be solved in polynomial time.*

Proof. By Proposition 5.5.7, we can assume, w.l.o.g., that A is given as a functional extended $\mathbb{K}$-weighted vset-automaton. As $\mathbb{K}$ is bipotent, it must hold that $a \oplus b \in \{a, b\}$ for every $a, b \in \mathbb{K}$. Therefore, the weight of a tuple $t \in [\![A]\!]_{\mathbb{K}}(d)$ is always equal to the weight of one of the runs ρ with $\mathrm{t} = \mathrm{ref}(\rho)$. In order to find the tuple with maximal weight, we need to find the run of A on d with maximal weight. We define a directed acyclic graph (DAG) which is obtained by taking a "product" between A and d. Finding the run with the maximal weight then boils down to finding the path with maximal weight in this DAG.

Assume that $A = (V, Q, I, F, \delta)$. Recall that 2^{Γ_V} denotes the power set of Γ_V. We define a weighted, edge-labeled DAG $G = (N, E, w)$, where each edge e is in $N \times (\{\varepsilon\} \uplus (2^{\Gamma_V} \times \{1, \ldots, |d| + 1\})) \times N$ and w assigns a weight $w(e) \in \mathbb{K}$ to every edge e. We note that an edge $(p, (T, i), q) \in E$ will encodes that a transition, labeled T, is reached after reading $d_{[1,i\rangle}$.

More formally, let $N := \{s, t\} \uplus \{(q, i) \mid q \in Q \text{ and } 1 \leq i \leq |d| + 1\}$. We say that a node $n = (p, i)$ is in layer i of G, where s is in layer 0 and t in layer $|d| + 2$. Furthermore, let E be defined as follows:

$$\begin{aligned} E := &\{(s, \varepsilon, (q, 1)) \mid I(q) \neq \overline{0}\} \\ &\cup \{((q, |d| + 1), \varepsilon, t) \mid F(q) \neq \overline{0}\} \\ &\cup \{((p, i), (T, i), (q, i)) \mid T \subseteq \Gamma_V \text{ and } \delta(p, T, q) \neq \overline{0}\} \\ &\cup \{((p, i), \varepsilon, (q, i + 1)) \mid d_{[i,i+1\rangle} = a \text{ and } \delta(p, a, q) \neq \overline{0}\} \,. \end{aligned}$$

Furthermore, for $T \subseteq \Gamma_V$ and $a \in \Sigma$, we define the weight $w(e)$ for all $e \in E$ as follows:

$$\begin{aligned} w((s, \varepsilon, (q, 1))) &:= I(q) \\ w(((q, |d| + 1), \varepsilon, t)) &:= F(q) \\ w(((p, i), (T, i), (q, i))) &:= \delta(p, T, q) \\ w(((p, i), \varepsilon, (q, i + 1))) &:= \delta(p, a, q) \,. \end{aligned}$$

Recall that, in extended weighted vset-automata, the set of states Q is a disjoint union of letter- and variable states, such that all transitions labeled by $\sigma \in \Sigma$ originate in letter states and all transitions labeled by $T \subseteq \Gamma_V$ originate in variable states. Therefore, G must be acyclic, as all edges are either from a node in layer i to a node in layer $i + 1$ or from a variable state to a letter state within the same layer. Furthermore, there is a path from s to t in G with weight w if and only if there is a tuple $\mathrm{t} \in [\![A]\!]_{\mathbb{K}}(d)$ with the same weight. The Procedure BestWeightEvaluation shows how a path with maximal weight can be computed in polynomial time.[27] The correctness follows directly from $\mathbb{K}$

[27]We note that all semiring operations must be computable efficiently as we assume that only efficient encodings are used.

being positively ordered, thus order being preserved by addition and multiplication with an element $\ell \in \mathbb{K}$. ☐

Procedure BestWeightEvaluation(G,s,t)

Input: A weighted, edge-labeled DAG $G = (N, E, w)$, nodes s, t

Output: A path from s to t in G with maximal weight or Null, if no such path exists.

```
p(s) ← ε                              ▷ p(n) will store the best path from s to n.
w(s) ← 1̄                              ▷ w(n) will be the weight of the path p(n).
for s ≠ n ∈ N in topological order do
    if there is a node n′ and a label ℓ with (n′, ℓ, n) ∈ E then
        p(n) = ε
        w(n) = 0̄
    else
        e = arg max_{e:=(n′,ℓ,n)∈E} w(n′) ⊗ w(e)
        p(n) = p(n′) · e
        w(n) = w(n′) ⊗ w(e)
if w(t) ≠ 0̄ then
    output p(t)
else
    output Null
```

If the semiring is not bipotent, however, the THRESHOLD and MAXTUPLE problems quickly become intractable.

Theorem 5.6.4. *Let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring such that, for increasing values of m, $\bigoplus_{i=1}^{m} \overline{1}$ is strictly monotonously increasing. Furthermore, let A be a functional $\mathbb{K}$-weighted vset-automaton, let $d \in \Sigma^*$ be a document, and $k \in \mathbb{K}$ be a weight threshold. Then THRESHOLD for such inputs is* NP*-complete.*

Proof. It is obvious that THRESHOLD is in NP, as one can guess a tuple t and test in PTIME whether $w \preccurlyeq [\![A]\!]_{\mathbb{K}}(d, \mathrm{t})$, using Theorem 5.6.1.

For the NP-hardness, we will reduce from the MAX-3SAT problem. Given a 3CNF formula and a natural number k, the decision version of MAX-3SAT asks whether there is a valuation satisfying at least k clauses. Let $\psi = C_1 \wedge \cdots \wedge C_m$ be a Boolean formula in 3CNF over variables $x_1, \ldots, x_n$ such that each clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ is a disjunction of exactly three literals $\ell_{i,j} \in \{x_c, \neg x_c \mid 1 \leq c \leq n\}$, $1 \leq i \leq k, 1 \leq j \leq 3$. We can assume, w.l.o.g., that no clause has two literals corresponding to the same variable.[28] Observe that, for each clause C_i, there are $2^3 = 8$ assignments of the variables corresponding

[28] A clause $C = x \vee \neg x \vee y$ can be omitted, as it is satisfied by every valuation and a clause $C = x \vee x \vee y$ can be replaced by two new clauses $C_1 = x \vee z \vee y$ and $C_2 = x \vee \neg z \vee y$, where z is a new variable.

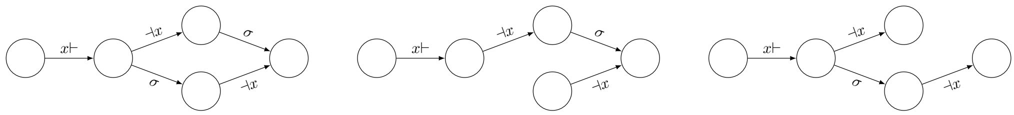

Figure 5.4: Example gadgets for variable x.

to the literals of C_i of which exactly 7 satisfy the clause C_i. Formally, let f_{C_i} be the function that maps a variable assignment τ to a number between 1 and 8, depending on the assignments of the literals of the clause C_i. We assume, w.l.o.g., that $f_{C_i}(\tau) = 8$ if and only if C_i is not satisfied by τ.

We define a functional weighted automaton automaton A_ψ over the unary alphabet $\Sigma = \{\sigma\}$ such that $[\![A_\psi]\!]_{\mathbb{K}}(\sigma^n)(\mathrm{t}) = \bigoplus_{i=1}^{m} \overline{1}$ if and only if the assignment corresponding to t satisfies exactly m clauses in ψ and $[\![A_\psi]\!]_{\mathbb{K}}(d, \mathrm{t}) = \overline{0}$ if $d \neq \sigma^n$ or t does not encode a variable assignment. Each variable x_i of ψ is associated with a corresponding capture variable x_i of A_ψ. With each assignment τ we associate a tuple t_τ, such that

$$\mathrm{t}_\tau(x_i) = \begin{cases} [i, i\rangle & \text{if } \tau(x_i) = 0 \text{ , and} \\ [i, i+1\rangle & \text{if } \tau(x_i) = 1 \text{ .} \end{cases}$$

The automaton $A_\psi := (\Sigma, V, Q, I, F, \delta)$ consists of m disjoint branches, where each branch corresponds to a clause of ψ; we call these *clause branches*. Each clause branch is divided into 7 sub-branches, such that a path in the sub-branch j corresponds to a variable assignment τ if $f_{C_i}(\tau) = j$. Thus, each clause branch has exactly one run ρ with weight $\overline{1}$ for each tuple t_τ associated to a satisfying assignment τ of C_i.

More formally, the set of states $Q = \{q_{i,j}^{a,b} \mid 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq a \leq 7, 1 \leq b \leq 5\}$ contains $5n$ states for every of the 7 sub-branches of each clause branch. Intuitively, A_ψ has a gadget, consisting of 5 states, for each variable and each of the 7 satisfying assignments of each clause. Figure 5.4 depicts the three types of gadgets we use. Note that the weights of the drawn edges are all $\overline{1}$. We use the left gadget if x does not occur in the relevant clause and the middle (resp., right) gadget if the literal $\neg x$ (resp., x) occurs. Furthermore, within the same sub-branch of A_ψ, the last state of each gadget is the same state as the start state of the next variable, i.e., $q_{i,j}^{a,5} = q_{i,j+1}^{a,1}$ for all $1 \leq i \leq k, 1 \leq j < n, 1 \leq a \leq 7$.

We illustrate the crucial part of the construction on an example. Let $\psi = (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$. The corresponding weighted vset-automaton A_ψ therefore has $14 = 2 \times 7$ disjoint branches. Figure 5.5 depicts the sub-branch for clause C_1 that corresponds to all assignments τ with $f_{C_i}(\tau) = 1$ which we assume is the case if $x_1 = x_2 = 1$ and $x_4 = 0$.

Formally, the initial weight function and the final weight function are defined as follows:

$$I(q_{i,j}^{a,b}) = \begin{cases} \overline{1} & \text{if } j = b = 1 \\ \overline{0} & \text{otherwise;} \end{cases} \qquad F(q_{i,j}^{a,b}) = \begin{cases} \overline{1} & \text{if } j = n, \text{ and } b = 5 \\ \overline{0} & \text{otherwise.} \end{cases}$$

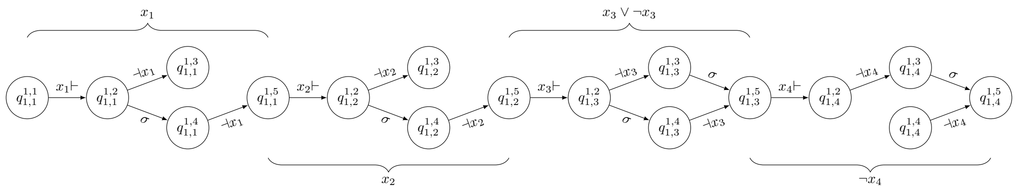

Figure 5.5: The sub-branch of A_ψ corresponding to C_1 and $x_1 = x_2 = 1, x_4 = 0$.

The transition function δ is defined as follows:

$$\delta(q_{i,j}^{a,b}, o, q_{i,j}^{a,b'}) = \begin{cases} \overline{1} & b = 1, b' = 2, o = x_j{\vdash} \\ \overline{1} & b = 2, b' = 3, o = {\dashv}x_j \\ \overline{1} & b = 2, b' = 4, o = \sigma, \text{ and there is a variable assignment } \tau \text{ with} \\ & \quad \tau(x_j) = 1 \text{ and } f_{C_i}(\tau) = a \\ \overline{1} & b = 3, b' = 5, o = \sigma, \text{ and there is a variable assignment } \tau \text{ with} \\ & \quad \tau(x_j) = 0 \text{ and } f_{C_i}(\tau) = a \\ \overline{1} & b = 4, b' = 5, o = {\dashv}x_j \end{cases}$$

All other transitions have weight $\overline{0}$.

We show that there is a tuple $\mathrm{t} \in [\![A_\psi]\!]_{\mathbb{K}}(\sigma^n)$ with weight $\mathsf{w}_\mathrm{t} = \bigoplus_{i=1}^{k} \overline{1}$ if and only if the corresponding assignment τ satisfies exactly k clauses of ψ. Let τ be an assignment of the variables $x_1, \ldots, x_n$. Thus, there is a run $\rho \in P(A_\psi, \sigma^n)$ with weight $\mathsf{w}_\rho = \overline{1}$ starting in $q_{i,1}^{a,1}$, such that $a = f_{C_i}(\tau)$ if and only if τ satisfies clause C_i. Due to $\bigoplus_{i=1}^{k} \overline{1}$ being strictly monotonously increasing it follows that $\bigoplus_{i=1}^{k} \overline{1} \preccurlyeq \mathsf{w}_{\mathrm{t}_\tau}$ if and only if the corresponding assignment to τ satisfies at least k clauses. Let $w = \bigoplus_{i=1}^{k} \overline{1}$. It follows directly that there is an assignment τ of ψ satisfying k clauses if and only if there is a tuple t with $w \preccurlyeq [\![A_\psi]\!]_{\mathbb{K}}(\sigma^n, \mathrm{t})$. $\square$

We note that Theorem 5.6.3 and Theorem 5.6.4 give us tight bounds for all semirings we defined in Example 5.1.1. Furthermore, since MAX-3SAT is hard to approximate, we can turn Theorem 5.6.4 into an even stronger inapproximability result for semirings where approximation makes sense. To this end, we focus on semirings that contain $(\mathbb{N}, +, \cdot, 0, 1)$ as a subsemiring in the following result. Note that this already implies that $\bigoplus_{i=1}^{m} \overline{1}$ is strictly monotonously increasing for increasing values of m.

Theorem 5.6.5. *Let $\mathbb{K}$ be a semiring that contains $(\mathbb{N}, +, \cdot, 0, 1)$ as a subsemiring and let A be a weighted vset-automaton over $\mathbb{K}$. Unless PTIME = NP, there is no algorithm that approximates the tuple with the best weight within a sub-exponential factor in polynomial time.*

Proof. Given a Boolean formula ψ in 3CNF, MAX-3SAT asks for the maximal number of clauses satisfied by a variable valuation. Håstad [69, Theorem 6.5] shows that, for every $\varepsilon > 0$, it is NP-hard to approximate MAX-3SAT within a factor $8/7 - \varepsilon$. In other words, unless PTIME = NP, there is no polynomial time algorithm which, given a 3CNF formula, returns a variable assignment satisfying at least $\frac{\mathrm{opt}}{8/7-\varepsilon}$ clauses, where opt is the maximal number of clauses which are satisfiable by a single variable assignment. We can leverage this, using the reduction from Theorem 5.6.4, to show that there is no polynomial time algorithm that approximates the tuple with the best weight with an sub-exponential approximation factor.

Let ψ be a 3CNF formula with m clauses and let A_ψ be the weighted vset-automaton and $d \in \Sigma^*$ be as constructed from ψ as in the proof of Theorem 5.6.4. Let $c = |A_\psi|$ be the size of A_ψ, which is linear in n. As shown in Theorem 5.6.4, there is a tuple t in A_ψ with weight j if and only if the variable assignment corresponding to t satisfies exactly j clauses. For a $k \in \mathbb{N}$ let A_ψ^k be the weighted vset-automaton, constructed by concatenating k copies of A_ψ, each of which using a set of n fresh variables, by inserting ε-edges with weight $\overline{1}$ from q_i to q_{i+1} where q_i is a final state of the i-th copy and q_{i+1} an initial state of the $i+1$-th copy. Observe that A_ψ^k has size $c \cdot k$, has nk variables, and each tuple $\mathrm{t} \in [\![A_\psi^k]\!]_{\mathbb{K}}(d^k)$ encodes k, possibly different, variable assignments for ψ.

For the sake of contradiction, assume there is a polynomial time algorithm approximating the best weight of A_ψ^k with a polynomial factor $p(c) = c^i$ for some constant i. That is, given a spanner A of size c and a document d of size $|d| \leq c$, the approximation algorithm returns a tuple t with $\mathsf{w}_\mathrm{t} \geq \frac{\mathrm{opt}}{p(c)}$, where opt is the maximal weight assigned to a tuple t over d by A. Let $\mathrm{t} \in [\![A_\psi^k]\!]_{\mathbb{K}}(d^k)$ be such an approximation and $\tau_1, \ldots, \tau_k$ be the corresponding variable assignments of ψ. Recall that $|A_\psi^k| = c \cdot k$ and $|d^k| = n \cdot k \leq c \cdot k$. Per assumption, there is an approximation algorithm, returning a tuple t with $\mathsf{w}_\mathrm{t} \geq \frac{\mathrm{opt}}{p(c)} \geq \frac{\mathrm{opt}}{(c \cdot k)^i}$. The tuple t encodes k variable assignments and the weight of the tuple is the product of the weights of the variable assignments. Let τ be one the variable assignments, encoded by t, which satisfy the most clauses.[29] Due to Håstad [69, Theorem 6.5], this procedure can at best lead to an $(8/7 - \varepsilon)$ approximation of the maximal number of satisfiable clauses. Therefore, it follows that $\mathsf{w}_\mathrm{t} \leq \frac{\mathrm{opt}}{(8/7-\varepsilon)^k}$. Thus, combining both inequalities, it must hold that $\frac{\mathrm{opt}}{(c \cdot k)^i} \leq \mathsf{w}_\mathrm{t} \leq \frac{\mathrm{opt}}{(8/7-\varepsilon)^k}$. Thus, $(8/7 - \varepsilon)^k \leq (ck)^i$. However, if $\frac{1}{8} > \varepsilon \geq 0$, this does not hold for arbitrarily large k, as i and c are constants, leading to the desired contradiction. □

5.7 Enumeration Problems

In this section we consider computing the output of annotators from the perspective of enumeration problems, where we try to enumerate all tuples with nonzero weight, possibly from large to small. Such problems are highly relevant for (variants of) vset-automata, as witnessed by the recent literature on the topic [6, 48].

[29] We note that there might be multiple assignments satisfying the same number of clauses.

An *enumeration problem* P is a (partial) function that maps each input i to a finite or countably infinite set of *outputs for* i, denoted by $P(i)$. Terminologically, we say that, given i, the task is to *enumerate* $P(i)$.

An *enumeration algorithm* for P is an algorithm that, given input i, writes a sequence of answers to the output such that every answer in $P(i)$ is written precisely once. If A is an enumeration algorithm for an enumeration problem P, we say that A runs in preprocessing p and delay d if the time before writing the first answer is $p(|i|)$, where $|i|$ is the size of the input i, and the time between writing every two consecutive answers is $d(|i|)$. By *between answers*, we mean the number of steps between writing the first symbol from an answer until writing the first symbol of the next answer. We generalize this terminology in the usual way to classes of functions. E.g., an algorithm with linear preprocessing and constant delay has a linear function for p and a constant function for d.

Given a $\mathbb{K}$-weighted vset-automaton A and a document d, let $f(A,d)$ be the maximal time required for a single addition or multiplication while computing the weight $[\![A]\!]_{\mathbb{K}}(d, \mathrm{t})$ for some tuple t. We note that, due to the assumption that $\mathbb{K}$ has an efficient encoding, $f(A,d)$ is at most polynomial in $|A|$ and $|d|$. Furthermore, for instance for finite semirings (like the Boolean semiring or the access control semiring), $f(A,d)$ is constant. If the order of the answers does not matter and the semiring is positive, we can guarantee an enumeration algorithm which has linear preprocessing time and constant delay in the size of the document and polynomial time and delay in the size of A and $f(A,d)$.[30] Note that the proof of the theorem essentially requires to go through the entire proof of the main result of Amarilli et al. [6, Theorem 1.1].

Theorem 5.7.1. *Given a weighted functional vset-automaton A over a positive semiring $\mathbb{K}$, and a document d, the $\mathbb{K}$-Relation $[\![A]\!]_{\mathbb{K}}(d)$ can be enumerated with preprocessing linear in $|d|$ and polynomial in $|A|$ and $f(A,d)$, and delay constant in $|d|$ and polynomial in $|A|$ and $f(A,d)$.*

Proof Sketch. Amarilli et al. [6, Theorem 1.1] showed that, given a sequential vset-automaton A and a document d, one can enumerate $[\![A]\!](d)$ with preprocessing time $O((|Q|^{\omega+1} + |A|) \times |d|)$ and with delay $O(|V| \times (|Q|^2 + |A| \times |V|^2))$, where $2 \leq \omega \leq 3$ is an exponent for matrix multiplication, V is the set of variables, and Q the set of states in A. In other words, $[\![A]\!](d)$ can be enumerated with linear preprocessing and constant delay in d, and polynomial preprocessing and delay in A. To obtain this result, they view the transition function of A as a (Boolean) transition matrix. Their methods easily extend from the Boolean case to transition matrices over positive semirings.[31] The claimed complexity for enumeration of the $\mathbb{K}$-Relation $[\![A]\!]_{\mathbb{K}}(d)$ can be achieved by computing all matrix multiplications over $\mathbb{K}$ instead of $\mathbb{B}$. Furthermore, instead of storing the set Λ

[30] We note that $f(A,d)$ can be polynomial in the size of the document. Thus, strictly speaking, preprocessing (resp., delay) might not be linear (resp., constant) in the size of the document. However, stating the theorem like this enables us to give two direct corollaries (Corollaries 5.7.2 and 5.7.3) depending on whether or not $f(A,d)$ is constant.

[31] Note that positivity is required as otherwise weights might sum up or multiply to zero, which may violate the constant delay.

of current states, one has to store a set of (state,weight)-tuples in order to compute the correct weights of the returned tuples. □

Depending on whether or not $f(A, d)$ is constant, we have the following two corollaries.

Corollary 5.7.2. *Given a weighted functional vset-automaton A over a positive semiring $\mathbb{K}$, and a document d, such that $f(A, d)$ is constant. Then the $\mathbb{K}$-Relation $[\![A]\!]_{\mathbb{K}}(d)$ can be enumerated with preprocessing linear in $|d|$ and polynomial in $|A|$, and delay constant in $|d|$ and polynomial in $|A|$.* □

Corollary 5.7.3. *Given a weighted functional vset-automaton A over a positive semiring $\mathbb{K}$, and a document d, such that $f(A, d)$ is polynomial in $|A|$ and $|d|$. Then the $\mathbb{K}$-Relation $[\![A]\!]_{\mathbb{K}}(d)$ can be enumerated with preprocessing linear in $|d|$ and polynomial in $f(A, d)$, and delay constant in $|d|$ and polynomial in $f(A, d)$.* □

We now consider cases in which answers are required to arrive in a certain ordering.

Ranked Annotator Enumeration (RA-Enum)	
Input:	Regular functional annotator A over an ordered semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ and a document d.
Task:	Enumerate all tuples $\mathrm{t} \in [\![A]\!]_{\mathbb{K}}(d)$ in descending order on $\mathbb{K}$.

Theorem 5.7.4. *Let $\mathbb{K}$ be an positively ordered, bipotent semiring, let A be a $\mathbb{K}$-weighted functional vset-automaton, and let $d \in \Sigma^*$ be a document. Then RA-Enum can be solved with polynomial delay and preprocessing.*

Proof. By Proposition 5.5.7, we can assume that A is an extended functional $\mathbb{K}$-weighted vset-automaton. Therefore, all runs of A which accept a tuple $\mathrm{t} \in [\![A]\!]_{\mathbb{K}}(d)$ have the same label. We will use the DAG G we defined in the proof of Theorem 5.6.3 and run a slight adaptation of Yen's algorithm [168] on G.

From the proof of Theorem 5.6.3 it follows that, given G we can find a path from s to t in G with maximal weight in polynomial time. Let $p = n_0 \cdot e_0 \cdot n_1 \cdots e_{k-1} \cdot n_k$ be a path in G, where $n_i \in N$ and $e_j \in E$ for $0 \le i \le k$ and $0 \le j < k$. We denote by $p[i, i]$ the node n_i, by $p[i, j]$ the path $n_i \cdot e_i \cdots e_{j-1} \cdot n_j$ and by $N(p)$ the set $\{n_i \mid 0 \le i \le k\}$ the set of nodes used by p. The Procedure BestWeightEnumeration shows how Yen's algorithm can be adapted for the RA-Enum problem. Recall that per construction of D, all edges which correspond to variable edges of A, are labeled by a tuple (T, i), which encodes that the set T is processed after reading $d_{[1,i\rangle}$. Thus, line 12 ensures that, whenever the algorithm reaches line 13, all paths $p[0, i] \cdot p'$ where p' is a path from $p[i, i]$ to t in G' differ from the paths in the set Out in at least one edge label and therefore, no tuple is enumerated multiple times. Observe that the first output of Algorithm BestWeightEnumeration is generated after polynomial time. Furthermore, every iteration of the while loop line 4, takes polynomial time. Thus, the algorithm runs with polynomial preprocessing and delay. □

Procedure BestWeightEnumeration(G,s,t).

Input: A weighted, edge-labeled DAG $G = (N, E, w)$, as constructed in Theorem 5.7.4, nodes s, t

Output: All paths from s to t in G in decreasing order without repetitions of the same path labels.

Out $\leftarrow \emptyset$ ▷ Out is the set of paths already written to output.
Cand $\leftarrow \emptyset$ ▷ Cand is a set of candidate paths from s to t.
$p \leftarrow$ BestWeightEvaluation(G, s, t)
while $p \neq$ Null **do**
    output p
    Add p to Out
    for $i = 0$ to $|p| - 1$ **do**
        $G' \leftarrow (N, E', w)$, where E' is a copy of E
        for every path p_1 in Out with $p_1[0, i] = p[0, i]$ **do**
            $n_i \cdot (n_i, \ell_i, n_{i+1}) \cdot n_{i+1} \leftarrow p_1[i, i+1]$
            for every $p, q \in N$ with $(p, \ell_i, q) \in E'$ **do**
                Remove the edge $\big(n_i, \ell_i, n\big)$ from E' ▷ Delete all ℓ_i-labeled edges.
        $p_2 \leftarrow$ BestWeightEvaluation$(G', p[i, i], t)$
        if p_2 is not Null **then**
            Add $p[0, i] \cdot p_2$ to Cand
    $p \leftarrow$ a path in Cand with maximal weight ▷ $p \leftarrow$ Null if Cand $= \emptyset$.
    Remove p from Cand

Chapter 6

Aggregation Functions for Document Spanners

In this chapter we study the computational complexity of aggregation functions over regular document spanners. Given an aggregation function α, a spanner S, and a document d, our main objective is to understand when it is tractable to compute an aggregate $\alpha(S(d))$. Furthermore, when exact computation is intractable, we study whether or not the aggregate can be approximated. To the best of our knowledge, counting the number of tuples extracted by a vset-automaton (i.e., the Count aggregate function) is the only aggregation function for document spanners, which has been studied in literature.[1] That is, Florenzano et al. [48] study the problem of counting the number of extractions of a vset-automaton and approximation thereof is studied by Arenas et al. [12]. To be specific, Arenas et al. [12] give a polynomial-time uniform sampling algorithm from the space of words which are accepted by an NFA and have a given length. Using that sampling, they establish an FPRAS for the Count aggregate function. Our FPRAS results are also based on their results. Throughout this chapter, we explain the connection between the known results and our work in more detail. Yet, to the best of our knowledge, this work is the first to consider aggregate functions over numerical values extracted by document spanners.

Organization

This chapter is organized as follows. In Section 6.1, we give preliminary definitions and notation. We summarize the main results of this chapter in Section 6.2 and expand on these results in the later sections. In Section 6.3 we give some preliminary results. We describe our investigation for constant-width weight functions, polynomial-time weight functions and regular weight functions in Sections 6.4, 6.5 and 6.6, respectively. Finally, we study approximate evaluations in Section 6.7.

[1] Arguably we also implicitly discuss the counting problem in Chapter 5 and study the maximum aggregation in Section 5.6.2.

`T h e r e ⊔ a r e ⊔ 7 ⊔ e v e n t s ⊔ i n ⊔ B e l g i u m , ⊔ 1 0 - 1 5 ⊔ i n ⊔`

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 30 31 32 33 34 35 36 37 38 39 40

`F r a n c e , ⊔ 4 ⊔ i n ⊔ L u x e m b o u r g , ⊔ t h r e e ⊔ i n ⊔ B e r l i n .`

41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81

$d_{x_{\mathsf{loc}}}$	$d_{x_{\mathsf{events}}}$	$w(d, \mathrm{t})$
Belgium	7	7
France	10-15	10
Luxembourg	4	4
Berlin	three	3

x_{loc}	x_{events}	$w(d, \mathrm{t})$
$[23, 30\rangle$	$[11, 12\rangle$	7
$[41, 47\rangle$	$[32, 37\rangle$	10
$[54, 64\rangle$	$[49, 50\rangle$	4
$[75, 81\rangle$	$[66, 71\rangle$	3

Figure 6.1: A document d (top), a string relation with corresponding weights (bottom left), and the corresponding span relation R with weights (bottom right).

6.1 Preliminaries on Aggregates

In this section, we will give some additional preliminaries. Recall that given a document d and a vset-automaton A, testing whether $[\![A]\!](d) \neq \emptyset$ is NP-complete, even if A is deterministic (cf. Proposition 2.2.7). Therefore, we only consider functional document spanners in this chapter.[2]

6.1.1 Aggregate Queries

Aggregation functions, such as min, max, and sum operate on numerical values from database tuples, whereas all the values of d-tuples are spans. Yet, these spans may represent numerical values, from the document d, encoded by the captured words (e.g., "3," "three," "March" and so on). To connect spans to numerical values, we will use *weight functions* w that map document/tuple pairs to numbers in $\mathbb{Q}$, that is, if d is a document and t is a d-tuple then $w(d, \mathrm{t}) \in \mathbb{Q}$. We discuss weight functions in more detail in Section 6.2.3.

Example 6.1.1. Consider the document in Figure 6.1 and assume that we want to calculate the total number of mentioned events. The table at the bottom left depicts a possible extraction of locations with their number of events, where each tuple is annotated with a weight $w(d, \mathrm{t})$. The table on the bottom right depicts the corresponding span relation. To get an understanding of the total number of events, we may want to take the sum over the weights of the extracted tuples, namely $7 + 10 + 4 + 3 = 24$. □

For a spanner S, a document d, and weight function w, we denote by $\mathit{Img}(S, d, w)$ the set of weights of output tuples of S on d, that is, $\mathit{Img}(S, d, w) = \{w(d, \mathrm{t}) \mid \mathrm{t} \in S(d)\}$.

[2] We note in general it suffices for the emptiness problem that vset-automaton is sequential. However, some of our results build upon $\mathbb{K}$-Annotators, which correspond to functional spanners.

Furthermore, let $Img(w) \subseteq \mathbb{Q}$ be the set of weights assigned by w, that is, $k \in Img(w)$ if and only if there is a document d and a d-tuple t with $w(d, \mathrm{t}) = k$.

Definition 6.1.2. Let d be a document and A be a vset-automaton such that $[\![A]\!](d) \neq \emptyset$. Let $S = [\![A]\!]$, let w be a weight function, and $q \in \mathbb{Q}$ with $0 \leq q \leq 1$. We define the following spanner aggregation functions:

$$\begin{aligned}
\mathrm{Count}(S,d) &:= \ |S(d)| \\
\mathrm{Min}(S,d,w) &:= \min_{\mathrm{t} \in S(d)} w(d,\mathrm{t}) \\
\mathrm{Max}(S,d,w) &:= \max_{\mathrm{t} \in S(d)} w(d,\mathrm{t}) \\
\mathrm{Sum}(S,d,w) &:= \sum_{\mathrm{t} \in S(d)} w(d,\mathrm{t}) \\
\mathrm{Avg}(S,d,w) &:= \frac{\mathrm{Sum}(S,d,w)}{\mathrm{Count}(S,d)} \\
q\text{-Quantile}(S,d,w) &:= \ \min \left\{ r \in Img(S,d,w) \;\middle|\; \frac{|\{\mathrm{t} \in S(d) \mid w(d,\mathrm{t}) \leq r\}|}{|S(d)|} \geq q \right\}
\end{aligned}$$

Observe that $\mathrm{Min}(S,d,w) = 0\text{-Quantile}(S,d,w)$ and $\mathrm{Max}(S,d,w) = 1\text{-Quantile}(S,d,w)$.

6.1.2 Main Problems

Let $\mathcal{S}$ be a class of regular document spanners and $\mathcal{W}$ be a class of weight functions. We define the following problems.

	$\textsc{Count}[\mathcal{S}]$
Input:	Spanner $S \in \mathcal{S}$ and document $d \in \Sigma^*$.
Question:	Compute $\mathrm{Count}(S,d)$.

	$\textsc{Sum}[\mathcal{S},\mathcal{W}]$
Input:	Spanner $S \in \mathcal{S}$, document $d \in \Sigma^*$, a weight function $w \in \mathcal{W}$.
Question:	Compute $\mathrm{Sum}(S,d,w)$.

The problems $\textsc{Average}[\mathcal{S},\mathcal{W}], q\text{-}\textsc{Quantile}[\mathcal{S},\mathcal{W}], \textsc{Min}[\mathcal{S},\mathcal{W}]$, and $\textsc{Max}[\mathcal{S},\mathcal{W}]$ are defined analogously to $\textsc{Sum}[\mathcal{S},\mathcal{W}]$. Notice that all these problems study *combined complexity*. Since the number of tuples in $S(d)$ is always in $O(|d|^{2k})$, where k is the number of variables of the spanner S (cf. Corlollary 6.3.4), the *data complexity* of all the problems is in FP: One can just materialize $S(d)$ and apply the necessary aggregate. Under combined complexity, we will therefore need to find ways to avoid materializing $S(d)$ to achieve tractability.

6.1.3 Algorithms and Complexity Classes

We begin by giving the definitions of fully polynomial-time randomized approximation schemes (FPRAS).

Definition 6.1.3. Let f be a function that maps inputs x to rational numbers and let $\mathcal{A}$ be a probabilistic algorithm, which takes an input instance x and a parameter $\delta > 0$. Then $\mathcal{A}$ is called a *fully polynomial-time randomized approximation scheme* (FPRAS), if

- $\Pr\Big(\big|\mathcal{A}(x,\delta) - f(x)\big| \ \leq \ \delta \cdot \big|f(x)\big|\Big) \ \geq \ \frac{3}{4}$;
- the runtime of $\mathcal{A}$ is polynomial in $|x|$ and $\frac{1}{\delta}$.

We will now recall the definitions for some of the complexity classes we will use in the following sections, closely following the Handbook of Theoretical Computer Science [164]. The class FP (respectively, FEXPTIME) is the set of all functions that are computable in polynomial time (resp., in exponential time). A *counting Turing Machine* is an nondeterministic Turing Machine whose output for a given input is the number of accepting computations for that input. Given functions $f, g : \Sigma^* \to \mathbb{N}$, f is said to be *parsimoniously reducible to g* in polynomial time if there is a function $h : \Sigma^* \to \Sigma^*$, which is computable in polynomial time, such that for every $x \in \Sigma^*$ it holds that $f(x) = g(h(x))$. Furthermore, we say that f is *Turing reducible to g* in polynomial time, if f can be computed by a polynomial time Turing Machine M, which has access to an oracle for g.

The class #P is the set of all functions that are computable by polynomial-time counting Turing Machines. A problem X is #P-*hard* under parsimonious reductions (resp., Turing reductions) if there are polynomial time parsimonious reductions (resp., Turing reductions) to it from all problems in #P. If in addition $X \in$ #P, we say that X is #P-complete under parsimonious reductions (resp., Turing reductions).

The class $\mathrm{FP}^{\#\mathrm{P}}$ is the set of all functions that are computable in polynomial time by an oracle Turing Machine with a #P oracle. It is easy to see that, under Turing reductions, a problem is hard for the class #P if and only if it is hard for $\mathrm{FP}^{\#\mathrm{P}}$. We note that every problem which is #P-hard under parsimonious reductions is also #P-hard under Turing reductions. Therefore, unless mentioned otherwise, we always use parsimonious reductions.

The class spanL is the class of all functions $f : \Sigma^* \to \mathbb{N}$ for which there is an nondeterministic logarithmic space Turing Machine M with input alphabet Σ such that $f(x) = |M(x)|$.

The class OptP is the set of all functions computable by taking the maximum output value over all accepting computations of a polynomial-time nondeterministic Turing Machine that outputs natural numbers. Assume that Γ is the Turing Machine alphabet. Let $f, g : \Gamma^* \to \mathbb{N}$ be functions. A *metric reduction*, as introduced by Krentel [85], from f to g is a pair of polynomial-time computable functions T_1, T_2, where $T_1 : \Gamma^* \to \Gamma^*$ and $T_2 : \Gamma^* \times \mathbb{N} \to \mathbb{N}$, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Gamma^*$.

The class BPP is the set of all decision problems solvable in polynomial time by a probabilistic Turing Machine in which the answer always has probability at least $\frac{1}{2} + \delta$ of being correct for some fixed $\delta > 0$.

6.2 Main Results

In this section we present the main results of this chapter.

6.2.1 Known Results

We begin by giving an overview of the results on Count, which are known from the literature.

Theorem 6.2.1 (Arenas et al. [12], Florenzano et al. [48])**.** *Count*[ufVSA] *is in* FP *and Count*[fVSA] *is* spanL*-complete. Furthermore, Count*[fVSA] *can be approximated by an* FPRAS*.*

Proof. Follows from Arenas et al. [12, Corollaries 4.1 and 4.2], and Florenzano et al. [48, Theorem 5.2]. □

The spanL lower bound by Florenzano et al. [48, Theorem 5.2] is due to a parsimonious reduction from the #NFA(n)-problem[3] which is known to be #P-complete under Turing reductions (cf. Kannan et al. [75]). As every parsimonious reduction is also a Turing reduction, the following corollary follows immediately.

Corollary 6.2.2. *Count*[fVSA] *is* #P*-hard under Turing reductions.* □

Two observations can be made from these results. First, Count requires the input spanner to be *unambiguous* for tractability. This tractability implies that Count can be computed without materializing the possibly exponentially large set $S(d)$ if the spanner is unambiguous. Furthermore, if the spanner is not unambiguous then, due to spanL-completeness of Count, we do not know an efficient algorithm for its exact computation (and therefore may have to materialize $S(d)$), but Count can be *approximated* by an FPRAS. We will explore to which extent this picture generalizes to other aggregates.

6.2.2 Overview of New Results

The complexity results are summarized in Table 6.1. By now the reader is familiar with the aggregate problems and the types of spanners we study. In the next subsection (Section 6.2.3), we will define the different representations of weight functions that we use. Here, CWidth (respectively, $\text{CWidth}_{\mathbb{N}}$ and $\text{CWidth}_{\mathbb{Q}^+}$) are constant-width weight functions (which only assign natural numbers or positive rationals), Poly are polynomial-time computable weight functions, and Reg (resp., UReg) are weight functions represented by weighted (resp., unambiguous weighted) vset-automata.

Entries in the table should be read from left to right. For instance, the third row states that the Min problem, for both spanner classes ufVSA and fVSA, and for all three classes CWidth, $\text{UReg}_{\mathbb{T}}$, and $\text{Reg}_{\mathbb{T}}$ of weight functions is in FP. Likewise, the fourth row

[3]Given an NFA A and a natural number n, encoded in binary, the #NFA(n) problem asks for the number of words $w \in \mathcal{L}(A)$ of length n. The #NFA(n) problem is sometimes also called Census Problem.

Aggregate	Spanner	Weights	Complexity	Approximation
COUNT	ufVSA	-	in FP	-
	fVSA	-	#P-hard†	FPRAS
MIN	ufVSA, fVSA	CWIDTH, UREG, $\text{REG}_{\mathbb{T}}$	in FP	-
		$\text{REG}_{\mathbb{Q}}$, POLY	OptP-hard	no FPRAS
MAX	ufVSA, fVSA	CWIDTH, UREG	in FP	-
		REG, POLY	OptP-hard	no FPRAS
SUM	ufVSA	CWIDTH, UREG, $\text{REG}_{\mathbb{Q}}$	in FP	-
		$\text{REG}_{\mathbb{T}}$, POLY	#P-hard	no FPRAS
	fVSA	$\text{CWIDTH}_{\mathbb{N}}$	spanL-complete	FPRAS
		CWIDTH, UREG, REG, POLY	#P-hard	no FPRAS
AVERAGE	ufVSA	CWIDTH, UREG, $\text{REG}_{\mathbb{Q}}$	in FP	-
		$\text{REG}_{\mathbb{T}}$, POLY	#P-hard	no FPRAS
	fVSA	$\text{CWIDTH}_{\mathbb{Q}^+}$	#P-hard†	FPRAS
		CWIDTH, UREG, REG, POLY	#P-hard†	no FPRAS
q-QUANTILE	ufVSA	CWIDTH	in FP	-
		UREG, REG, POLY	#P-hard†	no FPRAS
	fVSA	CWIDTH, UREG, REG, POLY	#P-hard†	no FPRAS
q-QUANTILE (positional)	fVSA	POLY	-	FPRAS-like approximation

Table 6.1: Detailed overview of complexities of aggregate problems for document spanners. All problems are in FEXPTIME. The "no FPRAS" claims either assume that RP $\neq$ NP or assume that the polynomial hierarchy does not collapse. The #P-hardness results, marked with † rely on Turing reductions.

states that the same problems with $\text{REG}_{\mathbb{Q}}$ or POLY weight functions become OptP-hard and that the existence of an FPRAS would contradict commonly believed conjectures.

In general, the table gives a detailed overview of the impact of (1) unambiguity of spanners and (2) different weight function representations on the complexity of computing aggregates.

6.2.3 Results for Different Weight Functions

We formalize how we represent the weight functions for our new results. Recall that weight functions w map pairs consisting of a document d and d-tuple t to values in $\mathbb{Q}$.

Constant-Width Weight Functions

The simplest type of weight functions we consider are the *constant-width weight functions.*[4] Let $1 \leq c \in \mathbb{N}$ be a constant. A *constant-width weight function (CWIDTH) w* assigns values based on the strings selected by at most c variables. A constant-width weight function CWIDTH is given in the input as a relation R over the numerical semiring $\mathbb{Q} = (\mathbb{Q}, +, \times, 0, 1)$ and the variables X, where $X \subseteq \text{Vars}$ is a set of at most c variables. Recall that d_{t} denotes the tuple $(d_{\text{t}(x_1)}, \ldots, d_{\text{t}(x_n)})$, where $\text{Vars}(\text{t}) = \{x_1, \ldots, x_n\}$. To facilitate presentation, we assume that the variables in X are always present in t, that is, $X \subseteq \text{Vars}(\text{t})$. The weight function $w(d, \text{t})$ is defined as

$$w(d, \text{t}) = R(d_{\pi_X \text{t}}) \ .$$

As we will see in Section 6.4, MAX[fVSA, CWIDTH] and MIN[fVSA, CWIDTH] are in FP (Theorem 6.4.1). Furthermore, we show that the problems SUM[$\mathcal{S}$, CWIDTH], AVERAGE[$\mathcal{S}$, CWIDTH], and q-QUANTILE[$\mathcal{S}$, CWIDTH] behave similarly to COUNT[$\mathcal{S}$], that is, they are in FP if $\mathcal{S}$ = ufVSA (Theorem 6.4.3) and intractable if $\mathcal{S}$ = fVSA (Theorems 6.4.4, 6.4.5, and 6.4.6).

Polynomial-Time Weight Functions

How far can we push our tractability results? Next, we consider more general ways of mapping d-tuples into numbers. The most general class of weight functions we consider is the set of *polynomial-time weight functions (POLY)*. A function w from POLY is given in the input as a polynomial-time Turing Machine M that maps (d, t) pairs to values in $\mathbb{Q}$ and defines $w(d, \text{t}) = M(d, \text{t})$. Not surprisingly there are multiple drawbacks of having arbitrary polynomial time weight functions. The first is that all considered aggregates become intractable, even if we only consider unambiguous vset-automata (Theorems 6.5.1, and 6.5.2). However, all aggregates can at least be computed in exponential time (Theorem 6.5.3).

Regular Weight Functions

As the class of polynomial-time weight functions quickly leads to intractability, we focus on a restricted class that is less restrictive than CWIDTH but not as general as POLY, such that we can understand the structure of the representation towards efficient algorithms. Our final classes of weight functions are based on $\mathbb{K}$-*Annotators* as defined in Chapter 5. More precisely, we consider (unambiguous) functional weighted vset-automata over the tropical semiring $\mathbb{T} = (\mathbb{Q} \cup \{\infty\}, \min, +, \infty, 0)$ and the numerical semiring $\mathbb{Q} = (\mathbb{Q}, +, \times, 0, 1)$.[5] Formally, let $\text{REG} := \text{REG}_{\mathbb{T}} \cup \text{REG}_{\mathbb{Q}}$ be the class of all Annotators over the tropical or numerical semiring. We observe that due to Propositions 5.4.2

[4] We note that this is an extension of the single-variable weight functions, which where studied by Doleschal et al. [31].

[5] One can also consider the tropical semiring with max/plus, in which case the complexity results are analogous to the ones we have for the tropical semiring with min/plus, with MIN and MAX interchanged.

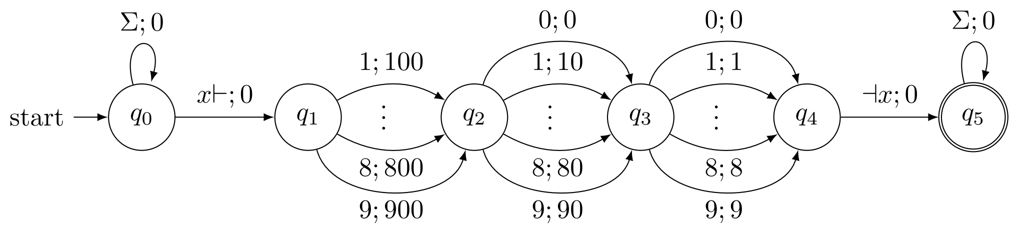

Figure 6.2: An unambiguous functional weighted vset-automaton over the tropical semiring with initial state q_0 (with weight 0) and accepting state q_5 (with weight 0), extracting three digit natural numbers captured in variable x. Recall that, over the tropical semiring, the weight of a run is the sum of all its edge weights.

and 5.4.4, both semirings have efficient encodings. Therefore all complexity results of Chapter 5 hold. A *regular (*R*EG) weight function* w is represented by a functional weighted vset-automaton $W \in \text{Reg}$ and defines $w(d, \mathrm{t}) = [\![W]\!](d, \pi_{\text{Vars}(W)}(\mathrm{t}))$. Furthermore, as for constant width weight functions, we assume that the variables used by W are always present in t, that is, $\text{Vars}(W) \subseteq \text{Vars}(\mathrm{t})$.

The set of *unambiguous regular (UReg) weight functions* is the subset of Reg that is represented by unambiguous functional weighted vset-automata, that is, $\text{UReg} := \text{UReg}_{\mathbb{T}} \cup \text{UReg}_{\mathbb{Q}}$.

Example 6.2.3. Figure 6.2 gives an unambiguous functional weighted vset-automaton over the tropical semiring that extracts the values of three-digit natural numbers from text. It can easily be extended to extract natural numbers of up to a constant number of digits by adding nondeterminism. Likewise, it is possible to extend it to extract weights as in Example 6.1.1. If a single variable captures a list of numbers, similar to $d_{[32,37\rangle} = 10{-}15$, one may use ambiguity to extract the minimal number represented in this range.

Our results for regular and unambiguous regular weight functions are that the situation is similar to CWidth when it comes to Min, Max, Sum, and Average. The main difference is that, depending on the semiring, we require more unambiguity. For instance, for the tropical semiring, one needs unambiguity of the regular weight function for Max and for Sum, and Average one needs unambiguity for *both* the spanner and the regular weight function to achieve tractability. Contrary, over the numerical semiring, one needs unambiguity of the regular weight function for Min and Max, whereas for Sum and Average unambiguity of the spanner is sufficient for tractability. For q-Quantile, the situation is different from CWidth in the sense that regular weight functions render the problem intractable. We refer to Table 6.1 for an overview.

6.2.4 Approximation

In the cases where exact computation of the aggregate problem is intractable, we consider the question of approximation. It turns out that there exist FPRAS's in two settings that we believe to be interesting. Firstly, in the case of SUM and AVERAGE and constant-width weight functions, the restriction of unambiguity in the spanner can be dropped if the weight function uses only nonnegative weights. Secondly, although q-QUANTILE is #P-hard under Turing reductions for general fVSA, it is possible to *positionally* approximate the Quantile element in an FPRAS-like fashion, even with the very general polynomial-time weight functions. We discuss this problem in more detail in Section 6.7.

6.3 Preliminary Results

In this section, we give some basic results for document spanners and weight functions, which we use throughout this chapter. That is, we study the relative expressiveness of the previously defined classes of weight functions in Section 6.3.1 and give some preliminary results on document spanners in Section 6.3.2.

6.3.1 Relative Expressiveness of Weight Functions

We begin by showing that every constant-width weight function is also an unambiguous regular weight function.

Proposition 6.3.1. *CWIDTH* $\subseteq$ *UREG*$_{\mathbb{Q}} \cap$ *UREG*$_{\mathbb{T}}$.

Proof. Let $w \in$ CWIDTH be a constant-width weight function, represented by a $\mathbb{Q}$-relation R over X. We begin by showing that $w \in \text{UREG}_{\mathbb{Q}}$. Let $X = \{x_1, \dots, x_n\}$. We construct a $\mathbb{Q}$-annotator W representing w. Recall that the tuples in R are over the domain of documents and not over spans. We define an unambiguous vset-automaton A_{t}, for every tuple $\mathrm{t} \in R$, such that $\mathrm{t}' \in [\![A_{\mathrm{t}}]\!]_{\mathbb{B}}(d)$ if and only if $d_{\mathrm{t}'} = \mathrm{t}$. Let $\mathrm{t} \in R$. For every $x \in X$, let

$$A_{\mathrm{t}}^{x} := \Sigma^* \cdot x\{\mathrm{t}(x)\} \cdot \Sigma^*$$

and

$$A_{\mathrm{t}} := A_{\mathrm{t}}^{x_1} \bowtie \cdots \bowtie A_{\mathrm{t}}^{x_n} \ .$$

It is straightforward to verify that all A_{t}^{x} are unambiguous. Thus, due to Corollary 5.5.10, the automaton A_{t} is also unambiguous.

We define W_{t} as the unambiguous functional $\mathbb{Q}$-weighted vset-automaton, such that

$$[\![W_{\mathrm{t}}]\!]_{\mathbb{Q}}(d, \mathrm{t}') = \begin{cases} R(\mathrm{t}) & \text{if } d_{\mathrm{t}'} = \mathrm{t} \\ \overline{0} & \text{otherwise.} \end{cases}$$

This can be achieved by interpreting A_{t} as a $\mathbb{Q}$-weighted vset-automaton, where all edges have weight $\overline{1}$, the final weight function assigns weight $\overline{1}$ to all accepting states, and the initial weight function assigns weight $R(\mathrm{t})$ to the initial state of A_{t}. We finally define W as the union of all W_{t}. That is,

$$W = \bigcup_{\mathrm{t} \in R} W_{\mathrm{t}} \ .$$

We observe that, by Lemma 5.5.5, W must be unambiguous as all W_{t} are unambiguous and the automata W_{t} are pairwise disjoint. Recall that $[\![W]\!]_{\mathbb{Q}}(d, \mathrm{t}) = \overline{0} = 0$ if there is no run of W on $\mathrm{ref}(d, \mathrm{t})$, i.e. $d_{\mathrm{t}} \notin R$. Therefore, $[\![W]\!]_{\mathbb{Q}}(d, \mathrm{t}) = R(d_{\mathrm{t}})$ as desired.

The proof for CWIDTH $\subseteq$ UREG$_{\mathbb{T}}$ follows the same lines. However, the zero element of the tropical semiring is ∞ which implies that the automaton W must have exactly one run ρ for every tuple t, even if $w(d, \mathrm{t}) = 0$. To this end, let W_{t} be as defined before, but interpreted over the tropical semiring. We construct an unambiguous functional $\mathbb{T}$-weighted vset-automaton $W_{\overline{R}}$, such that $[\![W_{\overline{R}}]\!]_{\mathbb{T}}(d, \mathrm{t}) = 0$ if $d_{\mathrm{t}} \notin R$ and $W_{\overline{R}}$ has no run for t otherwise. Observe that R is a recognizable string relation.[6] Therefore, due to Theorem 5.5.11, there is a document spanner A_R, with $\mathrm{t} \in [\![A_R]\!](d)$ if and only if $d_{\mathrm{t}} \in R$. Furthermore, let $A_{\overline{R}}$ be the complement of A_R, that is, $\mathrm{t} \in [\![A_{\overline{R}}]\!](d)$ if and only if $d_{\mathrm{t}} \notin R$. Note that $A_{\overline{R}} \in$ VSA as regular document spanners are closed under difference (cf. Fagin et al. [45, Theorem 5.1]). By Proposition 2.2.6, we can assume, w.l.o.g., that $A_{\overline{R}} \in$ dfVSA. Let $W_{\overline{R}}$ be $A_{\overline{R}}$, interpreted as $\mathbb{T}$-weighted vset-automaton, that is, each transition, initial and final state gets weight $\overline{1} = 0$. Note that, due to $A_{\overline{R}} \in$ dfVSA, $W_{\overline{R}}$ is unambiguous and functional. It follows that $[\![W_{\overline{R}}]\!]_{\mathbb{T}}(d, \mathrm{t}) = 0$ if $d_{\mathrm{t}} \notin R$ and $W_{\overline{R}}$ has no run for t otherwise. Let

$$W = W_{\overline{R}} \cup \bigcup_{\mathrm{t} \in R} W_{\mathrm{t}} \ .$$

Again, we observe that, by Lemma 5.5.5, W must be unambiguous as all involved automata are unambiguous and pairwise disjoint. Furthermore,

$$[\![W]\!]_{\mathbb{T}}(d, \mathrm{t}) = \begin{cases} R(d_{\mathrm{t}}) & \text{if } d_{\mathrm{t}} \in R \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $[\![W]\!]_{\mathbb{T}}(d, \mathrm{t}) = R(d_{\mathrm{t}})$ as desired. ☐

Recall that, given a document d and a d-tuple t, the weight $w(d, \mathrm{t})$ can be computed in polynomial time (cf. Theorem 5.6.1). We can therefore make the following observation.

Observation 6.3.2. *REG* $\subseteq$ *POLY*. ☐

6.3.2 Technical Foundations

We give some preliminary results which will be used throughout this chapter. That is, we first show that the number of spans over a document d is polynomial in the size of the document.

[6] Recall that a k-ary string relation is 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. Note that R is recognizable as it is the union over all tuples $\mathrm{t} \in R$, where each tuple is represented by the Cartesian product $\{\mathrm{t}(x_1)\} \times \cdots \times \{\mathrm{t}(x_n)\}$ with $\mathrm{Vars}(\mathrm{t}) = \{x_1, \ldots, x_n\}$.

Lemma 6.3.3. *Given a document d the number of spans over d is polynomial in the size of d. That is, $|\mathsf{Spans}(d)| = \frac{(|d|+1)\cdot(|d|+2)}{2}$, for every $d \in \Sigma^*$.*

Proof. For a span $[i,j\rangle$, let $\ell = j - i$ be the length of the span. It is easy to see that for any document d, there is exactly one span of length $|d|$, two spans of length $|d|-1$, three spans of length $|d|-2$, etc. Thus, there are $1 + 2 + \cdots + (|d|+1) = \frac{(|d|+1)\cdot(|d|+2)}{2}$ spans over a document d. Therefore, $\mathsf{Spans}(d) = \frac{(|d|+1)\cdot(|d|+2)}{2}$, concluding the proof. $\square$

It follows directly that the maximal number of tuples, extracted by a functional document spanner is exponential in the size of the spanner.

Corollary 6.3.4. *Let $A \in \mathrm{fVSA}$ be a vset-automaton and $d \in \Sigma^*$ be a document. Then* $\mathrm{Count}(S,d) \leq |\mathsf{Spans}(d)|^{|\operatorname{Vars}(A)|} = \left(\frac{(|d|+1)\cdot(|d|+2)}{2}\right)^{|\operatorname{Vars}(A)|}$. $\square$

As we show next, given a number of variables, a document d and a number k of tuples, we can construct an unambiguous functional vset-automaton A and a document d' such that A extracts exactly k tuples on d'. We will use this technical lemma throughout this chapter for multiple proofs regarding q-Quantile aggregation.

Lemma 6.3.5. *Let $X := \{x_1, \ldots, x_v\} \in \mathrm{Vars}$ be a set of variables, $d \in \Sigma^*$ be a document, and $0 \leq k \leq |\mathsf{Spans}(d)|^{|X|}$. Then there is a vset-automaton $A \in \mathrm{ufVSA}$ with $\mathrm{Vars}(A) = X$ and a document $d' \in \Sigma^*$ such that $|[\![A]\!](d')| = k$. Furthermore, A and d' can be constructed in time polynomial in $|X|$ and d.*

Proof. We observe that the statement holds for $k = 0$. Therefore we assume, w.l.o.g., that $1 \leq k \leq |\mathsf{Spans}(d)|^v$.

We begin by proving the statement for $|X| = 1$. Let $1 \leq k \leq |\mathsf{Spans}(d)|$. Recalling the proof of Lemma 6.3.3, we observe that k can be written as a sum $k = k_1 + \cdots + k_n$ of $n \leq |d|+1$ different natural numbers with $0 \leq k_1 < \cdots < k_n \leq |d|+1$. We construct an automaton $A_k \in \mathrm{ufVSA}$, which consists of n branches, corresponding to $k_1, \ldots, k_n$. On document d, the branch corresponding to k_i selects all spans of length $\ell_i := |d|+1-k_i$. Slightly overloading notation, each of these branches can be constructed as an unambiguous vset-automaton $A_{k_i} := \Sigma^* \cdot x\{\Sigma^{\ell_i}\} \cdot \Sigma^*$. We observe that there are exactly k_i spans over d with length ℓ_i, and therefore $|[\![A_{k_i}]\!](d)| = k_i$. The automaton A_k is defined as

$$A_k := A_{k_1} \cup \cdots \cup A_{k_n} \ .$$

It is straightforward to verify that all automata A_{k_i} are unambiguous and functional. Thus, due all A_{k_i} being pairwise disjoint, it holds that $A_k \in \mathrm{ufVSA}$ (cf. Lemma 5.5.5). Furthermore, we observe that

$$|[\![A_k]\!](d)| = |[\![A_{k_1}]\!](d)| + \cdots + |[\![A_{k_n}]\!](d)| = k_1 + \cdots + k_n = k \ .$$

It remains to show the statement for $v := |X| > 1$. Let $\# \notin \Sigma$ be a new alphabet symbol. We build upon the encoding for $|X| = 1$. That is, for every $1 \leq k \leq |\mathsf{Spans}(d)|$ let A_k^x be the automaton A_k, using variable x, as defined previously. We observe that every $1 \leq k \leq |\mathsf{Spans}(d)|^v$ has an encoding $k = k_1 \cdots k_v$ in base $|\mathsf{Spans}(d)|$ of length v. The document d' consists of v copies of $d \cdot \#$, more formally

$$d' := (d \cdot \#)^v \ .$$

For every $1 \leq i \leq v$, we construct an automaton A'_{k_i}, which selects exactly $k_i \cdot |\mathsf{Spans}(d)|^{v-i}$ tuples over document d'. More formally,

$$A'_{k_i} := d \cdot x_1\{\#\} \cdot d \cdot x_2\{\#\} \cdots d \cdot x_{i-1}\{\#\} \cdot A^{x_i}_{k_i} \cdot \# \cdot A^{x_{i+1}}_{|\mathsf{Spans}(d)|} \cdot \# \cdots \# \cdot A^{x_v}_{|\mathsf{Spans}(d)|} \cdot \# \ .$$

The automaton A'_k is then defined as the union of all A'_{k_i}, that is,

$$A'_k := A'_{k_1} \cup \cdots \cup A'_{k_v} \ .$$

We observe that $A'_{k_i} \in \text{ufVSA}$ and due to all A'_{k_i} being pairwise disjoint, $A'_k \in \text{ufVSA}$ (cf. Lemma 5.5.5). Furthermore, we observe that

$$|[\![A'_k]\!](d')| = |[\![A'_{k_1}]\!](d')| + \cdots |[\![A'_v]\!](d)'| = k_1 + \cdots + k_n = k \ .$$

This concludes the proof. □

6.4 Constant-Width Weight Functions

We begin this section by showing that Min and Max are tractable for constant-width weight functions. The reason for their tractability is that, for a constant number of variables $X \subseteq \text{Vars}(A)$, the spans associated to X in output tuples can be computed in polynomial time. Building upon Corollary 6.3.4, we show that Min and Max are in FP for constant-width weight functions and functional vset-automata. We immediately have:

Theorem 6.4.1. *Min*[fVSA, *CWidth*] *and Max*[fVSA, *CWidth*] *are in* FP.

Proof. Let $A \in \text{fVSA}$, $d \in \Sigma^*$, $X \subseteq \text{Vars}(A)$ with $|X| \leq c$, and $w \in \text{CWidth}$ be given as a $\mathbb{Q}$-Relation R over X. We first show that the set $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}$ can be computed in time polynomial in the sizes of A and d.

To this end, we observe that, per definition of projection for document spanners, $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\} = \big(\pi_X([\![A]\!])\big)(d)$. Since A is functional, a vset-automaton for $\pi_X([\![A]\!])$ can be computed in polynomial time (cf. Freydenberger et al. [54, Lemma 3.8]). Due to $|X| \leq c$, it follows from Corollary 6.3.4 that there are at most polynomially many tuples in $\big(\pi_X([\![A]\!])\big)(d)$. Thus, the set $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}$ can be materialized in polynomial time.

In order to compute Min and Max, a polynomial time algorithm can iterate over all tuples t in $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}$, evaluate $R(d, \mathrm{t})$ and maintain the minimum and the maximum of these numbers. □

Algorithm 1: Calculate the multiset $\mathcal{S}_{A,d}$.

Input: An unambiguous, functional vset-automaton $A \in$ ufVSA, a document $d \in \Sigma^*$.

Output: The multiset $\mathcal{S}_{A,d}$.

$\mathcal{S} \leftarrow \{\!\!\{\,\}\!\!\}$

$\mathrm{S} \leftarrow \pi_X(⟦A⟧)(d)$

for $\mathrm{t} \in \mathrm{S}$ **do**

  $A_{\mathrm{t}} \leftarrow A \bowtie A_{\mathrm{ref}(d,\mathrm{t})}$ ▷ $A_{\mathrm{ref}(d,\mathrm{t})}$ is the ufVSA that only accepts $\mathrm{ref}(d,\mathrm{t})$.

  $\mathcal{S}(\pi_X \mathrm{t}) \leftarrow \mathrm{Count}(⟦A_{\mathrm{t}}⟧, d)$

output $\mathcal{S}$

In order to calculate aggregates like Sum, Avg, or q-Quantile, it is not sufficient to know which weights are assigned, but also the multiplicity of each weight is necessary. Recall that counting the number of output tuples is tractable if the vset-automaton is functional and unambiguous (Theorem 6.2.1) and spanL-complete if the spanner is only functional. We now show that we can achieve tractability of the mentioned aggregate problems if the vset-automaton is functional and unambiguous. The reason is that we can compute in polynomial time the multiset $\mathcal{S}_{A,d} := \{\!\!\{\pi_X \mathrm{t} \mid \mathrm{t} \in ⟦A⟧(d)\}\!\!\}$, where we represent the multiplicity of each tuple t' (i.e., the number of tuples $\mathrm{t} \in ⟦A⟧(d)$ such that $\pi_X \mathrm{t} = \mathrm{t}'$) in binary.

Lemma 6.4.2. *Given a vset-automaton A and a document d, the multiset $\mathcal{S}_{A,d}$ can be computed in* FP *if $A \in$ ufVSA.*

Proof. The procedure is given as Algorithm 1. It is straightforward to verify that the algorithm is correct. Due to Corollary 6.3.4, the set $(\pi_X⟦A⟧)(d)$ is at most of polynomial size. Furthermore, slightly overloading notation, the automaton $A_{\mathrm{ref}(d,\mathrm{t})} := \mathrm{ref}(d,\mathrm{t}) \in$ ufVSA can be constructed in polynomial time and due to Corollary 5.5.10 an unambiguous functional vset-automaton for A_{t} can be computed in polynomial time as well. By Theorem 6.2.1, each iteration of the for-loop also only requires polynomial time. Thus, the whole algorithm terminates after polynomially many steps. □

It follows that all remaining aggregate functions can be efficiently computed if the spanner is given as an unambiguous functional vset-automaton.

Theorem 6.4.3. *The problems* SUM[ufVSA, CWIDTH], AVERAGE[ufVSA, CWIDTH], *and q-*QUANTILE[ufVSA, CWIDTH] *are in* FP*, for every $0 \leq q \leq 1$.*

Proof. Let $A \in$ ufVSA be a vset-automaton, $d \in \Sigma^*$ be a document, $w \in$ CWIDTH be a weight function, represented by a $\mathbb{Q}$-relation R over X. Due to Lemma 6.4.2 the multiset $\mathcal{S}_{A,d}$ can be computed in polynomial time. Thus one can compute the multiset $W := \{\!\!\{R(d_{\mathrm{t}}) \mid \mathrm{t} \in \mathcal{S}_{A,d}\}\!\!\}$ in polynomial time. It is straightforward to compute the aggregates in polynomial time from W. □

We conclude this section by showing that Sum, Avg, and q-Quantile are not tractable, if the spanner is given as a functional vset-automaton.

Theorem 6.4.4. *$\textsc{Sum}[\text{fVSA}, \textsc{CWidth}]$ is #P-hard, even if w is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with*

$$R(d) := \begin{cases} 1 & \textit{if } d = 1 \\ -1 & \textit{if } d = -1 \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. We will give a reduction from $\#CNF$ which is #P-complete under parsimonious reductions. Let ϕ be a Boolean formula in CNF over variables $x_1, \ldots, x_n$ and let $w \in \textsc{CWidth}$ be the weight function which is represented by the $\mathbb{Q}$-Relation R, which is as defined in the theorem statement.

We construct a vset-automaton $A \in \text{fVSA}$ and a document $d := a^n \cdot - \cdot 1$, such that $\text{Sum}([\![A]\!], d, w) = c$, where c is the number of variable assignments which satisfy ϕ.

We begin by defining two vset-automata A_1, A_{-1}, with $\text{Vars}(A_1) = \text{Vars}(A_{-1}) = \{x_1, \ldots, x_n, x\}$. Slightly overloading notation, we define both automata by regex formulas.

The automaton A_1 selects exactly 2^n tuples on document d, all of which get assigned weight 1 by w. More formally,

$$A_1 := (x_1\{a\} \vee x_1\{\varepsilon\} \cdot a) \cdots (x_n\{a\} \vee x_n\{\varepsilon\} \cdot a) \cdot - \cdot x\{1\} \, .$$

Therefore, $\text{Sum}([\![A_1]\!], d, w) = \text{Count}([\![A_1]\!], d) = 2^n$.

As in the proof of Theorem 5.6.4, we encode variable assignments into tuples. That is, each variable x_i of ϕ is associated with a corresponding capture variable x_i of A_{-1}. With each assignment τ we associate the tuple t_τ, such that

$$\mathrm{t}_\tau(x_i) := \begin{cases} [i, i\rangle & \text{if } \tau(x_i) = 0, \text{ and} \\ [i, i+1\rangle & \text{if } \tau(x_i) = 1 \, . \end{cases}$$

We construct the automaton A_{-1} as a regex formula α, such that there is a one-to-one correspondence between the non-satisfying assignments for ϕ and tuples in $[\![\alpha]\!](d)$. More formally, for each clause C_j of ϕ and each variable x_i, we construct a regex-formula

$$\alpha_{i,j} := \begin{cases} x_i\{\varepsilon\} \cdot a & \text{if } x_i \text{ appears in } C_j, \\ x_i\{a\} & \text{if } \neg x_i \text{ appears in } C_j, \\ (x_i\{\varepsilon\} \cdot a) \vee x_i\{a\} & \text{otherwise.} \end{cases}$$

Consequently, we define $\alpha_j := \alpha_{1,j} \cdots \alpha_{n,j} \cdot x\{-1\}$.

For example, if we use variables x_1, x_2, x_3, x_4 and $C_j = x_1 \vee x_3 \vee \neg x_4$ is a clause, then

$$\alpha_j = x_1\{\varepsilon\} \cdot a \cdot (x_2\{\varepsilon\} \cdot a \vee x_2\{a\}) \cdot x_3\{\varepsilon\} \cdot a \cdot x_4\{a\} \cdot x\{-1\}.$$

We observe that $\mathrm{t} \in [\![\alpha_j]\!](d)$ if and only if the variable assignment τ of ϕ with $\mathrm{t} = \mathrm{t}_\tau$ does not satisfy clause C_j.

We finally define $\alpha := \alpha_1 \vee \cdots \vee \alpha_m$, that is, the disjunction of all α_i and A_{-1} as the vset-automaton, corresponding to α.[7] Therefore, $\mathrm{Count}(\llbracket A_{-1} \rrbracket, d) = s$, where $s = 2^n - c$ is the number of variable assignments which do not satisfy ϕ. Furthermore, per definition of A_{-1} and w, it follows that

$$\mathrm{Sum}(\llbracket A_{-1} \rrbracket, d, w) = -1 \cdot s = -s \; .$$

We finally define the vset-automaton A as the union of A_1 and A_{-1}. We observe that every tuple $\mathrm{t} \in \llbracket A \rrbracket(d)$ is either selected by A_1 (if $d_{\mathrm{t}(x)} = 1$) or by A_{-1} (if $d_{\mathrm{t}(x)} = -1$), but never by both automata. Recall that c is the number of assignments which satisfy ϕ and $s = 2^n - c$ is the number non-satisfying assignments of ϕ. Therefore, we have that

$$\mathrm{Sum}(\llbracket A \rrbracket, d, w) = \mathrm{Sum}(A_1, d, w) + \mathrm{Sum}(A_{-1}, d, w) = 2^n + (-s) = 2^n - (2^n - c) = c \; .$$

This concludes the proof. □

If the weights are restricted to natural numbers, Sum becomes spanL-complete. Note that we restrict weight functions to natural numbers, because spanL is a class of functions that return natural numbers. Allowing positive rational numbers does not fundamentally change the complexity of the problems though. We will see in Section 6.7 that this enables us to approximate Sum aggregates.

Theorem 6.4.5. *Sum[fVSA, CWidth$_{\mathbb{N}}$] is* spanL*-complete, even if w is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with*

$$R(d) := \begin{cases} 1 & \textit{if } d = 1 \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. Recall that a function f is in spanL, if there is a nondeterministic logarithmic space Turing Machine M such that $f(x) = |M(x)|$. Let $A \in \mathrm{fVSA}$ be a vset-automaton, $d \in \Sigma^*$ be a document, and $w \in \mathrm{CWIDTH}_{\mathbb{Q}_+}$ be a weight function. We define M as the Turing Machine, which guesses a d-tuple t and checks whether $\mathrm{t} \in \llbracket A \rrbracket(d)$. If yes, M computes the weight $w(d, \mathrm{t})$, which can be done in NL, since w is given by a $\mathbb{Q}$-Relation. The Turing Machine M then branches into $w(d, \mathrm{t})$ accepting branches. If $\mathrm{t} \notin \llbracket A \rrbracket(d)$, M rejects. Thus, $|M(A, d)| = \mathrm{Sum}(S, d, w)$, and therefore $\mathrm{Sum}[\mathrm{fVSA}, \mathrm{CWIDTH}_{\mathbb{N}}]$ is in spanL.

For the lower bound, we give a reduction from Count[fVSA], which is spanL-complete (cf. Theorem 6.2.1). Let $A \in \mathrm{fVSA}$, $d \in \Sigma^*$. We assume, w.l.o.g., that $1 \notin \Sigma$ and $x \notin \mathrm{Vars}(A)$. We construct a document $d' := d \cdot 1$ and a vset-automaton $A' := A \cdot x\{1\}$. We observe that $\mathrm{Sum}(\llbracket A' \rrbracket, d', w) = \mathrm{Count}(\llbracket A \rrbracket, d)$, concluding the proof. □

We conclude this section by showing that Average and q-Quantile are #P-hard under Turing reductions.

[7] It is easy to verify that the automaton $A_{-1} \in \mathrm{fVSA}$ can be constructed in polynomial time from α.

Theorem 6.4.6. *Let $0 < q < 1$. Then, the problems* $\textsc{Average}[\text{fVSA}, C\textsc{Width}]$ *and* q-$\textsc{Quantile}[\text{fVSA}, C\textsc{Width}]$ *are #P-hard under Turing reductions, even if w is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with*

$$R(d) := \begin{cases} 1 & \textit{if } d = 1 \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. Recall that $\textsc{Count}[\text{fVSA}]$ is #P-hard under Turing reductions. We begin by giving a Turing reduction from $\textsc{Count}[\text{fVSA}]$ to $\textsc{Average}[\text{fVSA}, \text{C}\textsc{Width}]$. Let A, d, and d' be as defined in the proof of Theorem 6.4.5. The vset-automaton A' builds upon A but selects a single additional tuple t with $\mathrm{t}(x) = [|d| + 2, |d| + 2\rangle$ for all variables. As we will see later, this tuple is used to calculate $\text{Count}([\![A]\!], d)$ from $\text{Avg}([\![A']\!], d')$. Let $\text{Vars}(A) = \{x_1, \ldots, x_n\}$. We define

$$A' \ := \ (A \cdot x\{1\}) \ \vee \ (d \cdot 1 \cdot x_1\{x_2\{\cdots x_n\{x\{\varepsilon\}\}\cdots\}) \ .$$

Observe that, for all $\mathrm{t} \in A'(d')$ it holds that $d_{\mathrm{t}(x)} = 1$ if and only if $\pi_{\text{Vars}(A)}\mathrm{t} \in [\![A]\!](d)$. Thus, per definition of A' and w, $\text{Sum}([\![A']\!], d', w) = \text{Count}([\![A]\!], d)$ and $\text{Count}([\![A']\!], d') = \text{Count}([\![A]\!], d) + 1$. Therefore, it holds that

$$\text{Avg}([\![A']\!], d', w) = \frac{\text{Count}([\![A]\!], d)}{\text{Count}([\![A]\!], d) + 1} \ .$$

Solving the equation for $\text{Count}([\![A]\!], d)$, we have that

$$\text{Count}([\![A]\!], d) = \frac{\text{Avg}([\![A']\!], d', w)}{1 - \text{Avg}([\![A']\!], d', w)} \ .$$

This concludes the proof that $\textsc{Average}[\text{fVSA}, \text{C}\textsc{Width}]$ is #P-hard under Turing reductions.

It remains to show that q-$\textsc{Quantile}[\text{fVSA}, \text{C}\textsc{Width}]$ is also #P-hard under Turing reductions. Let $A \in \text{fVSA}$ be a functional vset-automaton and $d \in \Sigma^*$ be a document. We will show the lower bound for $q = \frac{1}{2}$ first and study the general case of $0 < q < 1$ afterwards. Let $x \notin \text{Vars}(A)$ be a new variable. Let $0 \leq r \leq |\mathsf{Spans}(d)|^{|\text{Vars}(A)|}$. By Lemma 6.3.5 there is a vset-automaton A' and a document d' with $\text{Count}([\![A']\!], d') = |[\![A']\!](d')| = r$. Let $0, 1 \notin \Sigma$ be a new alphabet symbol. Let $d_r = 0 \cdot d \cdot 1 \cdot d'$ and

$$A_r = \big(x\{0\} \cdot A \cdot 1 \cdot d'\big) \vee \big(0 \cdot d \cdot x\{1\} \cdot A'\big) \ .$$

Thus, $\text{Count}([\![A_r]\!], d_r) = \text{Count}([\![A]\!], d) + \text{Count}([\![A']\!], d')$. Recalling the definition of w it holds, for every tuple $\mathrm{t} \in [\![A_r]\!]$, that $w(d_r, \mathrm{t}) = 1$ if t was selected by A' and $w(d_r, \mathrm{t}) = 0$ otherwise, i.e., t was selected by A. Therefore, $\frac{1}{2}$-$\text{Quantile}([\![A_r]\!], d_r, w) = 0$ if and only if $\text{Count}([\![A]\!], d) \geq \text{Count}([\![A']\!], d') = r$. Let $r_{\max}$ be the biggest r such that $\frac{1}{2}$-$\text{Quantile}([\![A_r]\!], d_r, w) = 0$. Using binary search, we can calculate $r_{\max}$ with a polynomial number of calls to an $\frac{1}{2}$-$\textsc{Quantile}$ oracle. Furthermore, due to $\text{Count}([\![A]\!], d) \in \mathbb{N}$ and

$R_{\max}$ being maximal, it must hold that $\mathrm{Count}(\llbracket A \rrbracket, d) = r_{\max}$, concluding this part of the proof.

The general case of $0 < q < 1$ follows by slightly adopting the above reduction. Let $q = \frac{a}{b}$ with $a, b \in \mathbb{N}$ be given by its numerator and denominator. Observe that $b > a$ as $0 < \frac{a}{b} < 1$. Let A', d' be as above and let $c := \mathrm{Count}(\llbracket A \rrbracket, d)$. The document d_r consists of a copies of d, separated by $0's$ and $(b-a)$ copies of d' separated by $1's$. Formally, $d_r = 0 \cdot d_1 \cdot 0 \cdot d_2 \cdot 0 \cdots d_a \cdot 0 \cdot 1 \cdot d'_1 \cdot 1 \cdot d'_2 \cdot 1 \cdots d'_{b-a} \cdot 1$, where each d_i (resp. d'_i) is a copy of d (resp. d'). Furthermore, let

$$A_r = \left(\Sigma_0^* \cdot x\{0\} \cdot A \cdot 0 \cdot \Sigma_0^* \cdot \Sigma_1^*\right) \vee \left(\Sigma_1^* \cdot \Sigma_1^* \cdot x\{1\} \cdot A' \cdot 1 \cdot \Sigma_1^*\right) ,$$

where $\Sigma_0 := \Sigma \cup \{0\}$ (resp. $\Sigma_1 := \Sigma \cup \{1\}$). Observe that w assigns 0 to exactly $c \cdot a$ tuples in $\llbracket A_r \rrbracket(d_r)$ and $\mathrm{Count}(\llbracket A_r \rrbracket, d_r) = c \cdot a + r \cdot (b-a)$. Thus, $\frac{a}{b}$-Quantile$(A_r, d_r, w) = 0$ if and only if $\frac{c \cdot a}{c \cdot a + r \cdot (b-a)} \geq \frac{a}{b}$. We now show that $c \geq r$ if and only if $\frac{a}{b}$-Quantile$(\llbracket A_r \rrbracket, d_r, w) = 0$. Assume that $c \geq r$. Then,

$$\frac{c \cdot a}{c \cdot a + r \cdot (b-a)} \geq \frac{c \cdot a}{c \cdot a + c \cdot (b-a)} = \frac{c \cdot a}{c \cdot b} = \frac{a}{b} .$$

Therefore, $\frac{a}{b}$-Quantile$(\llbracket A_r \rrbracket, d_r, w) = 0$. On the other hand, if $c < r$,

$$\frac{c \cdot a}{c \cdot a + r \cdot (b-a)} < \frac{c \cdot a}{c \cdot a + c \cdot (b-a)} = \frac{c \cdot a}{c \cdot b} = \frac{a}{b} .$$

Thus, $\frac{a}{b}$-Quantile$(\llbracket A_r \rrbracket, d_r, w) = 1$.

Recall that $c = \mathrm{Count}(\llbracket A \rrbracket, d)$. As for $q = \frac{1}{2}$, let $r_{\max}$ be the biggest r such that $\frac{a}{b}$-Quantile$(\llbracket A_r \rrbracket, d_r, w) = 0$. Using binary search, we can calculate $r_{\max}$ with a polynomial number of calls to an $\frac{a}{b}$-Quantile oracle. Again it holds that $\mathrm{Count}(\llbracket A \rrbracket, d) = r_{\max}$, concluding the proof. □

6.5 Polynomial-Time Weight Functions

Before we study regular weight functions, we make a few observations on the very general polynomial-time computable weight functions. For weight functions $w \in$ Poly, we assume that w is represented as a Turing Machine A that returns a value $A(d, \mathrm{t})$ in polynomially many steps for some fixed polynomial of choice (e.g., n^2).[8] Furthermore, to avoid complexity due to the need to verify whether A is indeed a valid input (i.e., timely termination), we will assume that $w(d, \mathrm{t}) = 0$, if A does not produce a value within the allocated time.

We first observe that polynomial-time weight functions make all our aggregation problems intractable, which is not surprising.

Theorem 6.5.1. *The problems Min*[ufVSA, *Poly*] *and Max*[ufVSA, *Poly*] *are* OptP-*hard. Furthermore, Sum*[ufVSA, *Poly*] *and Average*[ufVSA, *Poly*] *are* #P-*hard.*

[8] Our complexity results are independent of the choice of this polynomial.

Proof. Follows directly from Theorems 6.6.3, and 6.6.7. $\square$

Theorem 6.5.2. *Let* $0 < q < 1$. q-QUANTILE[ufVSA, POLY] *is* #P*-hard under Turing reductions.*

Proof. Follows directly from Theorem 6.6.9. $\square$

In fact, all lower bounds already hold for regular weight functions. We note that all studied problems can be solved in exponential time, by first constructing the relation $[\![A]\!](d)$, which might be of exponential size, computing the weights associated to all tuples, and finally computing the desired aggregate.

Theorem 6.5.3. *Let* $0 < q < 1$*. Then* AGG[fVSA, POLY] *is in* FEXPTIME *for every* AGG $\in$ {MIN, MAX, SUM, AVERAGE, q-QUANTILE}.

Proof. Let $A \in$ fVSA, $d \in \Sigma^*$, and $w \in$ POLY. The algorithm first computes the multiset

$$W_{A,d,w} := \{\!\{ w(d, \mathrm{t}) \mid \mathrm{t} \in [\![A]\!](d) \}\!\} ,$$

which might be exponentially large. It is easy to see that $W_{A,d,w}$ can be computed in exponential time. Furthermore, it follows directly that AGG[fVSA, POLY] is in FEXPTIME for every AGG $\in$ {MIN, MAX, SUM, AVERAGE, q-QUANTILE}. $\square$

Throughout this chapter, we do not study excessively whether we can give a more precise upper bound than the general FEXPTIME upper bound. However, we sometimes give such bounds. For instance, we are able to provide OptP and $\mathrm{FP}^{\#\mathrm{P}}$ upper bounds if the weight functions return natural numbers (or integers in the case of the $\mathrm{FP}^{\#\mathrm{P}}$ upper bounds).

Theorem 6.5.4. MIN[fVSA, POLY] *and* MAX[fVSA, POLY] *are in* OptP *if the weight function only assigns natural numbers.*

Proof. We only give the upper bound for MAX. The proof for MIN is analogous. Let $A \in$ fVSA, $d \in \Sigma^*$, and $w \in$ POLY be a weight function which only assigns natural numbers. The Turing Machine N guesses a d-tuple t and accepts with output 0 if $\mathrm{t} \notin A(d)$. Otherwise, N computes the weight $w(d, \mathrm{t})$ and accepts with output $w(d, \mathrm{t})$. It is easy to see that the maximum output value of N is exactly $\mathrm{Max}([\![A]\!], d, w)$. $\square$

In the following theorem we show that SUM, AVERAGE, and q-QUANTILE can be computed in $\mathrm{FP}^{\#\mathrm{P}}$ if all weights are integers. The key idea is that, due to the restriction to integer weights, we can compute the aggregates by multiple calls to an #P oracle. For instance for SUM, we define two weight functions, w^+ and w^-, such that w^+ computes the sum of all positive and w^- the sum of all negative weights. Each of these sums can be computed by a single call to an #P oracle.

Theorem 6.5.5. *For every* $0 \leq q \leq 1$, SUM[fVSA, POLY], AVERAGE[fVSA, POLY], *and* q-QUANTILE[fVSA, POLY] *are in* $\mathrm{FP}^{\#\mathrm{P}}$ *if the weight function only assigns integers.*

Proof. We first prove that $\textsc{Sum}[\text{fVSA}, \textsc{Poly}]$ is in #P if the weight function only assigns natural numbers. We will use this as an oracle for the general upper bound. Let A be a vset-automaton, $d \in \Sigma^*$ be a document and $w \in \textsc{Poly}$ be a weight function that only assigns natural numbers. A counting Turing Machine M for solving the problem in #P would have $w(d, \mathrm{t})$ accepting runs for every tuple in $A(d)$. More precisely, M guesses a d-tuple t over $\mathrm{Vars}(A)$ and checks whether $\mathrm{t} \in [\![A]\!](d)$. If $\mathrm{t} \in [\![A]\!](d)$ and $w(d, \mathrm{t}) > 0$, then M branches into $w(d, \mathrm{t})$ accepting branches, which it can do because w is given in the input as a polynomial-time deterministic Turing Machine. Otherwise, M rejects. Per construction, M has exactly $w(d, \mathrm{t})$ accepting branches for every tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) > 0$. Thus, the number of accepting runs is exactly $\sum_{\mathrm{t} \in [\![A]\!](d)} w(d, \mathrm{t}) = \mathrm{Sum}([\![A]\!], d, w)$.

We now continue by showing that $\textsc{Sum}[\text{fVSA}, \textsc{Poly}]$ is in $\mathrm{FP}^{\#\mathrm{P}}$ if the weight function only assigns integers. Let A be a vset-automaton, $d \in \Sigma^*$ be a document, and $w \in \textsc{Poly}$ be a weight function, which only assigns integers.

We define two weight functions $w^+, w^- \in \textsc{Poly}$, such that

$$\mathrm{Sum}(A, d, w) = \mathrm{Sum}(A, d, w^+) - \mathrm{Sum}(A, d, w^-) \; .$$

Formally, we define the following two weight functions:

$$w^+(d, \mathrm{t}) := \begin{cases} w(d, \mathrm{t}) & \text{if } w(d, \mathrm{t}) \geq 0 \text{, and} \\ 0 & \text{otherwise;} \end{cases}$$

$$w^-(d, \mathrm{t}) := \begin{cases} -w(d, \mathrm{t}) & \text{if } w(d, \mathrm{t}) < 0 \text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $\mathrm{Sum}([\![A]\!], d, w) = \mathrm{Sum}([\![A]\!], d, w^+) - \mathrm{Sum}([\![A]\!], d, w^-)$ and the answer to $\textsc{Sum}[\mathcal{S}, \textsc{Poly}]$ can be obtained by taking the difference of the answers of two calls to the $\textsc{Sum}[\mathcal{S}, \textsc{Poly}]$ #P oracle. The upper bound for $\textsc{Average}[\text{fVSA}, \textsc{Poly}]$ is immediate from the upper bound of $\textsc{Sum}[\text{fVSA}, \textsc{Poly}]$ and Theorem 6.2.1. For the upper bound of q-$\textsc{Quantile}[\text{fVSA}, \textsc{Poly}]$ we define the weight function

$$w_{\leq k}(d, \mathrm{t}) = \begin{cases} 1 & \text{if } w(d, \mathrm{t}) \leq k \text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

Recall that

$$q\text{-Quantile}(S, d, w) := \min \left\{ r \in \mathit{Img}(S, d, w) \;\middle|\; \frac{|\{\mathrm{t} \in S(d) \mid w(d, \mathrm{t}) \leq r\}|}{|S(d)|} \geq q \right\} .$$

Therefore

$$q\text{-Quantile}(S, d, w) = \min \left\{ r \in \mathit{Img}(S, d, w) \;\middle|\; \frac{\mathrm{Sum}([\![A]\!], d, w_{\leq k})}{\mathrm{Count}([\![A]\!], d)} \geq q \right\} .$$

Thus, the upper bound of q-$\textsc{Quantile}[\text{fVSA}, \textsc{Poly}]$ can be obtained by performing binary search, using the upper bound of $\textsc{Sum}[\text{fVSA}, \textsc{Poly}]$ and Theorem 6.2.1. □

6.6 Regular Weight Functions

We now turn to Reg and UReg weight functions. As we have shown in Proposition 6.3.1, every CWidth weight functions can be translated into an equivalent UReg weight function. Furthermore, the weight functions which where used for the lower bounds can be represented by unambiguous functional weighted vset-automata of constant size. Therefore, all lower bounds for CWidth also hold for UReg.

6.6.1 Compact DAG Representation

As we show next, aggregation problems for regular weight functions can often be reduced to problems about paths on weighted *directed acyclic graphs (DAGs)*, where the weights come from the semiring of the weight function. Let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring. A $\mathbb{K}$-*weighted DAG* is a DAG $D = (N, E)$, where N is a set of nodes, $E \subseteq N \times \mathbb{K} \times N$ is a finite set of weighted edges, and *src* (resp., *snk*) is a unique node in N without incoming (resp., outgoing) edges. We define $\ell(e) = \ell$, where $e = (v, \ell, v') \in E$. Furthermore, we define paths p in the obvious manner as sequences of edges and the length $\ell(p)$ of p as the product ($\otimes$) of the lengths of its edges. More formally, a path

$$p := n_1 \ell_1 n_2 \cdots \ell_{n-1} n_j$$

is a sequence of nodes $n_i \in N$ with $1 \leq i \leq j$ and $(n_i, \ell_i, n_{i+1},) \in E$, for all $1 \leq i < j$, and the length

$$\ell(p) := \ell_1 \otimes \cdots \otimes \ell_{j-1} \ .$$

We denote the set of all paths in D from *src* to *snk* by $\mathrm{Paths}(src, snk)$.

Given a document d, a functional vset-automaton A and a regular weight function $w \in \mathrm{Reg}_{\mathbb{K}}$, we will construct a DAG D which plays the role of a compact representation of the materialized intermediate result. The DAG D is obtained by a product construction between A, W, and d, such that every path from *src* to *snk* corresponds to an accepting run of W that represents a tuple in $[\![A]\!](d)$. If A and W are unambiguous this correspondence is actually a bijection.

Lemma 6.6.1. *Let $\mathbb{K} \in \{\mathbb{Q}, \mathbb{T}\}$ be either the numerical or the tropical semiring. Let d be a document, $A \in$ fVSA, and W be the functional weighted vset-automaton representing $w \in \mathrm{Reg}_{\mathbb{K}}$. We can compute, in polynomial time, a $\mathbb{K}$-weighted DAG D, such that there is a surjective mapping m from paths $p \in Paths(src, snk)$ in D to tuples $\mathrm{t} \in [\![A]\!](d)$. Furthermore,*

(1) the mapping m is a bijection, if A and W are unambiguous, and

(2) $w(d, \mathrm{t}) = \bigoplus_{p \in Paths(src,snk), m(p)=\mathrm{t}} \ell(p)$, for every $\mathrm{t} \in [\![A]\!](d)$, if $A \in$ ufVSA or $\mathbb{K} = \mathbb{T}$.

Proof. Let $d \in \Sigma^*$, $A \in$ fVSA, and W be the functional weighted vset-automaton representing $w \in \mathrm{Reg}_{\mathbb{K}}$. By Proposition 5.5.1, we can assume, w.l.o.g., that all vset-automata used in this proof do not contain ε-transitions.

We begin by giving the construction of D. Let W_A be the functional weighted vset-automaton obtained by interpreting A as a $\mathbb{K}$-weighted vset-automaton. More formally, every transition in A is interpreted as an weighted transition with weight $\overline{1}$ and every transition which is not in A is interpreted as a transition with weight $\overline{0}$. Furthermore, let $W_d := d$ be the functional weighted vset-automaton with $\text{Vars}(W_d) = \emptyset$ that assigns the weight $\overline{1}$ to the empty tuple on input d and $\overline{0}$ to every tuple on input $d' \neq d$. By Lemma 5.5.9 the join of functional weighted vset-automata can be computed in polynomial time. Let

$$W_D := W \bowtie W_A \bowtie W_d \ .$$

Per definition of join for $\mathbb{K}$-relations, it holds that

$$[\![W_D]\!]_{\mathbb{K}}(d,\mathrm{t}) = [\![W]\!]_{\mathbb{K}}(d,\pi_{\text{Vars}(W)}(\mathrm{t})) \otimes [\![W_A]\!]_{\mathbb{K}}(d,\pi_{\text{Vars}(W_A)}(\mathrm{t})) \otimes [\![W_d]\!]_{\mathbb{K}}(d,\pi_{\text{Vars}(W_d)}(\mathrm{t})) \ .$$

Let $A \in \text{ufVSA}$ be unambiguous or $\mathbb{K} = \mathbb{T}$. In both cases, it holds that

$$[\![W_A]\!]_{\mathbb{K}}(d,\mathrm{t}) = \begin{cases} \overline{1} & \text{if } \mathrm{t} \in [\![A]\!](d), \text{ and} \\ \overline{0} & \text{otherwise.} \end{cases}$$

Furthermore,

$$[\![W_d]\!]_{\mathbb{K}}(d',\mathrm{t}) = \begin{cases} \overline{1} & \text{if } \text{Vars}(\mathrm{t}) = \emptyset \text{ and } d' = d, \text{ and} \\ \overline{0} & \text{otherwise.} \end{cases}$$

Therefore, if $A \in \text{ufVSA}$ or $\mathbb{K} = \mathbb{T}$, it holds, for every tuple $\mathrm{t} \in [\![A]\!](d)$. that

$$[\![W_D]\!]_{\mathbb{K}}(d,\mathrm{t}) = [\![W]\!]_{\mathbb{K}}(d,\pi_{\text{Vars}(W)}(\mathrm{t})) \quad (\dagger)$$

We will use this equality in the proof of condition (2).

The DAG $D = (N_D, E_D)$ is obtained from $W_D = (\Sigma, V, Q, I, F, \delta)$ as follows. The set of nodes $N_D := \big(Q \times (\Sigma \cup \Gamma_V \cup \emptyset)\big) \uplus \{src, snk\}$ contains the nodes src, snk, plus a state (q, σ) for each $q \in Q$ and $\sigma \in (\Sigma \cup \Gamma_V \cup \emptyset)$, where $\sigma \neq \emptyset$ encodes the label of the last transition and q the state. The set of edges is defined as follows:

$$\begin{aligned} E_D :=& \{(src, \ell, (x, \emptyset)) \mid I(x) = \ell \neq \infty\} \\ &\uplus \{((x_1, \sigma_1), \ell, (x_2, \sigma_2) \mid \delta(x_1, \sigma_2, x_2) = \ell \neq \overline{0}, \text{ where } \sigma_1 \in (\Sigma \cup \Gamma_V \cup \emptyset)\} \\ &\uplus \{((x, \sigma), \ell, snk) \mid F(x) = \ell \neq \infty, \text{ where } \sigma \in (\Sigma \cup \Gamma_V \cup \emptyset)\} \ . \end{aligned}$$

In the following we assume that D is trimmed, that is, for every node $n \in N_D$ there is at least one path from src to snk, which visits n.[9]

We observe that the construction of D only requires polynomial time. Note that there is a one-to-one correspondence between paths $p \in \text{Paths}(src, snk)$ and accepting runs of W_D on d. That is,

$$p = src \cdot \ell_0 \cdot (q_0, \emptyset) \cdot \ell_1 \cdot (q_1, \sigma_1) \cdots (q_n, \sigma_n) \cdot \ell_{n+1} \cdot snk$$

[9]Note that this condition can be enforced in linear time by two graph traversals (e.g. using breadth first search), one starting from src to identify all states which can be reached from src and one starting from snk to identify all states which can reach snk. We remove all states which are not marked by both graph traversals.

is a path from *src* to *snk* in D if and only if

$$\rho = q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} q_n \ ,$$

with $I(q_0) = \ell_0$ and $F(q_n) = \ell_{n+1}$ is an accepting run of W_D on d. Furthermore, we observe that the weight of p is exactly the weight assigned to the run ρ by W_D, that is, $\ell(p) = \mathsf{w}_(\rho)$.

For the sake of contradiction, assume that D is cyclic. Per assumption, all nodes $n \in N$ are on an path from *src* to *snk*, thus, D must have a path p from *src* to *snk*, which contains a cycle. Let ρ be the run of W_D corresponding to p. The automaton W_d is acyclic. Observe that W_D is functional as W, W_A, and W_d are functional. Thus, $\mathrm{ref}(\rho)$ is valid and therefore the cycle can not contain an edge labeled by a variable operation. Per assumption, all involved vset-automata do not contain ε-transitions. Therefore, the cycle must only consist of edges, labeled by alphabet symbols. Let ρ' be the run, obtained from ρ by removing all cycles. Due to commutativity of $\otimes$, it follows that $\mathsf{w}_{\rho'} = \mathsf{w}_\rho \otimes x$ for some $x \neq \overline{0}$. We observe that $\mathrm{doc}(\mathrm{ref}(\rho')) \neq d$. Therefore, there is a run ρ' of W_D on $\mathrm{doc}(\mathrm{ref}(\rho')) \neq d$ with weight $\mathsf{w}_{\rho'} \neq \overline{0}$, which is the desired contradiction to the observation that for all runs ρ of W_D it holds that $\mathsf{w}_\rho \neq \overline{0}$ if and only if $\mathrm{doc}(\mathrm{ref}(\rho)) = d$.

We now define the mapping m. Let $p \in \mathrm{Paths}(src, snk)$ and let ρ be the corresponding run of W_D. We define the mapping $m(p) := \mathrm{tup}(\rho)$. It follows directly that m is surjective. If $A \in \mathrm{ufVSA}$ or $\mathbb{K} = \mathbb{T}$ and for $\mathrm{t} \in [\![A]\!](d)$, we have that

$$\begin{aligned} w(d, \mathrm{t}) = {} & [\![W]\!]_{\mathbb{K}}(d, \pi_{\mathrm{Vars}(W)}(\mathrm{t})) \\ \overset{(\dagger)}{=} {} & [\![W_D]\!]_{\mathbb{K}}(d, \mathrm{t}) \\ = {} & \bigoplus_{\rho \in P(W_D, d) \text{ and } \mathrm{t} = \mathrm{tup}(\rho)} \mathsf{w}_\rho \\ = {} & \bigoplus_{p \in \mathrm{Paths}(src, snk), m(p) = \mathrm{t}} \ell(p) \ . \end{aligned}$$

The first and the third equalities follow from the definitions of Reg weight functions and $\mathbb{K}$-annotators. The last equality follows from the definition of D. This concludes the proof of condition (2).

It remains to show that condition (1) holds. To this end, assume that $A \in \mathrm{ufVSA}$ and W are unambiguous. Then, by Lemma 5.5.9, W_D is unambiguous.[10] Assume that there are two paths $p_1 \neq p_2$ such that $p_1, p_2 \in \mathrm{Paths}(src, snk)$ with $m(p_1) = m(p_2)$. Let $\rho_1 \neq \rho_2$ be the corresponding runs of W_D. Due to $m(p) = \mathrm{tup}(\rho)$, it must hold that ρ_1 and ρ_2 are two runs of W_D, encoding the same tuple t. Due to unambiguity condition (C3), both runs must encode a different ref-word, that is, $\mathrm{ref}(\rho_1) \neq \mathrm{ref}(\rho_2)$. This implies that either $\mathrm{ref}(\rho_1)$ or $\mathrm{ref}(\rho_2)$ must violate the variable order condition, contradicting unambiguity condition (C2). Thus, m must be a bijection. □

[10] Recall that W_d is unambiguous.

6.6.2 Min and Max Aggregation

We will now study the computational complexity of $\textsc{Min}$ and $\textsc{Max}$ aggregation. We begin by giving the tractable cases which are based on Lemma 6.6.1. The weighted DAG from Lemma 6.6.1 allows us to reduce $\textsc{Min}$ to the shortest path problem in DAGs. If the weight function is unambiguous, $\textsc{Max}$ can be reduced to the longest path problem in DAGs. Notice that, although the longest path problem is intractable in general, it is tractable for DAGs.

Theorem 6.6.2. *$\textsc{Min}[\text{fVSA}, \textsc{Reg}_{\mathbb{T}}]$, $\textsc{Min}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$, $\textsc{Max}[\text{fVSA}, \textsc{UReg}_{\mathbb{T}}]$, and $\textsc{Max}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$ are in* FP.

Proof. Let d be a document, $A \in \text{fVSA}$, and W be the functional weighted vset-automaton representing $w \in \textsc{Reg}_{\mathbb{T}}$ or $w \in \textsc{UReg}_{\mathbb{Q}}$. Let D and m be the DAG and the surjective mapping as guaranteed by Lemma 6.6.1. In the following, we will reduce all four cases to finding the path with minimal (resp., maximal) length in D. Note that given a weighted DAG D, one can compute the path with minimal (resp., maximal) length in polynomial time, via dynamic programming, e.g. using the Bellman-Ford algorithm.[11]

We begin by giving the proofs for the numerical semiring. If $A \in \text{ufVSA}$ and $W \in \textsc{UReg}_{\mathbb{Q}}$, it follows directly from property (1) of Lemma 6.6.1 that m is a bijection. Therefore, for every tuple $\mathrm{t} \in [\![A]\!](d)$, there is exactly one path $p \in \text{Paths}(src, snk)$ with $m(p) = \mathrm{t}$. Thus, $w(d, \mathrm{t}) = \ell(p)$, where $p \in \text{Paths}(src, snk)$ with $m(p) = \mathrm{t}$. It follows directly that $\text{Min}([\![A]\!], d, w)$ and $\text{Max}([\![A]\!], d, w)$ can be computed from D by searching for the path p with minimal (respectively maximal) length.

It remains to give the proofs for the tropical semiring. We begin by giving the proof for $\textsc{Min}[\text{fVSA}, \textsc{Reg}_{\mathbb{T}}]$. Due to property (2) of Lemma 6.6.1,

$$\text{Min}([\![A]\!], d, w) = \min_{\mathrm{t} \in [\![A]\!](d)} \min_{p \in \text{Paths}(src,snk), m(p)=\mathrm{t}} \ell(p) = \min_{p \in \text{Paths}(src,snk)} \ell(p)$$

and therefore $\textsc{Min}[\text{fVSA}, \textsc{Reg}_{\mathbb{T}}]$ again reduces to computing the path of minimal length in D.

For $\textsc{Max}$, the situation is different, because the maximal weight of an output tuple is

$$\text{Max}([\![A]\!], d, w) = \max_{\mathrm{t} \in [\![A]\!](d)} \min_{p \in \text{Paths}(src,snk), m(p)=\mathrm{t}} \ell(p) \; .$$

However, if W is unambiguous, it must hold that $\ell(p) = \ell(p')$ for all runs $p, p' \in \text{Paths}(src, snk)$ with $m(p) = m(p')$. Otherwise W would be required to have at least two runs which accept the same tuple but assign different weights. Thus, W would not be unambiguous. We can therefore conclude that,

$$\text{Max}(S, d, w) = \max_{\mathrm{t} \in [\![A]\!](d)} \min_{\{p \mid m(p)=\mathrm{t}\}} \ell(p) = \max_{p \in \text{Paths}(src,snk)} \ell(p) \; .$$

Again, we can reduce $\textsc{Max}[\text{fVSA}, \textsc{UReg}_{\mathbb{T}}]$ to the max length problem on D. □

[11] One has to be careful in the case of the numeric semiring as the lengths along the path are multiplied. Therefore one has to maintain the minimal as well as the maximal length between two nodes, as edges with negative length change the sign, resulting in minimal paths to be maximal and vice versa.

As we show now, the results of Theorem 6.6.2 are close to the tractability frontier: For instance, if we relax the unambiguity condition in the weight function, the problem Max does not correspond to finding the longest paths in DAGs and becomes intractable.

Theorem 6.6.3. *Min*[ufVSA, $\textit{Reg}_{\mathbb{Q}}$], *Max*[ufVSA, $\textit{Reg}_{\mathbb{T}}$], *and Max*[ufVSA, $\textit{Reg}_{\mathbb{Q}}$] *are* OptP*-hard.*

Proof. We begin by giving the proofs for Max[ufVSA, $\text{Reg}_{\mathbb{T}}$]. We give a metric reduction[12] from the OptP-complete problem Maximum Satisfying Assignment (MSA) [85], which is defined as follows. Let $\phi(x_1, \ldots, x_n)$ be a propositional formula in CNF and let $v = v_1 \cdots v_n \in \mathbb{B}^n$ be an variable assignment of ϕ. Furthermore, let $n_v \in \mathbb{N}$ be the natural number encoded by v in binary. MSA asks, given the CNF formula $\phi(x_1, \ldots, x_n)$, for the maximum $n_v \in \mathbb{N}$ such that v satisfies ϕ, or 0 if ϕ is not satisfiable. In the following, we denote by MSA(ϕ) the output of MSA on input ϕ.

Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula in CNF. We use a similar construction as in Theorems 5.6.4 and 6.4.4 to encode the CNF formula ϕ. Let $d = a^n$ be the document. We define

$$A := ((x_1{\vdash}\, \varepsilon \dashv\! x_1\, a) \vee (x_1{\vdash}\, a \dashv\! x_1)) \cdots ((x_n{\vdash}\, \varepsilon \dashv\! x_n\, a) \vee (x_n{\vdash}\, a \dashv\! x_n)) \; .$$

Notice that A can be defined with a polynomial-time constructible ufVSA. Observe that there is a one-to-one correspondence between tuples t in $[\![A]\!](d)$ and variable assignments α_{t} for ϕ: we can set $\alpha_{\mathrm{t}}(x_i) = 1$ if and only if $\mathrm{t}(x_i) = [i, i+1\rangle$. We construct a weight function $w \in \text{Reg}_{\mathbb{T}}$ such that

$$w(d, \mathrm{t}) = \begin{cases} n_{\alpha_{\mathrm{t}}} & \text{if } \alpha_{\mathrm{t}} \models \phi \\ 0 & \text{otherwise.} \end{cases}$$

Recall that $n_{\alpha_{\mathrm{t}}}$ is the natural number which is encoded by the variable assignment α_{t}. It follows directly that $\text{MSA}(\phi) = \text{Max}([\![A]\!], d, w)$. Defining $T_2(x, y) \mapsto y$ gives the desired reduction.

It remains to construct an weighted vset-automaton W which encodes w. We define the functional weighted vset-automaton W as the union of two automata. Let V be the set of variables of ϕ. The first automaton W_A is a copy of A, assigning weight 0 to all edges, which are present in A. Furthermore, let δ assign weight 2^{i-1} to the a labeled edge between opening and closing variable x_i (that is, $x_i{\vdash}$ and $\dashv\! x_i$). Let $I(q) = 0$ if q is the start state of A and ∞, otherwise. Analogously, let $F(q) = 0$ if q is an accepting state of A and ∞ otherwise. It follows directly that $[\![W_A]\!]_{\mathbb{K}}(a^n, \mathrm{t}) = n_{\alpha_{\mathrm{t}}}$.

The second automaton, W' consists of m disjoint branches, where each branch corresponds to a clause C_i of ϕ; we call these *clause branches*. Each branch has exactly one run ρ with weight $\overline{1}$ for each tuple t associated to an assignment α_{t} which does not satisfy the clause C_i.[13]

[12] Recall that a metric reduction from f to g is a pair of polynomial-time computable functions T_1, T_2, where $T_1 : \Sigma^* \to \Sigma^*$ and $T_2 : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Sigma^*$.

[13] We note that this construction is quite similar to the construction in the proof of Theorem 5.6.4. However, this time, there is only one branch for each clause, encoding all valuations which *do not* satisfy the clause.

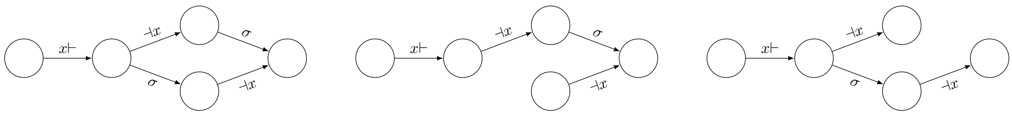

Figure 6.3: Example gadgets for variable x.

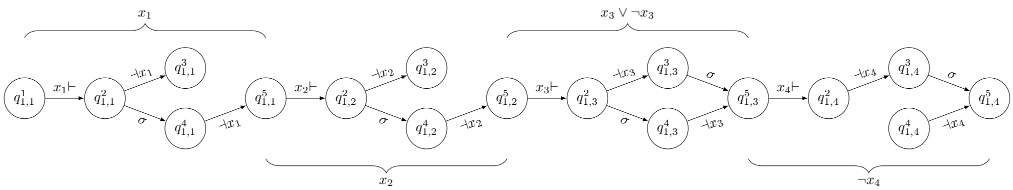

Figure 6.4: The clause branch of W corresponding to C_1 and $x_1 = x_2 = 1, x_4 = 0$.

We now give a formal construction of W'. The set of states $Q := \{q^a_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq a \leq 5\}$ contains $5n$ states for each clause branch. Intuitively, W' has a gadget, consisting of 5 states, for each variable and each clause branch. Figure 6.3 depicts the three types of gadgets we use here. Note that the weights of the drawn edges are all 0. We use the left gadget if x does not occur in the relevant clause and the middle (resp., right) gadget if the literal $\neg x$ (resp., x) occurs. Furthermore, within the same branch of W', the last state of each gadget is the same state as the start state of the next variable, i.e., $q^5_{i,j} = q^1_{i,j+1}$ for all $1 \leq i \leq k, 1 \leq j < n$.

We illustrate the crucial part of the construction on an example. Let $\phi = (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$. The corresponding functional weighted vset-automaton W' therefore has two disjoint branches, one for each clause of ϕ. Figure 6.4 depicts the clause branch C_1 that corresponds to all assignments which do not satisfy C_i, that is, all assignments with $x_1 = x_2 = 1$ and $x_4 = 0$.

Formally, the initial weight function is $I(q^a_{i,j}) = \overline{1}$ if $j = 1 = a$ and $I(q^a_{i,j}) = \overline{0}$ otherwise. The final weight function $F(q^a_{i,j}) = \overline{1}$ if $j = n$ and $a = 5$ and $F(q^a_{i,j}) = \overline{0}$, otherwise. The transition function δ is defined as follows:

$$\delta(q^a_{i,j}, o, q^{a'}_{i,j}) = \begin{cases} \overline{1} & a = 1, a' = 2, o = x_j\vdash \\ \overline{1} & a = 2, a' = 3, o = \dashv x_j \\ \overline{1} & a = 2, a' = 4, \text{ and there is a variable assignment } \tau \text{ with} \\ & \quad \tau(x_j) = 1 \text{ and } \tau \not\models C_i \\ \overline{1} & a = 3, a' = 5, o = a, \text{ and there is a variable assignment } \tau \text{ with} \\ & \quad \tau(x_j) = 0 \text{ and } \tau \not\models C_i \\ \overline{1} & a = 4, a' = 5, o = \dashv x_j \end{cases}$$

All other transitions have weight $\overline{0}$.

We claim that W' represents w', where $w'(d, \mathrm{t}) = \overline{1}$ if $\alpha_{\mathrm{t}} \not\models \phi$ and $w'(d, \mathrm{t}) = \overline{0}$ otherwise. Let $\mathrm{t} \in [\![A]\!](d)$ be a tuple and let $\tau = \alpha_{\mathrm{t}}$ be the variable assignment encoded by t. It is easy to see that there is an accepting run ρ of W' for r with weight $\mathsf{w}_\rho = \overline{1}$, starting in $q^a_{i,0}$, if and only if τ does not satisfy clause C_i.

As mentioned before, the functional weighted vset-automaton W is the union of W' and W_A. Recall that, over the tropical semiring, $\overline{0} = \infty, \overline{1} = 0$, and the weight of a tuple t is the minimal weight over all accepting runs which encode t. Thus, the weight function represented by W is exactly w, as claimed. This concludes the proof that $\textsc{Max}[\text{ufVSA}, \textsc{Reg}_{\mathbb{T}}]$ is OptP-hard.

It remains to show that $\textsc{Min}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$ and $\textsc{Max}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$ are OptP-hard. We first show OptP-hardness for $\textsc{Max}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$.

We give a metric reduction from the OptP-complete problem of weighted satisfiability (WSAT) [85], which is defined as follows. Let $\phi(x_1, \dots, x_n)$ be a propositional formula in CNF with binary weights. WSAT asks, given the CNF formula $\phi(x_1, \dots, x_n)$ with m clauses and weights $w_1, \dots, w_m$, for the maximal weight of an assignment, where the weight of an assignment is the sum of the weights of the satisfied clauses.

Denote by WSAT(ϕ) the output of WSAT on input ϕ. Let $\phi(x_1, \dots, x_n)$ be a Boolean formula in CNF. Let d, A, W be as defined before. However, the weights in W are defined differently. That is, W is the union of W_A and W', where W_A is a copy of A, where all transitions have weight $\overline{1}$. Furthermore, let x be the sum of all clause weights and $F(q) = x$, if q is an accepting state of A. The automaton W' is defined exactly as before, however, accepting with final weight $F(q) = -w_i$ if q is the final weight of the branch of clause C_i and w_i is the weight of C_i. Observe that $w(d, \mathrm{t}) = [\![W]\!]_{\mathbb{Q}}(d, \mathrm{t})$ is exactly the weighted sum of all clauses, which are satisfied by the valuation α_{t} encoded by t. It follows that $\text{Max}(S, d, w) = \text{WSAT}(\phi)$. Defining $T_2(x, y) \mapsto y$ concludes the proof for $\textsc{Max}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$.

The proof for $\textsc{Min}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$ is analogous, replacing the weight x with $-x$ and $-w_i$ with weight w_i. Therefore, $\text{Min}(S, d, w) = -\text{WSAT}(\phi)$. Defining $T_2(x, y) \mapsto -y$ concludes the proof. ☐

6.6.3 Sum and Average Aggregation

Since Sum and Average are already intractable for fVSA spanners and CWidth weight functions (Theorems 6.4.4, 6.4.5, and 6.4.6), they are intractable for fVSA spanners and Reg/UReg weight functions as well. In a similar vein as in Section 6.4, the problems become tractable if we have unambiguity. However, in the case of the tropical semiring, we require unambiguity of *both* the spanner and the representation of the weight function. We begin by showing that $\textsc{Sum}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$ and $\textsc{Sum}[\text{ufVSA}, \textsc{UReg}_{\mathbb{T}}]$ are in FP.

Theorem 6.6.4. *$\textsc{Sum}[\text{ufVSA}, \textsc{Reg}_{\mathbb{Q}}]$ is in* FP.

Proof. Let $d \in \Sigma^*, A \in \text{ufVSA}$, and W be a functional weighted vset-automaton representing $w \in \textsc{Reg}_{\mathbb{Q}}$. Let $D = (N, E)$ and m be as guaranteed by Lemma 6.6.1. It follows that

$$\mathrm{Sum}(\llbracket A \rrbracket, d, w) = \sum_{\mathrm{t} \in \llbracket A \rrbracket(d)} \; \sum_{p \in \mathrm{Paths}(src, snk), m(p) = \mathrm{t}} \ell(p) = \sum_{p \in \mathrm{Paths}(src, snk)} \ell(p) \; .$$

All paths $p \in \mathrm{Paths}(src, snk)$ consist of $|d| + 1 + 2 \cdot |\,\mathrm{Vars}(A)|$ edges. We assume, w.l.o.g., that $N = \{1, \ldots, n\}$, with $src = 1$ and $snk = n$. Therefore, $\mathrm{Sum}(\llbracket A \rrbracket, d, w)$ can be computed by interpreting the edge relation E as a $\mathbb{Q}^{n \times n}$ matrix M and computing the weight

$$I \times M^{|d|+1+2\cdot|\,\mathrm{Vars}(A)|} \times F^T \; ,$$

where $I = (1, 0, \ldots, 0)$ (resp., $F = (0, \ldots, 0, 1)$) is the vector which assigns 0 to all nodes but 1 (resp., n), which is assigned the weight 1. Recall that the numerical semiring has an efficient encoding. Therefore, $\mathrm{Sum}(S, d, w)$ can indeed be computed in polynomial time. $\square$

Theorem 6.6.5. *SUM*[ufVSA, *UREG*$_{\mathbb{T}}$] *is in* FP.

Proof. Let D, m be the DAG and the bijection guaranteed by Lemma 6.6.1. We have that

$$\begin{aligned} \mathrm{Sum}(\llbracket A \rrbracket, d, w) &\overset{(1)}{=} \sum_{\mathrm{t} \in \llbracket A \rrbracket} w(d, \mathrm{t}) \\ &\overset{(2)}{=} \sum_{\mathrm{t} \in \llbracket A \rrbracket} \min_{p \in \mathrm{Paths}(src, snk), m(p) = \mathrm{t}} \ell(p) \\ &\overset{(3)}{=} \sum_{p \in \mathrm{Paths}(src, snk)} \ell(p) \; . \end{aligned}$$

The first equation follows from the definition of SUM. The second equation follows from property (2) of Lemma 6.6.1. The third equation must hold due to m being a bijection between tuples $\mathrm{t} \in \llbracket A \rrbracket$ and paths $p \in \mathrm{Paths}(src, snk)$.

It remains to show that the sum of the lengths of source-to-target paths in a DAG $D = (N, E)$ can be computed in polynomial time. We begin by observing that given two nodes $x, y \in D$ the number of paths from x to y in D can be computed in polynomial time via dynamic programming. Furthermore, given an edge $e = (x, y) \in E$ one can compute the number of paths from src to snk which use e by multiplying the number of paths from src to x with the number of paths from y to snk. Therefore, the function $c : E \to \mathbb{N}$ which, given an edge $e \in E$ assigns the number of paths using e can be computed in polynomial time. Recall that over the tropical semiring, $\otimes = +$ and therefore

$\ell(p) = \sum_{e \in p} \ell(e)$. It therefore follows that

$$\begin{aligned}
\mathrm{Sum}([\![A]\!], d, w) &= \sum_{p \in \mathrm{Paths}(src, snk)} \ell(p) \\
&= \sum_{p \in \mathrm{Paths}(src, snk)} \sum_{e \in p} \ell(e) \\
&= \sum_{e \in E} (\ell(e) \times c(e)) \; .
\end{aligned}$$

Therefore, Sum can be computed by representing the weights $\ell(e)$ as a vector I and the counts $c(e)$ as a vector F. Thus, $\mathrm{Sum}([\![A]\!], d, w) = I \times F^T$, which can be computed in polynomial time, as $\mathrm{Reg}_{\mathbb{T}}$ has an efficient encoding. □

We observe that FP upper bounds for Average follow directly from the corresponding upper bound for Sum and the FP upper bound for Count (Theorem 6.2.1).

Corollary 6.6.6. *Average*[ufVSA, $\mathrm{Reg}_{\mathbb{Q}}$] *and Average*[ufVSA, $\mathrm{UReg}_{\mathbb{T}}$] *are in* FP. □

If we relax the restriction that weight functions are given as unambiguous automata, Sum and Average become #P-hard again.

Theorem 6.6.7. *Sum*[ufVSA, $\mathrm{Reg}_{\mathbb{T}}$] *and Average*[ufVSA, $\mathrm{Reg}_{\mathbb{T}}$] *are* #P*-hard.*

Proof. We begin by giving a parsimonious reduction from the #P-complete problem of #CNF. To this end, let $c = 1$ in the case of Sum and $c = 2^n$ in the case of Average.

Let $\phi(x_1, \ldots, x_n)$ be a propositional formula in conjunctive normal form. Let A, d be as constructed in the proof of Theorem 6.6.3 and let w be the weight function such that $w(d, \mathrm{t}) = c$ if the corresponding assignment α_{t} satisfies ϕ and $w(d, \mathrm{t}) = 0$ otherwise. Therefore, with $c := 1$ it follows directly that $\#\mathrm{CNF}(\phi) = \mathrm{Sum}([\![A]\!], d, w)$, which shows that the problem is #P-hard. For Average let $c := 2^n$. It follows that $\#\mathrm{CNF}(\phi) = x = \frac{x \cdot 2^n}{2^n} = \frac{x \cdot c}{2^n} = \mathrm{Avg}([\![A]\!], d, w)$, implying that Average[ufVSA, $\mathrm{Reg}_{\mathbb{T}}$] is also #P-hard.

It remains to show that there is a weighed automaton W representing $w \in \mathrm{Reg}_{\mathbb{T}}$. As in the proof of Theorem 6.6.3, W is the union of two weighted vset-automata W_A and W', where W_A is a copy of A, assigning weight 0 to all initial states and transitions of A and weight c to all final states. Furthermore, W' is as defined, that is,

$$[\![W']\!]_{\mathbb{T}}(a^n, \mathrm{t}) = \begin{cases} 0 & \text{if } \alpha_{\mathrm{t}} \not\models \phi \\ \infty & \text{otherwise.} \end{cases}$$

It follows directly that W encodes the weight function w, concluding the proof. □

Finally, we show that Sum and Average for $\mathrm{Reg}_{\mathbb{T}}$ weight functions are in $\mathrm{FP}^{\#\mathrm{P}}$.

Theorem 6.6.8. *$\textsc{Sum}$[fVSA, $\textsc{Reg}$] and $\textsc{Average}$[fVSA, $\textsc{Reg}$] are in* $\mathrm{FP}^{\#\mathrm{P}}$.

Proof. We will begin by showing that $\textsc{Sum}[\text{fVSA}, \textsc{Reg}]$ is in $\mathrm{FP}^{\#\mathrm{P}}$ if all weights assigned by w are natural numbers. We will use this as an oracle for the general upper bound. Let A be a vset-automaton, $d \in \Sigma^*$ be a document and $w \in \textsc{Reg}$ be a weight function, which only assigns natural numbers and is represented by a functional weighted vset-automaton W. A counting Turing Machine M for solving the problem in #P would have $w(d, \mathrm{t})$ accepting runs for every tuple in $A(d)$. More precisely, M guesses a d-tuple t over $\mathrm{Vars}(A)$ and can checks whether $\mathrm{t} \in [\![A]\!](d)$ and $w(d, \mathrm{t}) > 0$. If so, M branches into $w(d, \mathrm{t})$ accepting branches. Otherwise, M rejects. Per construction, M has exactly $w(d, \mathrm{t})$ accepting branches for every tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) > 0$. Thus, the number of accepting runs is exactly $\sum_{\mathrm{t} \in [\![A]\!](d)} w(d, \mathrm{t}) = \mathrm{Sum}([\![A]\!], d, w)$.

Now, let $w \in \textsc{Reg}$ be a weight function, represented by the functional weighted vset-automaton W. Following the same lines as the proof of Proposition 5.4.4, we can assume, w.l.o.g., that all rationals in W have the denominator d_{lcm}.[14] We recall that $w(d, \mathrm{t}) = [\![W]\!](d, \pi_{\mathrm{Vars}(W)}(\mathrm{t}))$. Thus, $w(d, \mathrm{t})$ is the product of $|d| + 1 + 2 * |\mathrm{Vars}(A)|$ rationals, where each factor has the denominator d_{lcm}. Therefore, $[\![W]\!](d, \pi_{\mathrm{Vars}(W)}(\mathrm{t}))$ must have the denominator $d_{\mathrm{lcm}}^{|d|+1+2|\mathrm{Vars}(A)|}$[15], which has an encoding length linear in W and d. Thus, $\textsc{Sum}[\text{fVSA}, \textsc{Reg}]$ can be computed by two calls to an $\textsc{Sum}[\text{fVSA}, \textsc{Reg}]$ oracle. The first call only considers positive numerators, whereas the second call only considers negative numerators. Then, $\textsc{Sum}[\text{fVSA}, \textsc{Reg}]$ is the difference of the results of both oracle calls, divided by $d_{\mathrm{lcm}}^{|d|+1+2*|\mathrm{Vars}(A)|}$.

The upper bound for $\textsc{Average}[\text{fVSA}, \textsc{Reg}_{\mathbb{T}}]$ is immediate from the upper bound of $\textsc{Sum}[\text{fVSA}, \textsc{Reg}_{\mathbb{T}}]$ and Theorem 6.2.1. ☐

6.6.4 Quantile Aggregation

The situation for q-$\textsc{Quantile}$ is different from the other aggregation problems, since it remains hard, even when both the spanner and weight function are unambiguous. The reason is that the problem reduces to counting the number of paths in a weighted DAG that are shorter than a given target weight, which is #P-complete due to Mihalák et al. [109].

Theorem 6.6.9. *q-$\textsc{Quantile}$[ufVSA, $\textsc{UReg}$] is #P-hard under Turing reductions, for every $0 < q < 1$.*

At the core of the quantile problem is the problem of counting up to a threshold $k \neq \infty$:

$$\mathrm{Count}_{<k}(S, d, w) := |\{\mathrm{t} \in P(d) \mid w(d, \mathrm{t}) \leq k\}|.$$

The problems $\mathrm{Count}_{>k}(S, d, w)$ and $\mathrm{Count}_{=k}(S, d, w)$ are defined analogously. The decision problem $\textsc{Count}_{<k}[\mathcal{S}, \mathcal{W}]$ is defined analogously to $\textsc{Sum}[\mathcal{S}, \mathcal{W}]$. We begin by

[14]Following the same lines as the proof of Proposition 5.4.4, this can be achieved by computing the least common multiple of all denominators d_{lcm} in W and expanding all fractions $\frac{a}{b}$ by $\frac{b}{d_{\mathrm{lcm}}}$.

[15]Actually for the tropical semiring the denominator is d_{lcm}, as $\otimes = +$ does not increase the denominator if both summands have the same denominator.

showing that $\textsc{Count}_{<k}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$ and $\textsc{Count}_{<k}[\text{ufVSA}, \textsc{UReg}_{\mathbb{T}}]$ #P-hard under Turing reductions. We reduce from #Partition and #-Product-Partition.

Given a set $N = \{n_1, \ldots, n_n\}$ of natural numbers. Two sets N_1, N_2 are a partition of N if $N_1 \cup N_2 = N$ and $N_1 \cap N_2 = \emptyset$. Furthermore, a partition is perfect, if the sums of the natural numbers in both sets are equal. Given such a set $N = \{n_1, \ldots, n_n\}$, the #Partition problem asks for the number of perfect partitions.

Analogously, a partition N_1, N_2 is called perfect product partition, if the products of the natural numbers in both sets are equal. Furthermore, the Product-Partition Problem asks whether there is a perfect product partition and the problem #Product-Partition asks for the number of perfect product partitions.

Proposition 6.6.10. *#Partition and #Product-Partition are* #P*-complete under Turing reductions.*

Proof. Mihalák et al. [109, Theorem 1] shows that #Partition is #P-complete.

The #P-completeness of #Product-Partition follows by a reduction of Ng et al. [113, Theorem 1], who give a reduction from Exact Cover by 3-sets (X3C) to Product-Partition. We note that this reduction is weakly parsimonious, as defined by Hunt et al. [70, Definition 2.5]. That is, for every solution of an X3C instance, there are exactly 2 solutions for the constructed Product-Partition instance. Furthermore, Hunt et al. [70, implicit in Theorem 3.8] show that #X3C is #P-hard. Therefore, the reduction of Ng et al. [113, Theorem 1] can be used to give a Turing reduction from #X3C to #Product-Partition, which implies that #Product-Partition is also #P-hard under Turing reductions. It is easy to see that #Product-Partition is in #P. $\square$

Lemma 6.6.11. *Let $k \in \mathbb{Q}$. Then $\textsc{Count}_{<k}[\text{ufVSA}, \textsc{UReg}_{\mathbb{T}}]$ is* #P*-hard under Turing reductions.*

Proof. We use the same idea as Mihalák et al. [109, Theorem 1] to encode #Partition. Let $N = \{n_1, \ldots, n_n\}$ be an instance of #Partition. Let $d = a^n$. We construct A and W such that every tuple $\mathrm{t} \in [\![A]\!](d)$ corresponds to a partition of N. Furthermore, $w(d, \mathrm{t}) = k$ if and only if the partition encoded by t is perfect.

More formally, $A := (\Sigma, V, Q, q_0, Q_F, \delta)$, where $\Sigma := \{a\}, V := \{x_1, \ldots, x_n\}, Q := \{q_i^j \mid 1 \le i \le n, 1 \le j \le 5\}$, where $q_i^5 = q_{i+1}^1$ for all $1 \le i < n$, $q_0 := q_1^1, Q_F := \{q_n^5\}$, and for $1 \le i \le n$, δ is defined as follows:

$$\delta(q_i^j, \sigma) := \begin{cases} \{q_i^2\} & \text{if } 1 \le i \le n, \sigma = x_i\vdash, \text{ and } j = 1 \\ \{q_i^3\} & \text{if } 1 \le i \le n, \sigma = \dashv x_i, \text{ and } j = 2 \\ \{q_i^4\} & \text{if } 1 \le i \le n, \sigma = a, \text{ and } j = 2 \\ \{q_i^5\} & \text{if } 1 \le i \le n, \sigma = a, \text{ and } j = 3 \\ \{q_i^5\} & \text{if } 1 \le i \le n, \sigma = \dashv x_i, \text{ and } j = 4\ . \end{cases}$$

Recall, that $q_i^5 = q_{i+1}^1$ for all $1 \le i < n$.

Furthermore, we define the functional weighted vset-automaton W encoding w the same way as A. That is, all transitions labeled by an variable operation $x \in \Gamma_V$ are assigned weight $\overline{1}$, $\delta(q_i^3, a, q_i^5) = n_i$ and $\delta(q_i^2, a, q_i^4) = -n_i$, the initial- and final weight functions:

$$I(q) := \begin{cases} \overline{1} & \text{if } q = q_0 \\ \overline{0} & \text{otherwise ;} \end{cases}$$

$$F(q) := \begin{cases} k & \text{if } q \in Q_F \\ \overline{0} & \text{otherwise .} \end{cases}$$

We observe that every tuple $\mathrm{t} \in [\![A]\!](d)$ encodes a partition of N, that is, $n_i \in N_1$ if $\mathrm{t}(x_i) = [i, i\rangle$ and $n_i \in N_2$ if $\mathrm{t}(x_i) = [i, i+1\rangle$. Furthermore, for every tuple $\mathrm{t} \in [\![A]\!](d)$, the weight $w(d, \mathrm{t})$ is exactly k plus the difference of the sum of all elements in N_1 and the sum of all elements in N_2. We make some observations about A, d, and w.

(1) The number of perfect partitions is exactly $\mathrm{Count}_{=k}([\![A]\!], d, w)$;

(2) $\mathrm{Count}_{<k}([\![A]\!], d, w) = \mathrm{Count}_{>k}([\![A]\!], d, w)$;

(3) $\mathrm{Count}([\![A]\!], d) = 2 \cdot \mathrm{Count}_{<k}([\![A]\!], d, w) + \mathrm{Count}_{=k}([\![A]\!], d, w)$;

(4) $\mathrm{Count}([\![A]\!], d) = 2^{n+1}$;

(5) $\mathrm{Count}_{=k}([\![A]\!], d, w) = 2^{n+1} - 2 \cdot \mathrm{Count}_{<k}([\![A]\!], d, w)$.

Due to Observations (1) and (5) it follows that the number of perfect partitions can be computed by a single call to an $\mathrm{Count}_{<k}([\![A]\!], d, w)$ oracle.

It remains to argue that the observations $(1) - (5)$ hold. Observation (1) follows directly from the previous observation that the weight of each tuple is k plus the difference of the sum of all elements in N_1 and the sum of all elements in N_2. Observation (2) follows from the fact that the partition problem is symmetric, that is, for every partition N_1, N_2 of N there is also a partition N_2, N_1. Observation (3) follows from (2), and (4) from the fact that there are 2^n subsets of N and therefore $2 \cdot 2^n$ possible partitions. The last observation (5) follows from (3) and (4). This concludes the proof. □

Along the same lines we show that $\textsc{Count}_{<1}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$ is #P-hard under Turing reductions. Note that we do not show hardness for $\textsc{Count}_{<k}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$, but only for the case $k = 1$.[16]

Lemma 6.6.12. *$\textsc{Count}_{<1}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$ is #P-hard under Turing reductions.*

Proof. Let N be an instance of #Product-Partition. We construct A, d, w and W, as constructed in the proof of Lemma 6.6.11. However, in W, $\delta(q_i^3, a, q_i^5) = n_i$ and $\delta(q_i^2, a, q_i^4) = \frac{1}{n_i}$. Observe that $w(d, \mathrm{t})$ is exactly the product of all elements in N_1 divided by the product of all elements in N_2, where $n_i \in N_1$ if and only if $\mathrm{t}(x_i) = [i, i\rangle$ and

[16] Recall that, in the proof for the tropical semiring, we add k to all accepting runs by having $F(q) = k$, if $q \in Q_F$. This is not possible over the numerical semiring, as the multiplicative operation is the numerical multiplication $\cdot$ and not the numerical addition $+$.

$n_i \in N_2$ if and only if $\mathrm{t}(x_i) = [i, i+1\rangle$. Therefore, the number of perfect product partitions is exactly the number of tuples $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = 1$. Using the same argument as in the proof of Lemma 6.6.11, it follows that

$$\#\text{Product-Partition} = 2^{n+1} - 2 \cdot \mathrm{Count}_{<1}([\![A]\!], d, w) \ ,$$

and thus, #Product-Partition can be computed by a single $\textsc{Count}_{<1}[\text{ufVSA}, \textsc{UReg}_{\mathbb{Q}}]$ oracle call. ☐

The following corollary follows directly from the Lemmas 6.6.11 and 6.6.12.

Corollary 6.6.13. *$\textsc{Count}_{<1}[\text{ufVSA}, \textsc{UReg}]$ is #P-hard under Turing reductions.* ☐

We are finally ready to give the proof of Theorem 6.6.9.

Proof of Theorem 6.6.9. We show that $\mathrm{Count}_{<1}([\![A]\!], d, w)$ can be computed in polynomial time, using a q-$\textsc{Quantile}[\text{ufVSA}, \textsc{UReg}]$ oracle therefore, concluding that the problem q-$\textsc{Quantile}[\text{ufVSA}, \textsc{UReg}]$ is also #P-hard under Turing reductions.

Let $A \in \text{ufVSA}$, $d \in \Sigma^*$, and $w \in \textsc{UReg}$ represented by an unambiguous functional weighted vset-automaton W. Furthermore, let $0 < q < 1$, such that $q = \frac{a}{b}$. Due to Theorem 6.2.1, $c := \mathrm{Count}([\![A]\!], d)$ can be computed in polynomial time. Let $0 \leq r \leq c \cdot (b-1)$. By Lemma 6.3.5[17], there are vset-automata $A_r, A'_r \in \text{ufVSA}$ and documents d_r, d'_r, such that $\mathrm{Count}([\![A_r]\!], d_r) = r$ and $\mathrm{Count}([\![A'_r]\!], d'_r) = c \cdot (b-1) - r$. Let W_r (resp., W'_r) be A_r (resp., A'_r), interpreted as unambiguous functional weighted vset-automaton, where all transitions of A_r (resp., A'_R) have weight $\overline{1}$, the initial weight function assigns weight $\overline{1}$ to the initial state of A_r (resp., A'_r), and the final weight function assigns weight 0 (resp., 1)[18] to all accepting states of A_r (resp., A'_r). Slightly overloading notation, we define

$$A' := (A \cdot d_r \cdot d'_r) \vee (d \cdot A_r \cdot d'_r) \vee (d \cdot d_r \cdot A'_r)$$

and

$$W' := (W \cdot d_r \cdot d'_r) \vee (d \cdot W_r \cdot d'_r) \vee (d \cdot d_r \cdot W'_r)$$

It is straightforward to verify that both, A' and W' are unambiguous. Let $d' = d \cdot d_r \cdot d'_r$ and let w' (resp, w_r, w'_r) be the weight function, represented by W' (resp., W_r, W'_r). It follows from the definition that

$$\begin{aligned} \mathrm{Count}([\![A']\!], d') &= \mathrm{Count}([\![A]\!], d) + \mathrm{Count}([\![A_r]\!], d_r) + \mathrm{Count}([\![A'_r]\!], d'_r) \\ &= c + r + (c \cdot (b-1) - r) = c \cdot b \ . \end{aligned}$$

Furthermore, recalling that $w(d, \mathrm{t}) = 0$ for all tuples $\mathrm{t} \in [\![A_r]\!](d_r)$ and $w(d, \mathrm{t}) = 1$ for all tuples $\mathrm{t} \in [\![A'_r]\!](d'_r)$, we have that

$$\begin{aligned} &\mathrm{Count}_{<1}([\![A']\!], d', w') \\ &= \mathrm{Count}_{<1}([\![A]\!], d, w) + \mathrm{Count}_{<1}([\![A_r]\!], d_r, w_r) + \mathrm{Count}_{<1}([\![A'_r]\!], d'_r, w'_r) \\ &= \mathrm{Count}_{<1}([\![A]\!], d, w) + r + 0 \ . \end{aligned}$$

[17]For instance with $v = \mathrm{Vars}(A) \cdot b$.

[18]Note that we use 0 and 1 instead of $\overline{0}$ and $\overline{1}$ on purpose. The reason is that we want to assign the same weights for both semirings.

Using binary search, we compute r_{min} as the smallest r with q-Quantile$([\![A']\!], d', w') < 1$. Thus,

$$\frac{\text{Count}_{<1}([\![A']\!], d', w')}{\text{Count}([\![A']\!], d')} = \frac{\text{Count}_{<1}([\![A]\!], d, w) + r_{min}}{c \cdot b} \geq q \ .$$

For the sake of contradiction, assume that $\frac{\text{Count}_{<1}([\![A]\!], d, w) + r_{min}}{c \cdot b} > q = \frac{c \cdot a}{c \cdot b}$. It follows that, $\text{Count}_{<1}([\![A]\!], d, w) + r_{min} > c \cdot a$ and therefore, as all involved numbers are natural numbers, $\text{Count}_{<1}([\![A]\!], d, w) + r_{min} - 1 \geq c \cdot a$. Thus, $\frac{\text{Count}_{<1}([\![A]\!], d, w) + (r_{min} - 1)}{c \cdot b} \geq q$, leading to the desired contradiction, as r_{min} was assumed to be minimal.

We have that $\frac{\text{Count}_{<1}([\![A]\!], d, w) + r_{min}}{c \cdot b} = q = \frac{c \cdot a}{c \cdot b}$. It follows that

$$\text{Count}_{<1}([\![A]\!], d, w) = c \cdot a - r_{min} \ ,$$

which concludes the proof. □

6.7 Aggregate Approximation

Now that we have a detailed understanding on the complexity of computing exact aggregates, we want to see in which cases the result can be approximated. We only consider the situation where the exact problems are intractable and want to understand when the considered aggregation problems can be approximated by fully polynomial-time randomized approximation schemes (FPRAS), and when the existence of such an FPRAS would contradict commonly believed conjectures, like RP $\neq$ NP and the conjecture that the polynomial hierarchy does not collapse.

Based on the results for the computation of exact aggregates, we can already give some insights into the possibility of approximation. That is, Zuckerman [171] shows that #SAT can not be approximated by an FPRAS unless NP = RP. Furthermore, as shown by Dyer et al. [41], this characterization extends to all problems which are #P-complete under parsimonious reductions. Therefore, due to Theorems 6.4.4, and 6.6.7, we have the following corollary.

Corollary 6.7.1. *Unless* NP = RP, *Sum*[fVSA, *CWidth*], *Sum*[ufVSA, *Reg*$_{\mathbb{T}}$], *and* *Average*[ufVSA, *Reg*$_{\mathbb{T}}$] *can not be approximated by an* FPRAS. □

Arenas et at. [12, Corollary 3.3] showed that every function in spanL admits an FPRAS. Therefore, due to Theorem 6.4.5, we have the following corollary.

Corollary 6.7.2. *Sum*[fVSA, *CWidth*$_{\mathbb{N}}$] *can be approximated by an* FPRAS. □

In the remainder of this section, we will revisit the other intractable cases of spanner aggregation and study whether or not approximation is possible.

6.7.1 Approximation is Hard at First Sight

We begin with some inapproximability results. For instance, as we show now, the existence of an FPRAS for the problems Min, Max with Reg$_{\mathbb{Q}}$ weight functions would imply a collapse of the polynomial hierarchy, even when spanners are unambiguous. Furthermore, for MAX and $\mathrm{REG}_{\mathbb{T}}$ weight functions the same result holds.

Theorem 6.7.3. *MIN*[ufVSA, *$\mathrm{REG}_{\mathbb{Q}}$*] *and MAX*[ufVSA, *$\mathrm{REG}_{\mathbb{Q}}$*] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. Assume there is an FPRAS for MIN[ufVSA, $\mathrm{REG}_{\mathbb{Q}}$]. We will show that such an FPRAS implies that the NP-complete problem SAT is in BPP, which implies that the polynomial hierarchy collapses to the second level.[19]

Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula, given in CNF, and let A, d, and W' be as defined in the proof for MAX[ufVSA, $\mathrm{REG}_{\mathbb{T}}$] of Theorem 6.6.3, where W' is interpreted as an weighted vset-automaton over the numerical semiring. Observe that, due to $\overline{1} = 1$ and $\overline{0} = 0$, it follows that $[\![W']\!]_{\mathbb{Q}}(d, \mathrm{t}) \geq 1$ if the valuation α_{t} encoded by t does not satisfy at least one clause of ϕ and 0 otherwise. Let w be the weight function encoded by W'.

For the sake of contradiction, assume that there is an FPRAS for MIN[ufVSA, $\mathrm{REG}_{\mathbb{Q}}$] and let $\delta = 0.4$. Assume that ϕ is satisfiable, thus $\mathrm{Min}([\![A]\!], d, w) = 0$. Then the FPRAS must return 0 with probability at least $\frac{3}{4}$. On the other hand, if ϕ is not satisfiable, the FPRAS must return a value $x \geq (1 - \delta) \cdot 1 = 0.6$ with probability at least $\frac{3}{4}$. Consider the algorithm which calls the FPRAS and accepts if the approximation is 0, and rejects otherwise. This algorithm is a BPP algorithm for SAT, resulting in the desired contradiction.

The proof for MAX[ufVSA, $\mathrm{REG}_{\mathbb{Q}}$] is analogous. The only difference is that the final weight function of W' is multiplied by -1, that is, W' assigns weight $-x$ to each tuple, encoding a valuation α which does not satisfy x clauses of ϕ. □

Theorem 6.7.4. *MAX*[ufVSA, *$\mathrm{REG}_{\mathbb{T}}$*] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula, given in CNF. We assume, w.l.o.g., that the valuation which assigns false to all variables does not satisfy ϕ. Let A, d, and w be as defined in the proof for MAX[ufVSA, $\mathrm{REG}_{\mathbb{T}}$] in the proof of Theorem 6.6.3. Thus, $\mathrm{Max}([\![A]\!], d, w) \geq 1$ if ϕ is satisfiable and $\mathrm{Max}([\![A]\!], d, w) = 0$ if ϕ is not satisfiable.

For the sake of contradiction, assume that there is an FPRAS for MAX[ufVSA, $\mathrm{REG}_{\mathbb{T}}$] and let $\delta = 0.4$. Assume that ϕ is satisfiable, thus $\mathrm{Max}([\![A]\!], d, w) \geq 1$. Then the FPRAS must return a value $x \geq (1 - \delta) \cdot 1 = 0.6$ with probability at least $\frac{3}{4}$. On the other hand, if ϕ is not satisfiable, the FPRAS must return 0 with probability at least $\frac{3}{4}$. Therefore, we can obtain a BPP algorithm for SAT as follows. The algorithm first calls the FPRAS, accepts if the approximation is bigger than 0, and rejects otherwise. □

Concerning SUM and AVERAGE the only case which is not resolved by Corollary 6.7.1 is the case of AVERAGE[fVSA, CWIDTH]. We show now that, under reasonable complexity assumptions, this problem can also not be approximated by an FPRAS.

[19] NP $\subseteq$ BPP implies that PH $\subseteq$ BPP (cf. Zachos [169]) and as BPP $\subseteq (\Pi_2^{\mathrm{P}} \cap \Sigma_2^{\mathrm{P}})$ (cf. Lautemann [88]) the polynomial hierarchy collapses on the second level. Furthermore, as BPP is closed under complement, coNP $\subseteq$ BPP implies that NP $\subseteq$ BPP resulting in the same collapse of the polynomial hierarchy.

Theorem 6.7.5. *$\textsc{Average}$[fVSA, $\textsc{CWidth}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. We will show that such an FPRAS implies that the NP-complete problem SAT is in BPP, which implies that the polynomial hierarchy collapses to the second level.

Let A, d and w be as constructed in the proof of Theorem 6.4.4. Recall that given a propositional formula ϕ in CNF, we have that $\mathrm{Sum}(\llbracket A\rrbracket, d, w) = c$, where c is the number of satisfying assignments of ϕ.

Assume there is an FPRAS for $\textsc{Average}$[fVSA, $\textsc{CWidth}$] and let $\delta = 0.5$. Assume that ϕ is not satisfiable. Then the FPRAS on input A, d, w must return 0 with probability at least $\frac{3}{4}$. On the other hand, if ϕ is satisfiable, thus $c > 0$, the FPRAS must return a value $x \geq (1-\delta)\cdot \mathrm{Avg}(\llbracket A\rrbracket, d, w) = \frac{1}{2}\cdot\frac{c}{\mathrm{Count}(\llbracket A\rrbracket, d)} > 0$, with probability at least $\frac{3}{4}$. Therefore, the algorithm which first approximates $\mathrm{Avg}(\llbracket A\rrbracket, d, w)$ with $\delta = 0.5$, rejects if the approximation is 0 and accepts otherwise is an BPP algorithm for SAT, implying that NP $\subseteq$ BPP, which implies that the polynomial hierarchy collapses to the second level. □

We now turn to the quantile problem. It turns out that this problem is difficult to approximate even if the weight functions only return 0 or 1.

Theorem 6.7.6. *Let $0 < q < 1$. Then, q-$\textsc{Quantile}$[fVSA, $\textsc{CWidth}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. We will show that an FPRAS for q-$\textsc{Quantile}$[fVSA, $\textsc{CWidth}$] implies a BPP algorithm for SAT. Let ϕ be a propositional formula ϕ in CNF. Assume that $q = \frac{1}{2}$ and let A and d be as constructed in the proof of Theorem 6.4.4. However, let w be the weight function which is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with

$$R(d) := \begin{cases} 1 & \text{if } d = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Recall from the construction of A and d that A is the union of two automata A_1, A_{-1}, such that $\mathrm{Count}(\llbracket A_1\rrbracket, d) = 2^n$ and $\mathrm{Count}(\llbracket A_{-1}\rrbracket, d) = s$, where s is the number of non-satisfying assignments for ϕ, furthermore, $\mathrm{t} \in \llbracket A_1\rrbracket(d)$ if and only if $d_{\mathrm{t}(x)} = 1$ and $\mathrm{t} \in \llbracket A_{-1}\rrbracket(d)$ if and only if $d_{\mathrm{t}(x)} = -1$. We observe that $R(-1) = 0$ and therefore, for every $\mathrm{t} \in \llbracket A\rrbracket(d)$ we have that

$$w(d, \mathrm{t}) = \begin{cases} 1 & \text{if } \mathrm{t} \in \llbracket A_1\rrbracket(d) \\ 0 & \text{if } \mathrm{t} \in \llbracket A_{-1}\rrbracket(d)\,. \end{cases}$$

Thus, $\frac{1}{2}$-$\mathrm{Quantile}(\llbracket A\rrbracket, d, w) = 0$ if and only if ϕ is not satisfiable.

Assuming there is an FPRAS for q-$\textsc{Quantile}$[fVSA, $\textsc{CWidth}$], one can decide SAT with a probability of $\frac{3}{4}$ by approximating q-$\mathrm{Quantile}(\llbracket A\rrbracket, d, w)$ with $\delta = 0.5$, rejecting if the approximation is 0 and accepting otherwise. This, however, implies that NP $\subseteq$ BPP, which implies a collapse of the polynomial hierarchy on the second level.

The general case for $0 < q < 1$ follows by slightly adopting the previous construction. That is, assume that $q = \frac{a}{b}$. Due to $0 \leq q \leq 1$, it must hold that $1 \leq a < b$. We construct a vset-automaton A' and a document d' as follows. Let $\sigma \notin \Sigma$ be a new alphabet symbol. The document d' consists of b copies of d, separated by σ and A' consists of a copies of A_{-1} and $b-a$ copies of A_1. More formally,

$$d' := (d \cdot \sigma)^b \, .$$

Furthermore, slightly abusing notation, we define

$$A' := (A_{-1} \cdot \sigma)^a \cdot (A_1 \cdot \sigma)^{b-a} \, .$$

We observe that on input document d', the automaton A' accepts exactly $2^n \cdot (b-a)$ tuples t with $w(d', \mathrm{t}) = 1$ and $s \cdot a$ tuples with weight 0. Therefore, $\frac{a}{b}$-Quantile$(S, d, w) = 0$ if and only if

$$\frac{s \cdot a}{2^n \cdot (b-a) + s \cdot a} \geq \frac{a}{b} \, .$$

Solving this equation for s, it holds that $\frac{a}{b}$-Quantile$(S, d, w) = 0$ if and only if $s = 2^n$ and therefore $\frac{a}{b}$-Quantile$(S, d, w) = 0$ if and only if ϕ is not satisfiable.

The rest of the proof is analogous to the case that $q = \frac{1}{2}$. □

When the spanners are unambiguous, the simplest intractable case for q-Quantile is the one with UReg weight functions (see Table 6.1). Again, we can show that approximation is hard.

Theorem 6.7.7. *Let $0 < q < 1$. Then, q-Quantile*[ufVSA, *UReg$_{\mathbb{T}}$*] *cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses on the second level.*

Proof. We show that an FPRAS for q-Quantile[ufVSA, UReg$_{\mathbb{T}}$] implies a BPP algorithm for the NP-complete Partition problem. Let $S = \{s_1, \ldots, s_n\}$ be a set of natural numbers. Furthermore, let A, d, w be constructed from S as in the proof of Lemma 6.6.11 with $k = 0$.

Per construction of A, d and w, every tuple $\mathrm{t} \in [\![A]\!](d)$ corresponds to a partition of S, such that the partition is perfect if and only if $w(d, \mathrm{t}) = 0$. Furthermore, due to the partition problem being symmetrical, for every tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = k$ there is a tuple $\mathrm{t}' \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = -k$. Thus, $\frac{1}{2}$-Quantile$([\![A]\!], d, w) = 1$ if and only if there is a tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = 0$.

Let $q = \frac{1}{2}$. Assuming there is an FPRAS for q-Quantile[ufVSA, UReg$_{\mathbb{T}}$], one can decide Partition with a probability of $\frac{3}{4}$ by approximating q-Quantile$([\![A]\!], d, w)$ with $\delta = 0.5$, accepting if the approximation is 0 rejecting otherwise. This implies that the algorithm accepts if and only if there is a perfect partition and therefore, NP $\subseteq$ BPP, which implies a collapse of the polynomial hierarchy on the second level.

For the general case, assume that $q = \frac{a}{b}$. We observe that due to $0 < q < 1$, it must hold that $a < b$. By Observation (4) in the proof of Lemma 6.6.11, Count$([\![A]\!], d) = 2^{n+1}$. As in the proof of Theorem 6.6.9, we construct a vset-automaton A', a document d' and a weight function w', represented by the weighted automaton $W' \in$ UReg$_{\mathbb{T}}$, such that q-Quantile$(A', d', w') = 0$ if and only if S has a perfect partition. By Lemma 6.3.5, there are vset-automata $A_{-1}, A_1 \in$ ufVSA and documents $d_{-1}, d_1 \in \Sigma^*$ such that $\text{Count}(\llbracket A_{-1} \rrbracket, d_{-1}) = (a-1) \cdot 2^n$ and $\text{Count}(\llbracket A_1 \rrbracket, d_1) = (b-a-1) \cdot 2^n$. Let W_{-1} (resp., W_1) be the same as A_{-1} (resp., A_1) interpreted as weighted automaton over the tropical semiring, such that all transitions are assigned weight 0 and the final weight function assigns weight -1 (resp., 1) to all accepting states. Let w_{-1} (resp., w_1) be the weight function, represented by W_{-1} (resp., W_1) Thus, $w_{-1}(d_{-1}, \mathrm{t}) = -1$ if and only if $\mathrm{t} \in \llbracket A_{-1} \rrbracket(d_{-1})$ and $w_1(d_1, \mathrm{t}) = 1$ if and only if $\mathrm{t} \in \llbracket A_1 \rrbracket(d_1)$. Let σ be a new alphabet symbol. We construct A', d', and W' as follows.

$$\begin{aligned}
d' &= d_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot d_1 \\
A' &= (A_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot A \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot A_1) \\
W' &= (W_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot W \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot W_1) \;.
\end{aligned}$$

Furthermore, let w' be the weight function, represented by W'. It follows that

$$\begin{aligned}
\text{Count}_{<0}(\llbracket A' \rrbracket, d', w') &= (a-1) \cdot 2^n + \text{Count}_{<0}(\llbracket A \rrbracket, d, w) \\
\text{Count}_{\leq 0}(\llbracket A' \rrbracket, d', w') &= (a-1) \cdot 2^n + \text{Count}_{\leq 0}(\llbracket A \rrbracket, d, w) \\
\text{Count}(\llbracket A' \rrbracket, d') &= (a-1) \cdot 2^n + 2 \cdot 2^n + (b-a-1) \cdot 2^n = b \cdot 2^n \;.
\end{aligned}$$

We make a case distinction on S. If S has a perfect partition, $\text{Count}_{<0}(\llbracket A \rrbracket, d, w) < 2^n$ and $\text{Count}_{\leq 0}(\llbracket A \rrbracket, d, w) \geq 2^n$. Thus, q-Quantile$(A', d', w') = 0$. Otherwise, if S has no perfect partition, $\text{Count}_{<0}(\llbracket A \rrbracket, d, w) = 2^n$ and therefore q-Quantile$(A', d', w') < 0$. Therefore, q-Quantile$(A', d', w') = 0$ if and only if S has a perfect partition. This concludes the proof. □

We note that the case of approximating q-Quantile[ufVSA, $\text{UReg}_{\mathbb{Q}}$] does not follow analogous to the proof for q-Quantile[ufVSA, $\text{UReg}_{\mathbb{T}}$]. The main reason is the fact that #Partition can be encoded into an weight function automaton $w_{\mathbb{T}} \in \text{UReg}_{\mathbb{T}}$, such that perfect partitions correspond to tuples with weight 0, whereas #Product-Partition is encoded into an weight function $w_{\mathbb{Q}} \in \text{UReg}_{\mathbb{Q}}$, such that perfect product partitions correspond to tuples with weight 1. Furthermore, all weights assigned by $w_{\mathbb{T}}$ are integers, whereas $w_{\mathbb{Q}}$ assigns rational numbers. Therefore it is not obvious whether or not q-Quantile[ufVSA, $\text{UReg}_{\mathbb{Q}}$] can be approximated by an FPRAS. This case is left open for future research.

6.7.2 When an FPRAS is Possible

We show that Theorem 6.7.5 is very much on the intractability frontier: it shows that approximation is intractable if weight functions can assign 1 and -1. On the other hand, if the weight functions are restricted to *nonnegative* numbers, then approximating Sum and Average is possible with an FPRAS.

Theorem 6.7.8. *Sum*[fVSA, $\text{CWidth}_{\mathbb{Q}_+}$] *and Average*[fVSA, $\text{CWidth}_{\mathbb{Q}_+}$] *can be approximated by an* FPRAS.

Proof. By Corollary 6.7.2 and Theorem 6.2.1, there are FPRAS for the problems $\text{Sum}[\text{fVSA}, \text{CWidth}_{\mathbb{N}}]$ and Count[fVSA]. We will use these FPRAS to give an FPRAS for $\text{Sum}[\text{fVSA}, \text{CWidth}_{\mathbb{Q}^+}]$ and $\text{Average}[\text{fVSA}, \text{CWidth}_{\mathbb{Q}^+}]$.

In the following, we will denote an FPRAS approximation with error rate δ of the problem $\text{Count}(\llbracket A \rrbracket, d)$ (resp., $\text{Sum}(\llbracket A \rrbracket, d, w)$ and $\text{Avg}(\llbracket A \rrbracket, d, w)$) by $\text{Count}(\llbracket A \rrbracket, d, \delta)$ (resp., $\text{Sum}(\llbracket A \rrbracket, d, w, \delta)$ and $\text{Avg}(\llbracket A \rrbracket, d, w, \delta)$).

We begin by showing that $\text{Sum}[\text{fVSA}, \text{CWidth}_{\mathbb{Q}^+}]$ admits an FPRAS. Let $A \in \text{fVSA}$ be a vset-automaton, $d \in \Sigma^*$ be a document, and $w \in \text{CWidth}_{\mathbb{Q}^+}$ be a weight function. Recall that every weight $x \in \mathbb{Q}^+$ is encoded by its numerator and its denominator. Let D be the set of denominators used by w and let lcm be the least common multiple of all elements in D. We note that, as argued in the proof of Proposition 5.4.4, lcm can be computed in polynomial time. Let $w_{\mathbb{N}}(d, \mathrm{t}) = w(d, \mathrm{t}) \cdot \text{lcm}$. Per definition of lcm, $w_{\mathbb{N}} \in \text{CWidth}_{\mathbb{N}}$ only assigns natural numbers. Furthermore, $w(d, \mathrm{t}) = \frac{w_{\mathbb{N}}(d,\mathrm{t})}{\text{lcm}}$. It follows that $\text{Sum}(\llbracket A \rrbracket, d, w, \delta) := \frac{\text{Sum}(\llbracket A \rrbracket, d, w_{\mathbb{N}}, \delta)}{\text{lcm}}$ is an δ-approximation of $\text{Sum}(S, d, w)$ with success probability $\frac{3}{4}$, concluding this part of the proof.

It remains to show that $\text{Average}[\text{fVSA}, \text{CWidth}_{\mathbb{Q}_+}]$ admits an FPRAS. We show that the algorithm which, with success rate $(\frac{3}{4})^{0.5}$, calculates an $\frac{\delta}{3}$-approximations for Count and Sum, and then returns the quotient of the results, is an FPRAS for $\text{Average}[\text{fVSA}, \text{CWidth}_{\mathbb{Q}_+}]$. We note that the probability that both approximations are successful is $(\frac{3}{4})^{0.5} \cdot (\frac{3}{4})^{0.5} = \frac{3}{4}$.

It remains to show that the quotient of both results, $\text{Avg}(\llbracket A \rrbracket, d, w, \delta) := \frac{\text{Sum}(\llbracket A \rrbracket, d, w, \frac{\delta}{3})}{\text{Count}(\llbracket A \rrbracket, d, \frac{\delta}{3})}$, is indeed a δ-approximation of $\text{Avg}(\llbracket A \rrbracket, d, w)$. Formally, we have to show that

$$(1-\delta) \cdot \text{Avg}(S, d, w) \leq \text{Avg}(\llbracket A \rrbracket, d, w, \delta) \leq (1+\delta) \cdot \text{Avg}(\llbracket A \rrbracket, d, w) \; .$$

We begin with the first inequality:

$$\begin{aligned}
\text{Avg}(\llbracket A \rrbracket, d, w, \delta) &= \frac{\text{Sum}(\llbracket A \rrbracket, d, w, \frac{\delta}{3})}{\text{Count}(\llbracket A \rrbracket, d, \frac{\delta}{3})} \\
&\geq \frac{(1-\frac{\delta}{3}) \cdot \text{Sum}(\llbracket A \rrbracket, d, w)}{(1+\frac{\delta}{3}) \cdot \text{Count}(\llbracket A \rrbracket, d)} \\
&= \frac{1-\frac{\delta}{3}}{1+\frac{\delta}{3}} \cdot \frac{\text{Sum}(\llbracket A \rrbracket, d, w)}{\text{Count}(\llbracket A \rrbracket, d)} \\
&\geq (1-\delta) \cdot \text{Avg}(\llbracket A \rrbracket, d, w) \; .
\end{aligned}$$

Procedure PositionalQuantileApprox(A, d, w, q, δ)

Input: $A \in \text{fVSA}, d \in \Sigma^*, w \in \textsc{Poly}, 0 \leq q \leq 1, 0 \leq \delta \leq 1$
Output: A positional δ-approximation of q-Quantile$(\llbracket A \rrbracket, d, w)$ with success rate $\frac{3}{4}$.

$W \leftarrow \{\!\{\cdot\}\!\}$
for $1 \leq i \leq 4 \cdot \lceil \frac{\ln(16)}{2\delta^2} \rceil$ **do**
  t $\leftarrow$ Sample$(A, d, \frac{\delta}{3})$
  Add $w(d, \mathrm{t})$ to W
if $|W| < \lceil \frac{\ln(16)}{2\delta^2} \rceil$ **then**
  Fail ▷ Sample size to small
Return q-Quantile(W)

It is straightforward to verify that $\frac{1-\frac{\delta}{3}}{1+\frac{\delta}{3}} \geq (1-\delta)$ holds for every $0 \leq \delta \leq 1$. The second inequality follows analogously:

$$\begin{aligned}
\text{Avg}(\llbracket A \rrbracket, d, w, \delta) &= \frac{\text{Sum}(\llbracket A \rrbracket, d, w, \frac{\delta}{3})}{\text{Count}(\llbracket A \rrbracket, d, \frac{\delta}{3})} \\
&\leq \frac{(1+\frac{\delta}{3}) \cdot \text{Sum}(\llbracket A \rrbracket, d, w)}{(1-\frac{\delta}{3}) \cdot \text{Count}(\llbracket A \rrbracket, d)} \\
&= \frac{1+\frac{\delta}{3}}{1-\frac{\delta}{3}} \cdot \frac{\text{Sum}(\llbracket A \rrbracket, d, w)}{\text{Count}(\llbracket A \rrbracket, d)} \\
&\leq (1+\delta) \cdot \text{Avg}(\llbracket A \rrbracket, d, w)\ .
\end{aligned}$$

Again, it is straightforward to verify that $\frac{1+\frac{\delta}{3}}{1-\frac{\delta}{3}} \leq (1+\delta)$ holds for every $0 \leq \delta \leq 1$. □

Our second positive result is about approximating quantiles *in a positional manner*. Let d be a document, S be a document spanner, w be a weight function and $0 \leq q \leq 1$ with $q \in \mathbb{Q}$. Then, for $\delta > 0$, we say that $k \in \mathbb{Q}$ is a positional δ-approximation of q-Quantile(S, d, w) if there is a $q' \in \mathbb{Q}$, with $q - \delta \leq q' \leq q + \delta$ and $k = q'$-Quantile(S, d, w).[20]

Lemma 6.7.9 (Hoeffding's Inequality)**.** *Let $X_1, \ldots, X_n$ be independent random variables with $0 \leq X_i \leq 1$ for $1 \leq i \leq n$. Let $X = \Sigma_{i=1}^n X_i$ and let* EX *denote the expectation of X. Then, for any $\lambda > 0$, $Pr(X - \text{EX} \geq \lambda) \leq e^{\frac{-2\lambda^2}{n}}$.*

Theorem 6.7.10. *Let $0 \leq q \leq 1$. There is a probabilistic algorithm that calculates a positional δ-approximation of q-*$\textsc{Quantile}$[fVSA, *$\textsc{Poly}$*] *with success probability at least $\frac{3}{4}$. Furthermore, the run time of the algorithm is polynomial in the input and $\frac{1}{\delta}$.*

[20] The idea of positional quantile approximations was originally introduced by Manku et al. [100] in the context of quantile computations with limited memory.

Proof. Let $A \in$ fVSA be a functional vset-automaton and $d \in \Sigma^*$ be a document. Arenas et al. [12, Corollary 4.1] showed that given a functional vset-automaton, one can sample tuples $\mathrm{t} \in [\![A]\!](d)$ uniformly at random with success probability $\geq \frac{1}{2}$.[21] We will use this sampling algorithm to first create a sample of the assigned weights and then return the q-Quantile of this sample. The algorithm is depicted in Procedure PositionalQuantileApprox.

We note that this algorithm has two points of failure. On one hand, it can happen that less then $s := \lceil \frac{\ln(16)}{2\delta^2} \rceil$ calls to the sampling algorithm of Arenas et al. [12] are successful. On the other hand, it can happen that the returned quantile is no positional δ-approximation of the quantile. We show that both of these points of failure have a probability of less than $\frac{1}{8}$. Thus, the probability that the whole algorithm is successful is $\frac{7}{8} \cdot \frac{7}{8} > \frac{3}{4}$. We will first show that Line 6 is reached with probability less than $\frac{1}{8}$.

The success probability of each call to the sampling algorithm of Arenas et al. [12] is at least $\frac{1}{2}$. Thus, the expected number of samples, generated by $4s$ consecutive calls to the algorithm is at least $2s$. Using Hoeffding's Inequality, the probability that $4s$ consecutive calls to the sampling algorithm yield less than s samples is less than e^{-s} and therefore less than $\frac{1}{8}$ for every $s \geq 3$.[22]

It remains to show that a total of s samples is enough to guarantee that the q-Quantile of W is a positional δ-approximation of q-Quantile$([\![A]\!], d, w)$ with probability at least $\frac{7}{8}$.

Let $w_{q-\delta} = (q-\delta)$-Quantile$([\![A]\!], d, w)$ and $w_{q+\delta} = (q+\delta)$-Quantile$([\![A]\!], d, w)$. Furthermore, let $W_{q-\delta} = \{\!\{x \in W \mid x < w_{q-\delta}\}\!\}$ and $W_{q+\delta} = \{\!\{x \in W \mid x > w_{q+\delta}\}\!\}$. We say that a sample is bad, if either $|W_{q-\delta}| \geq q \cdot s$ or $|W_{q+\delta}| \geq (1-q) \cdot s$. We will first show that the probability that $|W_{q-\delta}| \geq q \cdot s$ is at most $e^{-2\delta^2 \cdot s}$. For each element $x \in W$ the probability that $x \in W_{q-\delta}$ is at most $(q-\delta)$. Thus, the expected size of $W_{q-\delta}$ is $(q-\delta) \cdot s$. Using Hoeffding's Inequality, with $\lambda = \delta \cdot s$ the probability that $|W_{q-\delta}| \geq q \cdot s$ is at most $e^{-2\delta^2 \cdot s}$. On the other hand, the for each element $x \in W$ the probability that $x \in W_{q+\delta}$ is at most $(1-(q+\delta)) = 1-q-\delta$. Thus, the expected size of $W_{q+\delta}$ is $(1-q-\delta) \cdot s$. Again, using Hoeffding's Inequality, with $\lambda = \delta \cdot s$ the probability that $|W_{q+\delta}| \geq (1-q) \cdot s$ is at most $e^{-2\delta^2 \cdot s}$. Therefore, the probability for a bad sample is at most $2 \cdot e^{-2\delta^2 \cdot s}$. Due to $s = \lceil \frac{\ln(32)}{2\delta^2} \rceil$, the probability of a bad sample is at most $\frac{1}{8}$, concluding the proof. $\square$

[21] We note that the sampling algorithm by Arenas et al. [12, Corollary 4.1] detects and reports failures.
[22] Obviously, we can call the sampling algorithm 16 times for $s = 1$ and $s = 2$ to ensure a failure rate of less than $\frac{1}{8}$.

Conclusions

Chapter 7

Summary and Directions for Future Research

Throughout this thesis, we studied multiple aspects of document spanners. That is, we studied parallel evaluation in Part I and quantitative aspects of document spanners in Part II. We will now summarize the results and discuss open problems and directions for future research.

7.1 Parallel Evaluation of Document Spanners

In Part I, we embarked on an exploration of the task of automating the distribution of information-extraction programs across splitters. Adopting the formalism of document spanners and the concept of parallel-correctness, our framework focuses on two computational problems, Split-Correctness and Splittability, as well as their special case of Self-Splittability. We presented an analysis of these problems and studied their complexity within the class of regular spanners. We have also discussed several natural extensions of the framework, considering the reasoning about splittability, schema constraints, and black-box spanners with split constraints. Our principal objective is to open up new directions for research within the framework, and indeed, several open problems are left for future investigation.

One open problem is the exact complexity of Splittability, as we do not have matching upper- and lower-bounds in the general case. The complexity is also open if the input is restricted to unambiguous sequential vset-automata and the highlander condition holds.

We know more about Split-Correctness and Self-Splittability, but there are some basic open problems there as well. For instance, when considering more expressive languages for spanners (e.g., the class of *core* spanners [45, 53] that allow for string equalities or context-free spanners [122]), all problems reopen.

A variant of Splittability that we barely touched upon is that of deciding, given a spanner S, whether it can be decomposed in a non-trivial way. We showed (Observation 4.5.1) that this variant closely relates to the Language-Primality problem—can a given regular language be decomposed as the concatenation of non-trivial regular languages? Interestingly, Martens et al. [103] showed that Language-Primality is also related to the work of Abiteboul et al. [1] on typing in distributed XML, which is quite reminiscent, yet different from, our work.

For the extensions of reasoning about splitters, and deciding on splittability with black-box spanners, we barely scratched the surface. Specifically, we believe that reasoning about split constraints over black-box extractors can have a profound implication on the usability of IE systems to developers at varying degrees of expertise, while embracing the advances of the Machine Leaning and Natural Language Processing communities on learning complex functions such as artificial neural networks.

7.2 Quantitative Aspects of Document Spanners

Part II consists of two Chapters, which we will summarize now.

7.2.1 Weight Annotators

In Chapter 5 we embarked on a study that incorporates *annotations* or *weights* in information extraction and propose $\mathbb{K}$-annotators as a candidate formalism to study this problem. The $\mathbb{K}$-annotators can be instantiated with *weighted vset-automata*, thereby obtaining *regular* $\mathbb{K}$*-annotators*, which are powerful enough to capture the extension of the traditional spanner framework with parametric factors. Furthermore, the regular $\mathbb{K}$-annotators have favorable closure properties, such as closure under union, projection, natural join, and, depending on the semiring, also under string selection using regular relations. The first complexity results on evaluation problems are encouraging: answer testing is tractable and, depending on the semiring, problems such as the threshold problem, the max tuple problem, and enumeration of answers are tractable too.

We note that the addition of weights to vset-automata also introduces new challenges. For instance, some questions which are typically studied in database theory are not yet fully understood for weighted automata, which are the basis of weighted vset-automata. Examples are equivalence and emptiness. Concerning equivalence, it is known that equivalence is undecidable for weighted vset-automata over the tropical semiring (cf. Proposition 5.3.3). In general, however, it is not completely clear for which semirings equivalence is decidable or not. Concerning emptiness, the definition in weighted automata literature is not as database theoreticians would expect. That is, it does not ask if there exists a document d such that the automaton returns at least one tuple with nonzero weight on d, but is additionally given a threshold (as in our Threshold problem) and asks if the automaton returns a tuple with at least the threshold weight (which requires an order on the semiring). It is not yet clear how much this threshold influences the complexity of the problem.

An additional challenge is *determinization* of weighted automata, which is a complex matter and not always possible. It is well-known to be possible for the Boolean semiring but, for the tropical semiring (defined as $(\mathbb{Q} \cup \{-\infty\}, \max, +, -\infty, 0)$) deterministic weighted automata are strictly less expressive than unambiguous weighted automata, which are strictly less expressive than general weighted automata (cf. Klimann et al. [83, Section 3.5]).

A possible direction for further exploration could be the study of annotators which use regular cost functions (cf. Colcombet [26]) instead of weighted automata. Since regular cost functions are restricted to the domain of the natural numbers, this would probably be most interesting in the case where the semiring domain is (a subset of) the natural numbers. Indeed, in this case, it is known that regular cost functions are strictly more expressive than weighted automata over the tropical semiring (cf. Colcombet et al. [27]) and therefore could provide a useful tool to annotate document spanners. On the other hand, it is not yet clear to us how to associate regular cost functions in a natural way to annotated relations, which require semirings.

7.2.2 Aggregation Functions for Document Spanners

In Chapter 6, we investigated the computational complexity of common aggregate functions over regular document spanners given as regex formulas and vset-automata. While each of the studied aggregate functions is intractable in the general case, there are polynomial-time algorithms under certain general assumptions. These include the assumption that the numerical value of the tuples is determined by a constant number of variables, or that the spanner is represented as an (unambiguous) vset-automaton. Moreover, we established quite general tractability results when randomized approximations (FPRAS) are possible. The upper bounds that we obtained for general (functional) vset-automata immediately generalize to aggregate functions over queries that involve relational-algebra operators and string-equality conditions on top of spanners, whenever these inner queries can be *efficiently* compiled into a single vset-automaton [54, 123]. Moreover, these upper bounds immediately generalize to allow for *grouping* (i.e., the GROUP BY operator) by computing the tuples of the grouping variables and applying the algorithms to each group separately.

We identified several interesting cases where the computation of $\alpha(S(d))$ can avoid the materialization of the exponentially large set $S(d)$, where, d is the document, S is the spanner, and α is the aggregate function. Notably, this is the case (1) for Min with general vset-spanners and weight functions in $\mathrm{Reg}_{\mathbb{T}}$, UReg, and CWidth, (2) for Max with general vset-spanners and weight functions in UReg and CWidth, (3) for Sum and Average with ufVSA-spanners and weight functions in $\mathrm{Reg}_{\mathbb{Q}}$, UReg and CWidth, and (4) for q-Quantile with ufVSA-spanners and CWidth weight functions.

Yet, several basic questions are left for future investigation. A natural next step would be to seek additional useful assumptions that cast the aggregate queries tractable: Can monotonicity properties of the numerical functions lead to efficient algorithms in cases that are otherwise intractable? What are the regex formulas that can be efficiently translated into unambiguous vset-automata (and, hence, allow to leverage the algorithms for such vset-automata)? Another important direction is to generalize the results in a more abstract framework, such as the *Functional Aggregate Queries* (FAQ) [81], in order to provide a uniform explanation of our findings and encompass general families of aggregate functions rather than specific ones. Finally, the practical side of our work remains to be studied: How do we make our algorithms efficient in practice? How effective is the sampling approach in terms of the balancing between accuracy and execution cost? Can we accurately compute estimators of aggregate functions over (joins of) spanners within the setting of *online aggregation* [66, 92]?

Appendix A

Proof of Lemma 4.4.3

Lemma 4.4.3. *The problems* Self-Splittability[dfVSA]*,* Splittability[dfVSA]*, and* Cover[dfVSA] *are* PSPACE*-hard, even if* P *is disjoint.*

Proof. In order to proof this result, we use a reduction by Smit [153, Proposition 3.3.7], who shows that Split-Correctness[dfVSA] is PSPACE-hard, even if P is disjoint.

We give a reduction from the PSPACE complete problem of DFA union universality [84]. Given deterministic finite automata $A_1, \ldots, A_n$ over the alphabet Σ, the union universality problem asks whether

$$\mathcal{L}(\Sigma^*) \subseteq \bigcup_{1 \leq i \leq n} \mathcal{L}(A_i) \,. \tag{$\dagger$}$$

Let $A_1, \ldots, A_n$ be DFAs over the alphabet Σ and let $a \notin \Sigma$ be a new alphabet symbol. Slightly abusing notation, we define the dfVSA by a hybrid regex-formula, where the automata A_i are plugged in. In particular,

$$A_S = a \vee a^{n+1}\Sigma^* \text{ , and}$$

$$A_P = x\{a\} \vee a \cdot x\{a\} \cdot a^{n-1} \cdot A_1 \vee a^2 \cdot x\{a\} \cdot a^{n-2} \cdot A_2 \vee \cdots \vee a^n \cdot x\{a\} \cdot A_n \,.$$

Furthermore, let $S = [\![A_S]\!]$ and $P = [\![A_P]\!]$. We show that the following statements are equivalent:

1. S is self-splittable by P,
2. S is splittable by P,
3. $\dagger$ holds,
4. P covers S.

We observe that (1) implies (2). Thus, we only need to show that (2) implies (3), (3) implies (4), and (4) implies (1).

(2) implies (3): Assume that $\dagger$ does not hold. Therefore, there is a document $d \in \Sigma^*$ with $d \notin \mathcal{L}(A_i)$, for every $1 \leq i \leq n$. Thus, $P(a^{n+1} \cdot d) = \emptyset$, but $() \in S(a^{n+1} \cdot d)$, which leads to the desired contradiction that S can not be splittable by P.

(3) implies (4): Assume that $\dagger$ holds. Let $d' \in (\Sigma \cup \{a\})^*$ be a document and $\mathrm{t} \in S(d')$ be a tuple. As S does not use variables, we have that $\mathrm{t} = ()$. We make a case distinction on d':

- $d' = a$,
- $d' \in \mathcal{L}(a^{n+1}\Sigma^*)$,
- $d' \notin \{a\} \cup \mathcal{L}(a^{n+1}\Sigma^*)$.

If $d' = a$, we have that $P(d') = \{[1, 2\rangle\}$ and therefore the cover condition is satisfied. On the other hand, if $d' \in \mathcal{L}(a^{n+1}\Sigma^*)$ there is a document $d \in \Sigma^*$ such that $d' = a^{n+1}d$. Thus, there is an index $1 \leq i \leq n$, such that $d \in \mathcal{L}(A_i)$ and therefore $[i+1, i+2\rangle \in P(d')$, covering $()$. In the last case, $S(d') = \emptyset$ which contradicts the assumption that $\mathrm{t} \in S(d')$.

(4) implies (1): We will show that $S = S \circ P$. Let $d \in (\Sigma \cup \{a\})^*$ be a document and let $\mathrm{t} \in S(d)$ be a d-tuple. Again, as S does not use variables, we have that $\mathrm{t} = ()$. As P covers S, there is a span $s \in P(d)$ which covers t. Using $() = () \gg s$ and $() \in (S \circ P)(d)$, we can conclude that $S \subseteq S \circ P$. For the other direction, let $d \in (\Sigma \cup \{a\})^*$ be a document and $\mathrm{t} \in (S \circ P)(d)$. As $S \circ P$ does not use any variables, we have that $\mathrm{t} = () = () \gg s$, for every $s \in P(d)$. By definition of S we have that $() \in S(d)$ showing $S \circ P \subseteq S$. $\square$

Appendix B

A Note on the CSV Schema Language SCULPT

Despite the availability of numerous standardized formats for semi-structured and semantic web data such as XML, RDF, and JSON, a very large percentage of data and open data published on the web remains tabular in nature.[1] Tabular data is most commonly published in the form of comma separated values (CSV) files because such files are open and therefore processable by numerous tools, and tailored for all sizes of files ranging from a number of KBs to several TBs. Despite these advantages, working with CSV files is often cumbersome [160] since they are typically not accompanied by a *schema* that describes the file's structure (i.e., "the second column is of integer datatype", "columns are delimited by tabs", ...) and captures its intended meaning. In fact, without schema information, already converting CSV-like data into a relational database is a challenging engineering problem [160]. In recognition of this problem, the *CSV on the Web* Working Group of the World Wide Web Consortium (W3C) [71] argues for the introduction of a schema language for tabular data to ensure higher interoperability when working with datasets using the CSV or similar formats. Inspired by the W3C effort towards a recommendation for tabular data and metadata on the Web, Martens et al. [101] proposed the tabular schema language SCULPT. At its core, SCULPT is a rule-based language with rules of the form $\varphi \to \rho$ where φ selects a set of regions[2] of the input table and ρ constrains the allowed structure and content of each such region. The region selection expressions φ are not limited to selecting columns but can navigate through a table, much like XPath expressions can navigate the nodes of an XML tree. This generalization beyond columns is necessary since there are natural cases in practice in which CSV-like data is not rectangular [14, 159] (see also Figure B.1).

In Doleschal et al. [37], we study static optimization of SCULPT schemas. In particular, we address the satisfiability problem that asks whether for a given SCULPT schema there is a CSV file that satisfies it. Not only is satisfiability a core problem in the foundations of database management field that has been studied in depth for a variety of formalisms, it is also particularly relevant for schema design as it allows to detect schemas that are not well-defined.

[1] Jeni Tennison, one of the two co-chairs of the W3C CSV on the Web working group claims that over 90% of the data published on data.gov.uk is tabular data [160].

[2] A region is a set of cells.

	1	2	3	4	5	6	7	8	9	10
1	subject	predicate	object	provenance						
2	:e4	type	PER							
3	:e4	mention	"Bart"	D00124	283-286					
4	:e4	mention	"JoJo"	D00124	145-149	0.9				
5	:e4	per:sibling	:e7	D00124	283-286	173-179	274-281			
6	:e4	per:age	"10"	D00124	180-181	173-179	182-191	0.9		
7	:e4	per:parent	:e9	D00124	180-181	381-380	399-406	D00101	220-225	230-233

Figure B.1: Fragment of a CSV-like file (added row and column numbers), inspired by use case 13 in [159].

Unsurprisingly, satisfiability of SCULPT quickly turns out to be undecidable, which we show by an easy reduction from the domino tiling problem [163]. Indeed, using only one rule, a region selection expression can be used to 'walk' over a grid testing all horizontal and vertical constraints, or alternatively many much simpler rules can be used to test all horizontal and vertical constraints in parallel for every domino type. Even though these observations are valid to demonstrate undecidability they use rather artificial constructions.

For this reason, we introduce a restricted variant of SCULPT called Lego SCULPT (L-SCULPT) that not only suffices to express the W3C use cases but also admits tractable satisfiability. In brief, L-SCULPT restricts region selection expressions to only select rectangular shaped areas, that is, (parts of) rows, columns, and rectangles, thereby constraining the structural power of the language. A second restriction is that L-SCULPT only considers tables on which no two selected regions intersect. Specifically, we make the following contributions:

1. We show that the safisfiability problem for the structural core of SCULPT is undecidable.

2. We define a fragment of SCULPT called L-SCULPT that is powerful enough to capture the structural core of the schemas for tabular data in the W3C recommendation [125, Section 5.5]. Intuitively, L-SCULPT allows selections of rows, columns, rectangles, and bounded-size regions in the directions up, left, down, and right, whereas the W3C's recommendation only allows column selection.[3]

3. Depending on which axes are used, we show that satisfiability of L-SCULPT is PTIME-complete or undecidable. Our main technical result shows that for L-SCULPT using only row, column, right, and rectangle selections satisfiability is in PTIME. The proof is an intricate reduction to the emptiness problem of nondeterministic tree automata where tables are encoded as trees.

Furthermore, in Doleschal et al. [32], we present CHISEL, a tool for flexible manipulation of CSV-like data. In brief, CHISEL supports SCULPT as an expressive built-in

[3] We only focus on the structural core of the languages. The W3C's proposal also supports key and foreign key constraints, which are out of scope here but easy to add to the language. (In fact, we implemented them in [32].)

schema language for CSV-like data, that can handle both tabular and non-tabular data. Furthermore, it supports a simple programming language for transforming tabular and non-tabular CSV-like data. Chisel enables the user to develop SCULPT schemas, build data transformations, and set up a pipeline for automatic conversion of “wild” CSV-like data into structured tabular data. The source code and the tool can be obtained at `https://github.com/PoDMR/Chisel`.

Bibliography

[1] Serge Abiteboul, Georg Gottlob, and Marco Manna. Distributed XML design. *Journal of Computer and System Sciences*, 77(6):936–964, 2011.

[2] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1974.

[3] Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *Automated Technology for Verification and Analysis*, pages 482–491, 2011.

[4] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *44th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 111:1–111:15, 2017.

[5] Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on trees under relabelings. In *21st International Conference on Database Theory (ICDT)*, pages 5:1–5:18, 2018.

[6] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *22nd International Conference on Database Theory (ICDT)*, pages 22:1–22:19, 2019.

[7] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. *SIGMOD Rec.*, 49(1):25–32, September 2020.

[8] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. *ACM Trans. Database Syst.*, 46(1):2:1–2:30, 2021.

[9] Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and transferability for conjunctive queries. *Journal of the ACM*, 64(5):36:1–36:38, 2017.

[10] Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Reasoning on data partitioning for single-round multi-join evaluation in massively parallel systems. *Communications of the ACM*, 60(3):93–100, 2017.

[11] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, 1980.

[12] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In *Proceedings of the 38th Symposium on Principles of Database Systems (PODS)*, pages 59–73, 2019.

[13] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. *SIGMOD Rec.*, 49(1):52–59, September 2020.

[14] Marcelo Arenas, Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. A framework for annotating csv-like data. *Proceedings of the VLDB Endownment (PVLDB)*, 9, 2016.

[15] Kaustubh Beedkar, Rainer Gemulla, and Wim Martens. A unified framework for frequent sequence mining with subsequence constraints. *ACM Trans. Database Syst.*, 44(3), June 2019.

[16] Henrik Björklund, Wouter Gelade, and Wim Martens. Incremental xpath evaluation. *ACM Trans. Database Syst.*, 35(4):29:1–29:43, 2010.

[17] Pierre Bourhis, Alejandro Grez, Louis Jachiet, and Cristian Riveros. Ranked enumeration of MSO logic on words. In *24th International Conference on Database Theory (ICDT)*, pages 20:1–20:19, 2021.

[18] Anne Brüggemann-Klein. Unambiguity of extended regular expressions in SGML document grammars. In *Algorithms - ESA '93, First Annual European Symposium*, pages 73–84, 1993.

[19] Anne Brüggemann-Klein and Derick Wood. Deterministic regular languages. In *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science*, pages 173–184, 1992.

[20] Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Inf. Comput.*, 140(2):229–253, 1998.

[21] Jinxiu Chen, Donghong Ji, Chew Lim Tan, and Zheng-Yu Niu. Unsupervised feature selection for relation extraction. In *International Joint Conference on Natural Language Processing (IJCNLP), Companion Volume*, 2005.

[22] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *Annual Meeting on Association for Computational Linguistics (ACL)*, pages 128–137, 2010.

[23] Jason P. C. Chiu and Eric Nichols. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the ACL*, 4:357–370, 2016.

[24] K. Y. Cockwell and I. G. Giles. Software tools for motif and pattern scanning: program descriptions including a universal sequence reading algorithm. *Computer Applications in the Biosciences*, 5(3):227–232, 1989.

[25] Thomas Colcombet. Forms of determinism for automata (invited talk). In *29th International Symposium on Theoretical Aspects of Computer Science, (STACS) 2012*, pages 1–23, 2012.

[26] Thomas Colcombet. Logic and regular cost functions. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–4, 2017.

[27] Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Torunczyk. Cost functions definable by min/max automata. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 29:1–29:13, 2016.

[28] Wojciech Czerwinski, Claire David, Katja Losemann, and Wim Martens. Deciding definability by deterministic regular expressions. *J. Comput. Syst. Sci.*, 88:75–89, 2017.

[29] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

[30] Joel D. Day, Daniel Reidenbach, and Markus L. Schmid. Closure properties of pattern languages. *J. Comput. Syst. Sci.*, 84:11–31, 2017.

[31] Johannes Doleschal, Noa Bratman, Benny Kimelfeld, and Wim Martens. The Complexity of Aggregates over Extractions by Regular Expressions. In *24th International Conference on Database Theory (ICDT)*, pages 10:1–10:20, 2021.

[32] Johannes Doleschal, Nico Höllerich, Wim Martens, and Frank Neven. Chisel: Sculpting tabular and non-tabular data on the web. In Pierre-Antoine Champin, Fabien L. Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis, editors, *Companion of the The Web Conference 2018 on The Web Conference 2018 (WWW)*, pages 139–142. ACM, 2018.

[33] Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *Proceedings of the 38th Symposium on Principles of Database Systems (PODS)*, pages 149–163, 2019.

[34] Johannes Doleschal, Benny Kimelfeld, Wim Martens, Frank Neven, and Matthias Niewerth. Split-correctness in information extraction. *CoRR*, abs/1810.03367, 2021.

[35] Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *23rd International Conference on Database Theory (ICDT)*, pages 8:1–8:18, 2020.

[36] Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. *CoRR*, 2020.

[37] Johannes Doleschal, Wim Martens, Frank Neven, and Adam Witkowski. Satisfiability for SCULPT-schemas for csv-like data. In *21st International Conference on Database Theory (ICDT)*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[38] Manfred Droste and Werner Kuich. *Semirings and Formal Power Series*, pages 3–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[39] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[40] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd Symposium on Principles of Database Systems (PODS)*, pages 121–131, 2014.

[41] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004.

[42] Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, Inc., Orlando, FL, USA, 1974.

[43] Zoltán Ésik and Werner Kuich. *Finite Automata*, pages 69–104. Springer Berlin Heidelberg, 2009.

[44] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Spanners: a formal framework for information extraction. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2013.

[45] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *Journal of the ACM*, 62(2):12:1–12:51, 2015.

[46] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. A relational framework for information extraction. *SIGMOD Record*, 44(4):5–16, 2015.

[47] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative cleaning of inconsistencies in information extraction. *ACM Transactions on Database Systems*, 41(1):6:1–6:44, 2016.

[48] Fernando Florenzano, Cristian Riveros, Martin Ugarte, Stijn Vansummeren, and Domagoj Vrgoč. Constant delay algorithms for regular document spanners. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 165–177, 2018.

[49] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoč. Efficient enumeration algorithms for regular document spanners. *ACM Transactions on Database Systems*, 45(1), February 2020.

[50] J. Nathan Foster, Todd J. Green, and Val Tannen. Annotated XML: queries and provenance. In *Proceedings of the 27th Symposium on Principles of Database Systems (PODS)*, pages 271–280, 2008.

[51] Dominik D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory Comput. Syst.*, 53(2):159–193, 2013.

[52] Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019.

[53] Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory of Computing Systems*, 62(4):854–898, 2018.

[54] Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 137–149, 2018.

[55] Dominik D. Freydenberger and Liat Peterfreund. Finite models and the theory of concatenation. *CoRR*, abs/1912.06110, 2019.

[56] Dominik D. Freydenberger and Markus L. Schmid. Deterministic regular expressions with back-references. *Journal of Computer and System Sciences*, 105:1–39, 2019.

[57] Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *23rd International Conference on Database Theory (ICDT)*, pages 11:1–11:21, 2020.

[58] Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and containment for conjunctive queries with union and negation. *ACM Trans. Comput. Log.*, 20(3):18:1–18:24, 2019.

[59] Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006.

[60] Jonathan S. Golan. *Semirings and their Applications*. Springer, Dordrecht, 1999.

[61] Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999.

[62] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the 26th Symposium on Principles of Database Systems (PODS)*, pages 31–40, 2007.

[63] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, page 328–339, 1995.

[64] Benotît Groz, Sebastian Maneth, and Slawek Staworko. Deterministic regular expressions in linear time. In *Proceedings of the 31st Symposium on Principles of Database Systems (PODS)*, page 49–60, New York, NY, USA, 2012.

[65] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, page 157–166, 1993.

[66] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298. ACM Press, 1999.

[67] Marti A. Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[68] Luisa Herrmann, Heiko Vogler, and Manfred Droste. Weighted automata with storage. *Inf. Comput.*, 269, 2019.

[69] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, July 2001.

[70] Harry B. Hunt, Madhav V. Marathe, Venkatesh Radhakrishnan, and Richard E. Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27(4):1142–1167, August 1998.

[71] Phil Archer Ivan Herman. CSV on the web working group charter. `https://www.w3.org/2013/05/lcsv-charter.html`, 2015. Visited on Sept. 18, 2017.

[72] Abhay Kumar Jha, Vibhor Rastogi, and Dan Suciu. Query evaluation with soft-key constraints. In *Proceedings of the 27th Symposium on Principles of Database Systems (PODS)*, pages 119–128, 2008.

[73] Tao Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, December 1993.

[74] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[75] Sampath Kannan, Z. Sweedyk, and Steve Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, SODA '95, pages 551–557. Society for Industrial and Applied Mathematics, 1995.

[76] Jens Keppeler. *Answering Conjunctive Queries and FO+MOD Queries under Updates.* PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2020.

[77] Bas Ketsman, Aws Albarghouthi, and Paraschos Koutris. Distribution policies for datalog. *Theory Comput. Syst.*, 64(5):965–998, 2020.

[78] Bas Ketsman, Frank Neven, and Brecht Vandevoort. Parallel-correctness and transferability for conjunctive queries under bag semantics. In *21st International Conference on Database Theory (ICDT)*, pages 18:1–18:16, 2018.

[79] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 325–340, 2018.

[80] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Learning models over relational data using sparse tensors and functional dependencies. *ACM Trans. Database Syst.*, 45(2):7:1–7:66, 2020.

[81] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *Proceedings of the 35th Symposium on Principles of Database Systems (PODS)*, pages 13–28, 2016.

[82] Daniel Kirsten. A burnside approach to the termination of mohri's algorithm for polynomially ambiguous min-plus-automata. *RAIRO - Theoretical Informatics and Applications*, 42(3):553–581, 2008.

[83] Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349 – 373, 2004.

[84] D. Kozen. Lower bounds for natural proof systems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 254–266. IEEE, 1977.

[85] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490 – 509, 1988.

[86] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. J. Algebra Comput.*, 4(3):405–426, 1994.

[87] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 260–270, 2016.

[88] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215 – 217, 1983.

[89] Robert Leaman and Graciela Gonzalez. BANNER: an executable survey of advances in biomedical named entity recognition. In *Pacific Symposium on Biocomputing (PSB)*, pages 652–663, 2008.

[90] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Conference on Computational Natural Language Learning (CoNLL), Shared Task*, pages 28–34, 2011.

[91] Domenico Lembo, Yunyao Li, Lucian Popa, and Federico Maria Scafoglieri. Ontology mediated information extraction in financial domain with mastro system-t. In *Proceedings of the Sixth International Workshop on Data Science for Macro-Modeling*, 2020.

[92] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *SIGMOD Conference*, pages 615–629. ACM, 2016.

[93] Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning*, Lecture Notes in Computer Science, pages 319–339, 2004.

[94] Katja Losemann and Wim Martens. MSO queries on trees: enumerating answers under updates. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) CSL-LICS*, pages 67:1–67:10, 2014.

[95] Katja Losemann, Wim Martens, and Matthias Niewerth. Descriptional complexity of deterministic regular expressions. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium (MFCS)*, pages 643–654, 2012.

[96] Katja Losemann, Wim Martens, and Matthias Niewerth. Closure properties and descriptional complexity of deterministic regular expressions. *Theor. Comput. Sci.*, 627:54–70, 2016.

[97] Ping Lu, Joachim Bremer, and Haiming Chen. Deciding determinism of regular languages. *Theory Comput. Syst.*, 57(1):97–139, 2015.

[98] Nizar R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv.*, 43(1), December 2010.

[99] Aman Madaan, Ashish Mittal, Mausam, Ganesh Ramakrishnan, and Sunita Sarawagi. Numerical relation extraction with minimal supervision. In *AAAI Conference on Artificial Intelligence*, pages 2764–2771, 2016.

[100] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, page 426–435, 1998.

[101] Wim Martens, Frank Neven, and Stijn Vansummeren. SCULPT: A schema language for tabular data on the web. In *World Wide Web Conference (WWW)*, pages 702–712, 2015.

[102] Wim Martens and Joachim Niehren. On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Sciences*, 73(4):550–583, 2007. Special Issue: Database Theory 2005.

[103] Wim Martens, Matthias Niewerth, and Thomas Schwentick. Schema design for XML repositories: complexity and tractability. In *Proceedings of the 29th Symposium on Principles of Database Systems (PODS)*, pages 239–250, 2010.

[104] Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. Document spanners for extracting incomplete information: Expressiveness and complexity. *CoRR*, abs/1707.00827, 2017.

[105] Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. Document spanners for extracting incomplete information: Expressiveness and complexity. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 125–136, 2018.

[106] Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In *CD-MAKE*, Lecture Notes in Computer Science, pages 350–369, 2018.

[107] Filip Mazowiecki and Cristian Riveros. Pumping lemmas for weighted automata. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 50:1–50:14, 2018.

[108] Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. In *ICLR (Poster)*, 2019.

[109] Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016.

[110] Mehryar Mohri. *Weighted Automata Algorithms*, pages 213–254. Springer Berlin Heidelberg, 2009.

[111] Yoav Nahshon, Liat Peterfreund, and Stijn Vansummeren. Incorporating information extraction in the relational database model. In *WebDB*, page 6. ACM, 2016.

[112] A. Neuwald and P. Green. Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712, 1994.

[113] C. T. Ng, M. S. Barketau, T. C. Edwin Cheng, and Mikhail Y. Kovalyov. "product partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *Eur. J. Oper. Res.*, 207(2):601–604, 2010.

[114] Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 769–778, 2018.

[115] Matthias Niewerth and Luc Segoufin. Enumeration of MSO queries on strings with constant delay and logarithmic updates. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 179–191, 2018.

[116] Galia Nordon, Gideon Koren, Varda Shalev, Benny Kimelfeld, Uri Shalit, and Kira Radinsky. Building causal graphs from medical literature and electronic medical records. In *AAAI*, pages 1102–1109. AAAI Press, 2019.

[117] Takamasa Okudono, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. Weighted automata extraction from recurrent neural networks via regression on state spaces. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5306–5314, 2020.

[118] Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.

[119] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Annual Meeting on Association for Computational Linguistics (ACL)*, pages 271–278, 2004.

[120] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[121] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Trans. Assoc. Comput. Linguistics (TACL)*, pages 101–115, 2017.

[122] Liat Peterfreund. Grammars for document spanners. In *24th International Conference on Database Theory (ICDT)*, LIPIcs, pages 7:1–7:18, 2021.

[123] Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *Proceedings of the 38th Symposium on Principles of Database Systems (PODS)*, pages 320–334, 2019.

[124] Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive Programs for Document Spanners. In *22nd International Conference on Database Theory (ICDT)*, pages 13:1–13:18, 2019.

[125] Rufus Pollock, Jeni Tennison, Gregg Kellogg, and Ivan Herman. Metadata vocabulary for tabular data. Technical report, World Wide Web Consortium (W3C), December 2015. `https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/`.

[126] Hoifung Poon and Pedro M. Domingos. Joint inference in information extraction. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 913–918, 2007.

[127] W. V. Quine. Concatenation as a basis for arithmetic. *Journal of Symbolic Logic*, 11(4):105–114, 1946.

[128] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230 – 245, 1963.

[129] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nate Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher D. Manning. A multi-pass sieve for coreference resolution. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 492–501, 2010.

[130] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282, 2017.

[131] Daniel Reidenbach and Markus L. Schmid. A polynomial time match test for large classes of extended regular expressions. *CoRR*, 2017.

[132] Matthew Richardson and Pedro Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

[133] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer, 1997.

[134] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A formal framework for probabilistic unclean databases. In *22nd International Conference on Database Theory (ICDT)*, pages 6:1–6:18, 2019.

[135] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[136] Kai Salomaa. Language decompositions, primality, and trajectory-based operations. In *Implementation and Applications of Automata*, pages 17–22. Springer Berlin Heidelberg, 2008.

[137] Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[138] F. M. Scafoglieri and D. Lembo. A formal framework for coupling document spanners with ontologies. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 155–162, 2019.

[139] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016*, pages 3–18, 2016.

[140] Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *24th International Conference on Database Theory (ICDT)*, pages 4:1–4:19, 2021.

[141] Markus L. Schmid and Nicole Schweikardt. Spanner evaluation over slp-compressed documents. In *Proceedings of the 40th Symposium on Principles of Database Systems (PODS)*, pages 153–165, 2021.

[142] Robert P. Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Trans. Inf. Syst.*, 27(2):12:1–12:19, 2009.

[143] Roy Schwartz, Sam Thomson, and Noah A. Smith. Bridging CNNs, RNNs, and weighted finite-state machines. In *ACL*, pages 295–305, 2018.

[144] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 151–163, 2018.

[145] Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Sketches of dynamic complexity. *SIGMOD Rec.*, 49(2):18–29, December 2020.

[146] Roberto Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR*, pages 64–78, 2006.

[147] Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 20:1–20:16, 2017.

[148] Helmut Seidl. Deciding equivalence of finite tree automata. In *STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, pages 480–492. Springer, 1989.

[149] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

[150] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative information extraction using Datalog with embedded extraction predicates. In *Conference on Very Large Data Bases (VLDB)*, pages 1033–1044, 2007.

[151] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using DeepDive. *Proceedings of the VLDB Endowment (PVLDB)*, 8(11):1310–1321, 2015.

[152] Takeshi Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposia on Software Science and Engineering*, pages 115–127, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[153] Benedict Smit. Aufteilungskorrektheit von abschnitsanfragen. Bachelors Thesis (in German), Technical University of Dortmund, Germany, 2020.

[154] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *International Conference on International Conference on Machine Learning (ICML)*, pages 129–136, 2011.

[155] Wee Meng Soon, Hwee Tou Ng, and Chung Yong Lim. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544, 2001.

[156] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Advances in Database Technology — EDBT '96*, pages 1–17, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[157] R. E. Stearns and H. B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985.

[158] Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[159] Jeremy Tandy, Davide Ceolin, and Eric Stephan. CSV on the Web: Use cases and requirements. Technical report, World Wide Web Consortium (W3C), Februrary 2016. `http://www.w3.org/TR/2016/NOTE-csvw-ucr-20160225/`.

[160] Jeni Tennison. 2014: The year of CSV. `http://theodi.org/blog/2014-the-year-of-csv`, 2014. Visited on Sept. 18, 2017.

[161] David Torrents, Mikita Suyama, Evgeny Zdobnov, and Peer Bork. A genome-wide survey of human pseudogenes. *Genome research*, 13(12):2559–2567, 2003.

[162] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2003.

[163] Peter van Emde Boas. The convenience of tilings. In *Complexity, Logic and Recursion Theory*, Lecture Notes in Pure and Applied Mathematics, pages 331–363. Marcel Dekker Inc., 1997.

[164] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity.* MIT Press, Cambridge, MA, USA, 1991.

[165] Alexandre Vigny. *Query enumeration and nowhere dense graphs. (Énumération des requêtes et graphes nulle-part denses).* PhD thesis, Paris Diderot University, France, 2018.

[166] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[167] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning, (ICML)*, pages 5244–5253, 2018.

[168] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[169] Stathis Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36(3):433 – 451, 1988.

[170] Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. Distant supervision for relation extraction via piecewise convolutional neural networks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1753–1762, 2015.

[171] David Zuckerman. On unapproximable versions of NP-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, December 1996.

List of Notations

Automata and Regex-formulas

RGX	Set of all regex-formulas
fRGX	Set of all functional regex-formulas
sRGX	Set of all sequential regex-formulas
VSA	Set of all variable-set automata
fVSA	Set of all functional variable-set automata
sVSA	Set of all sequential variable-set automata
dVSA	Set of all deterministic variable-set automata
dfVSA	Set of all deterministic functional variable-set automata
dsVSA	Set of all deterministic sequential variable-set automata
uVSA	Set of all unambiguous variable-set automata
ufVSA	Set of all unambiguous functional variable-set automata
usVSA	Set of all unambiguous sequential variable-set automata

Weight Functions

CWidth	Constant-width weight function
Poly	Polynomial-time weight function
Reg	Regular weight function (over the numerical or tropical semiring)
$\text{Reg}_{\mathbb{Q}}$	Regular weight function over the numerical semiring
$\text{Reg}_{\mathbb{T}}$	Regular weight function over the tropical semiring
UReg	Unambiguous regular weight function (over the numerical or tropical semiring)
$\text{UReg}_{\mathbb{Q}}$	Unambiguous regular weight function over the numerical semiring
$\text{UReg}_{\mathbb{T}}$	Unambiguous regular weight function over the tropical semiring

Other symbols

Γ_V	Set of variable operations over the variables in V
Σ	Alphabet
Σ^*	Set of all documents over Σ
$\mid x \mid$	Size of x

$[i, j\rangle$	Span from index i to index j
$[\![A]\!]$	Spanner defined by A
$a \gg b$	Shift a by b
$a \ll b$	Left shift a by b
D	Datavalues
d	Document
d_{t}	String tuple $(d_{\mathrm{t}(x_1)}, \dots, d_{\mathrm{t}(x_n)})$, where $\mathrm{Vars}(\mathrm{t}) = \{x_1, \dots, x_n\}$
$\mathrm{doc}(\mathrm{r})$	Document encoded by ref-word r
FPRAS	Fully polynomial-time randomized approximation scheme
$\mathcal{L}(A)$	Language accepted by A
$\mathcal{R}$	Ref-word Language
r	Ref-Word
$\mathrm{ref}(d, \mathrm{t})$	Ref-Word encoded by d and t, which satisfies the variable order condition
$S \circ P$	Composition of S and P
$\mathsf{Spans}(d)$	Set of all possible spans over document d
Spans	Set of all possible spans
t	Tuple
$\mathrm{tup}(\mathrm{r})$	Tuple encoded by r
Vars	Span variables
$\mathrm{Vars}(A)$	Variables of A
$V\text{-}\mathsf{Tup}$	Set of all V-tuples

Index