Database Principles in Information Extraction

Benny Kimelfeld*
LogicBlox, Inc.
Berkeley, CA, USA
bennyk@gmail.com

ABSTRACT

Information Extraction commonly refers to the task of populating a relational schema, having predefined underlying semantics, from textual content. This task is pervasive in contemporary computational challenges associated with Big Data. This tutorial gives an overview of the algorithmic concepts and techniques used for performing Information Extraction tasks, and describes some of the declarative frameworks that provide abstractions and infrastructure for programming extractors. In addition, the tutorial highlights opportunities for research impact through principles of data management, illustrates these opportunities through recent work, and proposes directions for future research.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.4 [**Database Management**]: Systems—*Textual databases, Relational databases, Rule-based databases*; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Algebraic language theory, Classes defined by grammars or automata, Operations on languages*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata, Relations between models*

General Terms

Theory

Keywords

Information extraction, document spanners, regular expressions, finite-state transducers, database inconsistency, database repairs, prioritized repairs

*The work of the author mentioned in this paper was done while at IBM Almaden – Research.

PODS'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2594538.2594563 .

1. INTRODUCTION

Information Extraction (IE) refers to the task of discovering structured information in textual content. More precisely, the goal in IE is to populate a predefined relational schema that has predetermined underlying semantics, by correctly detecting the values of records in a given text document or a collection of text documents. Popular tasks in the space of IE include *named entity recognition* [60] (identify proper names in text, and classify those into a predefined set of categories such as *person* and *organization*), *relation extraction* [67] (extract tuples of entities that satisfy a predefined relationship, such as *person-organization*), *event extraction* [3] (find events of predefined types along with their key players, such as *nomination* and *nominee*), *temporal information extraction* [21,44] (associate mentions of facts with mentions of their validity period, such as *nomination-date*), and *coreference resolution* [53] (match between phrases that refer to the same entity, such as "Obama," "the President," and "him").

As a discipline, IE had its start with the DARPA Message Understanding Conference in 1987 [29]. While early work in the area focused largely on military applications, this task is nowadays pervasive in a plethora of computational challenges, in particular those associated with Big Data, such as social media analysis [7], machine data analysis [26], healthcare analysis [66], customer relationship management [2], and indexing for semantic search [69]. Within a typical text-analytics pipeline (e.g., [65]), the output of IE is fed into a cleaning and/or fusion component, such as an entity-resolution algorithm, that in turn produces input for a global processing phase (e.g., statistical analysis or data mining). Contemporary business models like cloud computing, along with analytics platforms like Hadoop, facilitate such data analyses for a broad range of individuals and organizations.

This tutorial focuses on foundations of data management systems that involve IE over textual content. In Section 2 we give an overview of the methodologies used for carrying out IE tasks; we also highlight some programming paradigms and abstractions for developing IE solutions. Section 3 describes our recent work on a formal framework for IE, where we leverage known principles of database management. We conclude with Section 4, where we propose directions for future research.

2. PARADIGMS AND METHODOLOGIES FOR INFORMATION EXTRACTION

A plethora of methodologies have been proposed, studied and practiced for carrying out IE. We identify four main categories of methodologies: (1) rule invocation, (2) inference over probabilistic graphical models, (3) inference under soft logical constraints, and (4) classification. The approaches in the first category deploy

various rule languages as high-level specifications of deterministic IE programs. Within the second category, the IE task is encoded as a probabilistic model that combines the text tokens, annotations, and features as random variables with inter-dependencies, while the actual result is obtained by means of inference (e.g., finding the most likely assignment to the variables). The third category can be viewed as a combination of the former two: rules are specified, yet can be violated, and are in fact means of specifying a probability space over possible results (where the probability is measured by the cases of violation and satisfaction of the rules). Approaches in the fourth category treat the extraction task as a classification problem on candidate tuples. In each of these categories, components of the solution, such as rules and weight parameters, can be either manually encoded or automatically constructed from examples (by means of machine learning). Interestingly, a recent study [15] highlights and quantifies a contrast between the dominance of probabilistic approaches in scientific publications on IE in the research community of Natural Language Processing (NLP), and the dominance of rule-based approaches in industrial solutions.

Our focus here is on the first aforementioned category of methodologies for IE, namely rule invocation. Next, we describe the ideas underlying representative rule-based formalisms and systems. For completeness, we also give an overview of concepts in the other categories. The interested reader is referred to published tutorials and surveys on IE, such as Sarawagi's [57], for in-depth discussions on approaches to IE.

2.1 Rule Invocation

A typical rule-based system for IE supports two kinds of rules. Rules of the first kind declare the manner by which spans (intervals within the document) are annotated when scanning the text. Such a rule is usually a *finite-state transducer* that is represented by means of a pattern in some grammar. We refer to such rules as *direct extractors*. Rules of the second kind operate on top of the annotations produced by the direct extractors, and determine how those should be combined (e.g., by joining annotations into richer tuples, or by resolving conflicts among extracted tuples). In RAPIER [12], for instance, the rules are mostly direct extractors, and a rule is specified through three patterns: a *filler* that matches the extracted span, a *pre-filler* that matches the text right before the filler, and a *post-filler* that matches the text right after the filler. Each of these patterns is phrased as a regular expression (in a restricted grammar) over the words and part-of-speech tags. As another example, in WHISK [60] the transducers are defined as regular expressions with *capture variables*, which are embedded variables that are assigned the substrings that match the corresponding parts of the text; it also deploys rules to filter out conflicting extracted tuples (for example, different tuples should not contain overlapping spans), by imposing a special behavior of the transducers. As a third example, in FASTUS [4] the transducers are applied in two different phases, where the first phase ("Recognizing Phrases") produces annotations that are referenced by the rules of the second phase ("Recognizing Patterns"); following these two phases, the system applies rules that merge tuples produced by the second phase.

The earlier rule-based systems evolved into development platforms for programming IE. One of the most popular such platforms is the General Architecture for Text Engineering (GATE) [19], an open-source project by the University of Sheffield. GATE is an instantiation of the *cascaded finite-state transducers* specified by the *Common Pattern Specification Language* (CPSL) [5]. The core IE engine of GATE, called JAPE, processes a document via a sequence of *phases* (or *cascades*), each annotating spans with types by applying grammar rules over previous annotations and user-defined Java procedures. Another such a system is Xlog [59], which extends (non-recursive) Datalog with *documents* and *spans* as primitive data types, and built-in extractors (e.g., matchers of regular expressions) of spans from documents, and features a query plan optimizer. A system with substantial industrial and academic impact is SystemT [14], which evolved from the Avatar project [36, 54] and supports an SQL-like declarative language named *AQL* (Annotation Query Language). SystemT also includes a query-plan optimizer [54] and development tooling [45]. Conceptually, AQL provides a collection of direct extractors of relations from text (e.g., tokenizer, dictionary lookup, matchers of regular expressions, part-of-speech tagger, and other morphological analyzers), along with an algebra for relational manipulation.

There are also rule systems for IE that are designed for natural language, as their rules can be applied to structured data that expose pre-applied linguistic analyses. Examples include LIEP [33] (which exposes restricted linguistic information obtained through simple patterns) and INSTAREAD [30] (which exposes more thorough text processing such as deep parsing, coreference resolution, and named-entity recognition). Both of these systems use variants of (non-recursive) Datalog over base relations (EDB predicates) that store the linguistic information.

A significant research effort has been made towards the automation (or semi-automation) of rule engineering for IE. Such automation includes *dictionary learning*, *rule refinement*, and *rule induction*. In dictionary learning, a given seed dictionary is automatically expanded from a text corpus [17, 56]. Rule refinement aims to improve existing rules by (at least conceptually) exploring the space of syntactic revisions to the rules and observing the resulting impact on the performance on labeled data [45]. In rule induction, the rules are produced from scratch by learning from training data. The induction techniques include incremental rule specification (i.e., *top-down* induction), where the algorithm begins with high-recall rules and gradually restricts the rules to account for false positives [24, 60]; they also include the analogous rule generalization (i.e., *top-down* induction, where the algorithm generalizes rules that initially overfit the examples) [12, 16], and techniques from inductive logic programming [1, 47].

2.2 Inference and Classification

Designing rules to properly capture the given IE problem may be overly tedious and expensive, depending on the nature of the task and domain. In natural-language text, for example, there is a great variety of ways of expressing the same information, and the extraction should account for typical mistakes due to practices such as grammatical errors, jargon, slang, wishful thinking and sarcasm. An alternative approach, which has been extensively explored, is to capture the problem by means of a probabilistic model, where the actual IE task is done through inference over the probabilistic space. An important advantage of this approach is due to the fact that, very often, a significant portion of the specification of the probabilistic model (e.g., numeric parameters) can be effectively learned automatically by using well studied machine-learning techniques. Hence, machine learning can replace much of the work that would otherwise be done by the developer. IE development then entails labeling of examples (or generation of such examples from available resources) and engineering of *features* (informative components in the specification of the probabilistic model). Labeling of examples may be overly expensive if the learning technique requires too much training, and there have been approaches to reduce the effort of labeling, such as *weak supervision* [31]. Different methodologies within the probabilistic approach differ in the lan-

guage and structure that specify the probabilistic model, and consequently, in the deployed training and inference algorithms.

An example of a probabilistic model for IE is a *Naive Bayes classifier*, where the label of each token is determined independently by features of a (bounded-length) window that surrounds the token [25]. This model can be viewed as a simple case of *probabilistic graphical model*, which is a model where the text tokens, features, hidden states and annotations are presented as random variables organized in a graph structure, where edges represent probabilistic correlations using the Markov property stating that a node is independent from the rest of the nodes, given its neighbors. Such graph models that have been applied to IE include *Hidden Markov Models* [8,9,27,42] and *Maximum Entropy Markov Models* [39,46], which are Bayesian networks (i.e., directed acyclic graphical models) that can be viewed as generative models of text (in the former) or annotations (in the latter). A highly successful graph model is that of *Conditional Random Fields* [13, 41, 64], which is a non-generative model where the underlying graph is undirected.

Rules (or constraints) with soft interpretation provide means to program extractors by combining the simplicity and expressiveness of rules (allowing the developer to easily express her domain knowledge and insights) with the ability of probabilistic models to capture the uncertain nature of text extraction and to utilize techniques from machine learning. The deployed rules are logical formulas with free variables. An example of such a rule is the following: "If a location name y occurs in a sentence at most 3 tokens after a person name x, and one of these tokens is *born*, then $\mathsf{BornIn}(x, y)$ is true." The rules can be *grounded* by replacing the free variables with actual values (which are obtained by, e.g., deterministic rule invocation). For instance, a possible grounding of our example rule replaces x with "Einstein" and y with "Germany." These rules are usually weighted in order to express a-priory reliability therein. A *possible world* can be viewed as a collection of grounded predicates such as $\mathsf{BornIn}(\text{Einstein}, \text{Germany})$.

Different notions of soft constraints weight (that is, assign a certainty or probability measure to) a possible world by applying aggregate measures on the grounded rules that are satisfied and those that are violated. SOFIE [63], for example, attempts to satisfy the grounded rules to a maximum extent, using a MAX-SAT solver; hence, under this approach the probability of a possible world is (proportional to) the sum of weights across all the satisfied grounded rules. *Probabilistic Soft Logic* (PSL) [10] and *Markov Logic Networks* [50, 55] (MLNs), which have also been applied to IE [48, 51, 58], apply different such aggregate functions (which we do not define here) over the groundings. Interestingly, recent work has drawn strong connections between MLN inference and database research: Niu et al. [48] showed how a relational query engine can be used to improve the efficiency of inference over MLNs, and Jah and Suciu [37] showed that inference in MLNs can be carried out via a reduction to the problem of query evaluation over *tuple-independent probabilistic databases* [20]. Such probabilistic databases have been used by Dylla et al. [21] directly for IE.

An IE task can also be cast as a classification problem on candidate tuples. In such an approach, one first produces a set of candidate tuples by some simple means of extraction (featuring high recall but possibly low precision), and then applies classification to distinguish the correct tuples from the wrong ones (and therefore increase the precision). In standard classification, every instance (candidate tuple in our case) is mapped into a vector of features, each describing some property of the instance, and the classifier operates over that vector. The features can be either defined manually [35,43] or produced automatically from linguistic information, typically by graph querying (e.g., a feature is a sequence of labels on a path that connects the spans of a tuple in the dependency tree) [62] or *graph kernels* that translate a large (implicitly defined) space of features over graphs (e.g., paths connecting spans) to various measures of *similarity* to the training examples [11, 18].

3. THE FRAMEWORK OF DOCUMENT SPANNERS

In this section, we review a recent work by Fagin et al. [22, 23] who established a formal framework for IE programs, conducted an investigation of expressiveness, and proposed concepts that unify key mechanisms in existing rule systems by making connections to known principles from database research.

As mentioned earlier, a typical rule in an IE program expresses two kinds of functions. Functions of the first kind are *direct extractors* that produce tuples of spans by directly processing the text, usually via specified transducers (e.g., a regular expression with capture variables, or a dictionary lookup). Functions of the second kind apply *relational manipulation* to the relations obtained from the direct extraction. In XLog [59] and INSTAREAD [30] the relational manipulation is expressed through the (non-recursive) Datalog syntax and semantics, and in SystemT [14] the manipulation is by a variation of SQL. In CPSL [5] and JAPE [19], each cascade consists of transducer specifications that can reference the annotations of previous cascades; these specifications combine the two kinds of functions, but they can be viewed, conceptually, as relational joins between direct extractors in different cascades. Later, we will also discuss *cleaning* mechanisms that SystemT and JAPE further provide.

A rule language that features the above two kinds of functions can be abstracted as an ordinary relational query language, except that the base relations are replaced with the direct extractors. The framework of *document spanners* (or just *spanners* for short) is based on this abstraction, and in particular, aims to explore the relationship between the direct extraction (by different mechanisms) and the relational manipulation (through various relational operators); for example, a natural question to ask is to what extent the relational manipulation adds expressive power to the direct extractors. Next, we describe the spanner framework.

3.1 Spanners

We first need some notation. We assume a fixed, finite alphabet Σ. A *document* is a finite string over Σ (i.e., a member of the set Σ^*). A *span* of a document $\mathbf{d} = \sigma_1 \cdots \sigma_n$ represents the range of a substring of $\mathbf{d}$, and if has the form $[i, j\rangle$, where $1 \leqslant i \leqslant j \leqslant n+1$. The substring of $\mathbf{d}$ *spanned by* $[i, j\rangle$ is the string $\sigma_i \cdots \sigma_{j-1}$. For example, if $\mathbf{d}$ is `ACM_PODS_2013`, then the span $[5, 9\rangle$ refers to the part of $\mathbf{d}$ from the fifth to the eighth symbols inclusive, spanning the substring `PODS`. The *variables* we use in this framework are all assigned spans. A *spanner* extracts from a string a relation over its spans, and it is formally defined as follows. Let V be a finite set of *variables*. A $(V, \mathbf{d})$-*tuple* is a mapping that assigns a span of $\mathbf{d}$ to each variable in V. A $(V, \mathbf{d})$-*relation* is a set of $(V, \mathbf{d})$-tuples. Note that in a $(V, \mathbf{d})$-relation, the variables of V are playing the roles of the attribute names, and the spans themselves are used as attribute values. When V and $\mathbf{d}$ are clear from the context, we may write just *tuple* and *relation* instead of $(V, \mathbf{d})$-tuple and $(V, \mathbf{d})$-relation, respectively. A *spanner* is a mapping that is associated with a set V of variables, and that maps every document $\mathbf{d}$ to a $(V, \mathbf{d})$-relation.

As an example, Figure 1 depicts a document $\mathbf{d}$. The alphabet Σ consists of the lowercase and uppercase letters from the English alphabet (i.e., `a`,...,`z` and `A`,...,`Z`), the comma symbol (","), and the underscore symbol ("_") that stands for whitespace. (We use a

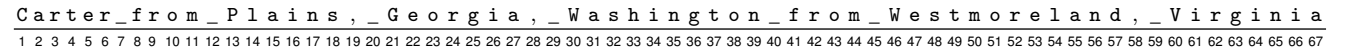

Figure 1: Example of a document

$\rho_{\mathsf{loc}}(\mathbf{d})$			
	x_1	x_2	y
μ_1	$[13, 19\rangle$	$[21, 28\rangle$	$[13, 28\rangle$
μ_2	$[21, 28\rangle$	$[30, 40\rangle$	$[21, 40\rangle$
μ_3	$[46, 58\rangle$	$[60, 68\rangle$	$[46, 68\rangle$

Figure 2: The result of applying the spanner ρ_{loc} to the document $\mathbf{d}$ of Figure 1

restricted alphabet for simplicity.) The figure also depicts the index of each character in $\mathbf{d}$. Later, we will define a spanner ρ_{loc}. Figure 2 shows the result of ρ_{loc} over $\mathbf{d}$; this result is a $(V, \mathbf{d})$-relation, where $V = \{x_1, x_2, y\}$.

Fagin et al. [22] focus on representations of spanners, and their associated expressive power. The main representation system they consider consists of the following.

- *Direct extractors* are defined by regular expressions with *capture variables* (and with some natural syntactic restrictions), which they call *regex formulas*. These regex formulas take the role of the base relations in the relational model.
- *Relational manipulation* is done by algebraic operators such as natural join, projection, union, difference, and different kinds of selection. Note that projection is based on *span* equality rather than *string* equality.

We do not give the formal definition of a regex formula here, but rather provide examples. For a formal definition of a regex formula, the reader is referred to the paper of Fagin et al. [22]. The following regex formula extracts tokens (which, for simplicity of presentation, are simply complete words) from text.

$$\gamma_{\mathsf{tkn}}[x] := \tag{1}$$
$$\big(\epsilon \vee (\Sigma^* \cdot \_)\big) \cdot x\{[\mathtt{a} - \mathtt{zA} - \mathtt{Z}]^+\} \cdot \big(\big((, \vee \_) \cdot \Sigma^*\big) \vee \epsilon\big)$$

In the syntax of regex formulas, "ϵ" denotes the empty string, "$\vee$" denotes disjunction, "*" denotes Kleene star, "$^+$" is the same as "*" with the exclusion of zero occurrences, and "$\cdot$" denotes concatenation. We often omit concatenation for readability. We are also using the convention that Σ denotes the disjunction over all the symbols, $[\mathtt{a} - \mathtt{z}]$ denotes the disjunction $\mathtt{a} \vee \cdots \vee \mathtt{z}$, $[\mathtt{A} - \mathtt{Z}]$ denotes the disjunction $\mathtt{A} \vee \cdots \vee \mathtt{Z}$, and $[\mathtt{a} - \mathtt{zA} - \mathtt{Z}]$ denotes the disjunction $[\mathtt{a} - \mathtt{z}] \vee [\mathtt{A} - \mathtt{Z}]$. Observe that the formula is applied to the entire document $\mathbf{d}$ (as opposed to every substring of $\mathbf{d}$).

The above regex formula (1) defines a spanner with the variable set $V = \{x\}$. The expression to the left of x matches either the empty string (meaning that x is a prefix of the document) or a string that ends with an underscore. The expression to the right of x matches either the empty string (meaning that x is a suffix of the document), or a string that begins with an underscore or a comma. The expression $[\mathtt{a} - \mathtt{zA} - \mathtt{Z}]^+$ inside $x\{\cdot\}$ means that x is a nonempty string that consists of only English letters (where each letter is in either lowercase or uppercase). When applied to the document $\mathbf{d}$ of Figure 1, the resulting $(V, \mathbf{d})$-tuples map x to $[1, 7\rangle$, $[8, 12\rangle$, and so on.

The following regex formula extracts spans that begin with a capital letter.

$$\gamma_{\mathsf{1Cap}}[x] := \Sigma^* \cdot x\{[\mathtt{A} - \mathtt{Z}] \cdot \Sigma^*\} \cdot \Sigma^*$$

When applied to the document $\mathbf{d}$ of Figure 1, the resulting spans include $[1, 7\rangle$, $[1, 3\rangle$, $[13, 19\rangle$, $[13, 20\rangle$, and so on.

The following regex formula extracts all the triples (x_1, x_2, y) of spans such that the string ", _" separates between x_1 and x_2, y is the span that begins where x_1 begins and ends where x_2 ends, and both x_1 and x_2 are tokens that begin with a capital letter.

$$\rho_{\mathsf{loc}}[x_1, x_2, y] := \big(\Sigma^* \cdot y\{x_1\{\Sigma^*\}, \_x_2\{\Sigma^*\}\} \cdot \Sigma^*\big) \tag{2}$$
$$\bowtie \gamma_{\mathsf{tkn}}[x_1] \bowtie \gamma_{\mathsf{tkn}}[x_2] \bowtie \gamma_{\mathsf{1Cap}}[x_1] \bowtie \gamma_{\mathsf{1Cap}}[x_2]$$

The result of applying $\rho_{\mathsf{loc}}[x_1, x_2, y]$ to the document $\mathbf{d}$ of Figure 1 is depicted in Figure 2. Note that the natural join is realized by using the same variable (namely x_1 or x_2) in multiple regex formulas.

All of the regex formulas given so far define what we call *hierarchical spanners*; this means that in every extracted $(V, \mathbf{d})$-tuple μ, and for every two variables $x, y \in V$, at least one of the following holds.

- $\mu(x)$ is contained in $\mu(y)$;
- $\mu(y)$ is contained in $\mu(x)$;
- $\mu(x)$ and $\mu(y)$ are disjoint.

An example of a non-hierarchical spanner is given by the following expression, which we will later reference in a different context. This spanner extract all the pairs (y, y') of spans, such that y and y' are different spans with a nonempty intersection (overlap), and either y begins before y' or y' is a prefix of y.

$$\pi_{y,y'}\Big(\big(\Sigma^* \cdot y\{\Sigma^+ \cdot z\{\Sigma^+\} \cdot \Sigma^*\} \cdot \Sigma^*\big) \bowtie \tag{3}$$
$$\big(\Sigma^* \cdot y'\{z\{\Sigma^+\} \cdot \Sigma^*\} \cdot \Sigma^*\big)\Big) \cup$$
$$\big(\Sigma^* \cdot y\{y'\{\Sigma^+\} \cdot \Sigma^+\} \cdot \Sigma^*\big)$$

Note that in the left operand of the union ($\cup$), the variable z represents a nonempty prefix of y', such that z is contained in y but is not a prefix of y. Also note that the spanner defined by this operand does not have the variable z, since z is projected out by the operation $\pi_{y,y'}$.

3.2 Expressiveness

To investigate the expressiveness of spanners defined by regex formulas and relational algebra, Fagin et al. [22] studied representation systems for spanners by means of simple transducers. Intuitively, those transducers are ordinary nondeterministic finite automata that open and close variables along their run, with the restriction that every variable should be opened and later closed precisely once. One such type of a transducer is the *variable-set automaton*, abbreviated *vset-automaton*. Figure 3 depicts an example of a vset-automaton. Observe that some of the transitions open variables (e.g., $x_1 \vdash$) and some close variables (e.g., $\dashv x_1$). A vset-automaton is a nondeterministic machine, and it can have multiple accepting runs over a document $\mathbf{d}$; each accepting run defines a beginning and an end index in $\mathbf{d}$ for each variable. The spanner defined by a vset-automaton produces one $(V, \mathbf{d})$-tuple (where V is the set of variables that occur in the automaton) for each accepting run. As an example, the vset-automaton of Figure 3 defines the same spanner as the one defined by the regex formula ρ_{loc} defined

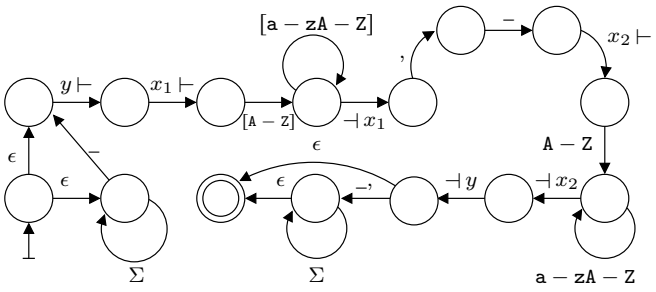

Figure 3: An example of a vset-automaton

in (2). Again, the reader is referred to Fagin et al. [22] for a precise definition of a vset-automaton and its translation into a spanner.

A vset-automaton can represent spanners that are not definable by any regex formula. In particular, regex formulas can define only hierarchical spanners, whereas it is easy to construct a vset-automaton that defines a non-hierarchical spanner. Fagin et al. [22] proved the following theorem, stating the relationship between regex formulas and vset-automata.

THEOREM 3.1. [22] *The following hold.*

1. *The class of spanners that are representable by regex formulas is precisely the class of* hierarchical *spanners that are representable by vset-automata.*
2. *The class of spanners that are representable by vset-automata is precisely the closure of the regex formulas under difference, natural join, projection, and union.*

Spanners representable by vset-automata are called *regular spanners*. Based on the above theorem, Fagin et al. [22] established additional results. For example, they proved that the expressive power of regular spanners is fully captured even without the difference operator in Part 2 of the theorem. They also give various fundamental results on the expressive power gained by adding *string equality* of spans (e.g., x and y span the same string, even though they may be different spans) as a selection operator; as an example, they show that in the presence of string equality, the difference operator strictly increases the expressive power.

3.3 Cleaning Inconsistencies through Database Repairs

Databases often involve inconsistency, due to human errors, integration of heterogeneous resources, imprecision entailed in Extract-Transform-Load (ETL) flows, and so on. The database research community has proposed principled ways to capture and manage data inconsistency [6]. A well-studied notion of inconsistency is the violation of *denial constraints* [6]; such constraints forbid cases such as two persons having the same driver's license number, or a person having multiple residential addresses. In IE tasks, inconsistency occurs even at a lower level. For example, an extractor may annotate multiple person mentions inside *"Martin Luther King Jr"*: *Martin Luther*, *Luther King, Martin Luther King Jr.,* etc. Moreover, all of these annotated spans may be contained in the larger span *"1805 Martin Luther King Jr Way, Berkeley, CA 94709,"* which by itself is annotated as an address, and which should not overlap with person mentions. As another example, consider again Figure 2, showing the result of applying the spanner (represented by) ρ_{loc} to the document $\mathbf{d}$ of Figure 1. If the goal is to extract locations, then the fact that the tuples μ_1 and μ_2 have overlapping assignments for y can be viewed as inconsistency, since we expect mentions of different locations to be disjoint spans.

To handle inconsistency, IE systems often provide cleaning mechanisms. JAPE [19], for instance, provides a collection of "controls" that represent different cleaning policies. These policies apply to unary annotators (that is, spanners with a single variable), and their specifications explain how the grammar should be translated into procedures (transducers) that avoid conflicts. As an example, in the Appelt control the procedure scans the document left to right; when at a specific location, it applies only the longest annotation, and continues scanning right after that annotation. In the Brill control, scanning is also done left to right, and when at a specific location, all the annotations that begin there are retained; but after that, scanning still continues after the longest annotation. In AQL [14], this cleaning mechanism comes in the form of "consolidation." Specifically, the AQL declaration of a view may include a command to filter out tuples by applying a consolidation policy to one of the columns. There is a built-in collection of such policies, like LeftToRight, which is similar to Appelt, and ContainedWithin that retains only the spans that are not strictly contained in other spans (but other than that, overlaps are permitted). WHISK [60] does not expose cleaning operations, but deploys implicit ones. For example, it deploys a rule which states that the substring captured by the wildcard ("*") should be as short as possible. Another rule states that after producing a tuple, the scan for the next tuple should proceed after (i.e., to the right of) all the spans of the first tuple (to avoid overlaps). In a sense, the well known POSIX semantics for matching regex formulas [34] can be viewed as an extreme cleaning policy that retains at most one tuple from the result.

The above cleaning mechanisms have been collected in an ad-hoc fashion in the course of use cases. Ideally, we would like to allow IE developers to declare their own policies. But the above policies are defined in a procedural way and, hence, it is not clear how to extend the built-in operations without requiring low-level coding of internal or external functions. In a recent work, Fagin et al. [23] used the framework of spanners to establish a formalism for declarative cleaning of inconsistencies in IE. This formalism unifies all of the above cleaning policies by adopting the concept of *prioritized repairs* [61], which extends the traditional notion of database repairs [6] with priorities among conflicting facts. Next, we briefly review this formalism.

In the framework of prioritized repairs, a relational database is augmented with two components. The first component is that of *integrity constraints*. To represent such constraints, Staworko et al. [61] use the conventional formalism of *denial constraints*, which are monotonic constraints (i.e., every subinstance of a consistent database instance is also consistent) that generalize common constraints such as functional dependencies (and key constraints in particular). The second component is a *priority* relation $\succ$, which is a binary relation over the facts; the meaning of $f_1 \succ f_2$ is that the fact f_1 is prioritized over the fact f_2. A *partial repair* of an inconsistent database instance I is a consistent subinstance J (i.e., J satisfies all the integrity constraints). A partial repair J is a *Pareto improvement* of a partial repair J' if J contains a fact f, such that $f \in J \backslash J'$, and $f \succ f'$ for every fact $f' \in J' \backslash J$. A *Pareto-optimal* repair (referred to as just *optimal* here) is a partial repair that does not have any Pareto improvement.[1] Conceptually, an inconsistent database instance is viewed as representing a set of consistent *possible worlds*—those are the optimal repairs.

[1]There are other notions of prioritized repairs, but here we consider only the Pareto semantics for simplicity of presentation. For a discussion on the relationship to other notions, see the work of Fagin et al. [23].

To adopt the concept of prioritized repairs, Fagin et al. [23] investigate extraction programs that are phrased as acyclic Datalog queries, where the *EDB predicates* (i.e., the relational symbols representing the base relations) are replaced with spanners. To phrase integrity constraints and priorities, they propose various formalisms. Constraints are specified by a spanner variant of denial constraints. Priority is specified by the formalism of *priority generating dependencies* (*pgds*), which is a spanner variant of the well known tuple-generating dependencies (tgds). The *denial pgds* jointly represent denial constraints and pgds. For simplicity, we will discuss only the latter formalism, namely pgds. An example of what one can express with a denial pgd is the following:

$$\mathsf{overlap}[x, y] \rightarrow \mathit{Address}(x) \rhd \mathit{Person}(y)$$

Here, $\mathsf{overlap}[x, y]$ is the spanner that produces all the pairs of overlapping spans, and *Address* and *Person* are unary relations that store extracted spans. This denial pgd states that whenever two facts $\mathit{Address}(x)$ and $\mathit{Person}(y)$ are such that x and y are overlapping spans, these two facts are in conflict, and moreover, the fact on the left side, namely $\mathit{Address}(x)$, has priority over the fact on the right side, namely $\mathit{Person}(y)$. (This can be due to the fact that the address extractor is deemed more precise than the person extractor.) Note, however, that the fact that $\mathit{Address}(x)$ and $\mathit{Person}(y)$ are in conflict does not necessarily mean that $\mathit{Person}(y)$ is excluded from every optimal repair; it may be the case that $\mathit{Address}(x)$ is excluded due to another declared denial pgd, in which case $\mathit{Person}(y)$ may survive.

As another example, consider again the spanner ρ_{loc} in (2). Suppose that R is a ternary relation that represents the result of the spanner ρ_{loc}; in the case of the document $\mathbf{d}$ of Figure 1, the relation R is given by the table in Figure 2. Let $\rho[y, y']$ denote the spanner defined in (3). Now consider the following denial pgd.

$$\rho[y, y'] \rightarrow \big(R(x_1, x_2, y) \rhd R(x'_1, x'_2, y')\big)$$

Similarly to the previous example, this denial pgd states that in R, whenever two facts $R(x_1, x_2, y)$ and $R(x'_1, x'_2, y')$ are such that the pair (y, y') is extracted by ρ, these two facts are in conflict, and moreover, $R(x_1, x_2, y)$ has priority over $R(x'_1, x'_2, y')$. This denial pgd actually captures the Appelt control of JAPE (as well as the LeftToRight consolidator of AQL). When applying this denial pgd to our example of R (Figure 2), we get a single optimal repair (possible world), namely the relation that consists of the tuples μ_1 and μ_3.

The property of defining a single optimal repair has practical importance, since systems are often not designed to support multiple possible worlds. Fagin et al. [23] explore the problem of whether a given specification with denial pgds (and/or a combination of denial constraints and pgds) is such that a single possible world is assured on every document. Unfortunately, they show undecidability results, such as the following one.

THEOREM 3.2. [23] *The following is undecidable. Given a denial pgd of the form $\rho[x, y] \rightarrow (R(x) \rhd R(y))$, where ρ is a regular spanner, is there an optimal repair for every document $\mathbf{d}$, assuming that R contains all the spans of $\mathbf{d}$?*

Practically, this theorem implies that more restricted safety conditions should be deployed. That work shows some tractable variants of this problem, but the exploration of robust variants that capture real-life examples is left for future research.

Another question they explore is whether the cleaning operations increase the expressive power of the extraction language (in the case of a single optimal repair). They show that in the case of regular spanners, it is possible to define a denial pgd that strictly increases the expressive power. Yet, interestingly, all of the policies mentioned in this section (include POSIX) do not increase the expressive power of regular spanners (but they increase the expressive power if string equality is allowed) [23]. This means that if the program uses only regular spanners with cleaning declarations, then it can be translated into an equivalent program that has no cleaning declarations. For some of the policies (e.g., ContainedWithin), this result is a special case of a more general phenomenon (in particular, this result holds for every transitive denial pgd that is phrased by means of a regular spanner). Also, for some of these cases (e.g., Appelt), this result required a fairly intricate proof.

4. CHALLENGES FOR FUTURE WORK

There are quite a few research challenges to pursue on the foundations and realization of data management systems that effectively incorporate IE. This section focuses on challenges that relate the framework of document spanners.

The theoretical research on spanners is still in its early stage, and some fundamental questions are still open. One class of such questions relates to the computational complexity of spanners: understanding the complexity of spanner evaluation under different representation systems, the complexity of translating spanners between representations (e.g., from algebraic expressions to and from automata with variables), and the complexity of traditional static-analysis questions such as emptiness, containment and equivalence of spanners. Another important direction is the extension of spanner representations with recursion. Specifically, we believe that some algorithms for IE can be naturally represented as recursive Datalog programs with spanner rules (while existing Datalog representations [23, 30, 59] do not support recursion). Another important direction is the incorporation of spanners within a more general database that may include multiple documents and ordinary relational data, as in Xlog [59]. Finally, Fagin et al. [23] leave open some questions on the expressive power of spanners, such as the effect of allowing both string equality and difference in the relational algebra. Obviously, some of these directions involve challenging system aspects towards the application to real-life data and tasks.

To maintain high precision, rule specifications such as spanners need to be highly elaborate, which often comes at the expense of recall and/or software complexity (e.g., many rules need to be deployed). It is then conceivable that in some practical scenarios one would like to enhance the rules with an inference engine that would reason about the precision of the rules and conflicts thereof. In principle, one could naturally extend the framework of spanners with notions of soft constraints, such as the aforementioned MLNs [50, 55]. It has been shown that MLNs properly capture the uncertainty involved in some IE tasks [48, 58]. We believe that an important research direction is to explore the foundations of a probabilistic database systems that combine Datalog, spanners and MLNs. This direction will hopefully allow us to leverage the research on learning soft constraints (e.g., [28, 40, 52]) within the task of rule induction for IE. A related direction is the application of spanners to probabilistic text (rather than deterministic text as discussed here), in a manner similar to the work of Kimelfeld and Ré [38] on the application of transducers to *Markov sequences*. This is especially useful when the text is generated by procedures that involve uncertainty, such as speech recognition [32], image processing [49], and normalization of informal text [68].

Acknowledgments

The author is grateful to Ronald Fagin for providing helpful comments on this paper.

5. REFERENCES

[1] J. S. Aitken. Learning information extraction rules: An inductive logic programming approach. In *ECAI*, pages 355–359. IOS Press, 2002.

[2] J. Ajmera, H.-I. Ahn, M. Nagarajan, A. Verma, D. Contractor, S. Dill, and M. Denesuk. A CRM system for social media: challenges and experiences. In *WWW*, pages 49–58, 2013.

[3] C. Aone and M. Ramos-Santacruz. Rees: A large-scale relation and event extraction system. In *ANLP*, pages 76–83, 2000.

[4] D. E. Appelt, J. R. Hobbs, J. Bear, D. J. Israel, and M. Tyson. FASTUS: A finite-state processor for information extraction from real-world text. In *IJCAI*, pages 1172–1178. Morgan Kaufmann, 1993.

[5] D. E. Appelt and B. Onyshkevych. The common pattern specification language. In *Proceedings of the TIPSTER Text Program: Phase III*, pages 23–30, Baltimore, Maryland, USA, 1998.

[6] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[7] E. Benson, A. Haghighi, and R. Barzilay. Event discovery in social media feeds. In *ACL*, pages 389–398, 2011.

[8] D. M. Bikel, S. Miller, R. M. Schwartz, and R. M. Weischedel. Nymble: a high-performance learning name-finder. In *ANLP*, pages 194–201, 1997.

[9] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *SIGMOD Conference*, pages 175–186. ACM, 2001.

[10] M. Bröcheler, L. Mihalkova, and L. Getoor. Probabilistic similarity logic. In *UAI*, pages 73–82. AUAI Press, 2010.

[11] R. C. Bunescu and R. J. Mooney. Subsequence kernels for relation extraction. In *NIPS*, 2005.

[12] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI/IAAI*, pages 328–334. AAAI Press / The MIT Press, 1999.

[13] F. Chen, X. Feng, C. Re, and M. Wang. Optimizing statistical information extraction programs over evolving text. In *ICDE*, pages 870–881. IEEE Computer Society, 2012.

[14] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *ACL*, pages 128–137, 2010.

[15] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! Long live rule-based information extraction systems! In *EMNLP*, pages 827–832. ACL, 2013.

[16] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *IJCAI*, pages 1251–1256. Morgan Kaufmann, 2001.

[17] A. Coden, D. Gruhl, N. Lewis, M. A. Tanenblatt, and J. Terdiman. Spot the drug! An unsupervised pattern matching method to extract drug names from very large clinical corpora. In *HISB*, pages 33–39. IEEE Computer Society, 2012.

[18] A. Culotta and J. S. Sorensen. Dependency tree kernels for relation extraction. In *ACL*, pages 423–429. ACL, 2004.

[19] H. Cunningham. GATE, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[20] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

[21] M. Dylla, I. Miliaraki, and M. Theobald. A temporal-probabilistic database model for information extraction. *PVLDB*, 6(14):1810–1821, 2013.

[22] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Spanners: a formal framework for information extraction. In *PODS*, pages 37–48. ACM, 2013.

[23] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *PODS*. ACM, 2014.

[24] D. Freitag. Toward general-purpose learning for information extraction. In *COLING-ACL*, pages 404–408, 1998.

[25] D. Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2/3):169–202, 2000.

[26] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *ICDM*, pages 149–158, 2009.

[27] S. Ginsburg and X. S. Wang. Regular sequence operations and their use in database queries. *J. Comput. Syst. Sci.*, 56(1):1–26, 1998.

[28] V. Gogate, W. A. Webb, and P. Domingos. Learning efficient Markov networks. In *NIPS*, pages 748–756. Curran Associates, Inc., 2010.

[29] R. Grishman and B. Sundheim. Message understanding conference 6: A brief history. In *COLING*, pages 466–471, 1996.

[30] R. Hoffmann. *Interactive Learning of Relation Extractors with Weak Supervision*. PhD thesis, University of Washington, 2012.

[31] R. Hoffmann, C. Zhang, X. Ling, L. S. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *ACL*, pages 541–550. The Association for Computer Linguistics, 2011.

[32] X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov models for speech recognition*, volume 2004. Edinburgh university press Edinburgh, 1990.

[33] S. B. Huffman. Learning information extraction patterns from examples. In S. Wermter, E. Riloff, and G. Scheler, editors, *Learning for Natural Language Processing*, volume 1040 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 1995.

[34] Institute of Electrical and Electronic Engineers and the Open group. The open group base specifications issue 7, 2013. IEEE Std 1003.1, 2013 Edition.

[35] H. Isozaki and H. Kazawa. Efficient support vector classifiers for named entity recognition. In *COLING*, 2002.

[36] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1):40–48, 2006.

[37] A. K. Jha and D. Suciu. Probabilistic databases with MarkoViews. *PVLDB*, 5(11):1160–1171, 2012.

[38] B. Kimelfeld and C. Ré. Transducing Markov sequences. In *PODS*, pages 15–26. ACM, 2010.

[39] D. Klein and C. D. Manning. Conditional structure versus conditional estimation in NLP models. In *EMNLP*, pages 9–16. Association for Computational Linguistics, 2002.

[40] S. Kok and P. Domingos. Using structural motifs for learning Markov logic networks. In *Statistical Relational Artificial*

Intelligence, volume WS-10-06 of *AAAI Workshops*. AAAI, 2010.

[41] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[42] T. R. Leek. Information extraction using hidden Markov models. Master's thesis, UC San Diego, 1997.

[43] Y. Li, K. Bontcheva, and H. Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2004.

[44] X. Ling and D. S. Weld. Temporal information extraction. In *AAAI*. AAAI Press, 2010.

[45] B. Liu, L. Chiticariu, V. Chu, H. V. Jagadish, and F. Reiss. Automatic rule refinement for information extraction. *PVLDB*, 3(1):588–597, 2010.

[46] A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *ICML*, pages 591–598, 2000.

[47] A. Nagesh, G. Ramakrishnan, L. Chiticariu, R. Krishnamurthy, A. Dharkar, and P. Bhattacharyya. Towards efficient named-entity rule induction for customizability. In *EMNLP-CoNLL*, pages 128–138. ACL, 2012.

[48] F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.

[49] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1):63–84, 2000.

[50] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 913–918. AAAI Press, 2007.

[51] J. Pujara, H. Miao, L. Getoor, and W. Cohen. Knowledge graph identification. In *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 542–557. Springer, 2013.

[52] L. D. Raedt and K. Kersting. Statistical relational learning. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning*, pages 916–924. Springer, 2010.

[53] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. D. Manning. A multi-pass sieve for coreference resolution. In *EMNLP*, pages 492–501. ACL, 2010.

[54] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan. An algebraic approach to rule-based information extraction. In *ICDE*, pages 933–942, 2008.

[55] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[56] E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479. AAAI Press / The MIT Press, 1999.

[57] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[58] S. Satpal, S. Bhadra, S. Sellamanickam, R. Rastogi, and P. Sen. Web information extraction using Markov logic networks. In *KDD*, pages 1406–1414. ACM, 2011.

[59] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB*, pages 1033–1044, 2007.

[60] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[61] S. Staworko, J. Chomicki, and J. Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.

[62] F. M. Suchanek, G. Ifrim, and G. Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In *KDD*, pages 712–717. ACM, 2006.

[63] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: a self-organizing framework for information extraction. In *WWW*, pages 631–640. ACM, 2009.

[64] D. Z. Wang, M. J. Franklin, M. N. Garofalakis, J. M. Hellerstein, and M. L. Wick. Hybrid in-database inference for declarative information extraction. In *SIGMOD Conference*, pages 517–528. ACM, 2011.

[65] R. Wisnesky, M. A. Hernández, and L. Popa. Mapping polymorphism. In *ICDT*, ACM International Conference Proceeding Series, pages 196–208. ACM, 2010.

[66] H. Xu, S. P. Stenner, S. Doan, K. B. Johnson, L. R. Waitman, and J. C. Denny. Application of information technology: Medex: a medication information extraction system for clinical narratives. *JAMIA*, 17(1):19–24, 2010.

[67] D. Zelenko, C. Aone, and A. Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106, 2003.

[68] C. Zhang, T. Baldwin, H. Ho, B. Kimelfeld, and Y. Li. Adaptive parser-centric text normalization. In *ACL (1)*, pages 1159–1168. The Association for Computer Linguistics, 2013.

[69] H. Zhu, S. Raghavan, S. Vaithyanathan, and A. Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500, 2007.