Avatar Information Extraction System

T.S. Jayram, Rajasekar Krishnamurthy, Sriram Raghavan,
Shivakumar Vaithyanathan, Huaiyu Zhu

IBM Almaden Research Center
{jayram,rajase,rsriram,vaithyan,huaiyu}@us.ibm.com

Abstract

The **AVATAR** *Information Extraction System (IES) at the IBM Almaden Research Center enables high-precision, rule-based, information extraction from text-documents. Drawing from our experience we propose the use of probabilistic database techniques as the formal underpinnings of information extraction systems so as to maintain high precision while increasing recall. This involves building a framework where rule-based annotators can be mapped to queries in a database system. We use examples from* **AVATAR** *IES to describe the challenges in achieving this goal. Finally, we show that deriving precision estimates in such a database system presents a significant challenge for probabilistic database systems.*

1 Introduction

Text analytics is a mature area of research concerned with the problem of automatically analyzing text to extract structured information. Examples of common text analytic tasks include *entity identification* (e.g., identifying persons, locations, organizations, etc.) [1], *relationship detection* (e.g., person X works in company Y)[9] and *co-reference resolution* (identifying different variants of the same entity either in the same document or different documents) [8]. Text analytic programs used for information extraction are called **annotators** and the objects extracted by them are called **annotations**. Traditionally, such annotations have been directly absorbed into applications. Increasingly, due to the complex needs of today's enterprise applications (such as Community Information Management [3]), there is a need for infrastructure that enables information extraction, manages the extracted objects and provides an easy interface to applications. Moreover, a very important pre-requisite for the use of annotations in enterprise applications is *high precision*.

At the IBM Almaden Research Center we are currently building the **AVATAR** Information Extraction System (IES) to tackle some of these challenges. Drawing from our experience in building the **AVATAR** IES infrastructure, in this paper, we make a case for the use of probabilistic database techniques as the formal underpinnings of such information extraction systems.

Annotations and Rules. Annotators in **AVATAR** IES are classified into two categories based on their input:

- **Base annotators** operate over the document text, independent of any other annotator.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

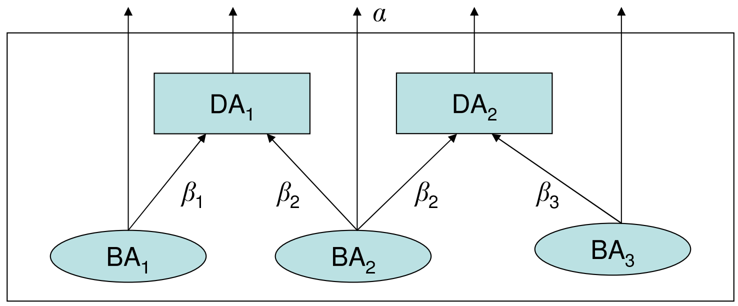

Figure 1: Thresholding annotator precisions

- **Derived annotators** operate on document text as well as the annotation objects produced by other annotators.

Every annotation object produced by an annotator is characterized by a well-defined set of operations. We refer to each such operation as a *rule*. A set of rules within an annotator that together produce an annotation object is called a *meta-rule*. As an example, consider a simple base annotator that identifies occurrences of person names in the text of a document. An example of a meta-rule used by such an annotator would be (informally) R_1: *look for the presence of a salutation such as Dr. or Mr. followed by a capitalized word*. Meta-rule R_1 would identify "Dr. Stonebraker" as a candidate Person annotation. Since a derived annotation depends on other annotation objects, the concept of meta-rule history is useful:

Definition 1 (Meta-Rule History): The *meta-rule history* $\mathcal{H}(a)$ of an annotation object a is defined as follows: If a is a base annotation produced by meta-rule R, then $\mathcal{H}(a) = \langle R \rangle$; if a is a derived annotation object produced by a meta-rule R that operates on previously defined (base or derived) annotation objects $a_1, \ldots, a_k$, then $\mathcal{H}(a) = \langle R, \mathcal{H}(a_1), \ldots, \mathcal{H}(a_k) \rangle$.

The confidence in the accuracy of an annotation is related to its meta-rule history. For example, the person annotator mentioned above may use a different meta-rule R_2 that looks for capitalized words that may or may not be person names (e.g., to identify "Bill" as a candidate Person). Intuitively, the annotator has higher confidence in R_1 and therefore, higher confidence in the accuracy of the objects produced by R_1. To formalize this intuition, we characterize the precision of individual annotation objects as follows:

Definition 2 (Annotation Object Precision): The precision $\mathrm{prec}(a)$ of an annotation object a is defined as the confidence value in $[0, 1]$ given by the annotator to all objects that can be produced with the same meta-rule history as that of a.

Definition 3 (High-precision Information Extraction (HPIE) System): An information-extraction system in which the precision of all annotation objects are above a threshold α (α close to 1.0)[1] is a high-precision information-extraction system.

In accordance with Definitions 1 and 2, the precision for derived annotations is computed using the entire meta-rule history of the corresponding base and derived annotators. This approach has two severe drawbacks. First, the combinatorial explosion in the space of meta-rules renders information-extraction systems of any reasonable size to be intractable. Second, there is a sparsity problem arising out of the fact that there may not be enough evidence (i.e., not enough annotation objects) to obtain precision estimates for all meta-rule histories.

[1]The choice of α will be driven by application requirements.

Current Implementation in AVATAR IES. The current implementation of **AVATAR** IES uses the UIMA [6] workflow engine to execute annotators. Annotations produced in **AVATAR** IES are stored in an annotation store implemented using a relational database (DB2). The store allows multiple derived annotators to be executed without having to re-execute the base annotators.

To overcome problems associated with maintaining meta-rule history, **AVATAR** IES implicitly assumes that all α-filtered input objects to a derived annotator are of equal precision. However, this approach has several problems that we explain below using Figure 1. In this figure, derived annotator (DA_1) has inputs from two base annotators BA_1 and BA_2. For an application that requires an HPIE system with $\alpha = 0.9$, the naive approach of α-filtering the objects of BA_1 and BA_2 may not be sufficient for DA_1 to produce derived objects with $\alpha \geq 0.9$.[2] The reason is that a derived annotator might require input annotations to be filtered at thresholds different from α in order to produce derived annotations above α. To account for this, annotators are thresholded differently (the β's in Figure 1) for consumption by derived annotators. This process can become complicated if multiple β's are required for a single base annotator whose output is consumed by different derived annotators. To minimize the need to tune a large number of β's, in the current implementation of **AVATAR** IES, we only allow two β settings for each annotator, namely, *high* and *medium*.

Motivation for Probabilistic Databases. Even under the simplistic assumptions made in our implementation, two problems remain. First, as **AVATAR** IES scales to a large number of annotators, the task of setting β's can quickly become intractable.[3] Second, the choice of β has a significant impact on the recall of derived annotators. As an example, consider a derived annotator PersonPhone (see Section 2.2) that uses Person annotations produced by a base annotator. In **AVATAR** IES, by switching the β for Person from *medium* to *high*, the number of annotation objects produced by PersonPhone over the Enron email data set [4] drops from 910 to 580.

Below, we motivate the use of probabilistic database techniques [2, 7] as a potential solution to these problems. Our first step in this direction is to view the precision of an annotation object in probabilistic terms.

Assumption 4: Consider an object a of type $\mathbf{type}(a)$. The precision $\mathrm{prec}(a)$ of a can be interpreted as a probability as follows: let $\widetilde{a}$ be drawn at random from the set of annotation objects whose meta-rule histories are identical to that of a. Then $\mathrm{prec}(a)$ equals the probability that $\widetilde{a}$ is *truly* an object of type $\mathbf{type}(a)$[4]. Formally, $\mathrm{prec}(a) = P(\mathbf{tt}(\widetilde{a}) = \mathbf{type}(a) \mid \mathcal{H}(a) = \mathcal{H}(\tilde{a}))$.

Assumption 5: Let R be a meta-rule in a derived annotator that takes as input k objects of specific types $T_1, \ldots, T_k$. Let the derived annotation object a be produced by R using annotation objects $a_1, a_2, \ldots, a_k$ of types $T_1, \ldots, T_k$, respectively, i.e., $a = R(a_1, \ldots, a_k)$. Let $\widetilde{a}$ and $\widetilde{a}_i$ correspond to a and a_i, for $i = 1 \ldots k$, as defined in Assumption 4. Then, $\mathbf{tt}(\widetilde{a}) = T \implies \forall\, i:\ \mathbf{tt}(\widetilde{a}_i) = T_i$.

Proposition 6: Using Assumptions 4 and 5, we can express the precision of an annotation object produced by a derived annotator as

$$\begin{aligned}
\mathrm{prec}(a) &= P(\mathbf{tt}(\widetilde{a}) = T \mid \mathcal{H}(a) = \mathcal{H}(\widetilde{a})) \\
&= P(\mathbf{tt}(\widetilde{a}) = T \mid \widetilde{a} = R(\widetilde{a}_1, \ldots, \widetilde{a}_k), \{\mathbf{tt}(\widetilde{a}_i) = T_i\}, \{\mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i)\}) && \text{(meta-rule-prec)} \\
&\qquad \cdot P(\{\mathbf{tt}(\widetilde{a}_i) = T_i\} \mid \widetilde{a} = R(\widetilde{a}_1, \ldots, \widetilde{a}_k), \{\mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i)\}) && \text{(input-prec)}
\end{aligned}$$

[2] Indeed for very high α a derived annotator may produce no objects.

[3] Today **AVATAR** IES has about a hundred annotators. However, the infrastructure for annotator development is sufficiently powerful that we expect this number to go up dramatically (potentially to tens of thousands).

[4] This is the result of an experimental procedure wherein an expert examines each object and determines whether it is truly of the intended type.

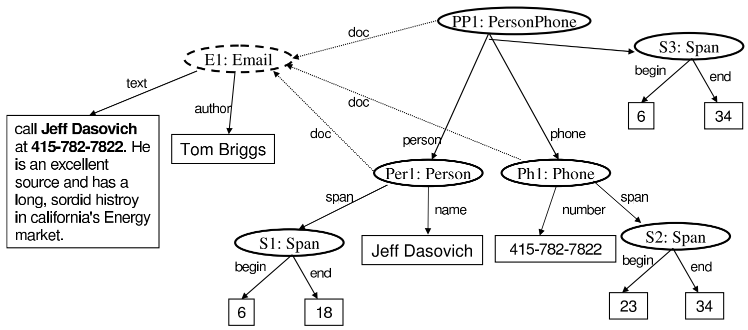

Figure 2: Annotation Store

In the proposition above, the expression (meta-rule-prec) represents meta-rule precision while the expression (input-prec) represents the overall input precision (see Section 4.3 for details and examples). While Assumption 4 has allowed us to reduce the problem of computing precision of derived object to one of computing probabilities for meta-rule precision and overall input precision, the expressions in Proposition 6 appear intractable. To obtain a practical solution, we believe that a better understanding of annotators and their dependencies is essential.

In the rest of the paper we describe the internals of **AVATAR** IES and connections to probabilistic databases as appropriate. In Section 2.2, we describe a generic template for **AVATAR** rule-based annotators. In Section 3, we consider a simple probability model for base annotators. Finally, in Section 4, we discuss briefly efficiency issues related to derived annotators.

2 Rule-based Annotators

2.1 Annotator Data Model

Each annotation produced by an IES can be viewed as a structured object with a well-defined type. The overall output of an IES, called an annotation store, is a collection of such objects as defined below:

Definition 7 (annotation store): An annotation store $\mathcal{S} = (\mathcal{T}, \mathcal{O})$ consists of a set of types $\mathcal{T}$, a set of objects $\mathcal{O}$, and two distinguished types $D, S \in \mathcal{T}$, such that :

- there is a special attribute text for type D
- $\forall x \in \mathcal{O}$ $\mathbf{type}(x) \in \mathcal{T}$
- $\forall x \in \mathcal{O}$ such that $\mathbf{type}(x) \neq D, \mathbf{type}(x) \neq S$, there exist attributes doc and span with $\mathbf{type}(x.\mathsf{doc}) = D$ and $\mathbf{type}(x.\mathsf{span}) = S$.

In this definition, D is called the *document type*, S is called the *span type*, and every other type in $\mathcal{T}$ is an *annotation type*. The special attribute text refers to the raw text of each document. The doc attribute of an annotation object A points to the source document from which A was extracted. Similarly, the span attribute of A describes the portion of A.doc.text where A was mentioned. Figure 2 shows a simple annotation store with one document object of type Email and three annotation objects. Each oval in the figure represents one object and is labeled as A:B where A is the ID of the object and B is the type. The rectangular boxes represent atomic attribute values. In this example the *span type* S contains a pair of integers begin and end that store the character offsets of the piece of the text corresponding to the annotation.

For the purposes of this paper, we assume that every annotation is extracted from a single document and that the span is interpreted relative to the text of this document.

procedure Annotator(d, $\mathcal{A}_d$)
R_f *is a set of rules to generate features*
R_g *is a set of rules to generate candidate annotations*
R_c *is a set of rules to consolidate the annotations produced by* R_g

1. *Features* = Compute_Features(R_f,d)
2. **foreach** $r \in R_g$
   Candidates = *Candidates* $\cup$ Apply_Rule(r,*Features*,$\mathcal{A}_d$)
3. *Results* = Consolidate(R_c,*Candidates*)
   return *Results*

Figure 3: **Generic Template for a Rule-based Annotator**

2.2 Rule-based Annotator Template

Figure 3 is a high-level template that describes the steps involved in a rule-based annotator. The input to the annotator is a document d and a set of annotation objects $\mathcal{A}_d$ such that $\forall x \in \mathcal{A}_d, x.\mathsf{doc} = d$ ($\mathcal{A}_d = \emptyset$ for base annotators). A rule-based annotator uses three kinds of rules: *feature extraction rules* R_f, *candidate generation rules* R_g, and *consolidation rules* R_c. Every rule operates on one or more annotation objects to produce other annotation objects. However, to distinguish between the objects produced by these different kinds of rules, we use the terms *features*, *candidates*, and *result annotations* respectively.

In the first step, a rule-based annotator uses the rules in R_f to produce a set of features. Note that all of the rules in R_f operate on the document and do not involve input annotations. In the second step, each rule in R_g is independently applied to the features and the input annotations. The output of each such rule is a set of candidates. Finally, the rules in R_c are used to consolidate candidates to produce a set of *Results*. Consolidation rules typically fall into two broad categories: (i) *discard rules* that discard some candidates, and (ii) *merge rules* that merge a set of candidates to produce a result annotation.

In **AVATAR** IES, the feature extraction rules are deterministic operations over the document that do not impact the precision of the result annotations. Therefore, the meta-rule for a result annotation A is the set of candidate generation and consolidation rules (rules from R_g and R_c) that are involved in producing A.

As concrete instantiations of the template in Figure 3, we describe below a base annotator called SimplePerson and a derived annotator called PersonPhone.

Base Annotator SimplePerson. Base annotator SimplePerson identifies persons using two dictionaries – D_u (containing unambiguous first or last names such as "michael", "stonebraker", etc.) and D_a (containing ambiguous first or last names such as "bill", "gray"). Expressed in terms of the template in Figure 3, SimplePerson works as follows:

- In the Compute_Features step, the input document D is tokenized and each individual token is added to the set of features *Features*. Further, for each feature F, an attribute dictU (resp. dictA) is set to true if the corresponding token matches an entry in D_u (resp. D_a).
- The set of rules R_g for identifying person names are: (i) r_{uu}: a pair of features that are adjacent to each other in the document text and both of which are labeled with D_u (e.g., michael stonebraker), (ii) r_{ua}: a pair of features that are adjacent to each other in the document text and are labeled with D_u and D_a respectively (e.g., james gray), (iii) r_u: a feature that is labeled with D_u (e.g., michael), and (iv) r_a: a feature that is labeled with D_a (e.g., gray).
- Step 3 consists of a single consolidation rule r_d that executes the following logic: *If two candidates* o_1 *and* o_2 *are such that the* o_1*.span contains* o_2*.span, discard* o_2. This rule is applied repeatedly to produce *Results*.

Note that SimplePerson is a simple but powerful annotator that we use in this paper to illustrate some key ideas. In reality, a person annotator will use a significantly larger set of rules that exploit feature attributes such as capitalization, the presence of salutations (e.g., Dr., Mr.), etc.

Derived annotator PersonPhone. Derived annotator PersonPhone identifies people's phone numbers from documents. This annotator takes in as input the document D and a set of Person, Phone and ContactPattern annotations identified by *Base* annotators. Here (i) Person annotations are produced by a person annotator such as SimplePerson, (ii) Phone annotations are produced by an annotator that identifies telephone numbers, and (iii) ContactPattern annotations are produced by an annotator that looks for occurrences of phrases such as *"at"*, *"can be (reached|called) at"* and *"'s number is"*. Given these inputs, the PersonPhone annotator is fully described by the following two rules:

- Candidate generation rule r_{seq}: *If a triple $\langle$Person,ContactPattern,Phone$\rangle$ appear sequentially in the document, create a candidate PersonPhone annotation.* An example instance identified using this rule is shown in Figure 2.
- Consolidation rule r_d as described earlier.

3 Probability Model for Base Annotators

Let us revisit the rules of the SimplePerson annotator that was described in Section 2.2. Since the only consolidation rule in this annotator r_d is a *discard rule*, it does not affect the probability assigned to the result objects. Therefore, each generation rule in SimplePerson fully describes a meta-rule. For each candidate generation rule in the SimplePerson annotator, let us assume the corresponding precision values are available to us (e.g., we are given $\text{prec}(r_{uu})$, $\text{prec}(r_{ua})$, etc.). The precision value associated with a rule is a measure of the confidence that the annotator writer has in the accuracy of that rule. An experimental procedure to compute such precision values would entail running the annotator on a labeled document collection and setting $\text{prec}(r)$ to be the fraction of objects produced by rule r that indeed turned out to be persons. Irrespective of how such precision values are computed, we make the same assumption about candidate generation rules as we did in Assumption 4 for meta-rules.

For example, if o is an object of type Person produced by rule r_{ua} upon examining the text "James Gray", we have $P(\texttt{tt}(o) = \text{Person}) = \text{prec}(r_{ua})$. Thus, for annotators with discard-only consolidation rules, knowing the precision values for generation rules is sufficient to assign probabilities to the result annotations. However, more complex base annotators use a mixture of *merge rules* and *discard rules* to perform consolidation. The task of computing annotation probabilities for such annotators is significantly more complicated.

To illustrate, let us consider a more sophisticated person annotator ComplexPerson that uses an additional dictionary D_s containing salutations (such as Dr., Mr., Prof., etc.). Besides the generation and consolidation rules of SimplePerson, the extended ComplexPerson annotator uses a generation rule r_s and a merge rule r_m where:

Rule r_s: generate a candidate annotation if there is a pair of features in the document text that are adjacent to each other and are such that the first one is labeled with D_s and the second one is labeled with either D_u or D_a (e.g., "Dr. Michael" and "Dr. Stonebraker" would both be matched by this rule).

Rule r_m: given two candidate o_1 and o_2 such that o_1.span overlaps with o_2.span, produce a new candidate object by merging the two spans into a single larger span[5]

[5] To fully specify ComplexPerson, we must now specify an order in which the consolidation rules r_m and r_d are executed. However, these details are omitted as they are not relevant for the discussion in this section.

For instance, given the piece of text “Dr. Michael Stonebraker”, rule r_s would produce object o_1 corresponding to “Dr. Michael”, rule r_{uu} would produce object o_2 corresponding to “Michael Stonebraker”, and rule r_m would merge o_1 and o_2 to produce o_3 corresponding to “Dr. Michael Stonebraker”. From our earlier assumption, we know that $P(\mathtt{tt}(o_1) = \mathsf{Person}) = \mathrm{prec}(r_s)$ and $P(\mathtt{tt}(o_2) = \mathsf{Person}) = \mathrm{prec}(r_{uu})$. However, to assign a probability to object o_3, we need a meaningful probability model to compute $P(\mathtt{tt}(o_3) = \mathsf{Person})$, given the above two probabilities. In general, designing a probability model to handle consolidation rules that involve merging of candidates to produce new annotation objects is an open problem. Today, in **AVATAR**, without such a model, we are forced to manually tune base annotators until we obtain desired precision levels.

4 Derived Annotators

Based on our experiences with **AVATAR** IES on various applications such as email, IBM intranet and Internet blogs, the real power is in the extraction of domain-specific derived annotations which are usually very large in number. As part of our attempts to make **AVATAR** IES more efficient we try to describe the rules in annotator template 2.2 as queries over the **AVATAR** annotation store. Such a view opens the door to exploit appropriate indices, evaluate alternate plans and in general perform cost-based optimization of *Derived* annotators.[6] We begin by describing some of our current challenges in mapping rules to queries. We then briefly discuss several issues that arise in computing precision of *Derived* annotations in the context of probabilistic databases.

4.1 Rules as Queries

In this section, we give examples that demonstrate that candidate generation and consolidation rules can be expressed as queries (using the standard Object Query Language (OQL) [5] syntax). For ease of exposition, we only consider discard rules in the consolidation step.

As our first example, the candidate generation rule r_{seq} for the PersonPhone annotator can be written as shown below:

Query q_1**.**

```
CREATE   P as person, Ph as phone, span as SpanMerge(P.span, CP.span, Ph.span), doc as P.doc
FROM     Person P, ContactPattern CP, Phone Ph
WHERE    ImmSucc(P.span,CP.span) AND ImmSucc(CP.span,Ph.span) AND P.doc = CP.doc AND CP.doc = Ph.doc
```

In the above, ImmSucc(span1,span2) is a user-defined function that returns true if “span2” occurs immediately after “span1” in the original document text. SpanMerge is another function that produces a new span by concatenating the set of spans provided as input. Note that we use a CREATE statement to indicate that a new annotation object is produced that contains the features P and Ph as attributes “person” and “phone” respectively. Similarly, rule r_d can be written as:

Query q_2**.**

```
Candidates -  (SELECT O2
               FROM Candidates O1, Candidates O2
               WHERE SpanContains(O1.span, O2.span))
```

where SpanContains(span1,span2) is a user-defined function that returns true if “span1” contains “span2”. Notice how the recursive application of rule r_d is captured using set difference.

4.2 Challenges in Modeling Complex Rules as Queries

While the example candidate generation rules presented above map easily to queries - **AVATAR** IES uses several more complex rules for which the mapping is not obvious. We present an example below from our suite of rule-based annotators to extract reviews from blogs - specifically, MusicReview annotator that identifies an

[6] Note that such queries can also be used to represent base annotator rules but the benefit is most significant for derived annotators.

informal music review from a blog. A simplistic version of this annotator works as follows. $\mathcal{A}_d$ consists of *MusicDescription* phrases identified by an earlier annotator (e.g., "lead singer sang well", "Danny Sigelman played drums"). The MusicReview annotator identifies contiguous occurrences of MusicDescription (based on some proximity condition across successive entries), groups them into blocks and marks each block as a MusicReview.

We now present the candidate generation and consolidation rules for a simplistic version of this annotator, which is interested in block size up to two, i.e., it identifies occurrences of the pattern MD and $\langle MD, MD\rangle$ in the document (MD is the input MusicDescription annotations).

$R_g = \{r_{md}, r_{2md}\}$ is a pair of rules that identifies occurrences of MD and $\langle MD, MD\rangle$ respectively. The corresponding queries are given below.

Query q_3**.**

```
CREATE   MD as first, MD as second
FROM     MusicDescription MD
```

Query q_4**.**

```
CREATE   MD_1 as first, MD_2 as second
FROM     MusicDescription MD_1, MusicDescription MD_2
WHERE    Window(MD_1.span, MD_2.span, m) AND BeginsBefore(MD_1.span, MD_2.span)
```

where Window(span1,span2,m) is a user-defined function that returns true if the two spans are within a distance of m, and BeginBefore(span1, span2) returns true if "span1" begins before "span2".

There is a single consolidation rule r_{d1} given below. This rule discards candidates that are completely contained in some other candidate, or have another MusicDescription inside them.

Query q_5**.**

```
Candidates -   (SELECT O_2
               FROM Candidates O_1, Candidates O_2
               WHERE BeginsBefore(O_1.first.span, O_2.first.span) AND EndsAfter(O_1.second.span, O_2.second.span)
               UNION
               SELECT O_2
               FROM Candidates O_1, Candidates O_2
               WHERE BeginsBefore(O_1.first.span,O_2.first.span) AND O_1.second = O_2.second
               UNION
               SELECT O_2
               FROM Candidates O_1, Candidates O_2
               WHERE O_1.first = O_2.first and EndsAfter(O_1.second.span, O_2.second.span))
```

For larger block sizes, note that the both the rules become significantly more complex. The following challenges arise in modeling the rules as queries: (i) ability to express proximity conditions across objects based on their location in the document (e.g., identify two MD's appearing adjacent to each other within m characters and no other MD between them) (ii) ability to group multiple such objects together, where each pair satisfies some proximity conditions and (iii) ability to retain groups in decreasing order of size.

4.3 Computing Precision of Derived Annotations

The precision of derived annotation objects is given by Proposition 6 in Section 1. In the expression given in Proposition 6, the precision of a derived annotation is the product of the rule-precision and the precision of the input objects. The computation of the rule precision for derived annotators is similar to that for base annotators, therefore all the issues discussed in Section 3 are relevant to derived annotators as well. We now turn to computing the second term involving the precision of the input objects. Without any additional information, one reasonable assumption is the following:

Assumption 8: The overall input precision of the input annotations in Proposition 6 is a product of the precisions of individual input objects and is independent of the rule R. In other words,

$$P(\{\mathtt{tt}(\widetilde{a}_i) = T_i\} \mid \widetilde{a} = R(\widetilde{a}_1, \ldots, \widetilde{a}_k), \{\mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i)\}) = \prod_i P(\mathtt{tt}(\widetilde{a}_i) = T_i \mid \mathcal{H}(a_i) = \mathcal{H}(\widetilde{a}_i))$$

Although the above assumption allows us to account for the overall input precision, this assumption is invalid for most derived annotators. In particular, we believe that most derived annotator rules enhance our confidence in the precision of the input annotations. For example, let us revisit the PersonPhone annotator described in Section 2.2. This annotator has a candidate generation rule r_{seq} that operates on annotations generated by three base annotators, namely, Person, ContactPattern and Phone. Consider the example annotation identified by this rule shown in Figure 2. The regular expressions used in the ContactPattern annotator and the sequencing condition (ImmSucc) in the rule r_{seq} have a strong correlation with the average precision of the Person annotation. In fact, in the current implementation of this annotator in **AVATAR** IES, the precision of the Person annotations that satisfy this additional constraint is 97.5% – significantly higher than the overall precision of Person annotations under $\beta =$ *medium* setting. Since Assumption 8 assumes that $P(\mathtt{tt}(Per1) =$ Person$)$ is independent of $\{\mathcal{H}(PP1)\}$ and the rule r_{seq}, this significantly lowers the computed precision for PersonPhone objects, and results in lowering the recall of the PersonPhone annotator.

The combination of mapping rules to queries and accounting for the dependencies as described above presents a significant challenge for probabilistic database systems. This is the focus of our ongoing work in this area. We believe that this requires enhancing the existing semantics for probabilistic database query evaluation, and will lead to a fresh set of open problems for efficient query evaluation under these enhanced semantics.

References

[1] H. Cunningham. Information extraction - a user guide. Technical Report CS-97-02, University of Sheffield, 1997.

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

[3] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. In *IEEE Data Engineering Bulletin*, March 2006.

[4] Enron Dataset. http://www-2.cs.cmu.edu/ enron/.

[5] O. Schadow et. al. *The Object Data Standard: ODMG 3.0*. Morgan Kauffman, 2000.

[6] D. Ferrucci and A. Lally. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, June 2004.

[7] N. Fuhr and T. Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[8] J. F. McCarthy and W. G. Lehnert. Using decision trees for coreference resolution. In *IJCAI*, pages 1050–1055, 1995.

[9] K. Nanda. Combining lexical, syntactic and semantic features with maximum entropy models for extracting relations. In *Proc. of the 42nd Anniversary Meeting of the Association for Computational Linguistics (ACL04)*, 2004.