Review of Relational Methods for Information Extraction

Mario J. Lorenzo

IBM

November 2018

Contact: mario@mjlorenzo.com or mjlorenz@us.ibm.com

Abstract

This paper will review a novel approach taken within the field of information extraction based on the relational database model. Current methods and frameworks for information extraction make use of grammar-based languages, such as regular expressions, in the form of rule patterns that are applied sequentially to unstructured text, such as a document, in order to produce spans of text that represent some meaning within a domain. These grammars are typically applied in a cascading fashion where rules are applied in a priority order over the result of a previous rule. This approach is known to become very difficult to maintain and scale as the rules for the information extraction tasks grow. The performance of this approach quickly degrades as the document sizes increase. This paper will survey the use of a relational model for constructing SQL-like statements to represent information extraction rules. These statements can then be translated into an execution plan where additional optimization is performed using well-known query optimization algorithms producing an optimized execution plan.

Table of Contents

Introduction

The volume of unstructured data produced each day by social media, user forums, and online journals continue to grow at rapid rate. This data is typically written in human natural languages and therefore lacks structure and definition required by computer systems in order to capture its semantic meaning. The field of information extraction seeks to develop methods to process this data in a systematic way to produce structured information that can be stored in conventional relational database for further analysis via relational querying and data mining techniques.

This paper will focus on relational methods and frameworks for addressing numerous challenges involved with information extraction through the application of relational database theory. This work includes a survey and review of the state-of-the-art of information extraction using relational methods that build on the success of the modern relational database system first pioneered by Astrahan et al (Astrahan et al 1976). This paper will also explore the current barriers faced by this approach and identify future work to address these challenges.

Background

The practice of extracting information from unstructured data with rules is not new or novel. The classic rules-based approach typically employs the use of regular expression grammars by experts to extract entities and relationships from text. Through a systematic application of these rules, information is transformed from unstructured to structured that can then be persisted in a relational database schema. Once the data is stored in a relational form, it can easily be analyzed through querying and data mining techniques.

The field of information extraction was formalized during several conferences held during the 1980's. Many of these conferences were organized by the DARPA (Defense Advanced Research Projects Agency) organization, which was looking to automate the processing of military intelligence reports by extracting information from these reports into specific structured fields. Their focus was around reducing the human labor required to process these reports with the intent of reducing the latency within the workflows that supported intelligence agencies. One of these major conferences was the Message Understanding

Conference (MUC) held in 1987 that invited academic and industry groups to define and formalize the information extraction as a field (Fagin et al, 2016).

The motivations for studying and improving the accuracy and efficiency of Information Extraction are rooted in the need for organizations to process the large volume of unstructured text that is produced daily in this information age. The ability for an organization to gather insights by mining information from social media feeds, online publications, blogs, and user forums can provide a competitive edge. Today, enterprise uses information extraction to support brand and reputation management (responding to social media comments), business intelligence (including sales leads and targeted marketing), automation of data entry, and monitoring business interests (such as within the area of pharmacovigilance monitoring for adverse effects of medication). All these areas get a considerable amount of attention and investment due to its impact to expanding revenue opportunities as well as improving efficiency (Chiticariu et al, 2010).

Information Extraction (IE) can be broken down into various tasks. These tasks include the detection of Entities, Relations, Events, Sentiments, Co-references, and Table extraction. These tasks can be carried out using various methods. For Entity extraction, a common approach is to use a construct called a Dictionary, which represents the terms of interest for a given domain. Dictionaries are typically used by pattern matchers that scan through a document and marks (or annotates) the span (or offset location) for terms matched within the dictionary. The resulting spans represent the entities extracted from the document. Another method for extracting entities involves the use of rules. These rules are typically represented as regular expression grammars. These expressions may look for certain patterns of characters such as capitalized words that may indicate that a proper noun, such as the name of a person, is being referenced. *Figure 1* demonstrates the extraction of two person names, ‘Bob Scott’ and ‘Holly Smith’ from a passage of text. The method for extracting these names could have been dictionary based that enumerates person names or a regular expression that looks for capitalized words.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin, in sagittis facilisis, volutpat dapibus, ultrices sit amet, sem , volutpat dapibus, ultrices sit amet, sem **Tomorrow, we will meet Bob Scott, Holly Smith and** amet lt arcu tincidunt orci. Pellentesque justo tellus , scelerisque quis, facilisis nunc volutpat enim, quis viverra lacus nulla sit lectus. Curabitur cursus tincidunt orci. Pellentesque justo tellus , scelerisque quis, facilisis quis, interdum non, ante. Suspendisse

Desired output is to find:

Person: Bob Scott
Person Holly Smith

Figure 1

Once entities spans have been identified within a document, they can be used to support the Relationship extraction task. Relationships describe some semantic meaning between two entities. These relationships can take various forms. The relationship can be taxonomic, such as a hierarchical 'IS-A' relationship between entities. For example: 'Cancer' IS-A 'Disease'. The relationships can be ontological, such as 'Advil' treats 'Headaches'. These relationships can also be syntactic, such as the relationship between a word within a sentence and it's corresponding part-of-speech (noun, verb, adjective, etc) or semantic role. Lastly, relationships can represent referential binding between words such as subject and objects or noun and pronouns. These tasks for extracting relationships may be represented using rules that enumerate the relationships of interests and these rules can be represented as a collection of regular expressions, annotation patterns, or other relationship graphs.

Review of Cascading Grammar Methods

Most modern programming languages today provide support for regular expressions. The availability of regular grammars makes them a popular choice for programmers to extract information from text. Regular expressions are typically interpreted by a Finite-State Transducer (FST) that accepts input as a sequence of characters and produces an output based on the transitions between pre-defined states within the FST. *Figure 2* illustrates an example of a finite-state machine (FS) that accepts characters (in blue) as input and produces characters (in green) as output. Regular expressions can be compiled into an FS machine that is interpreted by a FST at runtime to process input text and produce output (Boguraev et al, 2004).

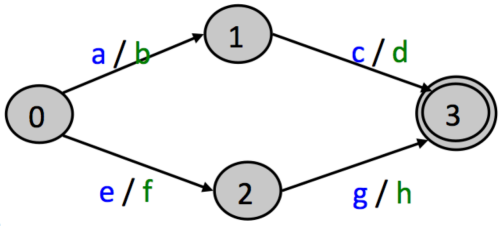


Figure 2

Boguraev et al assert that the finite-state matching and transduction over tokens and annotations provides flexibility and convenience through rapid configuration of custom analysis engines that can be layered within a pipelined architecture of interoperable components based on a formal language description (regular expression syntax) that sits on top of a separate FS abstraction which serves as the FST interpretation engine (Boguraev et al, 2004). Boguraev et al are the first to propose a finite-state cascading approach for supporting patterns of annotations in addition to tokens enabling the ability to layer regular grammar expressions on top of prior expressions. This approach addressed known limitations where multiple sub-expressions were composed into a single expression resulting in an exponential growth of total number of expressions required to cover all combinations of rule segments. For example, identifying a mailing address from documents may take on different forms. Ideally, this problem can be decomposed into smaller extraction tasks that scan for person names and then look for cities and states referenced within the surrounding text of the person name. This task is further complicated by the occurrence of intermediate tokens that represent a phone number or an email address. Rather than having to enumerate expressions for every combination of these tokens and allowing for the existence of certain intermediate tokens. Boguraev et al propose a method for decomposing such expressions into smaller parts by allowing for separate layered expressions that can detect atomic segments of the tasks. *Figure 3* is an example of a closing signature commonly found within an email that contains person names, cities, states, email, street address, and phone numbers. These parts can be extracted by atomic expressions, which in turn can be referenced by a series of cascading expressions. This approach is an example of a layered abstraction that helps reduce the complexity for describing the rules of a given extraction task. *Figure 4* shows an example of how these cascading expressions can be used to build upon previous expressions to produce a higher-level expression.

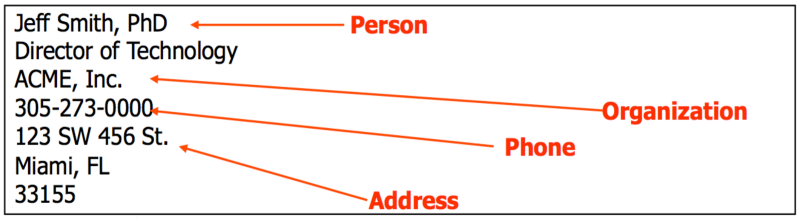


Figure 3, example of an email signature

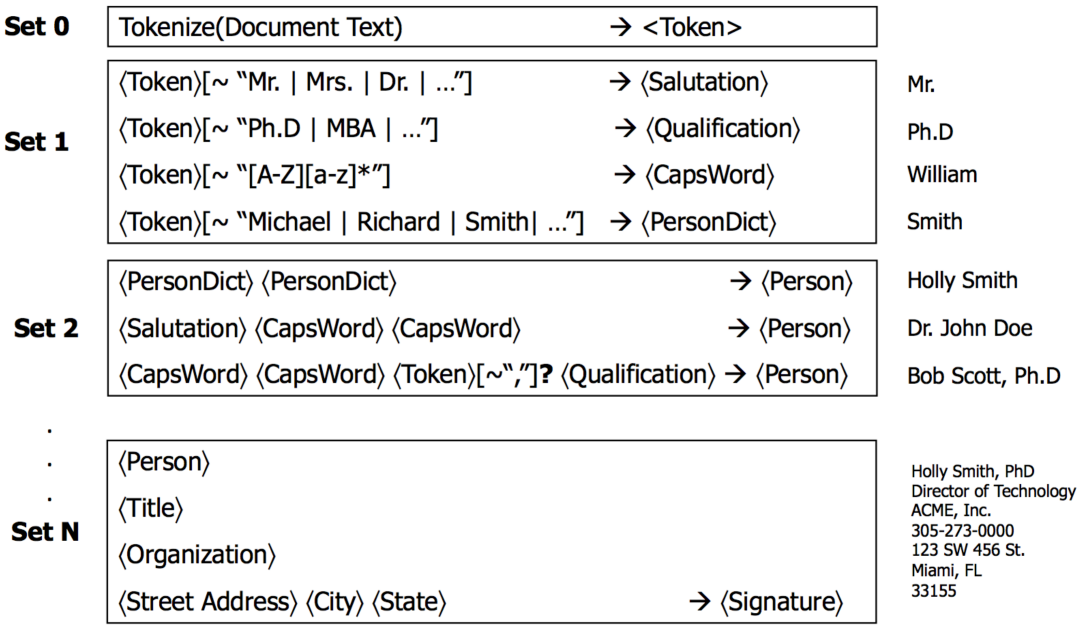


Figure 4, example of cascading grammar for extracting signature

The cascading approach also addressed issues when multiple expressions matched on overlapping spans producing ambiguity that could only be resolved by assigning priorities to the rules. Unfortunately this approach is very brittle since priorities are fixed and therefore may not always reflect the correct resolution for a given extraction task (Yunyao et al, 2011).

Another important aspect contribution to the grammar expression method for information extraction was the definition of a common specification for representing the rules. Appelt et al, presented the Common Pattern Specification Language (CPSL), which today is considered a de facto standard (Appelt et al 1998). This specification helped provide a uniform language for expressing these regular expressions that was independent of the underlying interpretation engine allowing for portability of these patterns across various frameworks. This was an important contribution and would prompt the creation of multiple frameworks, tools, and architectures for representing text analysis pipelines that leveraged CPSL-based annotators that could be pipelined to perform complex tasks. Many of these frameworks and architectures have been developed to help address the complexity of managing and executing large collections of rules for large data sets. Examples of these frameworks include the Java Annotation Pattern Engine (JAPE)

(Cunningham 1999), General Architecture for Text Engineering (GATE) (Cunningham 2002), and Unstructured Information Management Architecture (UIMA) (Ferrucci et al 2004).

A major drawback with using regular expression grammar approach for information extraction is quickly encountered when scaling to large data sets and/or a large collection of rules (Reiss et al 2008). Reiss et al demonstrate that these scaling issues are inherent limitations of cascading grammar languages that can only be resolved through applying additional parallel computing but lack direct optimization of its methods.

Another issue with the cascading grammars approach is observed when there exists ambiguity caused by variations in text that can produce overlapping results by several expressions. Although applying a priority to the rules can mitigate some issues, this is often not sufficient for more complex extraction patterns. The following example helps illustrate the ambiguity issue caused by overlapping grammar expressions.

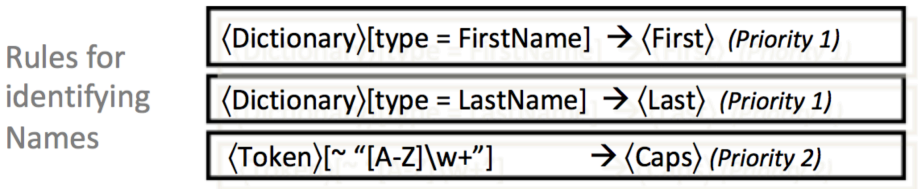


Figure 5, example rules for extracting person names

When applying the extraction rules shown in *figure 5* to the passage of text shown in *figure 1,* it produces multiple spans (annotations) for the tokens *"Tomorrow", "Bob", "Scott", "Holly", and "Smith"*. Some of these tokens received multiple overlapping spans such as "*Bob*", "*Scott*", and "*Holly*", which were identified by the *FirstName* rule and the *Cap* (or Capitalized name) rule. The token "*Scott*" was identified by all three rules including *FirstName* and *LastName*. *Figure 7* shows the resolution to the overlap ambiguity after applying the rule priority shown for each rule in *figure 5*. This resolution produces an incorrect result that assigns "*Scott*" as a *FirstName* instead of a *LastName*.

Figure 6, result of applying rules for extracting person names

Caps	First	Last
Tomorrow	Bob Scott Holly	Smith

Figure 7, shows the incorrect assignment of Scott to First Name

Due to the cascading dependencies of these rules, this incorrect assignment propagates through the downstream rules ultimately producing incorrect results. This is demonstrated when attempting to apply expressions that identify a *Person.* *Figure 8* illustrates an example of three rules that depend on the result of the previous rule set in order to identify a *Person.* These rules are not looking for patterns of tokens or characters but instead looking for patterns of previously labeled spans that trigger the production of a Person span.

⟨First⟩ ⟨Last⟩ → ⟨Person⟩
⟨First⟩ ⟨Caps⟩ → ⟨Person⟩
⟨First⟩ → ⟨Person⟩

Figure 8, rules for extracting Person from First Name, Last Name, and Cap spans.

When these rules are applied over the previous result, it incorrectly produces three persons as follows: *"Bob", "Scott"*, and "*Holly Smith*". The identification of "*Scott*" as a person was caused by the incorrect resolution to the ambiguity that "*Scott*" can sometimes refer to a first name and sometimes can refer to a last name. Because the first set of rules matched Scott as both, the priority of these rules was applied to ultimately assign Scott as a first name. This incorrect assignment caused the rule <First> to detect *Bob* and *Scott* and then <First> <Last> to detect *Holly Smith* producing a total of three persons instead of one.

Person
Bob
Scott
Holly Smith

Figure 9, Persons identified by cascading rules

This simple example helps demonstrate the issue known as Lossy Sequencing encountered when applying a cascading grammar approach where overlapping spans are dropped at a lower layer and therefore produces erroneous results at a later layer of rules. As the number of rules and cascading layers grows, the cost of maintenance increases dramatically resulting in a very brittle system where regression can be introduced even for very minor changes to these rules. Due to the cascading nature of these expressions and their interdependencies, changes made in one expression will also require a cascade of changes to avoid breaking rules (Chiticariu

et al, 2018). This growth of complexity is also observed in the degradation of system performance when executing large collections of rules over large documents. Reiss et al demonstrates that scaling issues are inherent limitations of cascading grammar languages (Reiss et al, 2008). Chiticariu et al present a thorough comparative assessment of cascading grammar performance across different CPSL-based (Appelt et al, 1998) frameworks such as JAPE (Cunningham et al, 1999) and GATE (Cunningham et al, 2000). The result of this assessment demonstrated the exponential growth in processing time as the complexity of rules and document sizes grow (Chiticariu et al, 2010).

Review of Relational Methods

An alternative method for information extraction proposed by Reiss (Reiss et al 2008) and further developed by Krishnamurthy (Krishnamurthy et al 2009), applied the relational database model (Astrahan, 1976) by leveraging relational algebra for representing information extraction rules and defined an extension to SQL called Annotation Query Language (AQL). This approach allows information extraction to benefit from years of database research in the areas of query optimization and execution strategies. AQL supports relational operators such as projection, selection, join operations (among others) and further extends it with special extraction operators over word-tokens allowing for dictionary-based and pattern-based matching. This approach also defines an extension to the relational data model by exposing an entity type called a "Span" that is represented as offsets <*begin, end*> within a document. Documents are represented as a relational table of tuples that have been previously tokenized (or chunked) into sentences. Each sentence is represented by a tuple with the *begin* and *end* offsets marking the start and ending position of each sentence relative to the whole document.

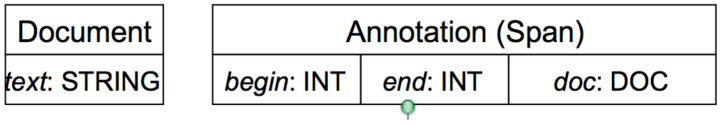


Figure 10, extended relational data model for information extraction

Information extraction occurs through the process of defining relational queries that represent annotations as relational views over the initial Document table produced by tokenizing the document into sentences. Through the use of "SELECT-FROM-WHERE" statements, information is extracted from unstructured form to a relational schema representation. The result

of these statements is placed in a new View that can be selected and projected to construct additional Views. These statements can be compiled into relational algebra where query-style optimization is performed. Lastly, the optimized algebra is assembled into an execution plan that is further optimized based on a cost-model. Figure 11 depicts the process of compiling and optimizing AQL into an execution plan.

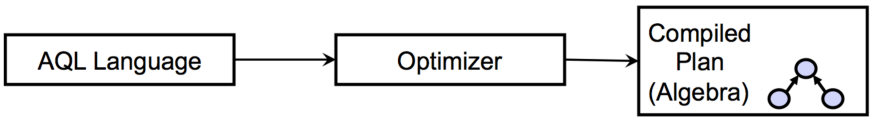


Figure 11, AQL Compilation process

Recent work by Fagin et al (Fagin et al 2016) formalizes this AQL relational algebra and language proposed by Reiss and Krishanmurthy in a theoretical framework called "Document Spanners" and proves that the expressivity of these relational languages is expressively equivalent to regular grammar languages. Further work by Fagin et al demonstrates how additional techniques such as "consolidation" of overlapping spans further enhances the expressivity of the framework (Fagin et al 2013). Fagin asserts that the balance between simplicity and expressivity of an IE language is crucial to its adoption in the real world. The language must have straightforward semantics so that it is easy to understand and debug but must be minimally expressive in order to address complex extraction tasks. Freydenberger et al build upon Fagin's work by comparing the expressive power of the Document Spanner framework to other extraction models including extended regular expressions (Freydenberger et al, 2017).

AQL statements can be aggregated together using SQL-like JOIN operations that combine the tuples into a new view that where the JOIN predicate condition evaluates to true. Aggregation is typically performed over previous views that extract additional fields (attributes) using a Dictionary or a Regular Expression pattern. The statement shown in *figure 12* below serves as an example of a rule for extracting capitalized words (i.e. names) from the document. The statement begins with a "create view" indicating that a new view will be produced within the relational model followed by the "extract regex" which indicates the use of a regular expression that will be applied across the tuples in the Document table. The "as name" defines the name for the field (or attribute) name that will be used to hold the matched text by the regular expression.

Applying this rule (*figure 12*) to the previous example in *figure 6* produces a new table view called *Caps* that contains tuples with the name values of *"Tomorrow", "Bob", "Scott", "Holly"*, and *"Smith"*. This rule is equivalent to the <CAPS> rule example shown in figure 5.

```
create view Caps as
extract regex /[A-Z](\w|-)+/ on D.text as name
from Document D
```

Figure 12, example of an AQL statement for extracting capitalized words from the Document.

Caps View:

name
Tomorrow
Bob
Scott
Holly
Smith

Figure 13, result of applying AQL rule in figure 12 to figure 6 text.

Another common extraction uses the "extract dictionary" operation that scans each tuple within the Document table looking for tokens that match words defined within the dictionary. The dictionary construct is a simple list of terms that can be enumerated within a plain text file or defined inline within AQL using a "create dictionary" statement followed by a comma separated list of string. The following example shows the AQL statement for extracting LastName from the Document table.

Dictionary

LastNamesDict
...
Rodriguez
Scott
Smith
Stevenson
...

```
create view Last as
extract dictionary LastDict on D.text as name
from Document D
```

Applying the above AQL to the text in *figure 6* produces a new view called *Last* with two tuples including the names "*Scott*" and "*Smith*". The same approach can be used to extract first names using a dictionary that contains a list of first names. These statements collectively provide

the atomic annotations for extracting first and last names annotations. Next these resulting views need to be joined together to produce the Persons. Figure 14 shows an example AQL program that extracts the Persons by combining the views produced by the First, Last, and Caps statements.

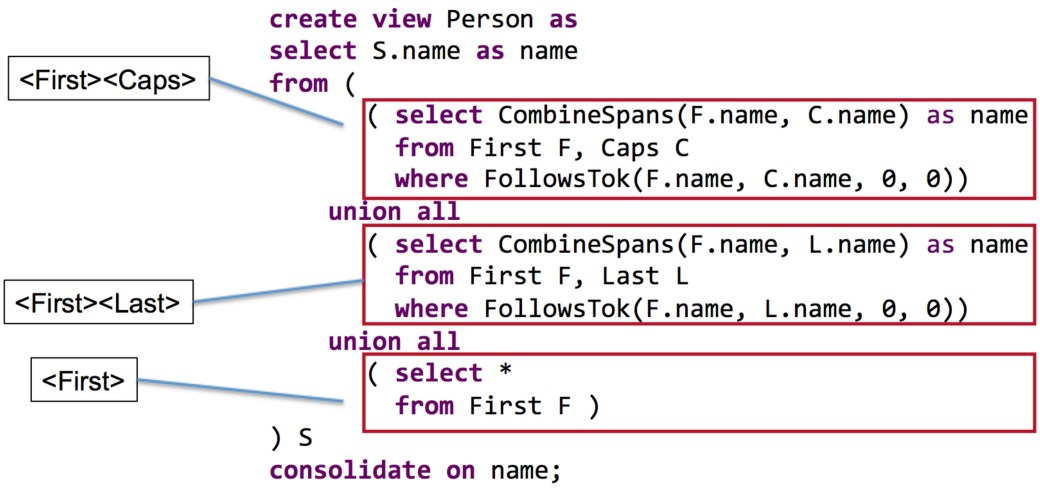
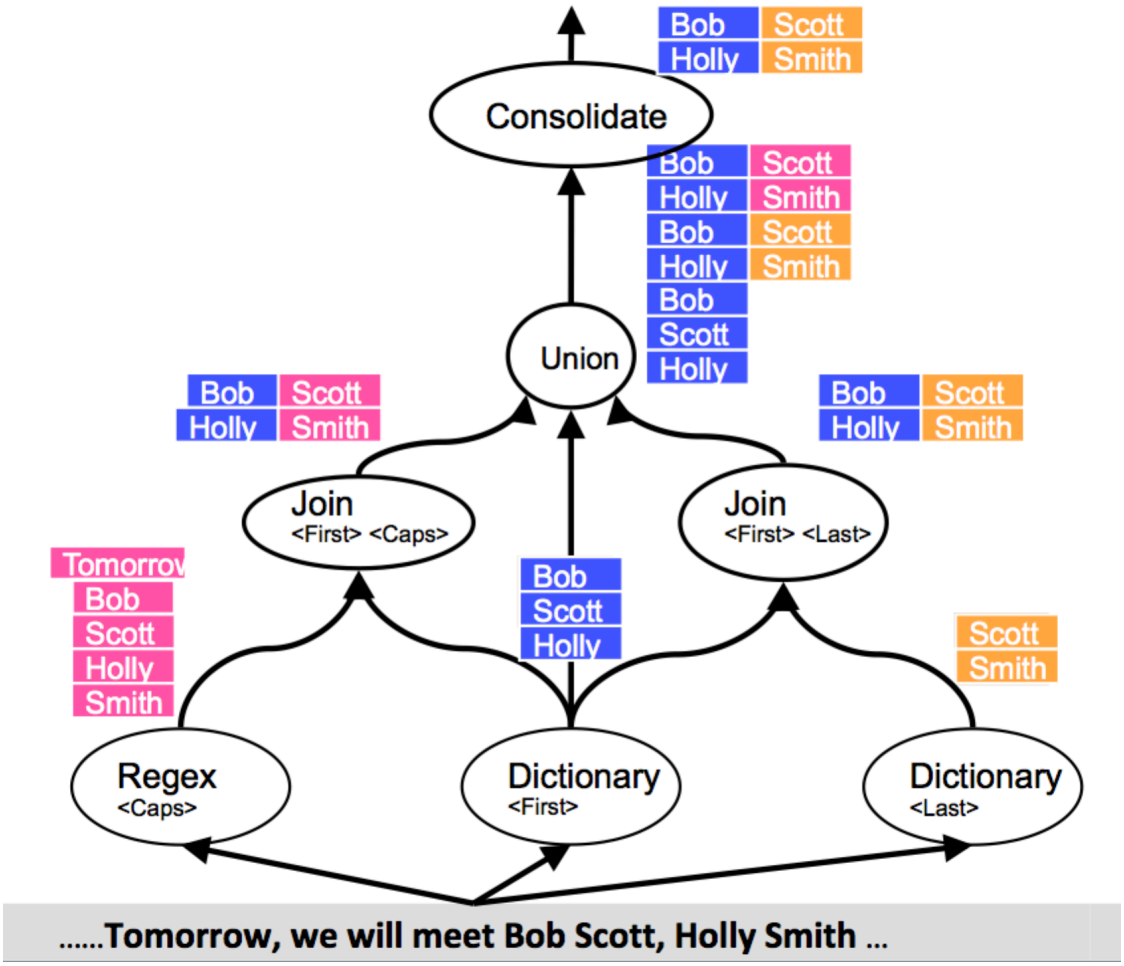


Figure 14, AQL statement that produces Person from First, Last, and Caps views. (Chiticariu et al, 2010)

The approach taken involves first performing a select query that joins <First> with <Caps>, <First> with <Last>, and <First> followed by the union of the resulting tuples. The operator "CombineSpans" is used to concatenate the span offsets between to Spans. This produces a new Span where the beginning offset is the smaller of the two and the ending offset is the greater of the two spans. This results in a longer span that covers both provided span. As an example, combining the first span in a document with the very last span would produce a new span that holds a string containing the entire document, where the first span starts at offset 0 and the second span ends at the length-1 of the document. Another operation used is "FollowsTok" that serves as a predicate within the select operation that requires that one span must occur after another span. This enforces a specific order. *Figure 14* includes two uses of "FollowsTok". The first use occurs where First Name must precede Cap spans. The second use requires First Name precedes Last Name. The ability to enforce a specific order helps prevent invalid (or spurious) tuples with patterns that are not recognized as Persons. This also serves as a performance optimization since it reduces the result set of tuples produced by each SELECT operation. Finally, the use of "consolidate" is used to combine final tuples produced in the Person view

using the name field as the consolidation field. The following illustration shows the step-by-step result when applying the AQL for extracting FirstName, LastName, Caps, and lastly Person views.



(Chiticariu et al, 2010)

This relational method for information extraction allows for overlapping spans to flow through to a higher-order rule (or layer) where it can then be disambiguated within the context of a particular rule objective. In this example, the consolidate operation is applied to all the names resulting in a correct final result that includes two persons: "*Bob Scott*" and "*Holly Smith*".

In addition to these basic operators, AQL also features semantic and syntactic role labeling (SRL) facility, expanding the expressivity of the language by allowing for additional qualifiers within conditional JOIN predicates (Chiticariu et al, 2010). This helps further reduce

ambiguity caused by words that can have multiple meanings or can be used in different parts-of-speech.

Optimizations and Performance of Relational Methods

The relational-based information extraction rules appear to have a natural performance advantage over a corresponding regular expression rule. In many cases a hand-tuned regular expression rule executed an order-of-magnitude slower than an un-optimized relational rule. Because regular expressions operate over characters and relational rules provide semantics for operating over whole tokens (such as terms defined by a dictionary table) or other views, the search space is reduced for a relational rule and therefore results in a faster processing time (Krishnamurthy et al 2009).

The primary advantage for a relational-rule is the ability to optimize an execution plan that represents a collection of rules at compile-time. Using relational query optimization theory, a search for the optimal execution plan is identified at compile time resulting in dramatic improvement in runtime performance (Krishnamurthy et al 2009). Krishnamurthy et al found an additional order-of-magnitude improvement when comparing optimized relational rules to un-optimized relational rules. The optimization techniques include cost-based optimization of operations such as with the "conditional evaluation join" (CEjoin) operator by avoiding evaluation of the right-hand argument when left-hand produces no tuples (Krishnamurthy et al 2009). An example of this optimization occurs when the evaluation of one side of the JOIN fails to produce any tuples. This allows for skipping the other side of the JOIN to avoid needless computation. For example, in *figure 15* if the left-hand side Dictionary does not produce any result, then there is no need to apply the expensive Regex (right-hand-side) of the Dictionary.

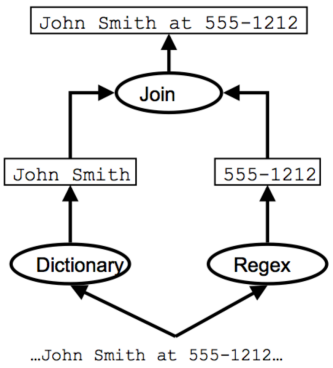
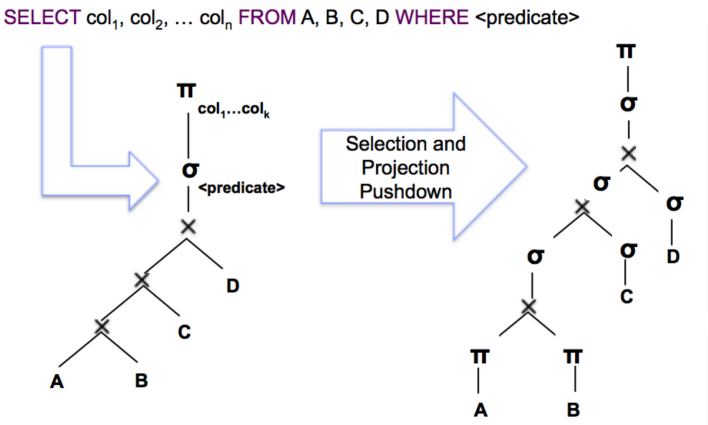


Figure 15

Another optimization technique involves pushing down selection and projection operators within the AQL operator graph. This approach works by reducing the intermediate

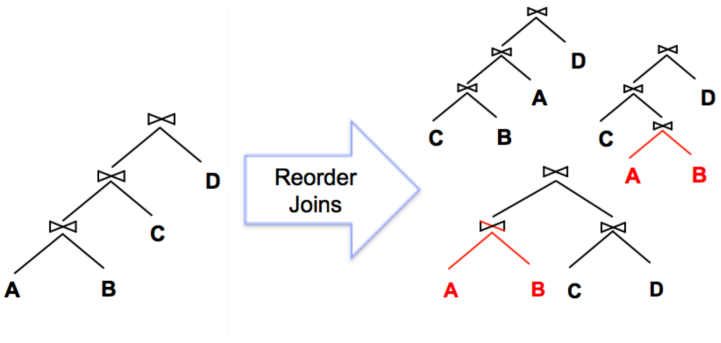
results produced at earlier stages of the operator graph and therefore reducing the computation required when evaluating predicates within Join operations (Reiss et al, 2008). The following operator graph shows an example of pushing down the projection and selection.



(Chiticariu et al, 2010)

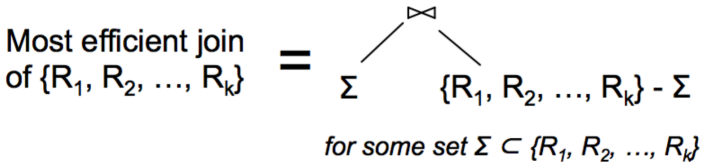
Because the projection applied to relation A and B is performed at an earlier point, the resulting number of fields (columns) that is carried forward is reduced. Similarly, by pushing down select operators, the number of tuples is reduced.

Another important optimization technique involves JOIN reordering. Because JOIN operations are commutative, it is possible to rearrange the order of the joins. The total number of unique arrangements that must be considered grows exponentially. A naïve approach would consider each possible join order evaluated using a cost-model to choose the lowest estimated cost. This however results in unnecessary overhead and wasted work since the cost of a sub-plan will be recomputed multiples times. Dynamic Programming algorithms are effective at optimally evaluating problems where recomputation can be avoided using a cache to store previously computed sub-plans. The following illustration shows multiple rearrangement options produced by reordering the joins from an initial plan. An example of the reoccurring sub-plans is shown in red. If the A ⋈ B sub-plan is evaluated once, it can be cached and used later to avoid duplicating the calculation.

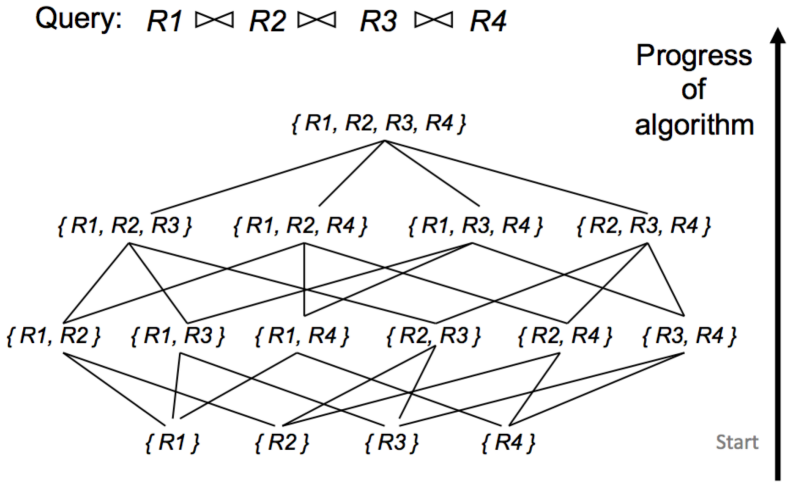


(Chiticariu et al, 2010)

Selinger et al proposed a dynamic programming algorithm for optimally evaluating to most efficient join ordering (Selinger et al, 1979). Selinger was a pioneering member of the IBM team that developed System R, considered the first relational database. Selinger et al presented the following recursive formulation for considering left-deep plans:

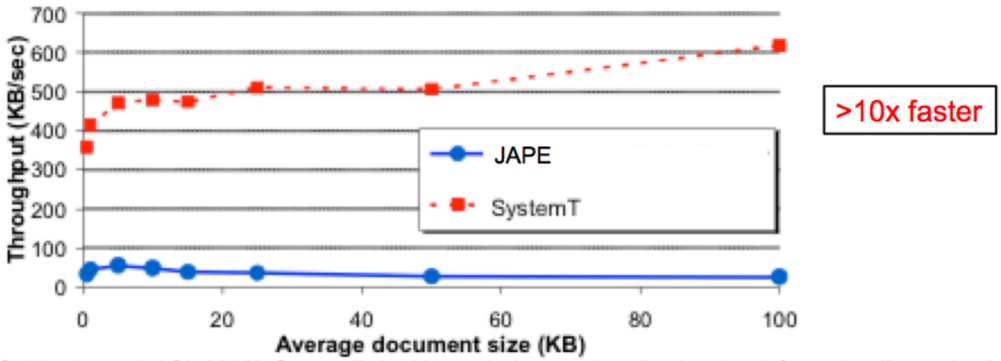


The following example shows the computation applied using Selinger's algorithm for R1 ⋈ R2 ⋈ R3 ⋈ R4:



In addition to query optimization techniques, other techniques focus on optimizing extraction operators such as "extract dictionary" and "extract regex" such as the examples shown earlier in *figure 14*. The first method is called Shared Dictionary Matching. This method helps combine all the dictionaries used by the AQL program into a single dictionary. This shared dictionary allowed for removing redundant dictionary entries and avoids duplicate copies of dictionaries held in memory (Reiss et al, 2008). Dictionaries can be represented as a tree of characters with terms located at the leaves allowing for linearithmic search times for matching dictionary tokens. By combining multiple dictionaries into one large dictionary, there is only one tree structured required that preserves its depth and only increases the number of leaves (Chiticariu et al, 2010).

The other optimization focuses on the "extract regex" operator. Reiss et al assert that some regular expressions can benefit from a simpler but less expressive regular expression matcher that does not allow for group capturing, look-back, or peek-ahead but has significantly better performance. By applying Regular Expression Strength Reduction, the evaluation of the regular expression can be performed faster because it does not require certain extended capabilities of full strength regular expressions (Reiss et al, 2008). This novel approach proves to be pivotal in speeding up the evaluation of AQL plans at runtime. Chiticariu et al demonstrate the performance evaluation of AQL compared to JAPE (Cunningham et al, 1999), which uses cascading grammar method. Their results show a 10x improvement in throughput when comparing to a hand-tuned (i.e. optimized) equivalent regular expressions.



[Chiticariu et al. ACL 2010]: SystemT: An Algebraic Approach to Declarative Information Extraction.]

Challenges and Issues

A critical aspect of information extraction is the accuracy of the information produced by the rules framework. Liu et al proposes a semi-automatic rule refinement technique that uses database tuple provenance to improve the precision while preserving recall accuracy. The rule refinements eliminate false positives using back-tracking over tuple provenance to correct or rewrite a rule (Liu et al 2010). Unfortunately this approach can only improve false-positives, which impact precision accuracy, but no method is presented for addressing false negatives or low recall accuracy.

Another challenge with the relational approach is the learning curve that must be overcome to educate programmers to build these programs. Most programmers are familiar with constructing and using regular expressions to extract strings from text. Learning a new paradigm and language such as AQL, creates a barrier to adoption and may explain why this approach has not become widely adopted by industry or academia.

AQL programs tend to be compiled as modules that include both the executable statements (i.e. the AQL program) as well as the data resources (such as dictionaries and tables) used by these programs resulting in an anti-pattern that makes it difficult to dynamically configure different dictionaries at runtime. This means that multiple versions of the same AQL program needs to be compiled and loaded for each domain that does not share the same dictionary resources.

Lastly, the dependency on a Database System introduces additional overhead and complexity to the deployment and maintenance of programs. The popularity of cloud-native application and micro-services has influenced the building of smaller lightweight programs that can be constantly started and stopped based on volume. This relational system for information extraction would require a redesign to support a more service-based approach that can allow for this dynamism.

Future Work and Conclusion

The relational methods for information extraction presented in this paper are based on a theoretical foundation that benefits from years of research in the areas of query optimization and execution. This approach shows great promise in its ability to reduce the complexity of large-scale information extraction systems while providing faster computation of IE tasks without the

maintenance issues encountered with cascading grammar alternatives. Furthermore, the relational methods demonstrate advantages in dealing with ambiguity and therefore increasing the relative accuracy of IE tasks.

The relational method for IE remains an active area of research. Additional work is needed to explore dynamic optimization by taking into account actual costs, instead of just estimated costs, at runtime. This improvement has the potential to provide an additional performance gains. Another area that can be improved is the development of visualization tools and intuitive editors for allowing non-technical subject matter experts to construct AQL programs without requiring formal education in computer science and AQL programming. Finally, the inclusion of additional operators such as String concatenation functions that are not available today, but often required when attempting to construct new view with attribute values that are composed of non-contiguous spans of text.

References

Appelt, D. E., & Onyshkevych, B. (1998, October). The common pattern specification language. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998* (pp. 23-30). Association for Computational Linguistics.

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., ... & Putzolu, G. R. (1976). System R: relational approach to database management. *ACM Transactions on Database Systems (TODS)*, *1*(2), 97-137.

Boguraev, B. K. (2004). Annotation-based finite state processing in a large-scale NLP architecture. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP 2003*, *260*, 61.

Chiticariu, L., Li, Y., Raghavan, S., & Reiss, F. R. (2010, June). Enterprise information extraction: recent developments and open challenges. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 1257-1258). ACM.

Chiticariu, L., Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F. R., & Vaithyanathan, S. (2010, July). SystemT: an algebraic approach to declarative information extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 128-137). Association for Computational Linguistics.

Cunningham, H., Maynard, D., & Tablan, V. (1999). JAPE: a Java annotation patterns engine.

Cunningham, H. (2002). GATE, a general architecture for text engineering. *Computers and the Humanities*, *36*(2), 223-254.

Fagin, R., Kimelfeld, B., Reiss, F., & Vansummeren, S. (2016). A relational framework for information extraction. *ACM SIGMOD Record*, *44*(4), 5-16.

Fagin, R., Kimelfeld, B., Reiss, F., & Vansummeren, S. (2013, June). Spanners: a formal framework for information extraction. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems* (pp. 37-48). ACM.

Ferrucci, D., & Lally, A. (2004). UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, *10*(3-4), 327-348.

Freydenberger, D. D., & Holldack, M. (2017). Document spanners: From expressive power to decision problems. Theory of Computing Systems, 1-45.

Jayram, T. S., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., & Zhu, H. (2006). Avatar information extraction system. *IEEE Data Eng. Bull.*, *29*(1), 40-48.

Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F., Vaithyanathan, S., & Zhu, H. (2009). SystemT: a system for declarative information extraction. *ACM SIGMOD Record*, *37*(4), 7-13.

Li, Y., Reiss, F. R., & Chiticariu, L. (2011, June). SystemT: A declarative information extraction system. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations* (pp. 109-114). Association for Computational Linguistics.

Liu, B., Chiticariu, L., Chu, V., Jagadish, H. V., & Reiss, F. R. (2010). Automatic rule refinement for information extraction. *Proceedings of the VLDB Endowment*, *3*(1-2), 588-597.

Neff, M. S., Byrd, R. J., & Boguraev, B. K. (2004). The Talent system: T EXTRACT architecture and data model. *Natural Language Engineering*, *10*(3-4), 307-326.

Reiss, F., Raghavan, S., Krishnamurthy, R., Zhu, H., & Vaithyanathan, S. (2008, April). An algebraic approach to rule-based information extraction. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on* (pp. 933-942). IEEE.

Shen, W., Doan, A., Naughton, J. F., & Ramakrishnan, R. (2007, September). Declarative information extraction using datalog with embedded extraction predicates. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 1033-1044). VLDB Endowment.