A Framework for Specifying Explicit Bias for Revision of Approximate Information Extraction Rules

Ronen Feldman
Yair Liberzon

Binyamin Rosenfeld
Jonathan Schler
Instinct Software Ltd.
Petah Tikva, Israel
Tel. 972 3 92 444 64

Jonathan Stoppi

ronen@instinct-soft.com
yair@instinct-soft.com
grur@instinct-soft.com
jonathan@instinct-soft.com
stops@instinct-soft.com

ABSTRACT

Information extraction is one of the most important techniques used in Text Mining. One of the main problems in building information extraction (IE) systems is that the knowledge elicited from domain experts tends to be only approximately correct. In addition, the knowledge acquisition phase for building IE rules usually takes a tremendous amount of time on the part of the expert and of the linguist creating the rules. We therefore need an effective means of revising our IE rules whenever we discover such an inaccuracy. The *IE revision problem* is how best to go about revising a deficient IE rules using information contained in examples that expose inaccuracies. The revision process is very sensitive to implicit and explicit biases encoded in the specific revision algorithm employed. In a sense, each revision algorithm must provide two forms of biases: bias as to the place of the revision and bias as to the type of the revision that should be performed. In this paper we present a framework for writing approximate IE rules that are provided with explicit bias. The proposed framework can be used by many existing revision algorithms. The purpose of the revision bias framework is to allow the user to declare his own bias in a simple and structured way, i.e. to express the conditions placed on the domain knowledge for a given revision operator to be applied. This language extends and generalizes the work reported in [Feldman et. al. 1993]. It attacks the problem of writing IE rules from a novel perspective, one which enables a much faster development of IE systems.

Keywords

Text Mining, Theory Revision, Information Extraction, User Guided Revision.

KDD 2000, Boston, MA USA
© ACM 2000 1-58113-233-6/00/08 ...$5.00

1. INTRODUCTION

One of the main problems in building information extraction (IE) systems [1,3,5,6,22,23] is that the knowledge elicited from domain experts tends to be only approximately correct. Although knowledge so obtained might make a good first approximation to the real world, it typically contains inaccuracies that are exposed when the model asserts a fact that does not agree with empirical observation. This paper proposes a means of automatically revising a set of IE rules - whenever we discover such an inaccuracy - using a predefined bias scheme.

Informally, the IE rule revision problem is about how best to go about revising a deficient set of IE rules using information contained in examples that expose inaccuracies. In order to characterize the attributes that are desirable in a IE rules revision system, let us briefly examine the process by which IE rules are constructed.

Typically, a domain expert sits with a linguist and encodes his or her expert knowledge as a collection of IE rules. Sometimes more than one domain expert is involved, leading to possible inconsistencies in the rule base, although inconsistencies are not uncommon in large rule bases even if there is only one domain expert.

After the knowledge has been encoded, it is used to extract events from documents. These events are usually obtained over a long period of time as the IE system is put to use. As errors and inaccuracies are encountered, the rule base must be refined manually by the linguist. Some may be simple encoding errors, while others may represent deeper, conceptual, errors in the expert's understanding of the domain. IE revision systems are in some sense nothing more than biased concept learning systems. The bias comes in the form of an approximately correct concept description (rule base) which is then patched and brought into line with the provided examples. The intuition is that patching an approximately correct IE rule set will be substantially cheaper and more accurate than an IE rule set built from scratch.

The IE rules revision problem is related to the problem of propositional and relational theory revision. Among the propositional theory revision systems which can handle both specialization and generalization we can count EITHER[18], KBANN [24] and RAPTURE [17]. Among the relational theory revision systems we can count AUDRY [25], FORTE [21] and FOCL [16]. A common theme to all these systems is that they do not distinguish between aspects of the original knowledge base that are firmly held and those which are more conjectural.

An IE rules revision system should, in practice, replace this knowledge refinement process. A useful solution to the IE rules revision problem should integrate smoothly with the traditional approach to knowledge engineering. Thus, it is important that an IE rules revision system have certain qualities. First, the system should be *incremental* - meaning that it should operate on examples as they are obtained. IE rules revision systems which require examples be provided all at once do not fit in well with the traditional knowledge engineering process. Second, a system which reconfigures the rule base so that it no longer makes sense to the linguist lacks *referential transparency*: a useful system should largely preserve the structure of the original rule base so that, at any time, the rule engineer may examine and understand its internal structure. Third, a domain expert may know *a priori* which portions of the rule base should be trusted and which are more conjectural. Thus the changes undertaken by the system should reflect the expert's intuitive *confidence* in the individual components of the rule base. Fourth, the domain expert may have preferences as to which revision operators should be used for revision specific elements should they be flawed.

In this paper we introduce our scheme for providing explicit revision bias in the revision of flawed IE rules.

Other research on learning IE rules [1,3,5,6,22,23] has focused on inducing new IE rules based on examples rather than revising existing IE rules based on examples. In addition, we use a more sophisticated extraction language, which is more suitable for handling real world tasks and achieving high precision and recall.

The rest of the paper is organized as follows: in Section 2 we define the basic concepts and establish the terminology used throughout the paper. In Section 3 we motivate the need for a special language for defining explicit revision bias. In Section 4 we describe our bias scheme and present examples. In Section 5 we provide experimental evaluation, and finally, in Section 6 we present our conclusions and cite the major contribution of this paper, outlining future research directions to extend this work.

2. FOUNDATIONS

In this section, we define and introduce the basic notions used throughout this paper.

2.1 Pattern Matching Elements (PME)

The basic entity in a rule base is a Pattern Matching Element, which is one of the following cases:

- String - e.g., "*merger*"
- Word class element: a phrase that is a member of a predefined set of phrases that share a common semantic meaning e.g., `WCCountries` (a word class that contains the names of all the countries in the world)
- Scanner feature (basic characteristic of a token) e.g., `@Capital` or `@HtmlTag`
- Compound feature: a phrase comprising several basic feature -*e.g.* `Match(@Capital & WCCountries)` will match a phrase that belongs both to the word class `WCCountries` and start with a capital letter.
- Part of Speech tag - e.g., noun or adj.
- Predicate Call - e.g., `Company(C)`
- Skip Pattern: a pattern that enables the system to skip up to a certain number of tokens until it reaches an instance of a predicate - provided that it does not encounter a phrase that satisfies the Fail condition. For instance, `skip(WCMerger, SkipFail, 20)` tells the system to skip up to 20 tokens until it reaches a member of the word class `WCMerger`, provided it did not encounter an end of sentence or HTML tag along the way (based on the current definition of `SkipFail`, which may be changed by the user).

2.2 Constraints

Constraints do not try matching text fragments to patterns, but carry out on-the-fly Boolean checks for specific attributes. In addition, they can do these checks on any bit of text: not just fragments in the source document but also on results thrown up during processing that never appear in the output. The marker for a Constraint is the word verify, followed by brackets containing a specific function, which governs what it is it is checking for.

For instance
`verify ( StartNotInPredicate ( c , @PersonName ) )`

makes sure that no prefix of the string assigned to variable `c` must match with the predicate `PersonName`.

2.3 IE Rule Bases

Throughout this paper we will view a rule base as a logic program. Thus, a *rule base*, Γ, is a conjunction of definite clauses C_i: $H_i \leftarrow B_i$ where C_i is a clause label, Hi (called the *head*) is a literal and $B_i = \{B_{i1}\ B_{i2}....\} = P_i \cup N_i$ (called the *body*) is a set of literals, where $P_i = \{p_{ij}\}$ is a set of Pattern Matching Elements and $N_i = \{n_{ij}\}$ is a set of constraints operating on P_i. The clause clause C_i: $H_i \leftarrow B_i$ represents the assertion that H_i is implied by the conjunction of the literals in P_i while satisfying all the constraints in N_i. The rules are written in a language called DIAL (Declarative Information Analysis Language). DIAL is a language designed specifically for writing IE rules. The complete syntax of DIAL is beyond the scope of this paper.

To follow is an example of a DIAL rule, which will be used throughout the paper:

```
FMergerCCM(C1, C2) :-

Company(Comp1) OptCompanyDetails "and"
skip(Company(x), SkipFail, 10)
Company(Comp2) OptCompanyDetails
skip(WCMergerVerbs, SkipFailComp, 20)
WCMergerVerbs skip(WCMerger, SkipFail, 20)
WCMerger

verify(WholeNotInPredicate(Comp1,
@PersonName))

verify(WholeNotInPredicate(Comp2,
@PersonName))
```

```
@% @!
{ C1 = Comp1; C2 = Comp2} ;
```

This is one of ten rules that define the notion of `Merger` between two companies (where the names of the companies appear before the merger-related verb or noun). The rule looks for a company name (carried out by the predicate `Company`, which returns back the parameter `Comp1`) followed by an optional phrase describing the company, and then the word "*and*". The system then skips (within the same sentence, and while not encountering any phrase matching the predicate `SkipFail`) up to ten tokens until it finds another company, followed by an optional company description clause. The system then skips up to 20 tokens until it finds one of the phrases belonging to the word class `WCMergerVerbs`. (This may be something like "*approved*", "*made an announcement*" etc.) Finally, the system skips up to ten tokens until it finds a phrase belonging to the word class `WCMerger`. Finally, the rule also contains two constraints ensuring that the names of the companies are not names of people.

2.4 Examples

In addition to the rule base, we have some additional background knowledge, denoted *K*. *K* is a collection of facts and clauses defining some background predicates: we assume that *K* is correct and no revision is attempted to *K*.

An example, E, is a tuple <*S,P,I*> made of three parts: a string *S* which is a text fragment, a top-level predicate *P*, and a ground instance *I* of *P*. Let Γ be a rule base, we denote $\Gamma+K \Rightarrow S\ E$, if when we apply the predicate *P* on the string *S*, we get the instance *I*. For instance if *S* = "*AOL and Time Warner announced a merger*", and *P* = `FMergerCCM`, then *I* = `FMergerCCM`("*AOL*", "*Time Warner*"). We define a function Γ such that for an example E=<S,P,I>, Γ(E) = true if $\Gamma+K \Rightarrow_S E$ and Γ(E) = false otherwise.

2.5 Relevant Examples

The decision to add new structures rather than delete a flawed element is made on the basis of the examples affected by the revision of the flawed element. Thus, before we decide which revision operator to apply to the flawed element, we must determine which of the examples are relevant to this revision. If we were to use all the given examples when adding a new structure, the new structure would be equivalent to an alternative rule base that handles all the examples correctly. In other words, we would be using the inductive algorithm to build a new IE rule base from scratch - one that reflects none of the structure of the original IE rule base. So instead, we focus the inductive algorithm by finding an appropriate smaller set of relevant examples -ones that are directly affected by the revision of the flawed element. In this way, by using only the relevant examples, we reduce the processing time needed for the inductive algorithm, while retaining the original structure of the rule base.

With regard to a specific rule base element *e* we divide the relevant examples into two sets:

- *needed examples* (denoted *N*): these are examples for which *e* contributes to their correct classification (i.e, *e* must be part of Γ in order to get a correct classification of the example).
- *obstructive examples* (denoted *O*) : examples for which e is obstructive to the goal of achieving their correct classification (*i.e*, *e* must not be part of Γ in order to get a correct classification of the example). The computation of *N* and *O* is done in a similar way to the algorithm described in [8].

2.6 Incremental Theory Revision

We provide a skeleton of an incremental revision algorithm that processes examples one at a time, and when elements of a rule base become candidates for revision we perform an appropriate revision based on the information known at the time. The input to the algorithm is an initial flawed rule base and a set of pre-classified examples {Ei} which are used to refine the initial rule base. The algorithm produces as output a revised version of Γ, Γ' which handles correctly all given examples.

```
While there exists any misclassified example do:
    For E ∈ {Ei} do:
        Find S - the set of elements to be revised
        For e ∈ S do:
            N = the set of needed examples
            O = the set of obstructive examples
            Pick a revision operator Δ
            Using Δ revise e based on the example sets N and O.
        end do.
    end do.
end do.
```

Figure 1 - A skeleton for an incremental theory revision algorithm

3. THE NEED FOR AN EXPLICIT REVISION BIAS

The revision process of a flawed IE rule base can be viewed as a search in an hypothesis space for the most appropriate concept definition - where the hypothesis space consists of all the candidate IE rules for the definition of the target concept consistent with all known examples.

Consider, for example, a vastly simplified rule base which includes the following IE rules:

```
Company(C):- CapitalWords -> head
WCCompanySuffix -> suffix {C=head+suffix};
Company(C):- NP->head WCCompanyVerb
            { C = head };
//NP = Noun Phrase
WCCompanySuffix = Inc Ltd Gmbh Ag;
WCCompanyVerb = announced merged "took
over";
```

Suppose that we then determine independently that the string "*Microsoft Corp signed a contract with Excite*" contains the instance `Company` (“*Microsoft Corp*”). None of the above rules will enable us to reach this conclusion - an indication that the original knowledge base is deficient.

There are many solutions to this deficiency. To name a few (among many other possible solutions):

Solution 1
Create a new clause for the predicate `Company` :

```
   Company(C) :- CapitalWords -> head
                 “Corp”-> suffix
                       { C = head+suffix };
```

Solution 2
Generalize the first clause by dropping the literal `WCCompanySuffix`:

```
   Company(C) :- CapitalWords -> head
                       { C = head };
```

Solution 3
Generalize the second clause by adding the phrase “*signed*” to `WCCompanyVerb`

Solution 4
Generalize the first clause by adding the phrase “*corp*” to `WCCompanySuffix.`

How do we decide which revision is most appropriate? We might apply a syntactic criterion (*e.g.*, minimize number of changes to the original theory) to prefer the third or fourth solutions, since these require adding one phrase to a word class rather then adding a new clause or dropping a complete PME. Note that such syntactic methods implicitly assume that every knowledge base element is of equal importance. In many cases, these heuristics may lead to performing the wrong revision.

Suppose the domain expert was able to supply additional information reflecting his or her confidence in the second rule. We can exploit this information to prefer the third revision, since it entails changes to a rule, which is *a priori* less credible. Our claim is that such bias knowledge that controls the selection of a revision operator should be clearly elicited, so that it can be integrated into the revision process.

Using declarative biases allows us to distinguish clearly between control and data, and furthermore between the parts of the revision algorithm that need fixing and those that may be modified by means of parameters. Revision biases could therefore be defined as parameters that are shifted depending on the application area and depending on the part of the theory that needs to be revised. These would be based on two main types of criteria: syntactic and semantic.

Our purpose here is to show how the user’s capacity to express flexible biases can be extended and systematized through a revision bias language. The purpose of such a language is to allow the user to declare his own bias in a simple and structured way, that is to express the conditions placed on the domain knowledge for a given revision operator to be applied. This language extends and generalizes the work reported in [7,8] by considering a larger family of conditions.

In the absence of a bias scheme the system will use a predefined cost scheme (each revision operator has a cost associated with its application), and suggest revisions that have a minimal cost.

3.1 Typical Situations In Which An Explicit Bias Is Needed

In this section we provide typical examples of situations in which the expert writing the approximate rule base can provide specific biases that direct the system to perform the correct revisions. In these cases, providing an explicit bias is the most natural way to guiding the revision system toward the desired theory.

Predicate stubs: In some cases the user only wants to specify that a certain predicate exists without supplying any of its definitions. In the absence of any bias information, a naive revision would just delete all instances of this predicate from the clauses it appears in, since clearly they will all fail. To remedy this problem, the user specifies that the suitable revision operator for all instances of the predicate is to add new clauses to it. When a literal which is an instance of the predicate is a candidate for revision, we activate the inductive component in order to learn a definition for that predicate. The user can also specify the primitive predicates from which the definition should be constructed, and the inductive component will give priority to these primitives in the construction of the clauses.

Extraneous literals: some users prefer to add many literals to the body of a clause just in case they are needed. The user can then specify for such literals that the appropriate revision operator is deletion. In such case, even if there are some negative examples that might be misclassified due to the deletion of the literal, the literal will be deleted anyway, and the negative examples will be taken care of in another place in the theory. This bias eliminates the addition of new intermediate concepts to this clause.

Under constrained clauses: there are times when the user would like to provide a general skeleton to the definition of a predicate, where it is clear that some literals are missing from the clauses of the predicate. In such cases, the user can provide this extra information by specifying that the revision operator of choice should be refinement of the clause by adding new literals to its body. The user can also specify the primitive (or intermediate) literals that should be used by the inductive learner.

Climbing up and down a hierarchy: There are two situations in which a hierarchy can be used. First, a hierarchy of predicates can be specified directly as part of the rule base. The clauses specifying the hierarchy are of the form Class :- SubClass, indicating that SubClass is a kind of Class. Each class can be a sub-class of several other classes, which implies that hierarchy is a dag. When we want to generalize a literal, we climb up the hierarchy; to specialize it, we descend the hierarchy. The user can specify which direction should be attempted for the literal, and even the maximum number of levels allowed in traversing the hierarchy. The second type of hierarchy is a *type hierarchy*. Type information can be of real use when specifying constraints on the arguments of certain literals. Each argument is assigned a type, from a given type hierarchy. The user can then specify the preferred revisions at argument level for each literal. The bias attached to specific arguments is similar to that of the predicate hierarchy.

4. A FRAMEWORK FOR EXPLICIT BIAS

4.1 The Language

Each clause in the approximate rule base has the following form:

$$C:\{CF_H,Bias_H\}\ H \text{ :- } B_1:\{CF_{B1},Bias_{B1}\}.....B_n:\{CF_{Bn},Bias_{Bn}\}.$$

C is the label of the clause, H is the head of the clause, and B1,....Bn are the literals of the clause. For each literal *L*, CF_L is a number between 1 and 0 that represents the expert's current degree of confidence that a given literal need not be revised. $Bias_L$ is an expression of the form {Revision-Operator/Pre-Conditions}*. The semantics of such a bias is: apply the first revision operator for which all the preconditions are satisfied.

Before we describe the current revision operators and the exact form of the biases allowed, let us consider the factors which might affect the selection of an appropriate revision operator:

- The location of the element in the rule base graph (in particular, its depth)
- The current contents of the rule base (the definition of one predicate might be affected by the definition of another)
- The negative examples that might be affected by the element's revision.
- The positive examples that might be affected by the element's revision.

In the next section we describe the current revision operators. As noted earlier, the system is designed to be easily extendible to accommodate new revision operators should they be required.

4.2 Revision Operators

We divide the revision operators into four classes according to the syntactic change they perform.

4.2.1 Deletion Operators

We have two deletion operators: we can either delete a literal from a clause (leading to greater generalization or specialization of the clause) or we can delete a clause from the definition of a predicate in order to specialize it. These operators perform radical revision since they delete complete elements from the rule base. In general, such operators will be used only if no additional problems can arise from their application.

4.2.2 Addition Operators

It is often the case that instead of deleting a clause *c*, we can remedy its original flaw by merely adding constraints to the body of *c* . These constraints should be chosen in such a way as to prevent the use of the clause by specifically those negative that have been using it to achieve an undesired proof. At the same time, it is equally important to ensure that these added conditions do not inadvertently prevent the acceptance of positive examples.

By analogy, when a literal *l* (where *l* is an instance of predicate *p*) becomes a candidate for revision, we can add clauses to the definition of *p* that produce alternative definitions of *l* under appropriate conditions. These additional clauses serve to generalize *l*, obviating the need to delete it from the clause in which it appears.

The decision to add new structures rather than deleting a flawed element is made on the basis of the examples affected by its revision. Therefore, before determining which revision operator to apply, we must decide which of the examples are relevant to this revision. If we were to make use of all given examples when adding a new structure, the new structure would be equivalent to an alternative rule base that correctly classifies all the examples. In other words, we would be using the inductive algorithm to build a new rule base from scratch – one that reflects none of the structure of the original rule base. Instead, we choose to focus our inductive IE rule learning algorithm (which is a FOIL-based algorithm) by finding an appropriate smaller set of relevant examples, ones that are directly affected by the revision of the flawed element. By using only these examples, we reduce the processing time needed for the inductive algorithm while retaining the original structure of the rule base.

4.2.3 Replacement Operators

These operators are actually a combination of deletion operators and addition operators. We delete one literal from a clause and immediately add a new set of literals to the clause instead. Since in this paper we are mainly concerned with incremental theory revision algorithms, and at each point we perform only a few revisions, the inclusion of macro operators such as replacement operators does make a difference. A special case of the replacement operators are literals that involve numeric constants such as, for example, skip elements (by changing the maximum number of skipped tokens allowed in them). Such literals are replaced by others where the numeric constants are changed in the appropriate direction in order to generalize or specialize the literal. In addition, when a word class becomes a candidate for revision, rather then deleting it, we can add another phrase to the word class.

Another important kind of the replacement operators are those related to dealing with the predicate hierarchy. We have two kinds: one that climbs up the hierarchy and generalizes the literal and another that descends the hierarchy and specializes the given predicate. Both operators take an extra argument - a number that represents the maximum number of levels to be tried in the hierarchy in the direction specified by the operator. Thus, for example, to allow the system to climb only one level, we provide 1 as an extra argument. In cases where there are several options for generalization or specialization, the program would pick the literal that best discriminates the positive examples from the negative ones in the set of relevant examples.

4.3 A Taxonomy of Biases

While in principle the preconditions attached to the revision operators might be any computable set of predicates, we propose to express preconditions using a small set of primitives. These are divided into three groups according to the type of information they examine.

4.3.1 Example-based preconditions

From our experience we have found that comparisons on the sets of needed and obstructive examples (N and O, respectively) are sufficient in most cases. In particular, we have found that three particular conditions are of interest:

1. *N* or $O \neq \varnothing$, meaning the set should not be empty
2. *N* or $O = \varnothing$, meaning the set should be empty
3. *N* or *O* = X, meaning we do not care about the set's contents.

For instance we might specify {delete/(*O* = X, *N* = X)} as the bias for the revision of a given clause. This bias implies that even if there are examples that need this clause to be correctly classified, we prefer to delete the clause when it becomes a candidate for revision. Similarly, we provide {add new clause /(*O* = X, *N* = X)} as a bias for the revision of a given intermediate literal. This expression says that even if we do not have any obstructive examples, we should not delete the literal, but prefer to add a new clause instead. This simple language, while not able to capture every possible bias with regard to the example set, is powerful enough to express most common biases given by experts. If we lift the constraint that the preconditions must be expressed using the sets *N* and *O* and allow any computable set of predicates, we can use this scheme to define any form of example bias.

4.3.2 Topology-based preconditions

These preconditions concern the location of the flawed element in the rule base graph. The primitives are:

`mindepth <Rel> <numeric constant>` where `mindepth` is the length of the shortest path between the flawed element and any top concept, and Rel ∈ {>,<,!=,≥,≤}.

`maxdepth <Rel> <numeric constant>` where `maxdepth` is the length of the longest path between the flawed element and any top concept, and Rel ∈ {>,<,!=,≥,≤}.

`minheight <Rel> <numeric constant>` where `minheight` is the length of the shortest path between the flawed element and any leaf in the graph, and Rel ∈ {>,<,!=,≥,≤}.

`maxheight <Rel> <numeric constant>` where `maxheight` is the length of the longest path between the flawed element and any leaf in the graph, and Rel ∈ {>,<,!=,≥,≤}.

Suppose we want to specify a bias that revisions should be performed only at the leaves of the rule base graph. We would then add the condition, `minheight = 0` to the list of preconditions of all elements. If this condition isn't met, the revision doesn't take place.

4.3.3 Hierarchy-based preconditions

These preconditions are related to the location of the element in the hierarchy and the topology of its ancestors and descendants.

The primitives are:

`ancestors(N)` - list of all Nth ancestors of the element (`ancestors(1)` is the list of direct parents).

`descendants (N)` - list of all Nth descendants of the element (`descendants(1)` is the list of direct children)

Thus, for example, we could specify a condition `size(ancestors(1)) = 1` as a precondition for climbing up in the hierarchy (*i.e.*, we may climb provided there is no ambiguity as to where to climb).

4.4 Bias Tables

In order to provide a friendly user interface, we view each of the biases attached to rule base elements as a structured table. Each element has a table that specifies its biases with regard to the selection of the appropriate revision operator. X represents "don't care", i.e., that there are no preconditions vis-a-vis that category.

The bias table of the literal `company(C)`, for example, would look as follows:

Table 1: Operator bias for Literal `company(c)`

Operator Name	Example preconditions	Topology preconditions	Hierarchy preconditions
Delete	N = ∅, O = ∅	X	X
Climb(1)	N = ∅, O = ∅	X	Size(parents(1)) = 1
User(climb)	N = ∅, O = ∅	X	Size(parents(1)) > 1
NOP	O = ∅	X	X

The above says that when the literal `company(C)` becomes a candidate for revision, we will delete it if there are no examples that rely on it to get a correct classification; while at the same time there are examples for which it is obstructive to correct classification. If any of these conditions are not met, we check if there is only one possible generalization of the literal. If that is the case, we replace it with its single generalization. Where there is more than one generalization we employ the interactive technique that involves the user picking the correct generalization operator. If there are no examples that benefit from the revision of this literal, we do nothing. This bias table was used for most the elements of the IE rule base.

4.5 Inheritance of Biases

Providing revision bias on an element-by-element basis might be unreasonable when dealing with large knowledge bases. We therefore propose using two hierarchies: the *syntactic hierarchy* and the *semantic hierarchy*. The syntactic hierarchy classifies elements according to their syntactic role (*e.g.*, intermediate literals, clauses, skip elements, word classes etc.). The syntactic hierarchy is shown in Figure 2. The semantic hierarchy classifies elements according to their meaning (e.g., people-related, company-relationship, technology-related, product-related etc.). While the syntactic hierarchy is domain-independent, the semantic hierarchy is domain dependent. A specification of an element cluster is formed by combining the syntactic description with the semantic one. For instance, one possible cluster description might be the set of leaf literals that correspond to company-relationship predicates (such as those for "*merger*", "*joint venture*", and "*take over*"). All such elements would be assigned identical revision bias. The biases provided for the different clusters form an *inheritance hierarchy*. An element will be affected by the bias declared for the most specific clusters that contain it. In the bias table we can define that some preconditions are inherited from more general clusters by indicating *I* in the suitable cell. We give the user the option of specifying a criterion for breaking ties in case of multiple inheritances. The default criterion is to pick the cluster that was defined previously.

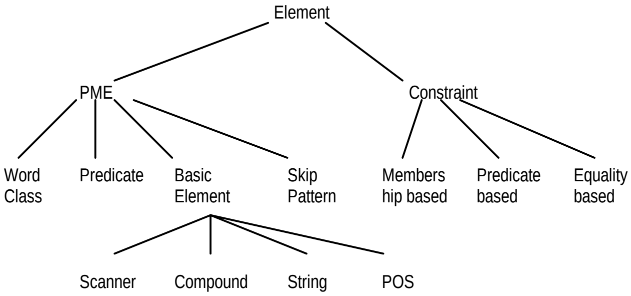
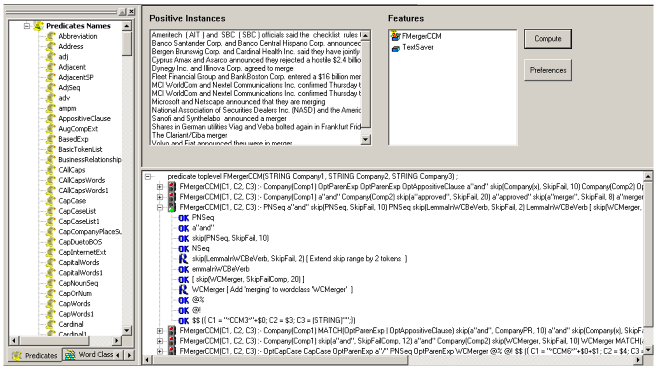


Fig. 2: The Syntactic hierarchy

5. EXPERIMENTAL EVALUATION

We have tested the accuracy of the IE engine by analyzing collections of documents extracted by the integrated Agent from MarketWatch.com (over the period Oct 1999 – Feb 2000). We started by extracting 15,950 articles from *MarketWatch.com* that mentioned the word "*merger*". We created 67 different event types centering on companies, people, locations, technologies, products and alliances. We defined 320 word classes and 2100 rules to extract the aforementioned event types. The advanced debugging tools proved to be very useful in the debugging and refinement of the rule set. After construction of the initial rule set we achieved an F-Score of 89.3%. The IE revision module enabled us to boost the F-Score to 96.7% in several hours. The actual revision of the rules is done interactively, enabling the user to pick the desired revision from the revisions proposed by the system.

In Figure 3 we can see how the system proposes the revisions to the user. In the left upper pane we see all positive instances of `FMergerCCM`, and in the bottom pane we can see the clauses of `FMergerCCM`, along with possible revision schemes. We expanded one of the clauses and show the proposed revisions. Here the system suggested two revisions (marked with "R" icon). The first revision is to increase the skip range from 2 tokens to 4 tokens, and the second revision is to add the word "*merging*" to the word class `WCMerger`. Performing these revisions enables the clause to extract the event `FmergerCCM` ("*Banco Santander Corp.*", "*Banco Central Hispano Corp.*") from the string "*Banco Santander Corp. and Banco Central Hispano Corp. announced that they are merging*". The revisions suggested by the system were based on the default bias scheme defined above.



```
Ameritech ( AIT ) and SBC ( SBC ) officials said the checklist rules t
Banco Santander Corp. and Banco Central Hispano Corp. announcec
Bergen Brunswig Corp. and Cardinal Health Inc. said they have jointly
Cyprus Amax and Asarco announced they rejected a hostile $2.4 billio
Dynegy Inc. and Illinova Corp. agreed to merge
Fleet Financial Group and BankBoston Corp. entered a $16 billion mer
MCI WorldCom and Nextel Communications Inc. confirmed Thursday t
MCI WorldCom and Nextel Communications Inc. confirmed Thursday t
Microsoft and Netscape announced that they are merging
National Association of Securities Dealers Inc. (NASD) and the Americ
Sanofi and Synthelabo  announced a merger
Shares in German utilities Viag and Veba bolted again in Frankfurt Frid
The Clariant/Ciba merger
Volvo and Fiat announced they were in merger
```

```
⊟    predicate toplevel FMergerCCM(STRING Company1, STRING Company2, STRING Company3) ;
  ⊞  FMergerCCM(C1, C2, C3) :- Company(Comp1) OptParenExp OptParenExp OptAppositiveClause a"and" skip(Company(x), SkipFail, 10) Company(Comp2) Op
  ⊞  FMergerCCM(C1, C2, C3) :- Company(Comp1) a"and" Company(Comp2) skip(a"approved", SkipFail, 20) a"approved" skip(a"merger", SkipFail, 8) a"merger
  ⊟  FMergerCCM(C1, C2, C3) :- PNSeq a"and" skip(PNSeq, SkipFail, 10) PNSeq skip(LemmaInWCBeVerb, SkipFail, 2) LemmaInWCBeVerb [ skip(WCMerger,
       OK PNSeq
       OK a"and"
       OK skip(PNSeq, SkipFail, 10)
       OK NSeq
       R  skip(LemmaInWCBeVerb, SkipFail, 2) [ Extend skip range by 2 tokens  ]
       OK emmaInWCBeVerb
       OK [ skip(WCMerger, SkipFailComp, 20) ]
       R  WCMerger [ Add 'merging' to wordclass 'WCMerger'  ]
       OK @%
       OK @!
       OK $$ ({ C1 = "*CCM3*"+$0; C2 = $3; C3 = (STRING)"";})
  ⊞  FMergerCCM(C1, C2, C3) :- Company(Comp1) MATCH(OptParenExp | OptAppositiveClause) skip(a"and", CompanyPR, 10) a"and" skip(Company(x), SkipFa
  ⊞  FMergerCCM(C1, C2, C3) :- Company(Comp1) skip(a"and", SkipFailComp, 12) a"and" Company(Comp2) skip(WCMerger, SkipFail, 10) WCMerger MATCH(a
  ⊞  FMergerCCM(C1, C2, C3) :- OptCapCase CapCase OptParenExp a"/" PNSeq OptParenExp WCMerger @% @! $$ ({ C1 = "*CCM6*"+$0+$1; C2 = $4; C3 =
```

Fig. 3: Interactive revision of the IE rules of `FMergerCCM`

We will now show how *Textoscope* [2,9,10,11,12,13] (The visual front-end of the Text Mining system) enables us to analyze the events and terms that were extracted from the 15,950 articles. In Figure 4 we can see an event map showing companies that are related to an event of "negative merger", i.e., denying a planned merger, or merger plans that did not materialize[1].

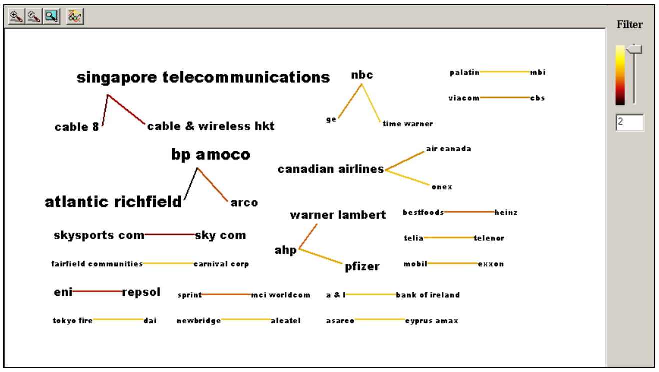


Fig. 4: "Negative merger" Event map.

In Fig. 5 we can see an Event Map (with filter set to 11, i.e. only events mentioned in at least 11 documents are shown) of actual and planned mergers. We can see that the Time Warner-AOL mega merger is one of the main events shown. In Fig. 6 we can see the companies related to Time Warner in this collection. In Fig. 7 we can see the titles of the documents that support the merger event between AOL and Time Warner. In addition to the title we can see the exact sentence in each document from which the merger event was extracted. In Fig. 8 we see one of the documents that supports the merger event between AOL and Time Warner.

[1] Larger font size indicates a higher occurrence of the term in the collection. The darker the color of the link between terms, the higher the support this event has in the collection.

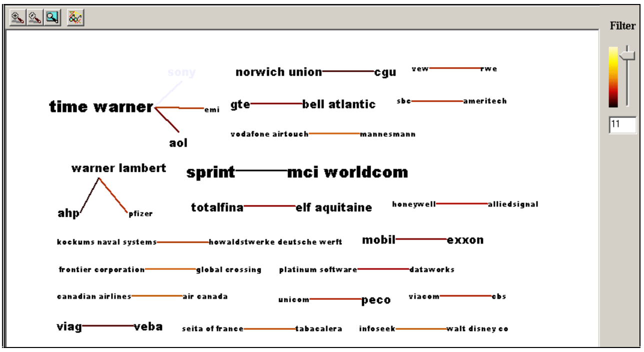


Fig. 5: "Merger" Event Map.

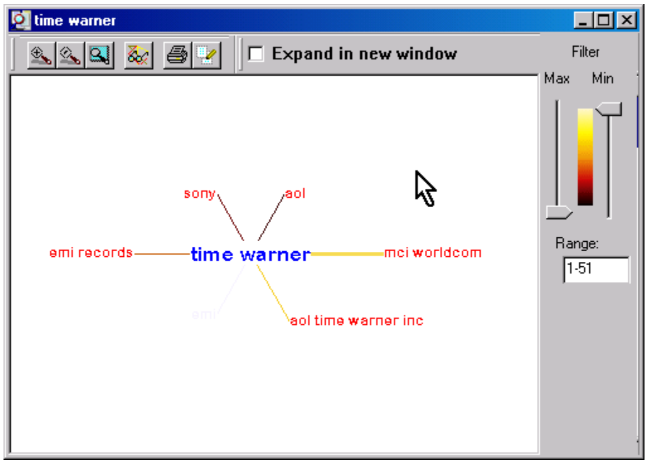
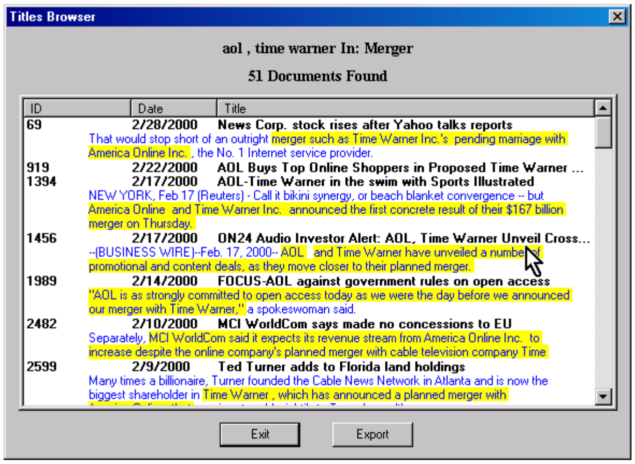


Fig. 6: Companies related to Time Warner

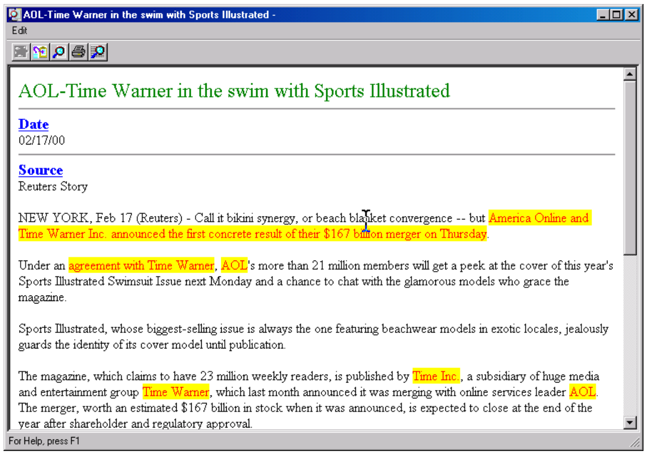


aol , time warner In: Merger

51 Documents Found

ID	Date	Title
69	2/28/2000	News Corp. stock rises after Yahoo talks reports That would stop short of an outright merger such as Time Warner Inc.'s pending marriage with America Online Inc. , the No. 1 Internet service provider.
919	2/22/2000	AOL Buys Top Online Shoppers in Proposed Time Warner ...
1394	2/17/2000	AOL-Time Warner in the swim with Sports Illustrated NEW YORK, Feb 17 (Reuters) - Call it bikini synergy, or beach blanket convergence -- but America Online and Time Warner Inc. announced the first concrete result of their $167 billion merger on Thursday.
1456	2/17/2000	ON24 Audio Investor Alert: AOL, Time Warner Unveil Cross... --(BUSINESS WIRE)--Feb. 17, 2000-- AOL and Time Warner have unveiled a number of promotional and content deals, as they move closer to their planned merger.
1989	2/14/2000	FOCUS-AOL against government rules on open access "AOL is as strongly committed to open access today as we were the day before we announced our merger with Time Warner," a spokeswoman said.
2482	2/10/2000	MCI WorldCom says made no concessions to EU Separately, MCI WorldCom said it expects its revenue stream from America Online Inc. to increase despite the online company's planned merger with cable television company Time
2599	2/9/2000	Ted Turner adds to Florida land holdings Many times a billionaire, Turner founded the Cable News Network in Atlanta and is now the biggest shareholder in Time Warner , which has announced a planned merger with

Fig. 7: Titles of Documents supporting the AOL-Time Warner Merger event.



AOL-Time Warner in the swim with Sports Illustrated

Date
02/17/00

Source
Reuters Story

NEW YORK, Feb 17 (Reuters) - Call it bikini synergy, or beach blanket convergence -- but America Online and Time Warner Inc. announced the first concrete result of their $167 billion merger on Thursday.

Under an agreement with Time Warner, AOL's more than 21 million members will get a peek at the cover of this year's Sports Illustrated Swimsuit Issue next Monday and a chance to chat with the glamorous models who grace the magazine.

Sports Illustrated, whose biggest-selling issue is always the one featuring beachwear models in exotic locales, jealously guards the identity of its cover model until publication.

The magazine, which claims to have 23 million weekly readers, is published by Time Inc., a subsidiary of huge media and entertainment group Time Warner, which last month announced it was merging with online services leader AOL. The merger, worth an estimated $167 billion in stock when it was announced, is expected to close at the end of the year after shareholder and regulatory approval.

Fig. 8: One of the Documents supporting the AOL-Time Warner Merger event.

6. Conclusions

This paper presents a framework for performing biased incremental IE rules revision based on explicit bias. We have introduced a modular architecture for specifying revision bias, and have described a family of revision operators and preconditions needed for their appropriate application. The architecture is designed to enable easy specification of biases of both the aggregates of elements based on predefined semantic and syntactic hierarchies and of specific rule base elements.

We view this paper as a further step toward building powerful bias-guided IE revision systems. In our case the bias comes from the operator bias attached to clusters of rule base elements. There are certainly other forms of biases, which might be used to construct efficient systems that perform accurate revisions, but we believe that those can be easily integrated into the proposed framework. It should be noted that the scheme of the operator bias presented here may be adopted by any incremental IE revision algorithm, and can be used in conjunction with any IE rule learning algorithm. This approach enabled us to achieve a much higher precision and recall than any of the other systems that were based only on inductive learning of IE rules.

7. ACKNOWLEDGMENTS

Our thanks to Oren Etzioni and Yonatan Aumann for helpful discussions on earlier drafts of this paper.

8. References

[1] Appelt, D. E., Hobbs J., Bear J., Israel D., and Tyson M., 1993. ''FASTUS: A Finite-State Processor for Information Extraction from Real-World Text", *Proceedings*. IJCAI-93, Chambery, France, August 1993.

[2] Aumann Y., Feldman R., Ben Yehuda Y., Landau D., Lipshtat O., Schler Y.: Circle Graphs: New

Visualization Tools for Text-Mining. *PKDD* 1999: 277-282

[3] Califf, M. E. and Mooney, R. (1997). Relational learning of pattern-match rules for information

[4] Cohen. W., "Compiling Prior Knowledge into an Explicit Bias". *Working notes of the 1992 AAAI spring symposium on knowledge assimilation*. Stanford, CA, March 1992.

[5] Craven M., DiPasquo D., Freitag D., McCallum A., Mitchell T., Nigam K. and Slattery S. Learning to Extract Symbolic Knowledge from the World Wide Web. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98).*

[6] Fisher D., Soderland S., McCarthy J., Feng F. and Lehnert W., "Description of the UMass Systems as Used for MUC-6," in *Proceedings of the 6th Message Understanding Conference*, November, 1995, pp. 127-140.

[7] Feldman R., Segre A. and Koppel M., "Incremental Refinement of Approximate Domain Theories". *Proceedings of the 8th International Conference on Machine Learning*, 500-504, Evanston,IL,1991.

[8] Feldman. R., "Probabilistic Revision of Logical Domain Theories". *Ph.D Thesis*, Computer Science Department, Cornell University, Ithaca NY, February 1993.

[9] Feldman R., Aumann Y., Fresko M., Lipshtat O., Rosenfeld B., Schler Y.: Text Mining via Information Extraction. *PKDD* 1999: 165-173

[10] Feldman R., Aumann Y., Zilberstein A., Ben-Yehuda Y. Trend Graphs: Visualizing the Evolution of Concept Relationships in Large Document Collections. PKDD 1998: 38-46

[11] Feldman R., Fresko M., Kinar Y., Lindell Y., Liphstat O., Rajman M., Schler Y., Zamir O. Text Mining at the Term Level. PKDD 1998: 65-73

[12] Feldman R., Dagan I., Hirsh H. Mining Text Using Keyword Distributions. JIIS 10(3): 281-300 (1998)

[13] Feldman R., Klösgen W., Zilberstein A. Visualization Techniques to Explore Data Mining Results for Document Collections. KDD 1997: 16-23

[14] Freitag, D. (1998). Multistrategy learning for information extraction. *Proceedings of the 15th International ML Conference*, 161-169.

[15] Huffman, S. (1996). Learning information extraction patterns from examples. In Wermter, *Learning for Natural Language Processing*. Berlin: Springer.

[16] Pazzani M. and Kibler D., "The Utility of Knowledge in Inductive Learning". *Technical Report*, University of California,Irvine, 1990.

[17] Mahoney J.J. and Mooney R.J., "Combining Neural and Symbolic Learning to Revise Probabilistic Rule Bases". *Advances in Neural Information Processing Systems*, Vol. 5. Morgan Kaufman, San Mateo, CA, 1993.

[18] Ourston D. and Mooney R. J.,"Changing the rules: A comprehensive approach to theory revision". *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815-820, Boston, MA, 1990.

[19] Pazzani M. J., "Detecting and Correcting Errors of Omission after Explanation-Based Learning", in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp 713-718, Detroit, Aug 1989.

[20] Quinlan J.R., "Induction on Decision trees". *Machine Learning*, 1, 81-106, 1986.

[21] Richards B.L., and Mooney R.J., "First-Order Theory Revision". *Proceedings of the 8th International Workshop on Machine Learning*, 447-451, Evanston,IL,1991.

[22] Riloff E. and Lehnert W., *Information Extraction as a Basis for High-Precision Text Classification*, ACM Transactions on Information Systems (special issue on text categorization) .

[23] Soderland S., “Learning Information Extraction Rules for Semi-structured and Free Text,” Machine Learning Journal, 1999, 1-44.

[24] Towell G.G., Shavlik J. and Noordewier M.O., "Refinement of approximately correct domain theories by knowledge-based neural networks". *Proceedings of the Eighth National Conference on Artificial Intelligence*, 861-866, Boston, 1990.

[25] Wogulis J., "Revising Relational Domain Theories". *Proceedings of the 8th International Workshop on Machine Learning*, 462-466, Evanston,IL,1991.