Recursive OpenIE

Keenon Werling*

Stanford University
keenon@stanford.edu

Abstract

This paper proposes an extension to the OpenIE paradigm, to allow the expression of recursive relations, and presents a fully implemented extension to the EXEMPLAR system, uncreatively referred to as Recursive-EXEMPLAR, to extract recursive n-ary relations automatically. A rule-based approach is shown to achieve very high accuracy. Attempts at the automatic learning of rules from hand-labelled data proved to be problematic, because I find that my labellings are inconsistent and contradictory, and automatic labeling methods are too compute intensive for the project timescale. recall.

I. Introduction

Open Information Extraction (Banko and Etzioni, 2008) is an attractive paradigm, because it frees us from the burden of manually describing every relation type we want to extract with large quantities of labelled training data. At the same time, it is notoriously difficult to use as input to higher level NLP and general reasoning tasks, because current state-of-the-art systems ignore many important limitations on the quality of their extractions.

Primarily, this is because related extractions are not linked. For example, "the prime minister proposed that women ascend the throne" would be extracted traditionally as two seperate relations: *proposed*(**prime minister**), *ascend*(**women**, **throne**). This repsentation suggests that "women ascend the throne" is already a fact, and doesn't give us any information on what the prime minister proposed, if he proposed anything. We could post-process the flat relations, after the OpenIE step, and try to guess that "the prime minister proposed **that** women ascend the throne", but this is a fraught activity, since we're missing much of the information that the OpenIE system used internally to assign relationships. Much more reliably, we could instead extract the recursive relation *proposed*(**prime minister**, *ascend*(**women**, **throne**)) from within the OpenIE extractor itself. This recursive representation has the benefit that limitations on truth that go beyond the complexity of basic prepositional relations are easily represented, and objects that are more complex than single phrases can still be linked without loss of information. The relative reliability of a speaker can be factored into the quality of an extraction if the speaker is known, and temporal and spatial limitations are still easily extracted from prepositional attachments at any point in the recursive relation.

II. Related Work

OpenIE has seen an explosion of methods in the last few years, with varying degrees of accuracy and computational cost. The original OpenIE systems, ReVerb (Fader et al. 2011) and SONEX (Merhav et al. 2012), putting a premium on scalability over recall, used only shallow syntactic information like POS tags for their extractions. As you would expect, these systems are very fast because preprocessing is so light, but their input doesn't allow them to distinguish long-range dependencies, and so accuracy suffers compared to later systems that use full syntactic parses. The syntactic parse OpenIE systems, PATTY (Nakashole et

*Many thanks to Gabor Angeli, and the Stanford NLP Lab in general, for data and advice

al., 2012), OLLIE (Mausam et al. 2012), and TreeKernel (Xu et al., 2013) all use automatically learned dependency parses to attempt to extract relations. OLLIE and PATTY use tree regular expressions, and TreeKernel uses an SVM over sub-trees to classify whether or not a relation is present between two named entities. Even more computationally expensive methods use a semantic parse as input, and can identify from that the n-ary relations in a sentence, though their accuracy is tied very tightly to the performance of the underlying semantic parser.

III. Original *EXEMPLAR*

A recent paper (Mequita et. al, 2013) did a fair comparison of several leading methods, and proposed an elegant and relatively computationally efficient method for OpenIE with the highest accuracy of any of the measured systems, *EXEMPLAR*. The two major innovations represented in the design of *EXEMPLAR* are

1. The ability to extract relation phrases independently of their arguments, like a semantic-parse-based system.
2. The return to a rule-based approach as a solution to a general lack of sufficient labeled data.

The system presented in this paper builds directly upon *EXEMPLAR*'s innovations to achieve a system for extracting n-ary, recursive relations with state-of-the-art accuracy.

IV. Design of *Recursive-EXEMPLAR*

The *Recursive-EXEMPLAR* extraction system works in a series of deterministic steps, repeatedly applied until no new information can be gained from data. Then post-processing is applied across the extracted relations, and the results are returned. The general algorithm is as follows:

function ExtractRelations(S)
  $P \leftarrow$ StanfordParser(S)
  $E \leftarrow$ NamedEntities(P)
  $R \leftarrow []$
  while true **do**
    $T \leftarrow$ DetectTriggers(P, E, R)
    $R_{new} \leftarrow$ DetectRoles(P, E, R, T)
    $R_{new} \leftarrow$ FilterRels(P, R_{new})
    if $|R_{new}| == 0$ **then**
      return PostProcess(R)
    else
      $R \leftarrow$ Merge(R, R_{new})
    end if
  end while
end function

Named entities are detected using the Stanford NER system. DetectTriggers(), DetectRoles(), FilterRels(), and PostProcess() will be explained in subsequent sections. The major change in the design of *Recursive-EXEMPLAR* over *EXEMPLAR* is the use of a loop to detect new relations given old relations, instead of a single pass to detect stand-alone relations all at once, and a final PostProcess() step to use parse data to make relations as easy to use as possible for other applications. The necessity for the loop design will become clear in the discussion of DetectRoles().

V. Detecting Triggers

The brilliant leap in the design of *EXEMPLAR* was to detect "triggers", defined to be words that indicate the presence of a relation, seperately from their arguments. *EXEMPLAR* identifies 3 kinds of triggers, **Verb**, **Copula+Noun**, and **Verb+Noun**. *Recursive-EXEMPLAR* adds the recursive trigger, **Conjunction**. Their relative frequencies, automatically collected from 3247 random sentences from Wikipedia, and 55173 sentences from NYT are as follows:

Table 1: *Wikipedia Trigger Frequencies*

Trigger Type	Freq.	Avg./Sentence
Verb	53.8%	1.32
Verb+noun	30.2%	0.74
Copula+noun	11.0%	0.27
Conjunction	5.0%	0.12

Table 2: *NYT Trigger Frequencies*

Trigger Type	Freq.	Avg./Sentence
Verb	53.2%	1.37
Verb+noun	32.3%	0.83
Copula+noun	12.9%	0.32
Conjunction	5.0%	0.13

Triggers are identified deterministically using rules on dependency parses. The striking similarity of the numbers between NYT and Wikipedia was suprising to me, and suggests that there is some underly distribution over the way English speakers communicate.

Verb Triggers: Any verb that does not have a noun as it's direct object is classified a verb trigger. Any noun that is the nominalized form of a verb is also classified as a verb trigger. Wordnet's Morphosemantic Database is used to find nominalized verbs, by checking if the classification of a noun is "event". Nominalized verbs are recorded as the original verb form relation. This can lead to spurious triggers, but those are filtered out during FilterRels().

Verb+Noun Triggers: In the original *EXEMPLAR*, any verb with a direct object is classified as a trigger containing two tokens, the verb, and the direct object. Either token can be the end of a trigger for a dependency rule described in the DetectRoles(), and the surface form of the relation is the concatenation of the lemmas of the verb and the direct object. To use one of the *EXEMPLAR* authors' examples, "NFL approves Falcons' new stadium in Atlanta" would extract the Verb+Noun trigger "approve new stadium". This entire chunk became the root of the relation. This is less useful for systems using the output relations, so instead *Recursive-EXEMPLAR* extracts only the Verb as the Verb+Noun trigger, and labels its direct object the direct object of the relation. Practically, the difference is that we get *approve*(NFL,new stadium) instead of *approve new stadium*(NFL).

Copula+Noun Triggers: Any noun to which a verb has a copula dependency is labeled a Copula+Noun trigger, with the surface form being the verb. Also, any noun with any apposition dependencies is considered as a Copula+Noun trigger. For appositions, the surface form used for the relation is "be". For example, "We, the people" is translated as *be*(We, the people).

Conjunction Triggers: This is an addition to the original *EXEMPLAR* set. Any preposition or subordinating conjunction with a "mark" dependency to any word is considered a Conjunction trigger. This is useful for catching the information conveyed by "that", which is traditionally ignored by OpenIE systems. To use the example in the introduction, "the prime minister proposed **that** women ascend the throne", "that" would be a Conjunction trigger.

VI. Detecting Roles

Once a list of triggers is collected, the next step assigns arguments to the triggers, which become preliminary n-ary relations. Each argument is given a weight of preference, and at this stage no effort is made to prevent an n-ary relation from having multiple subjects and multiple direct objects. The process is entirely rule-based. Rules take the form of simple dependency patterns from a trigger, with restrictions about what trigger type is applicable (Verb, Verb+Noun, etc), and what trigger POS is applicable.

Recursive-EXEMPLAR adds one more feature to rules. Rules may be marked "recursive" or "non-recursive". A rule marked recursive is only applicable to words that are already within an argument or trigger for an n-ary relation extracted in a previous round, and then the entire n-ary relation is taken as the argument. Rules marked non-recursive can only be applied to words that are not yet contained in any argument or trigger. In order to faccilitate recursive rules, which tend to have longer dependency paths, *Recursive-EXEMPLAR* al-

lows dependency path rules of arbitrary length, where *EXEMPLAR* considered only paths of length 1.

Below I reproduce the *Recursive-EXEMPLAR* rules, which are largely taken from the original *EXEMPLAR*. All bold rules are my additions. A dependency rule is written as seriese of "<" (indicating a parent dependency measured from the trigger) or ">" (indicating a child dependency measured from the trigger) followed by the type of dependency. For instance, ">dobj" would indicate a rule that applied to any direct child of a trigger with a "dobj" dependency. Read the a rule as "if POS, Dep. Type, and Recursive match, assign Role".

Recursive-Exemplar Rules:

Table 3: *Verb Trigger Rules*

POS	Dep. Type	Role	Recursive
Verb	>nsubj	subj	false
Verb	>agent	subj	false
Verb	<partmod	subj	false
Verb	<rcmod	subj	false
Verb	>dobj	dobj	false
Verb	>subjpass	dobj	false
Verb	>iobj	to_obj	false
Verb	>prep_*	prep_obj	false
Noun	>prep_by	subj	false
Noun	>amod	subj	false
Noun	>nn	subj	false
Noun	>poss	subj	false
Noun	>prep_of	dobj	false
Noun	>prep_*	prep_obj	false
Verb	**<xcomp**	**subj**	**true**
Verb	**>xcomp**	**dobj**	**true**
Verb	**<ccomp**	**subj**	**true**

Table 4: *Conjunction Rules*

POS	Dep. Type	Role	Recursive
IN	**<mark<***	**subj**	**true**
IN	**<mark**	**dobj**	**true**

Table 5: *Copula+Noun Trigger Rules*

POS	Dep. Type	Role	Recursive
Noun	>nsubj	subj	false
Noun	>appos	subj	false
Noun	<appos	subj	false
Noun	<partmod	subj	false
Noun	<rcmod	subj	false
Noun	>prep_of	of_obj	false
Noun	>amod	of_obj	false
Noun	>nn	of_obj	false
Noun	>poss	of_obj	false
Noun	>prep_*	prep_obj	false

Table 6: *Verb+Noun Trigger Rules*

POS	Dep. Type	Role	Recursive
Verb	>nsubj	subj	false
Verb	>agent	subj	false
Verb	<partmod	subj	false
Verb	<rcmod	subj	false
Verb	>iobj	to_obj	false
Verb	>prep_*	prep_obj	false
Noun	>amod	of_obj	false
Noun	>nn	of_obj	false
Noun	>poss	of_obj	false
Noun	>prep_*	prep_obj	false
Verb	**<xcomp**	**subj**	**true**
Verb	**>xcomp**	**dobj**	**true**
Verb	**<ccomp**	**subj**	**true**

The rules are given a descending (arbitrary) score at startup, so that rules higher up on the table trump rules lower down on the table during the filter step. The role detection algorithm is as follows:

function DETECTROLES(P,R,T)
  $P \leftarrow$ STANFORDPARSER(S)
  $E \leftarrow$ NAMEDENTITIES(P)
  $R_{new} \leftarrow []$
  for $t \in T.triggers$ **do**
    $r \leftarrow$ NRELFROMTRIGGER(t)
    for $rule \in RULES$ **do**
      for $W \in P.words$ **do**
        if CONFORMS($rule, t, W, R$)
then

```
                AddArgument(r, rule, W, R)
            end if
        end for
    end for
    Insert(R_new, r)
end for
return R_new
end function
```

The set of rules extracted last round is passed into Conforms() along with the trigger, word, and rule, so that the recursive restriction can be checked. Likewise, the last round's rules are passed into AddArgument(), so that if the rule is recursive, the argument can be made recursive as well.

VII. Filtering Rules

In the filtering step, for each n-ary relation, if it has multiple subject arguments or multiple direct object arguments, I remove all but the one generated by the rule the highest on the list. I allow as many other kinds of relations as were extracted, though I filter for duplicates. Any n-ary relation that doesn't have a subject is thrown it out, except if the trigger has any child dependencies of the type "nsubjpass", and then we guess that the action is expressed in passive voice, and so we use the psuedo-argument "PASSIVE", and keep the n-ary relation.

VIII. Post-processing

For the convenience of users of the output of *Recursive-EXEMPLAR*, a few post-processing steps are done once the algorithm has returned.

1. An attempt is made to collapse "that" relations. If the subject of the "that" relation has no direct object, then the direct object of the "that" relation is made in the direct object of the subject of the "that", and the original "that" is deleted from the output. To clarify that horrible sentence, using our running example, "the prime minister proposed that women ascend the throne" extracts *that*(*proposed*(**Prime Minister**),*ascend*(**women**,**throne**)), which is reduced to *proposed*(**Prime Minister**,*ascend*(**women**,**throne**)).

2. Coreference is applied to all arguments that contain pronouns, because extractions containing "he" and "she" are totally useless for some higher level applications. Coreference is *not* applied in general, because it tends to reduce the accuracy of the extractions, since any coreference system is imperfect. I'd rather leave the application of precision reducing tools as a choice for users when there's any ambiguity in it.

IX. Results

Results are measured against Wikipedia articles and NYT seperately. For each corpus I hand labeled 100 flat extractions, and 100 recursive extractions as correct or incorrect. I measured these two groups seperately because I am primarily interested in the recursive extractions, and since they are far less frequent, measuring together would mean that I would have a label many more flat extractions in order to get a reasonable sample of recursive extractions. The relative frequency of recursive extractions, as a percentage of total extractions, is noted in the first numeric column.

Table 7: *Hand Labeled Precision Scores*

Corpus	Rec. Freq.	Flat Prec.	Rec. Prec.
Wikipedia	10.9%	71.3%	74.1%
NYT	12.7%	73.1%	81.2%

These numbers are limited to precision only (no recall values) because I was unable to label *consistent* n-ary extractions from a body of sentences in the time provided for this project. See Discussion for more detail. I can make a reasonable lower bound on recall, however, since the original *EXEMPLAR* paper measured recall at 30%, and I can only have increased recall with additional rules (while potentially lowering precision) I can lower bound my recall at 30%. Having measured precision on

both NYT and Wikipedia, I can also claim that my recursive extraction rules are highly precise. The density of high-confidence information conveyed by recursive n-ary relations offsets somewhat their relative scarcity, but higher recall is none the less an area of further research.

Below are some examples of the recursive extractions produced by *Recursive-EXEMPLAR* on the test set during the labeling run to generate the precision numbers you see above:

1. Attorneys for the plaintiffs contended that Exxon bore responsibility for the accident because the company "put a drunk in charge of a tanker in Prince William Sound."

   because(
   bore(**Exxon**, **responsibility**),
   put(**company**, **drunk**)
   [*IN_OBJECT* **charge**]
)
   contended(
   Attorneys,
   bore(**Exxon**, **responsibility**)
)

2. The EDSAC's memory consisted of 1024 locations, though only 512 locations were initially implemented.

   though(
   consisted(**EDSAC memory**)
   [*OF_OBJECT* **1024 locations**],
   implemented(**PASSIVE**,
   only 512 locations)
)

3. Popular opinion holds that longer scale length contributes to greater amplitude.

   holds(
   Popular opinion,
   contributes(**longer scale length**)
   [*TO_OBJECT* **greater amplitude**]
)

4. The economy of the city of Cordova, Alaska was adversely affected after the spill damaged stocks of salmon and herring in the area.

   after(
   affected(**PASSIVE**, **economy**),
   damaged(**spill**, **stocks**)
   [*IN_OBJECT* **area**]
)

X. Error Analysis

Slightly over 40% of errors observed in both the Wikipedia and NYT tests came from misparsed sections of sentences. Sublocation, Location attachments accidentally treated as apposition was very common, extracting "London, England" as *be*(**London**,**England**). This could be corrected for by attaching a knowledge base to the extractor to look for impossible facts, and remove them. Since it's such a common parsing error in Wikipedia, where locations are often referenced in the "Small, Large" pattern, a simple location-be-location filter could go a long way for precision, and wouldn't hurt recall very much.

Recursive-EXEMPLAR is designed to attach arguments to trigger words. It does not have a step for attaching other words to triggers. This is the cause of may errors in the presence of a correct parse. For example, "in" was extracted where "in order to", "in case" would have been appropriate. "as" was extracted where "as long as" would have been appropriate. These idioms are hard to detect automatically, but are a major cause of confusion, and it's an issue that deserves further study.

Working through the errors for *Recursive-EXEMPLAR*'s non-recursive extractions, I found "to" consistently blocked traditional *EXEMPLAR* linkings. For example, "I came to win it" would parse as *came*(**I**) and *win*(**PASSIVE**,**it**). It would make more sense to parse as *came*(**I**)[*TO_OBJECT win*(**PASSIVE**,**it**)], and post-process to *came*(**I**)[*TO_OBJECT win*(**I**,**it**)]

but currently *Recursive-EXEMPLAR* doesn't support recursive linking of prepositions or post-processing passive recursive prepositions. This is an easy fix, and it's a shame I noticed it so late in the process. This is definately a low-hanging fruit for future research.

XI. Difficulties With Labeled Data

I labeled 100 sentences at the beginning of the project by hand, taking me the suprising length of 5 hours, and discovered almost immediately that none of my labels were consistent, and many of my extractions, though they looked fine to a human, were completely useless to the machine. An initial baseline system I implemented using the OLLIE model on my hand-labeled data could not achieve above 40% precision, which means that the largest cluster of interally consistent data I labeled, in terms of dependency parse rules, was 40%. I made a significant effort in the last two weeks of the project to build a system to automatically generate labeled data, as is briefly described in the *EXEMPLAR* paper, using a dismabiguation system (Cucerzan, 2007) and Freebase to find words in a sentence with two mapped entities that are Wordnet synonyms for a relation we know exists in Freebase between the entities. Freebase, which now clocks in at 280 gigs of text, comes in a messy proprietary format, and so I've had to apply several cleaning and filtering steps, each one taking several days to compute, and as I submit this, my computer is on its second day of building indexes for a Apache Jena database. A major point of interest, and further research, is what I could do with automatically labeled datasets.

XII. Further Research

This paper demonstrates an expansion of the OpenIE paradigm to allow more expressive first order logic expressions, and shows that it is possible with a simple rule-based approach to achieve suprisingly sophisticated and high precision extractions. I can see two clear areas that would be worth further exploration with relation to *Recursive-EXEMPLAR*:
The first is an idiom expansion system for the trigger words. Not properly expanding idioms, and failing to include negations and modifiers severely hurts the precision of the system.
The second is, if a reliable method of automatically generating training examples is available, an automated approach to learning and pruning rules based on large training sets. A large data set may also prove useful for idioms as well, although I can think of no automated approach that would get idioms right.

XIII. Thanks

Many thanks to Gabor Angeli in the Stanford AI lab for providing data, advice, and encouragement throughout the process of developing *Recursive-EXEMPLAR*.

References

[Dan Klein and Chistopher D. Manning, 2003] Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL'03, pages 423-430, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Silviu Cucerzan, 2007] Large-scale named entity disambiguation based on wikipedia data. *Proceedings of Coreference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL'12, pages 708-716.

[Anthony Fader et al., 2011] Identifying Relations for Open Information Extraction. *Proceedings of Coreference on Empirical Methods in Natural Language Processing*, EMNLP-CoNLL'12.

[Mausam et al., 2012] Open language learning for information extraction. *Proceedings of Coreference on Empirical Methods in Natural Language Processing and*

Computaitonal Natural Language Learning, EMNLP-CoNLL'12.

[Yuval Merhav et al., 2012] Extracting information networks from the blogosphere. *TWEB*, 6(3):11.

[Ndapandula Nakshole et al., 2012] Patty: a taxonomy of relational patterns with semantic types. *Proceedings of Coreference on Empirical Methods in Natural Language Processing and Computaitonal Natural Language Learning*, EMNLP-CoNLL'12, pages 1135-1145, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Ying Xu et al., 2013] Open information extraction with tree kernels. *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 868-877, Atlanta, Georgia, June. Association for Computational Linguistics.