# Commonsense from the Web: Relation Properties

**Thomas Lin** and **Mausam** and **Oren Etzioni**
Turing Center
University of Washington
Seattle, WA 98195, USA
{tlin,mausam,etzioni}@cs.washington.edu

## Abstract

When general purpose software agents fail, it's often because they're brittle and need more background commonsense knowledge. In this paper we present *relation properties* as a valuable type of commonsense knowledge that can be automatically inferred at scale by reading the Web. People base many commonsense inferences on their knowledge of relation properties such as *functionality*, *transitivity*, and others. For example, all people know that *bornIn(Year)* satisfies the functionality property, meaning that each person can be born in exactly one year. Thus inferences like "Obama was born in 1961, so he was not born in 2008", which computers do not know, are obvious even to children. We demonstrate scalable heuristics for learning relation functionality from noisy Web text that outperform existing approaches to detecting functionality. The heuristics we use address Web NLP challenges that are also common to learning other relation properties, and can be easily transferred. Each relation property we learn for a Web-scale set of relations will enable computers to solve real tasks, and the data from learning many such properties will be a useful addition to general commonsense knowledge bases.

## Introduction

People are more robust than software agents because they can fall back on commonsense knowledge when encountering unexpected situations. With the advent of the Web, projects such as WebKB (Craven et al. 2000), KnowItAll (Etzioni et al. 2004), Cyc (Matuszek et al. 2005), YAGO (Suchanek, Kasneci, and Weikum 2007), KNEXT (Gordon, Van Durme, and Schubert 2010), and many others have explored approaches for reading this type of knowledge off the Web. However, Web reading aspects of those projects have typically focused on retrieving explicitly stated knowledge.

A problem with commonsense is that much of it is not explicitly stated on the Web. For example, a statement like "a thirsty person would want water" is shared background knowledge that everyone knows, and thus isn't what people would typically post to the Web. Commonsense projects have used various methods to elicit this additional knowledge, including having it hand-encoded by human experts (Lenat 1995) or volunteers (Singh et al. 2002), inference (Schoenmackers, Etzioni, and Weld 2008) and generalization (Speer, Havasi, and Lieberman 2008).

An additional type of implicit commonsense knowledge, which projects have not typically investigated, is *relation properties*. Consider the following example from Ritter *et al.* (2008):

given: *Brad Pitt,* ***was born in****, Oklahoma*
do we know if: *Brad Pitt,* ***was born in****, California*

If we are given that Brad Pitt was born in Oklahoma (above), our common sense tells us that Brad Pitt was not also born in California. This is because the relation expressed by the phrase *was born in* can be characterized here as a *function* from people's names to their unique birthplaces. Not all relations are functional, for example if Brad Pitt *visited* Oklahoma, that tells us nothing about whether he also *visited* California.

Consider another example:

given: *Eiffel Tower,* ***is located in****, France*
given: *France,* ***is located in****, Europe*
*do we know if: Eiffel Tower,* ***is located in****, Europe*

It is common sense that if the Eiffel Tower is in France and France is in Europe, then the Eiffel Tower is also in Europe because we know that the *is located in* relation is *transitive*. However, just like with the *was born in* example, this is background knowledge that software applications do not have by default in the same way as people do. Because people understand the properties of these relations, it makes them more robust to noise when dealing with the real world.

In addition to functionality and transitivity, there are dozens of other relation properties that people have internalized such as *symmetry* (*f(a,b)* $\rightarrow$ *f(b,a)*, *e.g.*, *is friends with*), *1-to-1* (*e.g.*, *is president of*), *1-to-many* (each element maps to multiple elements, *e.g.*, *is co-advised by*) and *temporal permanence* (whether a relation depends on time). Having all this relation property information for a large set of relations would be a valuable and useful addition to a commonsense knowledge base.

But how can we obtain relation property knowledge? It would be difficult to hand encode all of it because there are a very large number of relations in the world (potentially millions), and for each we are interested in a large and potentially complex set of properties. Also because this knowl-

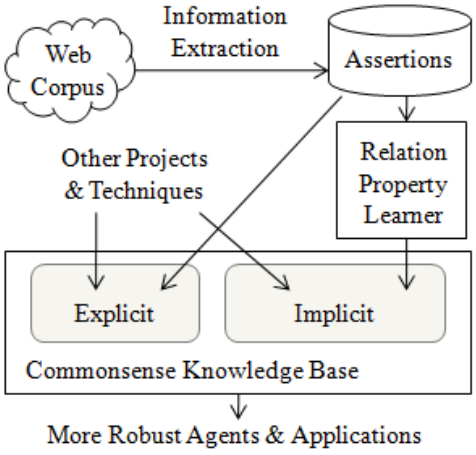


Figure 1: Our larger vision, in which learning Relation Properties provides us with commonsense to enable more robust agents and applications.

edge is implicit, it cannot just be read directly off the Web.

In this paper we propose that relation properties can be automatically *learned* from the Web in a scalable manner. To do this, we first employ Open Information Extraction (Open IE) (Banko et al. 2007) to extract a large set of explicit assertions from the Web. Given these assertions, we can then statistically infer properties from instances, and add the property information to a commonsense knowledge base. Figure 1 illustrates the proposed setup.

For each property that we would like to detect, we can obtain a logical formulation (Popescu 2007). Using *f(a,b)* to denote the relation *f* holding between arguments *a* and *b*, some examples of logical formulations are:

Functional: $\forall a,b,c: f(a,b) \& f(a,c) \rightarrow (b=c)$
Transitive: $\forall a,b,c: f(a,b) \& f(b,c) \rightarrow f(a,c)$

Once we have a logical formulation, we can analyze how often the formulation holds within our corpus of assertions. For example, for *was born in(State)*, we might observe that each arg1 only maps to a single arg2, and therefore the formula for the functionality property holds. However, this is a much more difficult problem than just checking for satisfaction of logical formula because of the large number of non-trivial challenges that we encounter when working with real-world noisy English on the Web. Issues like first argument ambiguity, second argument ambiguity, relation ambiguity, synonyms, hypernyms, extraction error, sparsity on certain topics and misinformation on others are all problems that can affect the accuracy of the method.

As a first step toward learning all interesting relation properties, we develop scalable heuristics for learning the *functionality* property from noisy Web text. Functionality exhibits most of the challenging characteristics of learning general relation properties, and many of the heuristics we explore will also generalize to learning other properties. Also, functionality detection has recently seen many applications such as contradiction detection (Ritter et al. 2008), synonym resolution (Yates and Etzioni 2009), quantifier scope disambiguation (Srinivasan and Yates 2009) and review mining (Popescu 2007).

The primary contributions of this paper are to frame the importance of relation properties as commonsense and describe effective methods for handling the functionality property that can also apply to learning other relation properties. In the next section we describe related work in relation properties and functionality detection. In the third section, we describe the methods we used for functionality detection from noisy Web text. We then compare several techniques via an evaluation on a set of test relations, and show that one of our function systems, named *Mr. Clean*, is able to achieve higher precision and recall than previous work. We conclude by showing how functionality can help on a disambiguation task and providing intuition for how the same techniques used for functionality can also apply to learning other relation properties.

## Related Work

Relations are often expressed using verbs, so an effort to detect properties for relations could conceivably be tied to a large online verb lexicon like VerbNet (Kipper-Schuler 2005). VerbNet does provide valuable syntactic and semantic information about verbs, but it does not currently have the type of property information we are interested in. Moreover, VerbNet contains 4,656 verb senses (5,257 in extended VerbNet) and this misses many of the *relation* strings in Web text. Our Open IE system identifies 16,247 high quality relations (resolved for synonymy, grounded by real-world entities in instances, and occuring multiple times on the Web), and over 1M distinct relation strings between arguments.

AuContraire (Ritter et al. 2008) is a contradiction detection system that also has a functional relation detection aspect. Ritter *et al.*'s idea is to determine a list of functional relations (*e.g.*, *wasBornIn*), and use that to find contradictions (*e.g.*, *Obama was born in 1961* and *Obama was born in 2008* are contradictions). AuContraire emphasizes requiring a lot of hand-engineered knowledge, which limits the scalability of their approach. They include type-specific sources such as geographic knowledge and lists of common first names in order to deal with the problems that arise in natural language functionality detection. Also, for polysemous relations where some arg2 types are functional but others aren't, they classify the entire relation as non-functional without separating types. This led to the vast majority of frequent relations being labeled as non-functional.

Srinivasan and Yates (2009) developed a Quantifier Scope Disambiguation (QSD) system that relied on first making judgments on relation functionality. In order to determine functionality, they look for *numeric phrases* following the relation.

*the fire,* ***destroyed****,* ***four*** *shops*
*the fire,* ***destroys****, the shop*

For example, you can tell that the *destroyed* relation is non-functional because the Web has evidence of something destroying multiple things. One problem with this approach is that it performs poorly for relations that naturally expect numbers as the target argument (*e.g.*, *has an atomic number of*). However, the key difference between their work

and ours is that they end up finding whether relations are functional when time is fixed (*temporal functionality*). This is because their functionality determinations are all based on evidence within single sentences (where time is fixed), rather than by combining evidence from multiple sentences (over which time may vary).

Temporal functionality does not satisfy our logical formulation for functionality. For example in Srinivasan's test set, *destroyed* is non-functional but *destroys* is functional, because you can only destroy one thing at a time but you could have destroyed multiple things in the past. If *destroys* is treated as functional then the following two statements would appear to contradict, but they don't:

*Hurricane Gilbert,* ***destroys****, Jamaica*
*Hurricane Gilbert,* ***destroys****, the Yucatán Peninsula*

While temporal functionality is appropriate for certain applications (*e.g.*, QSD), it does not help with others (*e.g.*, contradiction detection). By combining a temporal functionality detection system with an instance-based functionality detection system using a simple rule-based approach, we can create a classifier that distinguishes between functional, not functional, and temporally functional.

In a preliminary study Popescu (2007) aimed at detecting relation properties, but required human labeled type restrictions on the relations making their approach far less scalable than ours. Moreover, we demonstrate the effectiveness of using existing knowledge resources to this task.

## Scalable Functionality Detection

Following our logical definition $\forall a,b,c:\ f(a,b) \& f(a,c) \rightarrow (b=c)$, functionality is when a relation maps each arg1 to exactly one arg2. For example, *bornIn(Year)* is functional because someone can only be born in one year. Note that some relations that may seem functional at first (*e.g.*, *married*) are actually non-functional by our definition because people can marry multiple people. Functionality can also be considered as a distribution on expected number of arg2s for any arg1. If very few arg2 are expected, this can fall under a type of non-functionality known as approximate pseudo-functionality (Schoenmackers, Etzioni, and Weld 2008) where a relation is still functional enough for scalable inference.

### Acquiring Instances

We want to classify relation properties for as many relations as possible. We choose to get assertions via Open IE because it is able to extract assertions from the Web in a scalable relation-independent method with precision comparable to traditional extraction systems (Banko and Etzioni 2008). We use output from the TextRunner Open IE system (Banko and Etzioni 2008), so our assertions are of the form (arg1, relation, arg2). TextRunner has been run on over 500 million webpages, yielding over 800 million extractions. Our functionality detection method involves checking arg2 values for specific unambiguous arg1 values, so we restrict the assertions to those that have proper nouns as the first argument. We also disallow dictionary words, so this automatically filters out ambiguous arg1 values like "dad" that AuContraire needed EM to detect.

When extracting off the Web, redundancy is a valuable measure for ranking which extractions are more likely to be correct. If the same extraction arises from multiple sentences on different sites, it is more likely to be correct. However, not all redundancy indicates quality. For example, the same news headline ("Saudi collector buys a Picasso") or famous quotation ("a raven is like a writing desk") may appear in many places. A disclaimer sentence could be repeated a thousand times on different pages within a domain. We found that an effective method for filtering duplicate source sentences was to keep a running list of sentence fragments (when split on non-alphanumeric characters) and for any assertion, filter out evidence sentences that share sentence fragments as duplicates.

### Equality Checking and Connected Components

We first design a functionality system where the inputs are (1) a list of relations and (2) the instances (arg1 $\rightarrow$ arg2 mappings) for each of those relations, and the output is a probability that each relation is functional. To determine whether a relation is functional from a set of instances, we check whether it is functional for each distinct arg1 in that set (*local functionality*) by comparing all arg2 values for that arg1. If an arg1 maps to multiple arg2 that are not equal, then this supports non-functionality. If an arg1 maps to multiple arg2 that are all equal, then this supports functionality. As Figure 2 shows, this is a challenging problem because of many sources of noise. To find signal here, we need arg2 equality metrics that are resilient against some noise.

When checking arg2 equality, an important consideration is argument types. If two arg2 are of different types, we don't want to count that as evidence for non-functionality, *e.g.*, *bornIn(1961) vs. bornIn(Hawaii)*. To keep the method scalable, initially we only separate arg2s into types that can be determined without additional background knowledge: proper nouns (with capitalization) *vs.* common nouns *vs.* numbers. If two arg2s for an arg1 are of different types, we do not count that as evidence for either functionality or non-functionality.

Another large source of noise is the synonymy problem where the same concept is expressed in different ways in arg2. A method that we found useful for handling this is to examine all the arg2s for an arg1, and form connected equality clusters among the arg2 that share non-dictionary words. For example, for the query "Bluetooth, was named after, ?", we see arg2 values like (1) *Harald Bluetooth*, (2) *Harald Bluetooth, King of Denmark*, and (3) *the King of Denmark*. Our method connects (1) and (2) via the word *Harald* and then connects (2) and (3) via *Denmark*. It is then able to determine that *Harald Bluetooth* and *the King of Denmark* refer to the same entity without background knowledge like AuContraire required. Setting the arg2 as vertices in an undirected graph, this is the *connected components* problem from graph theory.

This method also helps solve meronymy issues such as locations. We see many cases where a famous person lived in: (1) *Boston*, (2) *Boston, Massachusetts*, (3) *Massachusetts*. Our method can automatically determine that *Boston* and *Massachusetts* refer to the same entity in the scope of the

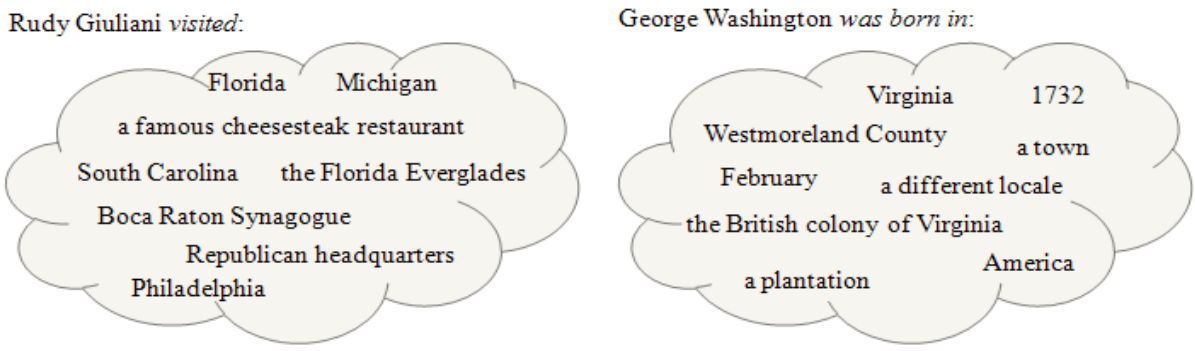


Figure 2: Sample second argument values that we see in Web text for a non-functional relation (*visited*) vs a functional relation (*was born in*) illustrate the challenge in discriminating properties such as *functionality*, *1-to-1* and *1-to-many* from Web text.

relation. Because local functionality looks at arg2 where the arg1 is fixed, we avoid ambiguity problems that we'd normally get with this type of matching. For example, this method would not confuse *Bank of America* and *the river bank*, because those two terms are unlikely to appear in the same arg2 list.

We then combine all the local functionality scores (*e.g.*, the functionality of *"Obama, was born in"*) to determine global functionality (the overall functionality of *was born in*). From a small representative set, we estimate values for $P(b{=}c|Rf)$ and $P(b{=}c|\neg Rf)$ where $b$ and $c$ are any two arg2s with the same arg1 and $Rf$ represents that relation being functional. For each relation we can then use Bayes' rule with our pairwise argument equality determinations over the instances to estimate $P(Rf)$. Our implemented system based on the methods described in this section is named *ConnComp*.

## Clean Lists and Freebase

Connected Components reduces error from synonymys (*Mars*, *the Red Planet*) and meronyms (*Paris*, *France*), but the exact arg2 values needed for it to work will not always be present, and it uses weak typing so it cannot distinguish between different senses of polysemous relations. We design an even stricter argument restriction approach to address those problems - namely, only consider the elements that match elements of predefined type lists.

For the intuition behind this, consider the two examples in Figure 2. If we knew that *visited(State)* and *was born in(State)* were appropriate typed relations, then we could check the functionality of relations in the cases where the arg2 matched on a predefined list of states. In this case we would see evidence of non-functionality for *visited(State)* (Giuliani visited multiple states including Florida, Michigan and South Carolina), and evidence of functionality for *was born in(State)* (George Washington was only born in one state - Virginia).

This method requires that we have enumerated lists of the popular elements for the types that we are interested in. We need the type lists to be *clean*, which we define as: (1) all of the elements are of that type, and (2) each element of the list appears only a single time in the list (so a list of people can have *Hillary Clinton* or *Hillary Rodham Clinton* but not both). To maintain scalability, we get our type lists from Freebase (Metaweb Technologies 2009). Freebase is a pre-existing community-generated freely available online database of over 12 million terms, and their downloadable data includes over 1,000 typed lists. They are also constantly expanding.

This *Clean Lists* method also requires that we know which types are valid as the second argument of our relations. Several methods for finding appropriate arg2 types for each relation include: (1) Hand-specifying the appropriate types for each relation, (2) Using a selectional preferences system *e.g.*, (Ritter, Mausam, and Etzioni 2010) to determine the matching types, and (3) Matching the observed arg2 against the type lists to determine which types are most prevalent in the arg2 set.

Mapping our relations to Freebase in this method also gives us some additional benefits. First, our relations are now typed, so we can identify polysemous relationships and come up with functionality determinations for each of their senses. This is important because for relations like weighs, senses like *weighs(Weight)* are functional while senses like *weighs(Option)* are not. Second, once the relations are associated with Freebase types, they can be connected to existing systems that use Freebase or WordNet (through Freebase-WordNet mappings). Third, Freebase also gives us a good way to filter remaining ambiguous arg1: filter arg1s that match on multiple elements in a type list. For example, "George" is ambiguous because it matches many elements within the people list (*e.g.*, George Bush, George Will).

A potential downside of mapping to Freebase is sparsity issues when the arg2 value doesn't exactly match the way it is expressed in a type list. One technique for addressing this is to admit not only exact string matches to our type lists, but also matches where the arg2 is an element of the type list plus one additional preceding word (*e.g.*, so that "the United States" can match "United States"). We make sure the additional word is not a preposition (*e.g.*, "over" or "near") because that changes the arg2 semantics. Another way to combat sparsity is to introduce more data. Our corpus is the Web, so more data can be easily obtained as needed.

We implement Clean Lists in a system named *Mr. Clean*.

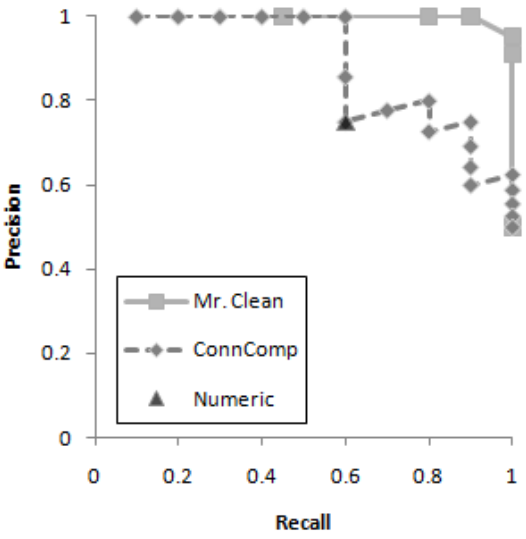


Figure 3: A comparison of *Mr. Clean*, *ConnComp*, and *NumericPhrases* on a test set for Functionality Detection. *Mr. Clean* achieves the highest precision/recall.

The input to *Mr. Clean* includes (1) a list of relations, (2) their instances, and (3) type lists from Freebase. The output includes a list of typed relations, and for each, a "% Functional" determination that corresponds to the % of arg1 that are locally functional for that typed relation.

## Evaluation

The "% Functional" output from *Mr. Clean* can be used to rank or classify relations. We found that because of extraction error and misinformation on the Web, functional relations tended to be in the 90-100% functional range, rather than 100%. Varying the threshold needed for functionality allows us to create a precision/recall curve.

We also implement a version of Srinivasan's numeric phrases system using singular pronouns in arg1, their exact list of numeric quantifiers, and the extraction pattern they used that corresponds best to relations of the form we consider. We ran that system, which we will refer to here as *NumericPhrases*, over 100 million English sentences from a crawl of high quality webpages.

We compare the precision/recall of *ConnComp*, *Mr. Clean*, and *NumericPhrases* on a set of test set of 20 relations. We chose 10 functional relations (*was born in*, *died in*, *is headquartered in*, *was founded in*, *assassinated*, *weighs*, *represents*, *comes from*, *is named after*, *was published in*) and 10 non-functional relations (*visited*, *traveled to*, *married*, *was banned in*, *attacked*, *sang*, *painted*, *acquired*, *looks like*, *wrote*). We chose these relations because their functionality determinations are fairly clear, and they are all very common relations so TextRunner has ample data.

The results in Figures 3 and 4 show that on this test set, *Mr. Clean* is able to achieve higher precision/recall than the other methods, and is also able to identify more typed relations from the original set. An example of where *NumericPhrases* did not work is the *looks like* relation. While it is possible for someone to look like multiple other people, it isn't stated explicitly in text with numeric phrases (nobody says "he looks like 2 celebrities.") *Mr. Clean* is the only system that can add types to relations - it can distinguish relations like *wrote(Song)* from *wrote(Person)*, or *wasBornIn(Year)* from *wasBornIn(Country)*. *Mr. Clean* appears to be the best, but note that it only works for relations where the arguments fit cleanly into type lists. For relations that don't (*e.g.*, *considered*), we should use *ConnComp* and *NumericPhrases*.

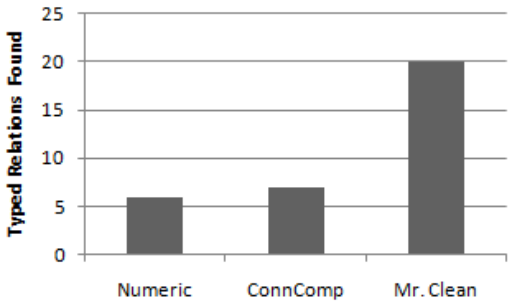


Figure 4: Clean Lists finds many more functional typed relations.

## Finding Prominent Senses using Functionality

To illustrate the utility of knowing functionality, we show here how it helps with a disambiguation task. The problem here is: "When a term belongs to multiple type lists, which sense is the most prominent?" For example, Freebase classifies *Oregon* as both a state and a city because it is also the name of a small city in Wisconsin. Is the primary sense of *Oregon* the state sense or the city sense?

One way to approach this problem is to count inlinks to Wikipedia articles to determine prominent senses (Fader, Soderland, and Etzioni 2009). However, this problem can also be solved with less work using functionality. *Mr. Clean* classifies *wasBornIn(City)* as functional, so if we see that someone was born in both Portland and Oregon and we know that Portland is only a city, then we have reason to believe that the primary sense of Oregon is a state rather than a city because we now know that you would not be born in two different cities. The algorithm is that when we have multiple potential types for an arg2 and a relation that is likely to be functional, we can find the prominent sense by disallowing potential types that are already used by unambiguous arg2 that share the same arg1 values.

As a test set, we consider the situation where we have *wasBornIn* with arg2 types: *city*, *state*, *country*, and *year*. There are 123 wasBornIn arg2 in our corpus that fall into

| | Assign All | Random Assignments | Functional Distribute |
|---|---|---|---|
| Correct | 117 | 58.16 | 95 |
| Incorrect | 131 | 64.83 | 4 |
| Accuracy | 47.18% | 47.29% | 95.96% |

Table 1: Knowing functionality allows us to assign prominent senses to ambiguous terms with much higher accuracy than naïve baselines.

multiple of those type lists. Of these, 121 matched in 2 Freebase lists, and 2 of them matched in 3 lists. There were 5 terms that matched multiple lists but did not fit into any of them. One term matched multiple lists but actually best fit in a qualifying list that it did not match.

Table 1 shows the number of correct and incorrect type assignments that we would get using a baseline of "assign all arg2 to all types," "randomly assign arg2 to qualifying types" (the expected number of values is reported), and from using our functionality method. We see that by taking advantage of functionality information, we are able to do this task very well. There's also a synergistic, recursive nature between this task and typed functionality detection: the better we can do this disambiguation, the more accurate the functionality results. And we can feed in more accurate functionality results for even better disambiguation.

## Conclusions and Future Work

In this paper we proposed relation properties as a valuable type of commonsense knowledge that we can automatically infer at scale by reading the Web. In a case study on the functionality property, we showed that with Open IE and techniques such as filtering, equality checking, connected components and clean lists, we are able to scalably and accurately determine relation functionality. Relation functionality is useful for numerous practical tasks such as contradiction detection, synonym resolution, quantifier scope disambiguation, and the prominent senses task that we showed. Lin *et al.* (2010) extends the discussed techniques and applies functionality detection to a large set of relations to generate a knowledge base of functional relations.

In the future we wish to extend our work to automatically learn other relation properties, as they generally all share similar NLP challenges. The Open IE-based method for acquiring instances is especially valuable for identifying properties such as *transitivity* and *functionality* that can best be detected via algorithms that incorporate tuples from varying sources. It is also useful as an independent method (with independent errors) for properties like *symmetry* where lexico-syntactic patterns may seem more natural. Precise equality checking techniques, such as connected components and other heuristics we used, are vital for identifying properties like *1-to-many*. Drawing in structured type information, such as from Freebase, to clean the data and type the relations will be helpul toward identifying almost all relation properties because it is so effective at reducing the noise from natural language.

The relation properties we end up learning for a wide range of relations and properties will be valuable implicit knowledge for a commonsense knowledge base to enable more robust software agents and applications. Harking back to the examples from the introduction, we envision a future where general purpose AI programs all have the common sense to figure out that Brad Pitt could not also have been born in California by knowing *functionality*, and that the Eiffel Tower must be located in Europe because it's located in France, by knowing *transitivity*.

## Acknowledgements

This research was supported in part by NSF grant IIS-0803481, ONR grant N00014-08-1-0431, DARPA contract FA8750-09-C-0179, a NDSEG Fellowship, and carried out at the University of Washington's Turing Center.

## References

Banko, M., and Etzioni, O. 2008. The tradeoffs between open and traditional relation extraction. In *ACL 2008*.

Banko, M.; Cafarella, M.; Soderland, S.; Broadhead, M.; and Etzioni, O. 2007. Open information extraction from the web. In *IJCAI 2007*.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 2000. Learning to construct knowledge bases from the world wide web. In *Artificial Intelligence*.

Etzioni, O.; Cafarella, M.; Downey, D.; Kok, S.; Popescu, A.-M.; Shaked, T.; Soderland, S.; Weld, D.; and Yates, A. 2004. Web-scale information extraction in KnowItAll. In *WWW 2004*.

Fader, A.; Soderland, S.; and Etzioni, O. 2009. Scaling wikipedia-based named entity disambiguation to arbitrary web text. In *WikiAI 2009*.

Gordon, J.; Van Durme, B.; and Schubert, L. 2010. Learning from the web: Extracting general world knowledge from noisy text. In *WikiAI 2010*.

Kipper-Schuler, K. 2005. Verbnet: A broad-coverage, comprehensive verb lexicon. In *Ph.D. thesis. University of Pennsylvania*.

Lenat, D. 1995. Cyc: A large-scale investment in knowledge infrastructure. In *CACM*, volume 38.

Lin, T.; Mausam; and Etzioni, O. 2010. Identifying functional relations in web text. In *EMNLP 2010*.

Matuszek, C.; Witbrock, M.; Kahlert, R.; Cabral, J.; Schneider, D.; Shah, P.; and Lenat, D. 2005. Searching for common sense: Populating Cyc from the web. In *AAAI 2005*.

Metaweb Technologies. 2009. Freebase data dumps. In *http://download.freebase.com/datadumps/*.

Popescu, A.-M. 2007. Information extraction from unstructured web text. In *Ph.D. thesis. University of Washington*.

Ritter, A.; Downey, D.; Soderland, S.; and Etzioni, O. 2008. Itś a contradiction - no, it's not: A case study using functional relations. In *EMNLP 2008*.

Ritter, A.; Mausam; and Etzioni, O. 2010. A latent dirichlet allocation method for selectional preferences. In *ACL 2010*.

Schoenmackers, S.; Etzioni, O.; and Weld, D. 2008. Scaling textual inference to the web. In *EMNLP 2008*.

Singh, P.; Lin, T.; Mueller, E.; Lim, G.; Perkins, T.; and Zhu, W. 2002. Open Mind Common Sense: Knowledge acquisition from the general public. In *ODBASE 2002*.

Speer, R.; Havasi, C.; and Lieberman, H. 2008. AnalogySpace: Reducing the dimensionality of common sense knowledge. In *AAAI 2008*.

Srinivasan, P., and Yates, A. 2009. Quantifier scope disambiguation using extracted pragmatic knowledge: Preliminary results. In *EMNLP 2009*.

Suchanek, F.; Kasneci, G.; and Weikum, G. 2007. Yago: A core of semantic knowledge unifying Wordnet and Wikipedia. In *WWW 2007*.

Yates, A., and Etzioni, O. 2009. Unsupervised methods for determining object and relation synonyms on the web. In *JAIR*, volume 34, 255–296.