Submitted 5/2019; published 8/2019

Disambiguation for Semi-Supervised Extraction of Complex Relations in Large Commonsense Knowledge Bases

Abhishek Sharma ABHISHEK@CYC.COM
Keith M. Goolsbey GOOLSBEY@CYC.COM
Dave Schneider DAVES@CYC.COM

Cycorp Inc., 7718 Wood Hollow Drive, Suite 250, Austin, TX 78731

Abstract

Cognitive systems require extensive commonsense knowledge to perform well in real-world tasks. Extracting knowledge from text to populate knowledge bases (KBs) would decrease the cost of knowledge engineering and expedite the process of building KBs. However, AI systems must address several disambiguation tasks to successfully learn useful knowledge from the Web. In this work, we propose two methods: (i) We discuss how contents of the Cyc knowledge base could be used to design a similarity-based disambiguation scheme to extract knowledge from text; (ii) We show how a large Web-scale corpus could be used with the Cyc knowledge base to aid in disambiguation tasks. Preliminary results show that these methods can extract complex relations from text with good accuracy.

1. Introduction

To ensure that AI systems can exhibit high levels of performance on complex real-world tasks, extensive knowledge about the world is first required (Feigenbaum 2003). After decades of research in knowledge-based systems, we are at a stage when learning should begin to accelerate due to the amount of knowledge already known to AI systems (Lenat & Feigenbaum 1991). Therefore, AI systems should expand the depth and breadth of their knowledge bases by acquiring more through reading from texts and the Web.

To build a large knowledge base from text repositories, AI systems need to understand semantic relations between concepts in texts. This relation extraction task aims to learn triples of the form $r(e_1, e_2)$, where r is a relation, and e_1 and e_2 are concepts. Because the supervised learning of relation extractors is expensive, researchers have focused on semi supervised learning of relations. These methods use a small number of seed examples from the KB and generate a training set. Given a triple $r(e_1, e_2)$, all sentences that mention both e_1 and e_2 are included as the training instances for the relation *r*. However, such methods can have low accuracy because some of these sentences might represent a relation that is unrelated to *r*.

In this paper, we propose that we can achieve higher levels of accuracy by leveraging Cyc's existing commonsense knowledge. In particular, we propose two methods: (i) We design a plausibility heuristic that ranks existing interpretations by their similarity to known facts in the knowledge base - interpretations that are highly similar to known facts are considered more likely to be true; (ii) we suggest another similarity based heuristic that searches for tuples similar to a given interpretation in a large corpus. Interpretations with a large number of matches are assessed to be more likely to be true.

We start by providing background to the Cyc knowledge base. Then, we discuss the semi-supervised pattern generation. We further discuss the two heuristics for disambiguation in the next section. We conclude after discussing experimental results.

2. Background

We assume a basic familiarity with the CycL language (the Cyc representation language) (Lenat & Guha 1990). In Cyc, concepts are represented as collections. For example, "Cat" is the collection of all cats and only cats. The subsumption relation between concepts is represented by the "genls" relation. For example, (genls Cat NonHumanAnimal) holds. Given its subsumption hierarchy, Cyc can compute the "nearest" generalization of any collection. (nearestGenls A B) holds if (genls A B) holds and there is no collection B' distinct from B such that both (genls A B') and (genls B' B) hold. Two collections A and B, are called *sibling collections* if there exists a collection C such that both (nearestGenls A C) and (nearestGenls A B) hold.[1]

Further, each instance of "Predicate" is either a property of things, or a relationship holding between two or more things. For example, (doneBy EmbargoAgainstIranByUS-1987 UnitedStatesOfAmerica) holds. Semantic well-formedness of assertions is mainly specified by two types of constraints: (i) argument type constraints, which specify argument types directly and impose constraints that require a certain argument being an instance of a specified collection. "argIsa" is an example of an argument type predicate. (argIsa P N COL) means that the nth argument of P must be an instance of COL (e.g., (argIsa doneBy 1 Event)) (ii) Inter-arg predicates specify constraints on polyadic relations in regard to two of their argument places. Each instance of inter-arg predicate specifies, with respect to a given relation, either that one of its argument places has a certain feature contingent upon one of its different argument places having a certain feature ("interargCondIsa1-2" is an example of inter-arg predicate. (interArgCondIsa1-2 P A B C) means that if (P A1 B1) holds, and A1 is an instance of A, and B1 is an instance of B, then B1

[1] For example, Dog is a sibling collection of Wolf because NonPersonAnimal is the common nearest common generalization of both collections.

would also be an instance of C. For example, (interArgCondIsa1-2 performedBy PsychologicalTreatment Doctor-Medical Psychiatrist) holds.)

Cyc maintains an estimate of a generality of every term in its KB. The leaf nodes of the ontology have a generality estimate of zero, whereas the root node has the highest generality estimate. These estimates satisfy the constraint that Generality (g) > Generality (s), if g is a generalization of s.

3. Pattern Extraction

Our algorithm for learning relation extractors is shown in Figure 1. The input to the learning system is the Cyc knowledge base, a target relation to be learned, and the desired number of relation extractors. Let us assume that we seek to learn facts involving the predicate makesProductType.[2]

Algorithm 1: Pattern Extractor

Input: A Knowledge Base (**KB**), a corpus **C**, a predicate (**P**), and desired number of patterns (**N**).

Output: Top **N** patterns for the predicate **P**.

```
1. for all facts of type (P arg1 arg2) in the KB:
      a. for (a, b) in NLRendition (arg1) × NLRendition(arg2)
            i.  S ← Sentences that mention both a and b in the corpus.
            ii. for all sentences s in S:
                  1. p ← Intervening sequence of words that occur between a and b.
                  2. Increment frequency of occurrence of pattern p.
2. Sort the patterns by frequencies and return the top N.
```

In Step 1 of the algorithm, we will iterate through the GAFs involving the predicate P and find sentences in the corpus[3] that mention the natural language rendition of both arguments[4]. Consider

[2] (makesProductType ORG TYPE) means that an organization ORG manufactures products of type TYPE. For example, (makesProductType AppleInc IPhone-AppleCellphone) holds.
[3] We use the Clueweb corpus [Callan & Hoy 2009] for the experiments reported in this work.
[4] The function "NLRendition" uses Cyc's natural language generation capabilities to generate a natural language rendition of the given term.

the assertion (makesProductType AppleInc IPhone-AppleCellphone). Cyc's natural language generation module would map the arguments of the aforementioned assertion to "Apple" and "iPhone," respectively. In Step 1.a.i, we find sentences mentioning these terms in the corpus. An example match is shown below.

Apple's iPhone still holds commanding lead in smartphone sales in Japan. …(S1)

Table 1: Examples of top patterns found by Algorithm 1. The third column shows some phrases that match one of the patterns shown in the second column.

Predicate	**Top features**	**Examples of phrases**
(makesProductType A B)	A's B, B from A, B is manufactured by A, B is produced by A.	Apple's iphone, iphone from Apple.
(physicalPartTypes A B)[5]	A's B, A of the B, A with B, A have B, A has B, A use their B.	Bear's claw, leg of the pig.
(typePrimaryFunction-DeviceUsed A B)	B in A, A for B, A is used to B.	bake in oven, bomber for bombing mission, needle used to give shots
(typeBehaviorCapable-PerformedBy A B)[6]	A B, B by A, A will B, A are B-ing, A can B	Kangaroo springs, bears are growling, bears can walk
(typicalLocationTypeOfEventType A B)[7]	A in the B, B for A	sleep in the bedroom, clinic for abortion

In the next step, we increment the frequencies of occurrence of the intervening pattern between the arguments of relation. For example, the sentence S1 will increment the frequency of pattern "A's B." Examples of patterns identified by Algorithm 1 are shown in Table 1. Due to the

[5] (physicalPartTypes A B) means that every instance of A has at least one instance of B as a physical part. For example, (physicalPartTypes Hand Finger) holds.

[6] (typeBehaviorCapable-PerformedBy A B) means that objects of type A can be performers in situations of type B due to their intrinsic properties. For example, (typeBehaviorCapable-PerformedBy Frog Jumping) holds.

[7] (typicalLocationTypeofEventType A B) means that for any instance of event type A, it is reasonable to assume, in the absence of information to the contrary, that A occurs at an instance of B. For instance, (typicalLocationTypeOfEventType Battle BattleZone) holds.

absence of negative examples, some frequently occurring noun phrases are among the top phrases [8]. These common phrases can be quite useful; however they are ambiguous[9]. In the next section, we propose a disambiguation scheme to resolve such ambiguity.

4. Knowledge Based Disambiguation

The patterns generated by the algorithm discussed above are used to generate lexical tuples that could be potentially translated into assertions.The knowledge-based disambiguation scheme (shown below) takes as input a lexical tuple, and performs disambiguation by comparing candidate interpretations to existing knowledge in the KB. The interpretations that are most similar to the contents are deemed to be the most plausible. We illustrate this approach with the help of a sample execution.

One of the patterns for the "makesProductType" predicate could lead to the tuple (Apple, mouse), which can have the following interpretations[10]:

(makesProductType AppleTree ComputerMouse)*

(makesProductType (FruitFn AppleTree) Mouse-Rodent)*

(makesProductType AppleInc ComputerMouse)

(makesProductType AppleInc Mouse-Rodent)*

In this section, we present a disambiguation scheme that uses the Cyc knowledge base to analyze existing assertions in the KB; further, it associates the most appropriate senses with the words. The input to our algorithm (see Algorithm 2 below) is a lexical tuple and a generalization threshold (as discussed below); and it returns top N interpretations for the input lexical tuple. In Step 1, S_1 and S_2 are the sets of all senses for words w_1 and w_2 respectively[11]. We initialize the scores for all possible interpretation to zero in the next step. In Step 3, we iterate over all assertions in the KB and compute the match score for all possible interpretations. In the final step, we return N interpretations that have the highest score. The performance of this disambiguation scheme depends on the *MatchScore* function, which we describe next.

[8] Although negative examples would improve the quality of patterns, we have found that it is hard to find *relevant* negative examples for general predicates. For taxonomic relations (e.g., 'isa' and 'genls'), we have found that sibling collections can be used to provide quite relevant examples. For example, while learning patterns for extracting instances of River, we could use instances of Desert to generate negative examples.

[9] For example, the pattern "A's B" could be translated to both makesProductType and physicalPartTypes sentences (see examples in Table 1). It can also be translated into an 'owns' assertion (e.g., John's shirt).

[10] Starred interpretations are incorrect.

[11] For simplicity, we include the most specific collections of individuals in the list of their senses. For example, "IBM" would be mapped to the terms {IBMInc, Business, SoftwareVendor, …}.

Algorithm 2: Knowledge-based Disambiguator

Input: A lexical tuple (w_1, w_2); a generality threshold $\boldsymbol{\alpha}$

Output: A list of best N interpretations of type (p_i, w_{1i}, w_{2i})

```
1. $S_1$ ←AllSenses ($w_1$) , $S_2$ ←AllSenses ($w_2$), *Predicates* ←All predicates in KB
2. score ($p, s_1, s_2$) ← 0  $\forall$ p ∈ *Predicates*, $s_1 \in S_1$ , $s_2 \in S_2$
3. *for* ($s_1$ , $s_2$) in $S_1 \times S_2$:
    a. *for* assertion *a* in KB:
        i. Increment score ($p, s_1, s_2$) by MatchScore ($a, s_1, s_2, \boldsymbol{\alpha}$) where *p* is the binary predicate in *a*.
4. Return top N ($p, s_i, s_j$) tuples.
```

Algorithm 2a (shown below) describes the MatchScore function. It takes as input an assertion *a* and two concepts s_1 and s_2. It returns an estimate of the similarity between the arguments of assertion a, and the input concepts. For example, to assess the plausibility of (makesProductType (FruitFn AppleTree) Mouse-Rodent), we compare its similarity with assertions in KB, e.g., (makesProductType NissanTheCompany ElectricCar). In this case, s_1 and s_2 will be set to (FruitFn AppleTree) and Mouse-Rodent, respectively. After initializing the local variables in step 1, we iterate over all generalizations of arguments of assertion *a* in Step 2. Therefore, g_1 and g_2 will range over all generalizations of NissanTheCompany and ElectricCar respectively. The input concepts ((FruitFn AppleTree) and Mouse-Rodent) will not be subsumed by any of the immediate or specific generalizations of Nissan and electric cars. The condition in Step 3 is satisfied when g_1 and g_2 are SolidTangibleThing. However, SolidTangibleThing is a fairly general collection and the condition in Step 4 will not be satisfied[12] for small values of $\boldsymbol{\alpha}$[13].

[12] When condition 3 is satisfied, then g_1 is a common feature of a_1 and s_1, and g_2 is a common feature of a_2 and s_2. When these common features are too general, then condition 4 is not satisfied.

[13] The generality estimates of Mouse-Rodent and SolidTangibleThing are 21 and 655,000. In this paper, $\boldsymbol{\alpha}$ was set to 100. Therefore, the condition in Step 4 was not satisfied.

Algorithm 2a: MatchScore

Input: An assertion in KB, *a*. Concepts, s_1 and s_2. Threshold $\boldsymbol{\alpha}$

Output: An estimate of the similarity of the arguments of a to concepts s1 and s2.

```
1. Let p ←Predicate (a), $a_1$ ←Argument 1 of a, $a_2$ ←Argument 2 of a, score ←0
2. for ($g_1$, $g_2$) ← All generalizations of $a_1$ × All generalizations of $a_2$
3.     if $s_1$ is a specialization of $g_1$ and $s_2$ is a specialization of $g_2$
4.         if Generality ($g_1$) / Generality ($s_1$) < $\alpha$ and Generality ($g_2$) / Generality ($s_2$) < $\alpha$
5.             score += Generality ($s_1$) * Generality ($s_2$) / Generality ($g_1$) * Generality ($g_2$)
6. return score
```

On the other hand, when we assess the plausibility of (makesProductType AppleInc ComputerMouse), s_1 and s_2 will be set to AppleInc and ComputerMouse, respectively. In Step 3a of Algorithm 2, we will iterate over all assertions in KB and compare them to our hypothesis interpretation. Let us consider the assertion (makesProductType IBMInc ComputerHardwareItem). Since both IBMInc and AppleInc are instances of a specific collection 'Business', the first condition in Step 4 would be satisfied. ComputerHardwareItem would be one of the values of g_2, and the second condition in Steps 3 and 4 would be satisfied as well. The score returned by the function is inversely proportional to the generality of common features between the input concepts and assertions in KB. Algorithm 2 and 2a leverage the information already known to Cyc KB to classify new information as plausible. However, even the largest of KBs have gaps, and the similarity-based approach would fail to identify plausible assertions if the KB does not have relevant information. To alleviate this problem, we propose another approach that uses a large corpus to estimate the plausibility of interpretations.

5. Corpus Based Disambiguation

The corpus-based disambiguation algorithm (shown below) takes as input two concepts s_1 and s_2, and a pattern *p*. It is expected to provide an estimate of the plausibility of s_1 co-occurring with the concept s_2. The algorithm first finds concepts that are similar to s_1 and s_2. To accomplish this, it limits the search to the sibling concepts of s_1 and s_2. Next, it finds whether these similar concepts co-occur with the pattern *p* in the Cluweweb corpus (Callan & Hoy 2009). A large number of search results leads to a higher plausibility estimate. We illustrate the workings of this algorithm with a sample execution.

Consider the phrase “walls of the temple”. In the disambiguation phase, we would like to infer that the fact (physicalPartTypes Temple WallOfAConstruction) is plausible. However, due to gaps, the KB might not have concepts similar to walls that are parts of concepts similar to temples. In Step 1 of the algorithm, g_1 and g_2 would range over concepts similar to Temple and WallOfConstruction, respectively. In one of the iterations, g_1 would be bound to Church, and g_2 would be bound to RoofOfAConstruction. In Step 4, we would search for the phrase “roof of church”, and several instances of this phrase would be found in any large web-scale corpus[14]. This is repeated for all sibling concepts of s_1 and s_2, and the average score (over all values of g_1 and g_2) is returned in Step 6.

Consider another candidate, i.e., “top of the shoe”. An erroneous interpretation of this phrase would be (physicalPartTypes Shoe Clothing-Top). In one of the iterations of Step 1, g_1 and g_2 would be set to Sock and Toga respectively. In Step 5, we would search for the phrase “toga of the sock”. This phrase (and other phrases generated from the sibling concepts of Shoe and Clothing-Top) are not found in the corpus, which leads to a low score for the aforementioned erroneous interpretation.

Algorithm 3: Corpus-Based Disambiguator

Input: Concepts s_1, s_2. A pattern *p*.

Output: An estimate of the plausibility of concept s_1 co-occurring with s_2.

```
1. for $(g_1, g_2) \leftarrow$ SiblingConcepts $(s_1) \times$ SiblingConcepts $(s_2)$
2.   Score $(g_1, g_2) \leftarrow 0$
3.   for $(w_1, w_2) \leftarrow$ NLRendition $(g_1) \times$ NLRendition $(g_2)$
4.       phrase $\leftarrow w_2\ p\ w_1$
5.       Score $(g_1, g_2) \leftarrow$ Score $(g_1, g_2)$ + NumOccurrences (phrase)
6. Return Average $_{(g1, g2)}$ (Score $(g_1, g_2)$).
```

[14] Table 2 shows some more examples of similar phrases found by executions of Algorithm 3. The function NumOccurrences (phrase) in Step 5 returns the number of occurrences of input phrase in the corpus.

Table 2: Examples of Similar phrases found in the corpus

Interpretation	**Pattern**	**Examples of similar phrases in corpus**
(physicalPartTypes DrinkingMug Handle)	A of B	base of teapot, handle of pot
(physicalPartTypes Shirt CollarOfGarment)	A of B	pocket of blouse
(properPhysicalPartTypes Submarine Hull)	A of B	deck of aircraft carrier

6. Experimental Evaluation

In this section, we discuss the experimental evaluation of these methods. Our choice of experiments was guided by three principles: (i) AI systems must learn complex relations from text to create realistic KBs. Therefore, instead of learning relations that can be extracted from structured information, we focus on relations that are implicitly stated in the text. (ii) Some previous work on relation extraction has failed to address the problem of synonym resolution and disambiguation (e.g., see (Carlson et al 2010a)). Therefore, we focus on the full interpretation cycle, i.e. we resolve strings to real-world entities and learn ground atomic formulas in first-order logic. (iii) Finally, because we are interested in populating large commonsense KBs, we focus on learning general knowledge. While learning instance level knowledge is important, we believe that learning type-level information is more critical for building general purpose AI systems[15]. We tested these methods on learning relations involving three predicates: physicalPartTypes, typeBehaviorCapable-PerformedBy and typicalLocationTypeOfEventType. For each of these predicates, we performed the following steps:

a. We used Algorithm 1 to extract patterns for extracting ground atomic formulas involving these predicates.
b. We then use Cyc's existing constraint checking mechanism to identify inconsistencies. For a given formula, Cyc uses argument constraints to identify well-formed formulas. Then we use deductive reasoning to identify additional inconsistencies, e.g., a formula P is deemed to be implausible if Cyc can prove ~P. This approach is referred to as 'baseline' in Table 3.

[15] Most of the predicates shown in Table 1 of (Carlson et al 2010b) represent information about individuals (e.g., athletePlaysSport, tvStationInCity).

c. We then used the KB-based disambiguation scheme (Algorithm 2a) to identify plausible interpretations of tuples that matched extracted patterns. The output of Steps (b) and (c) is referred to as "Baseline+KBDS" in Table 3.
d. In the final step, we used the corpus-based disambiguation scheme to identify plausible GAFs. In Table 3, the output of Steps (b), (c) and (d) is referred to as "Baseline+KBDS+CBDS."

Table 3: Experimental Results

Experiment	Method	Number of GAFs evaluated	Accuracy(%)
1	Baseline	250	14.8
	Baseline+KBDS	250	78.0
	Baseline+KBDS+CBDS	250	80.0
2	Baseline	250	12.8
	Baseline+KBDS	250	66.8
	Baseline+KBDS+CBDS	250	70.8
3	Baseline	250	18.0
	Baseline+KBDS	250	72.0
	Baseline+KBDS+CBDS	250	73.6

Although we used universally quantified sentences for extracting patterns, the output of these methods are not universally quantified sentences. Therefore, occurrence of a phrase like "submarine's nuclear reactor" in multiple documents does not justify the universally quantified sentence (physicalPartTypes Submarine NuclearReactor). Instead, we translate the output to (relationExistsManyExists physicalParts Submarine NuclearReactor).[16]

The pattern extraction process (Step a) was used to collect data for 3 days. This process led to identification of an average of 60,972 tuples per predicate. Then we used Cyc's consistency checking mechanism, along with knowledge and corpus-based disambiguation schemes, to identify plausible tuples (this corresponds to Steps b, c and d). These algorithms classified an

[16] Informally, this can be translated to "Nuclear reactors are a physical part of many submarines."

average of 32100, 2100 and 2000 tuples per predicate as plausible. Out of these, 250 were selected at random for manual accuracy evaluation. The accuracy of these tuples is shown in Table 3. We investigated the inaccurate tuples (from the final output produced by the "Baseline+KBDS+CBDS" method) and found that 33% of inaccuracies were due to the generality of patterns, whereas the rest were due to too-eager generalization performed by the knowledge-based disambiguation approach.

We conclude the following based on the experimental results (a) The knowledge-based disambiguation method leads to statistically significant improvement in accuracy ($p < 0.05$), and is quite effective in learning high-quality assertions from the text. (b) We see that the methods used in the baseline version were less effective in enforcing consistency. This is due to (i) hardness of deductive reasoning (i.e., we relied on proving negated sentences to identify inconsistencies, and most of these queries were timed out); (ii) broadness of argument constraints (i.e., many general predicates have quite broad argument constraints, and they have limited utility in identifying plausible scenarios); and (iii) insufficient inter-arg constraints (i.e., the results show that the KB does not have sufficient inter-arg constraints to enforce consistency in many cases. (c) Finally, we see that the corpus-based disambiguation method had only a small effect on improving accuracy. Due to the large number of sibling concepts, this algorithm was often timed out. We plan to address these problems in our future work.

7. Related work

Over the last decade, the problem of building large commonsense knowledge bases from large text corpora has received lots of attention (Carlson et al. 2010a, Carlson et al. 2010b). In (Pennacciotti & Pantel 2009), the authors use ensemble statistics to learn category instances. In (Downey, Etzioni & Soderland 2005), the authors presented a probabilistic model for learning extractors. In (Pasca et al. 2006) and (Chang, Ratinov and Roth 2007), the authors use argument constraints to check the validity of arguments. Researchers have also shown that pattern-based and list-based extraction methods are useful in information extraction tasks (Etzioni et al . 2004). In recent years, researchers have used deep learning to extract relations (Fu, Li & Ma 2019, Trisedya et al. 2019, Dixit & Al-Onaizan 2019, and Deng & Liu 2018), and in (Ling, Clark and Weld, 2013), the authors have used distant supervision to learn meronyms.

As mentioned above, the approach presented here is different from previous research in this area in two ways: (a) Unlike previous work that did not address the problem of resolving strings to real-world entities (Carlson et al. 2010a), we present an approach to learn ground atomic formulas for FOL KBs. This involves different disambiguation tasks. Several word-sense disambiguation algorithms exist (Navigli and Velardi 2005, Navigli 2009), but to the best of our knowledge, none of them have presented a similarity-based approach that fully leverages the concept hierarchy and other assertions in the KB. (b) While learning relations that represent information about individuals is important, we believe that learning general knowledge at type-level is critical for building general-purpose KBs. Therefore, we have shown that it is possible to learn predicates that represent relations between types from texts.

8. Conclusions

After decades of research in knowledge-based systems, we are at a stage when the knowledge already known to an AI system should expedite learning and construction of even larger KBs. However, learning from text requires design and implementation of several disambiguation methods. In this work, we have proposed two methods for this task: the first method flags a GAF as plausible if it is similar to already known facts in the KB. The second method uses a large corpus to verify that information similar to our interpretation occurs in text. We evaluated these methods on learning relations between collections (i.e., types). Such relations are often implicitly mentioned in texts. Preliminary results show that our methods are quite effective in extracting relations. However, there are many opportunities for improvement, including: (1) Performing larger scale experiments to better understand the properties of our learning algorithms; (2) including negative examples in our training set, as they would lead to higher-precision patterns; (3) analyzing trends in accuracy as we learn more facts from text repositories; (4) improving the performance of corpus-based disambiguation method and (5) improving the coverage of inter-arg constraints.

References

Callan, J., and Hoy, M. (2009). Clueweb09 data set. https://lemurproject.org/clueweb09/

Carlson, A., Betteridge, J., Wang, R. C., Hruschka, E. and Mitchell, T. (2010a). Coupled Semi-Supervised Learning for Information Extraction. *Proceeding of the WSDM.*

Carlson, A. Betteridge, J., Kisiel, B., Settles, B., Hruschka, E., and Mitchell, T. (2010b). Toward an Architecture for Never-Ending Language Learning, *Proceedings of AAAI.*

Chang, M.-W.; Ratinov, L.-A.; and Roth, D. (2007). Guiding semi-supervision with constraint-driven learning. *Proc. of ACL.*

Deng, Li; and Liu, Y. (2018). *Deep Learning in Natural Language Processing*. Springer.

Dixit, K.; and Al-Onaizan. (2019). Span-Level Model for Relation Extraction. *Proc. of ACL.*

Downey, D.; Etzioni, O.; and Soderland, S. (2005). A Probabilistic Model of Redundancy in Information Extraction. *Proc. of IJCAI.*

Etizioni, O.;Cafarella, M.;Downey, D.; Popescu, A.-M.; Shaked, T.; Soderland, S.; Weld, D. S.; and Yates, A. (2004). Methods for domain-independent Information Extraction from the Web: An Experimental Comparison. *Proc. of AAAI.*

Feigenbaum, E. (2003). Some Challenges and Grand Challenges for Computational Intelligence. *Journal of the ACM,* 50 (1), 32-40.

Fu, T.; Li, P.; and Ma. W. (2019). GraphRel: Modeling Text as Relational Graphs for Joint Entity and Relation Extraction. *Proc. of ACL.*

Lenat, D. B. and Feigenbaum, E. (1991). On the Thresholds of Knowledge, *Artificial Intelligence*, 47, 185-250.

Lenat, D. B. and Guha, R. (1990). *Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project*. Addison Wesley.

Ling, X.; Clark, P.; and Weld, D. (2013). Extracting Meronyms for a Biology Knowledge Base Using Distant Supervision. *Proc. of AKBC,* San Francisco, CA.

Navigli, R. (2009). Word Sense Disambiguation: A Survey. *ACM Computing Surveys.* 41 (2), 1-69.

Navigli, R. and Velardi, P. (2005). Structural Semantic Interconnections: A Knowledge-Based Approach to Word Sense Disambiguation. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 27 (7), 1075-1086.

Pasca, M.; Lin, D.; Bigham, J.; Lifchits, A.; and Jain, A. (2006). Names and Similarities on the Web: fast extraction in the fast lane. *Proc. of ACL.*

Pennacchioti, M., and Pantel, P. (2009). Entity Extraction via Ensemble Semantics., *Proc. of EMNLP.*

Trisedya, B. D.; Weikum, G.; Qi, J.; and Zhang, R.. (2019). Neural Relation Extraction for Knowledge Base Enrichment. *Proc. of ACL.*