Knowledge Authoring for Rule-based Reasoning

Tiantian Gao, Paul Fodor, Michael Kifer

Department of Computer Science
Stony Brook University, Stony Brook, NY, USA
{tiagao,pfodor,kifer}@cs.stonybrook.edu

Abstract. Modern knowledge bases have matured to the extent of being capable of complex reasoning at scale. Unfortunately, wide deployment of this technology is still hindered by the fact that specifying the requisite knowledge requires skills that most domain experts do not have, and skilled knowledge engineers are in short supply. A way around this problem could be to acquire knowledge from text. However, the current knowledge acquisition technologies for information extraction are not up to the task because logic reasoning systems are extremely sensitive to errors in the acquired knowledge, and existing techniques lack the required accuracy by too large of a margin. Because of the enormous complexity of the problem, *controlled natural languages* (CNLs) were proposed in the past, but even they lack high enough accuracy. Instead of tackling the general problem of text understanding, our interest is in a related, but different, area of *knowledge authoring*—a technology designed to enable domain experts to *manually create* formalized knowledge using CNL. Our approach adopts and formalizes the FrameNet methodology for representing the meaning, enables incrementally-learnable and explainable semantic parsing, and harnesses rich knowledge graphs like BabelNet in the quest to obtain unique, disambiguated meaning of CNL sentences. Our experiments show that this approach is 95.6% accurate in standardizing the semantic relations extracted from CNL sentences—far superior to alternative systems.

1 Introduction

Much of human knowledge can be represented as facts and logical rules and then fed into state of the art rule-based systems, such as XSB [23], Flora-2 [12, 28], or Clingo [8], to perform formal logical reasoning in order to answer questions, derive new conclusions and explain the validity of statements. However, human knowledge can be very complex and domain experts typically do not have the training needed to express their knowledge as logical rules, while trained knowledge engineers are, unfortunately, in short supply.

Ideally, one could try to extract the requisite knowledge from text, but this is an extremely complex task. Although impressive advances have been made in text understanding and information extraction (e.g., [16, 2, 9]) the technology is still very far from approaching the accuracy required for logic knowledge bases, which are extremely sensitive to errors (both wrong and missing data).

Controlled natural languages (CNLs) [13]—languages with restricted, yet fairly rich, grammars and unambiguous interpretations—were proposed as a

technology that might help bridge the gap. CNL systems allows domain experts who lack the experience in logic to specify knowledge that can be cast into logical statements suitable for reasoning. Such systems include Attempto Controlled English (*ACE*) [6], Processable English (PENG) [25], and BioQuery-CNL [5]. However, CNL systems perform rather limited semantic analysis of English sentences and do not provide for accurate authoring of knowledge. Specifically, they fail to recognize when sentences have the same meaning but are expressed in different syntactical forms or using different language constructs. For instance, the state of the art system ACE translates the sentences *Mary buys a car*, *Mary is the purchaser of a car*, *Mary makes a purchase of a car*, and many other equivalent sentences into very different logical representations. As a result, if any of these sentences is entered into the knowledge base, the reasoner would fail to answer questions like *Who purchases a car?* or *Who is the buyer of a car?* because ACE and others would translate these questions into logical sentences that are very different from the logical formulas used for the data. Clearly, this is a serious obstacle to using CNL as input to logical reasoning systems. The typically proposed "solution" is to manually specify bridge rules between equivalent forms, but this requires a huge number of such rules and is impractical.

Aim of this work. This work is *not* about text understanding or information extraction from general prose or even from technical manuals. Instead, we propose an approach to *knowledge authoring* with the aim of providing domain experts with tools that would allow them to translate their knowledge into logic by means of CNL. The difference between knowledge authoring and information extraction or knowledge acquisition is quite significant: whereas information extraction aims to enable machines to understand what humans write, knowledge authoring aims to enable humans to write in natural language so that machines could understand. At present, knowledge authoring technology (compared to knowledge acquisition and extraction) is in an embryonic state. Knowledge authoring was, in fact, the target that CNLs were eyeing, but failed to reach because not enough attention was paid to semantics. We believe that our work fills in much of the void left unfilled by CNLs, which will turn the latter into a widely accepted technology for creation of formalized knowledge.

Contributions. The contributions of this paper are four-fold:

(a) A formal, FrameNet-inspired [11] ontology *FrameOnt* that formalizes FrameNet frames and integrates linguistic resources from BabelNet [20] to represent the meaning of English sentences.

(b) An *incrementally-learned* semantic parser that disambiguates CNL sentences by mapping semantically equivalent sentences into the same *FrameOnt* frames and gives them *unique logical representation* (ULR). The parser is layered over the Attempto Parsing Engine (*APE*),[1] and utilizes *FrameOnt*, BabelNet, and our novel algorithms for frame-based parsing and ontology-driven role-filler disambiguation.

[1] `https://github.com/Attempto/APE`

(c) Explainability: the approach makes it possible to explain both why particular meanings are assigned to sentences and why mistakes were made (so they can be fixed).
(d) We developed the Knowledge Authoring Logic Machine (*KALM*)[2] to enable subject matter experts, who need not be proficient in knowledge representation, to formulate actionable logic via CNL. KALM achieves unmatched accuracy of 95.6% in standardizing the semantic relations extracted from CNL sentences—far superior to alternative systems.

Organization. Section 2 gives background information on APE (the Attempto parser), FrameNet, and BabelNet, that is necessary in order to understand the proposed approach and make the paper self-contained. Section 3 describes the framework of KALM, its frame-based parser and the role-filler disambiguation algorithms. Section 4 describes the parallelization and the other optimizations that make role-filler disambiguation feasible. Section 5 describes the explainability aspect of KALM. Section 6 presents an evaluation of our approach, which indicates very high accuracy. Section 7 concludes the paper with a discussion of related work and future extensions.

2 Background

This section provides background on the systems used by KALM; specifically, the *Attempto parsing engine* (APE), the *FrameNet* methodology for representing the meaning of sentences, and the *BabelNet* knowledge graph.

Attempto. KALM accepts sentences that follow the grammar and interpretation rules of Attempto Controlled English, *ACE*.[3] ACE represents the semantics of text in a logical form, called *discourse representation structure* (DRS) [7], relying on seven predicates: `object/6`, `predicate/4`, `property/3`, `relation/3`, `modifier_adv/3`, `modifier_pp/3`, and `has_part/2` (in `p/N`, `N` is the number of arguments in predicate `p`). An `object`-fact represents an entity—a noun-word with some properties (e.g., countable or uncountable, quantity). A `predicate`-fact represents an event—a verb-word and its participating entities. A `property`-fact represents the syntactic relation between a noun and its adjective modifier. A `modifier_adv`-fact represents the syntactic relation between a verb and its adverbial modifier. A `modifier_pp`-fact represents the syntactic relation of a verb, its prepositional modifier and its prepositional complement. A `relation`-fact represents the genitive relation between two noun-words. For conjunctions of noun phrases, the Attempto Parsing Engine, APE, uses an additional predicate, `has_part/2`, to represent the grouping of these entities. Each `object`-, `predicate`-, or `has_part`-fact has a unique identifier. Lastly, each fact has an *index* (e.g., `-1/2` in the first fact) showing the position of the word in the original sentence that generates this fact. For example, the sentence *A customer buys a watch for a friend* is represented using Prolog terms of the form:

```
object(A,customer,countable,na,eq,1)-1/2.
object(B,watch,countable,na,eq,1)-1/5.
```

[2] `https://github.com/tiantiangao7/kalm`
[3] `http://attempto.ifi.uzh.ch/site/docs/syntax_report.html`

```
object(C,friend,countable,na,eq,1)-1/8.
predicate(D,buy,A,B)-1/3.
modifier_pp(D,for,C)-1/6.
```

where `A`, `B`, and `C` are identifiers that represent the *customer*-, *watch*- and *friend*-entities, respectively, and `D` the *buy*-event. In each `object`-fact, the second argument represents the stem form of the word the fact represents; the rest of the arguments are the properties of this entity. In each `predicate`-fact, the third argument represents the subject of the event and the fourth argument represents the object of the event. In this case, the identifier `A` (resp. `B`) indicates that *customer* (resp. *watch*) is the subject (resp. object) of the event. A `modifier_pp`-fact connects the *buy*-event and its prepositional complement, the *friend*-entity.

As explained in the introduction, a very serious issue with Attempto Controlled English, ACE, is that sentences that have the same meaning may be represented by very different logical terms, preventing logic engines from making useful inferences from ACE parses. The often proposed workaround to manually build bridge rules is impractical. This paper solves this and related problems.

FrameNet. FrameNet is a knowledge base of semantic relations based on a theory of meaning called *frame semantics* [1]. The meaning of a sentence is understood as a semantic *frame* along with the semantic *roles* that the various words in the sentence play in the frame. FrameNet calls these roles *frame elements*. For example, in a sentence about purchasing goods, the frame `Commerce_Buy`[4] typically involves an individual purchasing a good (i.e., `Buyer`), the items that are purchased (i.e., `Goods`), the thing given in exchange for goods in the transaction (i.e., `Money`), the individual that has the possession of the goods and exchanges them with the buyer (i.e., `Seller`), the individual intended by the buyer to receive the goods (i.e., `Recipient`), the place and time of purchase (i.e., `Place` and `Time`), and so on. A frame is associated with a list of *lexical units* (words plus their part-of-speech). A lexical unit represents the basic language unit in a sentence that can *trigger* an application of the frame. For example, the lexical units `buy.v`, `purchase.v`, `buyer.n`, and `purchaser.n` can trigger the `Commerce_Buy` frame. But not only: the lexical unit `buy.v` can also trigger the frame, `Fall_for`.

Given the flexibility of natural languages, the same lexical unit can be used in multiple ways in sentences that match the same frame. To capture this, each pair (*lexical unit*, *frame*) in FrameNet is associated with a set of *valence patterns*, which represent the syntactic contexts in which a particular lexical unit and some of the frame elements can appear in sentences. For instance, one valence pattern for (`buy.v`, `Commerce_Buy`) says that the `Buyer`-role is the subject of the *buy*-event and the `Goods`-role is the object of that event. Therefore, the sentence *Mary buys a watch* matches the frame `Commerce_Buy` via the lexical unit `buy.v` and the valence pattern that extracts *Mary* as the filler for the `Buyer`-role and *watch* as the filler for the `Goods`-role.

BabelNet. BabelNet is a multilingual knowledge base that contains a rich semantic network for words with their linguistic and semantic information. It is

[4] `https://framenet2.icsi.berkeley.edu/fnReports/data/frame/Commerce_buy.xml`

constructed by integrating multiple well-known structured knowledge bases, such as WordNet [18], DBpedia [3], and Wikidata [27]. Similarly to WordNet, each word has a part-of-speech tag and a gloss representing its meaning. Words with similar meanings are grouped into *synsets* that have unique identifiers (of the form `bn:`*ddddddddp*, where *d* is a digit and *p* is a part of speech symbol `v`, `n`, etc.). BabelNet is structured as a knowledge graph where synset nodes are connected by directed edges representing semantic relations (i.e., *hypernym*, *hyponym*, etc.) Compared to WordNet, BabelNet has a much larger vocabulary and richer semantic relations. For instance, there are many *named entities* like famous people, locations, songs, books, etc. Besides, each edge in the knowledge graph has a weight that intends to capture the degree of relevance between two connected synset nodes with respect to the type of the edge. This abundance of information is the main reason we ended up using BabelNet as the underlying semantic network of words after trying many knowledge bases, such as WordNet, DBpedia, and Wikidata. Along with the richness, however, comes certain amount of noise and incompleteness—in part because integration of the data sources in BabelNet is done algorithmically and without much further curation by domain experts. While BabelNet is very large and rich in synsets and semantic relations, it also contains rarely used meanings of words, wrong semantic links, and words incorrectly associated with various synsets. Such errors can (and do) lead semantic parsers astray and therefore analysis and countermeasures are required to mitigate the impact of such noise in the knowledge base.

3 The KALM Framework

The KALM's frame-based parser is designed to parse CNL sentences and extract frame relations. Before embarking on building our own, we tried a number of semantic relation extraction tools, including Ollie [16], Stanford CoreNLP [15], the LCC system [14], SEMAFOR [4], and SLING [22]. These tools are designed for general text understanding and are strong contenders in that difficult domain, but their accuracy is far below the quality required for the knowledge that would be acceptable for logical knowledge bases. By *accuracy* we mean both precision and recall because, as mentioned earlier, our aim is to provide high-quality data and rules for logic-based reasoners, and these systems are very sensitive both to errors in the data as well as to missing data. In the KALM framework, high accuracy is achieved both through reliance on CNL, which makes our job much simpler compared to the aforesaid Open IE extractors, due to the extensive use of rich off-the-shelf knowledge bases, and due to the algorithms unique to KALM. Our frame-based parser uses a model that contains a set of *logical frames* and *logical valence patterns*. The logical frames are mostly modeled after FrameNet's frames, but we represent them in logical form and disambiguate the semantic meaning of each role via BabelNet synsets. The logical valence patterns are modeled after FrameNet's valence patterns, but we represent them in a form compatible with APE parses. These logical valence patterns are constructed in an automatic way by learning linguistic structures from annotated training sentences. For the rest of the paper, we use the acronyms *frame* and *lvp* to denote the concept of a logical frame and logical valence pattern, respectively.

3.1 *FrameOnt*—the Logical Model of Frames

FrameNet is not formal enough—it contains only textual descriptions of frames and valence patterns, so FrameNet cannot be directly used for frame-based parsing. We formalize FrameNet as *FrameOnt*, an ontology that models frames and valence patterns using logical facts and rules. A frame consists of a set of *roles* (or "frame elements" in the terminology of FrameNet), each representing the semantic role an entity plays in the frame relation. Unlike FrameNet, *FrameOnt* disambiguates each role via a set of BabelNet synsets that capture the relevant meanings of the role. In addition, *constraints* may be imposed on roles. For instance, a data type constraint may state that the *Money* role must be a number representing an amount in some currency.

The below `fp`-fact represents the `Commerce_Buy` frame that describes purchases involving buyers, sellers, goods, etc.

```
fp('Commerce_Buy',[
  role('Buyer',['bn:00014332n'],[]),
  role('Seller',['bn:00053479n'],[]),
  role('Goods',['bn:00006126n','bn:00021045n'],[]),
  role('Recipient',['bn:00066495n'],[]),
  role('Money',['bn:00017803n'],['Currency'])]).
```

The first argument here is the name of a frame; the second is a list of role descriptors in the frame. In each *role* descriptor, the first part is the name of the role, the second is a list of BabelNet synset IDs representing the meaning of the role (there can be several: `Goods` above can mean *article of commerce* or *article of a sale*), and the third lists data type constraints for that role. To extract an instance of the frame from a sentence, we use lexical units (from the previous section) and *logical valence patterns*, or *lvps*. We call the word extracted from a sentence to correspond to a role a *role-filler*. For instance, the `Commerce_Buy` frame has this logical valence pattern:

```
lvp(buy,v,'Commerce_Buy', [
    pattern('Buyer','verb→subject',required),
    pattern('Goods','verb→object',required),
    pattern('Recipient','verb→dep[for]',optnl),
    pattern('Money','verb→dep[for]',optnl),
    pattern('Seller','verb→dep[from]',optnl)]).
```
(1)

The first three arguments of an `lvp`-fact identify the lexical unit and the frame. The fourth argument is a set of `pattern`-terms, each having three parts: the first is the name of a role in the frame; the second is the *grammatical pattern* that specifies the grammatical context that relates the lexical unit, the role, and suitable role-filler words; and the third says whether the pattern is required in order to trigger the lvp or is optional. Each grammatical pattern is bound to a separate *parsing rule* that may be applied to extract the role-filler based on the APE parses. For example, in the first `pattern`-term, `verb→subject` says that the role-filler for the `Buyer`-role must be the subject of the *buy*-event. Based on the format of APE output described in Section 2, the corresponding parsing rule will find a suitable `object`-fact whose identifier equals the third argument

(the subject) of a `predicate`-fact representing the *buy*-event. If so, the word representing the `object`-fact will be extracted as the role-filler of the `Buyer`-role. For example, given the sentence *Mary buys a watch for John from Bob*, the above lvp applies the `Commerce_Buy` frame and extracts *Mary* as the `Buyer`, *Bob* as the `Seller`, *watch* as the `Goods`, and *John* as the `Recipient`. The lvp is used even though a filler for the optional `Money` role was not present.

Both the lexical units and the associated lvps are generated by KALM when a knowledge engineer designs and marks up sentences to train the parser to recognize frames and roles in various linguistic structures. This aspect is discussed next.

3.2 Frame Construction

Figure 1 shows the pipeline for frame and lvp construction. The frames and roles are designed by a knowledge engineer in advance, based on FrameNet and BabelNet. For each frame, the knowledge engineer must provide the semantics for the roles. In KALM, this requires searching BabelNet to find the most appropriate synsets for the roles in question. This has to be done because role-words tend to have multiple senses, and disambiguation of the role senses is key to ensuring accuracy of the extracted information. New frames may also have to be created, if the target domain requires that. (For instance, some important relations, such as human gender are not provided by FrameNet.) Some other frames in FrameNet must also be made more precise or expanded.

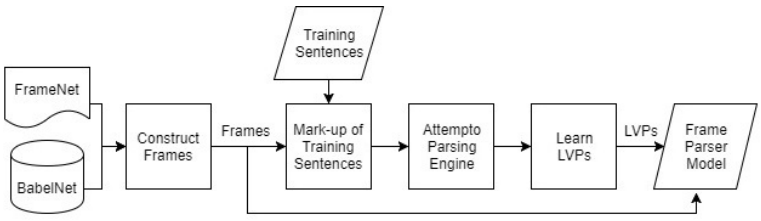

Fig. 1. Pipeline for frame and lvp construction

Once a frame and its roles are identified, the lvps are *learned* automatically from a set of marked-up *training sentences* designed by a knowledge engineer. In FrameNet, each valence pattern is associated with a set of exemplar sentences. Since KALM uses CNL, the knowledge engineer needs to rephrase sentences so that they will follow the ACE grammar. For each sentence, the engineer needs to mark the frame type, the lexical unit, the relevant roles, and the synonyms of the lexical unit. For instance, for a sentence like *Mary buys a watch for John from Bob for 200 dollars*, the knowledge engineer would create the following mark-up sentence and ask this query:

```
?- train('Mary buys a watch for John from Bob for 200 dollars',
     'Commerce_Buy', 'LUIdx'=2,
     ['Buyer'=1+required, 'Goods'=4+required,
      'Recipient'=6+optnl, 'Seller'=8+optnl, 'Money'=11+optnl] ,
     [purchase, acquire] ).
```

This says that the training sentence triggers the frame `Commerce_Buy` with the word *buy* as the lexical unit (*buy* is identified by the word index 2), that *Mary* (identified by the word index 1) is a filler for the *required* role `Buyer`, *Bob* (word index 8) fills in the *optional* role `Seller`, *watch* (word index 4) is a filler for the role `Goods` (also required), *200 dollars* (word 11) fills in the optional role `Money`, and *John* is the `Recipient`. The words `purchase` and `acquire` are defined as the synonyms of *buy* which can trigger an instance of the `Commerce_Buy` frame the same way as *buy* does.

Our *lvp generator* is a Prolog program that takes marked-up sentences described above and *learns* the appropriate grammatical patterns and parsing rules. These parsing rules can check if the syntactical context of the lexical unit with respect to the role-fillers in the marked-up sentence can be applied to new sentences and extract role-fillers from that sentence. For instance, the aforementioned marked-up sentence will lead to the lvp (1) shown earlier. An example of such a learned parsing rule is given below. It takes an APE parse and extracts the role-filler for `Buyer` according to the grammatical pattern `verb→subject`.

```
apply_pattern_to_target('verb→subject',APEParse,LUIdx,RoleFilIdx) :-
    get_pred_from_word_idx(APEParse,LUIdx,LUPred),                    (2)
    get_subj_from_verb(APEParse,LUPred,RoleFilIdx).
```

The rule takes an Attempto parse (`APEParse`) of a sentence and the word index for the lexical unit (`LUIdx`) as input, and outputs the word index of the extracted role-filler (`RoleFilIdx`) in the sentence. In the rule body, `get_pred_from_word_idx` takes `APEParse` and `LUIdx` and finds the corresponding predicate (`LUPred`) representing the lexical unit, which is an event in this case. Next, `get_subj_from_verb` searches `APEParse` to find an `object`-fact whose identifier matches the third argument (the subject) of `LUPred`. If found, the word index of the `object`-fact representing the role-filler is returned.

3.3 Frame Parsing

Having parsed a CNL sentence, the next step is to identify the frames and lvps (described in Section 3.2) that match the sentence. To this end, the sentence is scanned for lexical units of the existing lvps and then one checks if these lvps can be applied. If an lvp is applicable, all the extracted role-fillers from the sentence are collected and candidate frame-based parses are constructed. A *candidate frame-based parse* (abbr., *candidate parse*) has the form $\texttt{<FN,}\{\texttt{(RN}_i\texttt{,RF}_i\texttt{)}_{i=1,\ldots k}\}\texttt{>}$, where `FN` is the name of a frame the sentence possibly belongs to, and the second component in the tuple is a set of extracted role-name/role-filler pairs. This is a purely syntactic check and some parses may be rejected later on semantic grounds. For example, consider the following lvps:

```
lvp(buy,v,'Commerce_Buy',[pattern('Buyer','verb→subject',required),
  pattern('Goods','verb→object',required),
  pattern('Recipient','verb→dep[for]',optnl),                         (3)
  pattern('Money','verb→dep[at]→rel→dep',optnl),
  pattern('Seller','verb→dep[from]',optnl)]).
```

```
lvp(buy,v,'Commerce_Buy',[pattern('Buyer','verb→subject',required),
  pattern('Goods','verb→object',required),
  pattern('Recipient','verb→dep[for]',optnl),                    (4)
  pattern('Money','verb→dep[for]',optnl),
  pattern('Seller','verb→dep[from]',optnl)]).

lvp(buy,v,'Commerce_Buy', [pattern('Buyer,'verb→subject',required),
  pattern('Goods,'verb→object',required),
  pattern('Recipient,'verb→dep[for]',optnl),                     (5)
  pattern('Money,'verb→dep[for]→rel→dep',optnl),
  pattern('Seller','verb→dep[from]',optnl)]).
```

In our running example, the sentence *Mary buys a watch from John for Bob for 200 dollars*, the word *buys* triggers the above lvps. In all three cases, the pattern `verb→subject` lets rule (2) extract *Mary* as the `Buyer`. The pattern `verb→object` triggers another rule in the parser, which will extract *watch* as the `Goods`, and so on.

Based on these lvps, the system will construct several candidate parses, but some will be wrong, useless, or redundant. First, some candidate parses may be subsumed by others. For example, lvp (3) above yields a candidate parse where *Mary* is a `Buyer`, *watch* is the `Goods`, *Bob* is the `Recipient`, and *John* is the `Seller`. However, this parse is subsumed by the parse obtained from lvp (4) because our sentence contains all the components mentioned in lvp (4).

Second, the parser may misidentify the roles for the words extracted from the CNL sentence, so wrong role-fillers may get associated with some of the frame's roles in the candidate parses. For example, in lvp (4) the grammatical patterns for `Recipient` and `Money` are the same. Therefore, it will generate two candidate parses: in one case *Bob* is a role-filler for the `Recipient` role and *200 dollars* is a role-filler for `Money`; in another case, *Bob* is the role-filler for the `Money` and *200 dollars* is the role-filler for `Recipient`.

The third problem arises when a candidate parse extracts wrong role-fillers. For example, given the sentence *Mary buys a watch from John for Bob for a price of 200 dollars*, lvp (5) will give the right result. However, lvp (4) also applies, so we will get *price* as a role-filler for either the `Recipient` or the `Money`.

All of these problems are solved in the following subsection, via an algorithm for semantic *role-filler disambiguation*—a process related to word-sense disambiguation [19] but more narrow and so it has higher-accuracy solutions than the general problem of word-sense disambiguation.

3.4 Role-Filler Disambiguation

Role-filler disambiguation is akin to word-sense disambiguation but it does not try to disambiguate entire sentences. Instead, the goal is to disambiguate different senses of the extracted role-fillers and find the best sense for each role-filler with respect to the roles in particular logical frames. Consider the sentence *Mary grows a macintosh*, which belongs to the `Growing_Food` frame where *Mary* is the `Grower` and *macintosh* is the `Food`. In BabelNet, *macintosh* has several meanings like *an early-ripening apple* (`bn:00053981n`), *a computer sold by Apple Inc.* (`bn:21706136n`) and *a kind of waterproof fabric* (`bn:00052580n`). Since `Food` is much more semantically related to an apple than to a computer or a fabric, *macintosh* should be disambiguated with the synset `bn:00053981n` denoting *an apple*.

Role-filler disambiguation works on candidate parses produced by frame parsing, as explained in the previous subsection. Each role-filler is often associated with several synsets. The disambiguation process first scores BabelNet synsets in relation to the frame roles filled by the role-filler words and then combines the individual scores into scores for entire candidate parses, ranks the parses, and removes the ones that score below a threshold.

The disambiguation algorithm for candidate parses queries BabelNet for each role-filler and gets a list of *candidate role-filler synsets*, which are BabelNet synsets for the role-filler words. Then it performs a heuristic breadth-first search to find all semantic paths that start at each candidate role-filler synset and end at a role synset, or vice versa. A heuristic scoring function assigns a score to each path, prunes the unpromising paths, and selects the path with the highest score. The starting (or ending) point of that path is the role-filler synset chosen as the semantically most likely BabelNet synset for the role-filler in question. At this stage, each role-filler in a candidate parse is disambiguated, yielding a *disambiguated candidate parse* of the form <FN, {(RN$_i$,RF$_i$, BNSyn$_i$, Score$_i$)}>, where FN is a frame name, RN$_i$ a synset for a role in FN, RF$_i$ a role filler synset for the role RN$_i$, and Score$_i$ is a score that signifies the semantic relatedness of the role filler RF$_i$ to the particular meaning of the frame role represented by the synset RN$_i$. This disambiguated candidate parse thus extends the notion of a candidate parse described previously by adding the disambiguating information (in the form of the synsets RN$_i$ and RF$_i$ along with the relatedness score). For the score of the entire disambiguated parse, we take the geometric mean of all the individual role-filler scores. Candidate parses with lower scores are then discarded. The key to this process is choosing an appropriate scoring function, which is described next.

Computing semantic scores. Ideally, each role should be a direct or indirect *hypernym* of its role-filler, or vice versa. Consider the sentence *A person buys a car* that belongs to Commerce_Buy frame, where *person* is the Buyer and *car* is the Goods. Here, *person* is a hypernym of Buyer and Goods is a hypernym of *car*. At the first glance, one might try to focus on hypernym paths between the role-filler synsets and the role synsets, but this would have been too easy to actually work. First, BabelNet does not contain the entire knowledge of the world and many hypernym links are missing. Second, despite the overall high quality of this knowledge base, it still contains many wrong hypernym relations. Therefore, one must consider a broader class of semantic relations (like *derivationally related*, *gloss related*, and more) in building semantic paths between pairs of synsets. Note that some of the errors can be detected and corrected—see Section 5. Since it is impractical to fix all the wrong semantic relations or add all the missing ones, one must consider all kinds of semantic paths, not only the shortest ones. Also, not all links are created equal. As mentioned, hypernym links are probably a good bet, but following hyponym or gloss-related links (which connect words with related glossaries) is riskier. To compute the semantic score, we consider three factors: the semantic connection number (total number of semantic links connected to each synset node), the edge type and weight, and the path length. The first two can be obtained by querying BabelNet, but the edge weight information there is rather sparse and cannot be relied upon too much. For instance, many good hypernym links have the weight of zero in BabelNet. We therefore bump up the weight of hypernym and other links by various constants (e.g., larger for hypernyms, lower for hyponyms).

The scoring function in KALM was chosen to encourage the paths with higher semantic connection numbers and edge weights, and to penalize the longer paths. Additionally, different *relevance factors* are given to different types of edges in a path. For instance, the hypernym edges have the highest relevance factor. Derivationally-related

and gloss-related edges are given the next highest relevance factor, etc. Formally, let n_1 be a role-filler synset node, n_l be a role synset node and $L = \{n_1, e_{12}, n_2, \cdots, n_l\}$ be a semantic path from n_1 to n_l, where n_i represents a BabelNet synset node and $e_{i,i+1}$ represents an edge between n_i and n_{i+1}. The semantic score of the path is computed by the following formula based on the above principles:[5]

$$score = \frac{\sum_{i=1}^{n-1} \sqrt{f_n(n_i)} \times f_w(e_{i,i+1})}{5^{\sum_{i=1}^{n-1} f_p(e_{i,i+1})}} \tag{6}$$

where $f_n(n_i)$ is n_i's semantic connection number, $f_w(e_{i,i+1})$ is the sum of $e_{i,i+1}$'s BabelNet edge weight and its relevance factor, and $f_p(e_{i,i+1})$ is the penalty value for $e_{i,i+1}$, defined based on the edge type. The *base* of the exponent in the denominator is 5, which imposes a serious penalty for longer paths.

The above algorithm is fairly naive and takes hours to compute the score of a disambiguated candidate parse. This is because BabelNet is very large, the number of BabelNet queries required by this algorithm is in millions, and even a *local* such query takes about 10 ms. On average, each word is associated with 15 synsets and each synset is semantically related to a few hundred other synsets, so the number of semantic paths can be huge. Section 4 deals with this complexity, reducing the time to seconds.

3.5 Constructing Unique Logical Representation from Parses

We now show how frame parsing and role-filler disambiguation work in tandem to yield *unique logical representation* (*ULR*) for sentences. The process is shown in Figure 2.

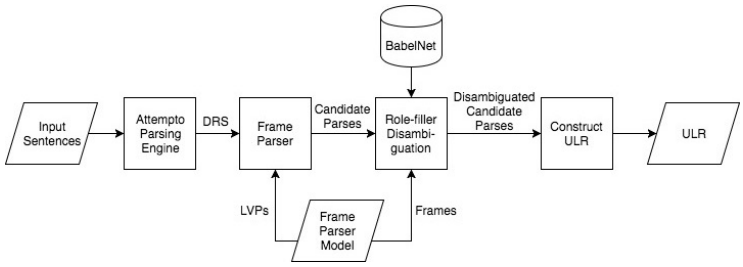

Fig. 2. Pipeline for translating a sentence into ULR

ULR uses the predicates `frame/2` and `role/2` for representing instances of the frames and the roles. The predicates `synset/2` and `text/2` are used to provide synset and textual information. Consider these sentences: *Mary buys a watch for Bob at a cost of 200 dollars*, *Mary buys a watch for Bob for 200 dollars*, and *Mary buys a watch for Bob for a price of 200 dollars*. Although these sentences are different in structure, they trigger the same frame (`Commerce_Buy`) because they match the lvps (3), (4), and (5) in Section 3.3, respectively, and this leads to *exactly the same* candidate parses. Therefore, when these sentences are stated as facts, they would be translated into exactly the same logical representation, shown below.

```
frame(id_1,'Commerce_Buy').
role(id_1,'Buyer',id_2).   role(id_1,'Recipient',id_3).
role(id_1,'Goods',id_4).   role(id_1,'Money',id_5).
```

[5] The parameters were chosen experimentally. As part of future work, we will explore using a neural net to fine-tune this formula.

```
synset(id_2,'bn:00046516n').   // person synset
text(id_2,'Mary').
synset(id_3,'bn:00046516n').   // person synset
text(id_3,'Bob').
synset(id_4,'bn:00077172n').   // watch synset
text(id_4,'watch').
synset(id_5,'bn:00024507n').   // currency synset
text(id_5,'200 dollars').
```

The symbol `id_1` here is a unique ID given to the event of Mary buying a watch. The other IDs are assigned to the various role-filler entities extracted from the sentence. For instance, `id_2` represents the entity corresponding to *Mary* and `id_3` to *Bob*. These entities are further described by the predicates `synset/2` and `text/2`.

4 Taming the Complexity of Role-Filler Disambiguation

BabelNet is a very large knowledge graph and the role-filler disambiguation relies on massive amount of querying of that graph, while each query is relatively expensive (even when BabelNet instance runs locally and is called directly, via Java). To solve the performance problems with the naive disambiguation algorithm of Section 3.4, we developed a number of optimizations whose collective effect is reducing the computation time from hours to seconds. These techniques are described below.

Parallel computation. The first obvious observation is that the base algorithm is easily parallelizable, since finding semantic paths connecting different synsets pairs can be done independently. To this end, we create a thread pool where each separate thread finds paths from a specific candidate role-filler synset to a role synset. Moreover, if one such path is found, the highest current score among such a paths is shared with all the parallel threads and is used as a cut-off for pruning the computation of any low-score path that is in progress. Our test machine had 12 CPU cores; with more cores, more threads could be created by elaborating on the above idea. Parallelization reduces the running time by an order of magnitude in some cases. Role-filler disambiguation could also be done in parallel across the different candidate parses, and one could further parallelize the process across sentences.

Caching BabelNet queries. BabelNet queries are relatively expensive and collectively take most of the computing time. Our experiments showed that in role-filler disambiguation about 70% of such queries are repeated more than once. Although BabelNet does some caching on its own, it is insufficient. To hasten the search for paths between pairs of synsets, KALM caches the results of BabelNet queries internally, which results in a big speedup (3-5 times, depending on specifics of the case).

No duplicate computation in role-filler disambiguation. For a sentence, different candidate parses (generated from different lvps) that represent the same frame often share some of the role/role-filler pairs. We avoid such duplicate computations by creating one thread for each unique role/role-filler pair.

Priority-based BabelNet path search. Given the complexity of BabelNet, any pair of synsets can have many connecting paths and even more paths wonder astray without connecting the requisite nodes. Although we prune away paths that score low, naive breadth-first search for connecting paths can get stuck exploring wrong parts of the graph for a long time. To avoid this problem, we use adaptive priority-based search with a priority queue in which unpromising paths get downgraded and eventually pruned.

Inverse path search. The aforementioned algorithm is designed to find a semantic path from the role-filler synset to the role synset. However, in the car-buying example

of Section 3.4, the role could be either a hypernym of the role-filler, as in (*car*, `Goods`), or a hyponym, as in (*person*, `Buyer`). In principle, we could use the same priority-queue based approach to find a hyponym path from the *person* synset to the `Buyer` synset. Once the path is found, we could compute one semantic score based on the hyponym path and another based on the inverse semantic path that goes from `Buyer` to *person* and uses hypernym links. We could then pick the best-scoring path.

However, not all BabelNet edges have semantically inverse edges (e.g., *entailment* edges do not). Besides, the fan-out in the BabelNet graph at a role-filler synset and that at the frame role synset can be very different. For example, the *person*-synset (`bn:00046516n`) has more than a thousand hyponyms, and starting the search from that synset is almost always costlier than going from, say, the role `Buyer` to the role-filler *person*. To take advantage of this asymmetry, separate threads are created to search from role synsets to their candidate role-filler synsets. This inverse path search is also based on the priority queue and is computed similarly.

5 Explaining Semantic Parses

This section discusses the explainability aspect of the KALM approach. This includes both explaining the correct semantic parses and also why errors are made.

5.1 Explaining Correct Parses

Section 3.4 explained the process of role-filler disambiguation, which assigns a BabelNet synset to each role-filler word. This is done by finding highest-scoring semantic paths that connect the candidate synsets for role-filler words and the synsets for the roles in selected *FrameOnt* frames. The KALM explanation mechanism is based on analysis of these paths. Consider the sentence *Robin Li is a founder of Baidu*, which has three candidate parses with the following lvps:

```
lvp(founder,n,Create_Organization, [
    pattern(Creator,object|→verb→subject,required),                (7)
    pattern(Organization,object→rel→object,required) ] ).

lvp(be,v,People_by_Origin, [
    pattern(Person,verb→subject,required),                         (8)
    pattern(Origin,verb→object,required) ] ).
lvp(be,v,Being_Employed, [
    pattern(Employee,verb→subject,required),                       (9)
    pattern(Position,verb→object,required) ] ).
```

Here, the lvp (7) belongs to the `Create_Organiazation` frame, where *Robin Li* fills the `Creator`-role and *Baidu* fills the `Organization`-role. The lvp (8) belongs to the `People_by_Origin` frame, where *Robin Li* fills the `Person`-role and *founder* fills the `Origin`-role. The last lvp, (9), belongs to the `Being_Employed` frame, where *Robin Li* fills the `Employee`-role and *founder* fills the `Position`-role. The parse corresponding to the first lvp (7) gets the highest score with the following semantic paths that can be shown to the user (who is the domain expert in this case) as explanations:

`Creator`: *Robin Li* (`bn:03307893n`) $-$*hypernym*$\rightarrow$ *a person who founds or establishes some institution* (`bn:00009631n`)

`Organization`: *Baidu* (`bn:00914124n`) $-$*hypernym*$\rightarrow$ *an institution created to conduct business* (`bn:00021286n`) $-$*hypernym*$\rightarrow$ *an organization* (`bn:00059480n`)

These paths justify the chosen parse by demonstrating the semantic connections in BabelNet between each role (`Creator` and `Organization` in this case) and their role-fillers. The alternative disambiguations, like `Origin`: *founder* (that stems from the lvp (8)) and `Position`: *founder* (that stems from the lvp (9)) receive very low scores because the concepts represented by these roles and their role-fillers (*founder* for both) are semantically incompatible, which results in low-scoring connecting BabelNet paths.

5.2 Explaining Erroneous Parses

Along with plethora of useful data, BabelNet contains fair amount of uncurated noise due to the fact that much of this knowledge graph was created automatically, by merging information from various sources with the help of sophisticated heuristics. Like any algorithm of that kind, this process admits certain amount of errors, including wrong synset assignments to words, incorrect semantic links, and missing links. For instance, BabelNet makes the concept of *job position* a hypernym of the concept of *womanhood* and the concept of *engineering science* a hypernym of the concept of *building structure*. It also wrongly assigns the concept of engineering science as one of the meanings of the word *engineer*. These errors can throw role-filler disambiguation off-course and hurt the accuracy of knowledge authoring.

To deal with such errors, it is necessary to be able to both explain (to a domain expert) why KALM has selected a wrong semantic parse and then to provide the means to correct or mitigate the noise in BabelNet that was responsible for that particular error. We illustrate these issues using three examples where errors in BabelNet cause wrong parsing results and show how a domain expert can deal with such problems.

First, consider the case of a wrong synset assignment to words in BabelNet. This problem arises in the sentence *Mary buys a watch for Susan's daughter*, among others. Here the frame parser selects the `Commerce_Buy` frame where *Mary* fills the role of `Buyer`, *watch* fills the role of `Goods`, and *daughter* fills the role of `Recipient`. The word *daughter* is disambiguated with the BabelNet synset `bn:00018346n` (*a human offspring—son or daughter—of any age*). Although this synset has a connection with the `Recipient` role, it is not equivalent to the *daughter* concept which only refers to *a female human offspring*. The domain expert can record this synset assignment error and add it to an *exception* list, so KALM will not associate *daughter* with synset `bn:00018346n` in the next run.

Now consider a case of incorrect semantic links using the sentence *Mary works in Rockefeller Center* as an example. Our semantic parser will generate three parses based on three different lvps—all belonging to the same frame `Being_Employed`:

```
{(Employee,Mary),(Place,Rockefeller Center)}
{(Employee,Mary),(Field,Rockefeller Center)}
{(Employee,Mary),(Employer,Rockefeller Center)}
```

Obviously, only the first parse is intended, but the second one gets the highest score. The reason is that *Rockefeller Center* is connected to `Field` (as a branch of knowledge) via the following path which has a very high score:

`Field`: *Rockefeller Center* (`bn:00897288n`) $-$*hypernym*$\rightarrow$ *a building structure* (`bn:00013722n`) $-$*hypernym*$\rightarrow$ *a discipline dealing with art or science* (`bn:00005105n`) $-$*hypernym*$\rightarrow$ *a branch of knowledge*(`bn:00007985n`).

Clearly, the second link is wrong because *a building structure* is not a special case of *a discipline dealing with art or science*. A domain expert can record this incorrect semantic link and KALM will not consider it as a valid link next time.

Finally, consider the case when BabelNet has a missing semantic link using the sentence *John travels to Los Angeles*. The frame parser would select the `Travel` frame where *John* is the `Traveler` and *Los Angeles* is the `Goal`. Surprisingly, the synset `bn:00019336n`, which refers to the city of *Los Angeles* in California, does not get the highest score because BabelNet is missing an important *hypernym* link connecting `bn:00019336n` (*Los Angeles, CA*) and the concept of municipality with the synset `bn:00056337n`. The highest score gets a relatively small city in Argentina (`bn:02084491n`) under the same name. This fact is immediately clear when one compares the semantic path from Los Angeles in Argentina to the role `Goal` (the highest-scoring path) to the path from the intended synset of Los Angeles, CA to that same role. This simple analysis immediately suggests to the user that the missing link should be added.

Another source of errors is when BabelNet associates different synsets to the same meaning. For instance, BabelNet synsets `bn:00071215n` (which comes from WordNet) and `bn:15385545n` (which comes from Wikidata) denote the same concept as *a place where items or services are sold*. This can be corrected by establishing an equivalence between these two synsets.

6 Evaluation

Dataset. Acting as knowledge engineers, we used the methodology described in Section 3.2 and created a total of 50 logical frames,[6] mostly derived from FrameNet but also some that FrameNet is missing (like *Restaurant*, *Human_Gender*). We then used KALM to learn 213 logical valence patterns from 213 *training* sentences.[6] We used 28 additional *tuning* sentences to adjust the parameters of the scoring function (6) used for role-filler disambiguation and to deal with noise in BabelNet.

We evaluated the KALM system using 250 test sentences[6] (distinct from the training sentences) and verified whether the system returns the expected frames and disambiguates each role-filler correctly. Note that our approach is based on CNL and so public and standardized data sets are not available for comparison. The test sentences were instead constructed by rephrasing the sentences from FrameNet into CNL. While Attempto CNL and KALM can handle quite complex sentences, the test sentences were on purpose selected to be very simple and common, to rule out the possibility of a bias towards KALM. We claim that to be of any use for knowledge acquisition or authoring, any system should grok such sentences out of the box. Here is a sample of the typical test sentences:

Mary buys a laptop.
Kate obtains a master degree in biology from Harvard University.
Kate makes a trip from Beijing to Shanghai.
John works at IBM.
Kate Winslet co-stars with Leonardo DiCaprio in Titanic.
Warren Buffett stays in Omaha.
A student borrows a textbook from a library.

Comparison Systems. The aim of this work is to create a technology to enable domain experts to author high-quality knowledge using CNL. This goal is quite unique in the literature and there do not seem to be systems that are directly comparable to our work. It is interesting, however, to compare KALM with systems designed for general text understanding, like SEMAFOR, SLING, and Stanford CoreNLP. Since these systems are much more general than KALM and tackle a much more difficult problem, it cannot be expected that they would produce the same quality of knowledge

[6] `https://datahub.csail.mit.edu/browse/pfodor/kalm/files`

as KALM. However, given the simplicity of our test sentences, it is still reasonable to anticipate that these systems would do well. Nevertheless, KALM bested all of them by a margin much wider than expected.

A few more observations about the differences between KALM and other systems are in order. First, none of the other systems do disambiguation or attempt to find synsets for role-fillers, so in this aspect KALM does more and is better attuned to the task of knowledge authoring. Second, none of these systems can explain their results, nor do they provide ways to analyze and correct errors. Third, two of the comparison systems use ontological frameworks that differ from *FrameOnt*. Whereas SEMAFOR is based on FrameNet and is similar in this respect to KALM, SLING is based on PropBank [21], and Stanford CoreNLP is based on Knowledge Base Population (KBP) relations [17]. In our view, PropBank is not well-suited to support disambiguation because it does not maintain equivalence among frames well enough. For instance, given the sentences *Mary buys a car* and *Mary purchases a car*, the word *buy* and *purchase* would trigger *buy.01* and *purchase.01* frames, respectively. Although these sentences mean the same, they are not mapped to the same frame in PropBank. As to KBP, it has too few types of semantic relations usable for knowledge acquisition compared to FrameNet and PropBank. In any case, although the three ontological frameworks are different, *FrameOnt*, FrameNet, and PropBank all cover the concepts used in the test sentences, and KBP covers many of them as well. Therefore, the comparison systems were expected to parse our test sentences and extract correct semantic relations.

Results. The evaluation is based on the following metrics.

FrSynC	all frames and roles (semantic relations) are identified correctly and all role-fillers are disambiguated
FrC	all frames and roles are identified correctly
PFrC	some frames/roles are identified, but some are not
Wrong	some frames or roles are misidentified

KALM: 239 sentences are *FrSynC* (**95.6**%), 248 sentences are *FrC* (> **99**%), and 2 sentences are *Wrong* (< 1%). Note that *FrSynC* applies only to KALM, since none of the comparison systems can disambiguate the senses of the extracted entities.

SEMAFOR: parses 236 sentences out of the 250 test sentences, where 59 sentences are *FrC* (**25**%), 44 sentences are *PFrC* (18.6%), and 133 sentences are *Wrong* (56.4%).

SLING: parses 233 sentences, where 98 sentences are *FrC* (**42.1**%), 63 are *PFrC* (27%), and 72 sentences are *Wrong* (30.9%).

Stanford CoreNLP: parses 26 sentences, out of which 14 sentences are *FrC* (**53.8**%), 10 sentences are *PrC* (38.5%), and 2 sentences are *Wrong* (7.7%).

Compared to the other three systems, KALM is by far more accurate: in only two cases it misidentifies the semantic frames. In one of these cases SEMAFOR succeeds partially (*PFrC*) and in the second case all other systems fail also. An example of such a difficult sentence is *Kate makes a purchase of a company*, which belongs to the frame `Commerce_Buy`, where *Kate* is supposed to fill the role of `Buyer` and *company* the role of `Goods`. However, KALM selects the `Building` frame instead and marks *Kate* as the `Agent` and *purchase* as the `Created_Entity`.

In the other 9 cases where KALM is less than perfect, all frames and roles are identified correctly, but some of the synsets are misidentified. For instance, in the sentence *Kate purchases a house*, KALM assigns the synset of a "public building for gambling and entertainment" to the role-filler *house*, but *house* was correctly extracted to fill the role `Goods`. Recall that the comparison systems do not have the means for assigning any synsets at all.

7 Conclusion

Controlled natural languages were proposed as a technology designed to enable domain experts who have no skills in knowledge representation to become effective as knowledge engineers. Unfortunately CNLs fell short of this promise due to insufficient attention to the semantics, which led to serious gaps in accuracy between what CNLs provide as knowledge authoring tools and what logic knowledge bases actually require. In this paper, we introduced a logic-based knowledge authoring approach, KALM, and demonstrated that bridging this gap is possible if CNLs are combined with linguistic frameworks, like FrameNet and knowledge bases like BabelNet, and with role-filler disambiguation. We have shown that this approach can be made efficient and that it achieves the unprecedented accuracy of 95.6% for CNL sentences. We also believe that this result can be further improved with machine learning techniques for tuning the relatedness scoring functions in KALM.

Before converging on FrameNet and BabelNet, we considered a number of other linguistic and general knowledge bases, such as ConceptNet [26], VerbNet [24], PropBank, DBpedia, Wikidata, and of course, the original WordNet. However, none of these systems could match the breadth of BabelNet, and we found the FrameNet-based methodology to be a good match for a logic-based approach such as ours. We also considered SEMAFOR, SLING, and Stanford CoreNLP as alternative frame extractors, but found that they target unrestricted natural language and have accuracy that is too low for our purposes.

For future work, we plan to extend the KALM framework to enable high-accuracy authoring of complex rules and to deal with rules that have exceptions, which are common in human knowledge.

Acknowledgements. We thank Niranjan Balasubramanian and H. Andrew Schwartz for the helpful discussions. This work was partially supported by NSF grant 1814457.

References

1. Allan, K.: Natural Language Semantics. Wiley, Hoboken, NJ, USA (2001)
2. Angeli, G., Premkumar, M.J.J., Manning, C.D.: Leveraging linguistic structure for open domain information extraction. In: 53rd Annual Meeting of the Association for Computational Linguistics. pp. 344–354. Beijing, China (2015)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A nucleus for a web of open data. In: 6th Intl. Semantic Web Conf. pp. 722–735 (2007)
4. Das, D., Chen, D., Martins, A.F.T., Schneider, N., Smith, N.A.: Frame-semantic parsing. Comp, Linguistics **40**(1), 9–56 (2014)
5. Erdem, E., Erdogan, H., Öztok, U.: BIOQUERY-ASP: querying biomedical ontologies using answer set programming. In: 5th Intl RuleML2011@BRF Challenge. pp. 1–8 (2011)
6. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto controlled english for knowledge representation. In: Reasoning Web. pp. 104–124. Springer (2008)
7. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Discourse Representation Structures for ACE 6.6. Tech. Rep. 2010.0010, Department of Informatics, University of Zurich, Switzerland (2010)
8. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.T.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011)

9. Gomez, F.: The acquisition of common sense knowledge by being told: an application of nlp to itself. In: International Conference on Application of Natural Language to Information Systems. pp. 40–51. Springer (2008)
10. Johansson, R., Nugues, P.: Lth: Semantic structure extraction using nonprojective dependency trees. In: 4th Intl. Workshop on Semantic Evaluations. pp. 227–230. SemEval '07, Assoc. for Computational Linguistics (2007)
11. Johnson, C.R., Fillmore, C.J., Petruck, M.R., Baker, C.F., Ellsworth, M.J., Ruppenhofer, J., Wood, E.J.: FrameNet: Theory and Practice (2002)
12. Kifer, M.: Knowledge representation & reasoning with Flora-2 (2018), `http://flora.sourceforge.net`
13. Kuhn, T.: A survey and classification of controlled natural languages. Comp. Linguistics **40**(1), 121–170 (2014)
14. Lehmann, J., Monahan, S., Nezda, L., Jung, A., Shi, Y.: LCC approaches to knowledge base population. In: 3d Text Analysis Conf., TAC. pp. 1–11. NIST, Gaithersburg, MA, USA (2010)
15. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: 52nd Annual Meeting of the Assoc. for Computational Linguistics, ACL, System Demonstrations. pp. 55–60. Baltimore, MD, USA (2014)
16. Mausam, Schmitz, M., Soderland, S., Bart, R., Etzioni, O.: Open language learning for information extraction. In: The Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL. pp. 523–534. Jeju Island, Korea (2012)
17. McNamee, P., Dang, H.T., Simpson, H., Schone, P., Strassel, S.M.: An evaluation of technologies for knowledge base population. In: 7th Intl. Conf. on Lang. Resources and Evaluation (LREC'10). p. 4. Valletta, Malta (May 2010)
18. Miller, G.A.: Wordnet: A lexical database for english. Commun. ACM **38**(11), 39–41 (1995)
19. Navigli, R.: Word sense disambiguation: A survey. ACM Comp. Surveys **41**(2), 10 (2009)
20. Navigli, R., Ponzetto, S.P.: BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence **193**, 217–250 (2012)
21. Palmer, M., Kingsbury, P., Gildea, D.: The PropBank: An annotated corpus of semantic roles. Comp. Linguistics **31**(1), 71–106 (2005)
22. Ringgaard, M., Gupta, R., Pereira, F.C.N.: SLING: A framework for frame semantic parsing. CoRR **1710.07032**, 1–9 (2017), `http://arxiv.org/abs/1710.07032`
23. Sagonas, K., Swift, T., Warren, D.S.: XSB as an efficient deductive database engine. In: ACM Conf. on the Management of Data. pp. 442–453. New York, USA (1994)
24. Schuler, K.K.: VerbNet: A Broad-coverage, Comprehensive Verb Lexicon. Ph.D. thesis, University of Pennsylvania (2005), aAI3179808
25. Schwitter, R.: English as a formal specification language. In: 13th Intl. Workshop on Database and Expert Systems Appl. (DEXA 2002). pp. 228–232. Aix-en-Provence, France (2002)
26. Speer, R., Chin, J., Havasi, C.: Conceptnet 5.5: An open multilingual graph of general knowledge. In: 31st AAAI Conf. on AI. pp. 4444–4451. San Francisco, CA, USA (2017)
27. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Comm. of the ACM **57**(10), 78–85 (2014)

28. Yang, G., Kifer, M., Zhao, C.: Flora-2: A rule-based knowledge representation and inference infrastructure for the Semantic Web. In: Intl. Conf. on Ontologies, Databases and Appl. of Semantics (ODBASE-2003). pp. 671–688 (2003)