HANDLING SENTENCE COMPLEXITY IN INFORMATION EXTRACTION FOR AUTOMATED COMPLIANCE CHECKING IN CONSTRUCTION

Jiansong Zhang, PhD Candidate, jzhang70@illinois.edu
Nora El-Gohary, Assistant Professor, gohary@illinois.edu
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA

ABSTRACT

Existing construction automated compliance checking (ACC) systems require manual effort for extracting requirements from textual regulatory documents (e.g. building codes) and encoding these requirements in a computer-processable format. To address this gap, we have proposed a new approach for ACC which automatically 1) extracts semantic information (concepts and relations) from construction regulatory documents, and 2) transforms the extracted information into Prolog logic clauses for automated reasoning. Due to the variety of natural language structures, the number of text patterns in one document could become extremely large. As a result, text processing (i.e. information extraction (IE) and information transformation (ITr)) becomes highly complex and difficult. In our approach, we are proposing two methods to handle sentence complexity: 1) top-down method: starting from the top level (i.e. full sentence) and proceeding down to identify and process complex sentence components, and 2) bottom-up method: starting from the lowest level (i.e. single terms in a sentence) and proceeding up to identify and process complex sentence components. Complex sentence components are intermediately processed segments of text that are composed of multiple concepts and relations. Further processing of complex sentence components results in recognition of concepts and relations for subsequent use in constructing logic clauses. We tested our proposed methods in processing quantitative requirements (i.e. IE and ITr) from the International Building Code. We compared the results against manually-developed gold standards; and evaluated the performance in terms of precision, recall, and F1 measure. Both methods achieved high performance, but the bottom-up method outperformed the top-down method. The bottom-up method achieved 0.962, 0.961, and 0.962, while the top-down method achieved 0.954, 0.925, and 0.939 for precision, recall, and F1 measure, respectively.

Keywords: automated compliance checking, natural language processing, information extraction, information transformation, sentence complexity

1. INTRODUCTION

Construction projects are regulated by a multitude of regulations. Manual compliance checking of those regulations is costly, time-consuming, and error-prone. Efforts in both academia and industry are being made to automate the process of compliance checking of construction projects. However, "formalizing the provisions into rules is done manually at present, which is time-consuming" (Zhong *et al*. 2012). For example, Solibri Model Checker (Corke 2013) currently includes a set of 300 proforma-based rules that allow for some degree of user customization of rules. However, such customization does not allow for the creation of new rules. The development of new rules in Solibri Model Checker, thus, requires professional software engineering expertise and deep understanding of the software's environment and data structure (Corke 2013). Similarly, for the SMARTcodes project (ICC 2012), the interpretation and encoding of regulatory provisions into SMARTcodes rules is conducted manually. As such, a major research gap still exists in the area of automated compliance

checking (ACC): existing ACC systems require manual effort for extracting requirements from textual regulatory documents and encoding these requirements in a computer-processable format. To address this gap, we developed a semantic natural language processing (NLP)-based approach for automatically extracting semantic information (concepts and relations) from regulatory textual documents, and subsequently transforming the extracted information into logic clauses (Zhang and El-Gohary 2013). Due to the variety of natural language structures, the number of text patterns in one document could become extremely large. As a result, text processing (i.e. information extraction (IE) and information transformation (ITr)) becomes highly complex and difficult. In this paper, we present and compare two methods that we developed for handling sentence complexity.

2. BACKGROUND

2.1 Natural Language Processing

Natural Language Processing (NLP) aims at enabling computers to analyze and process natural text and speech in a human-like manner (Cherpas 1992). NLP belongs to the domain of Artificial Intelligence (AI). AI is concerned with symbolic inference by computers and symbolic knowledge representation for use in making inferences (Tierney 2012). Many techniques in NLP, such as Part-of-Speech (POS) tagging (Charniak 1997), morphological analysis (Spencer 1991), named entity recognition (McCallum and Li 2003), etc., have been practically-used in various information systems applications. However, deep NLP, such as full understanding of sentences, is difficult. Because human language is irregular and diversified (Tierney 2012), deep NLP requires complex knowledge representation and reasoning to accurately understand the meaning of the text, which remains to be a fundamental challenge in AI. There are two main types of approaches taken in NLP: 1) rule-based approach, and 2) machine learning (ML)-based approach. Rule-based NLP uses manually-coded rules for text processing. These rules are iteratively-constructed and refined to improve the accuracy of text processing. ML-based NLP uses ML algorithms to train text processing models based on the text features of a given training text (Tierney 2012). Rule-based NLP tends to show better text processing performance (in terms of precision and recall), but requires more human effort. We are taking the rule-based approach (i.e. using human-developed text processing rules), because a high performance is important in supporting automated compliance checking.

2.2 Information Extraction

Information Extraction (IE) is a subfield of NLP that aims at extracting predefined target information from given text sources. IE mainly relies on text patterns that consist of multiple text features to recognize specific types of information from the text. There are two main types of text features that could be used in IE: syntactic (i.e. grammatical) and semantic (i.e. meaning related). Semantic features (concepts and relations) could be recognized based on an ontology. An ontology models domain knowledge in a semantic way, in the form of concept hierarchies, relationships between concepts, and axioms (El-Gohary and El-Diraby 2010). Semantic IE utilizes both syntactic and semantic features. In comparison to syntactic-only IE (i.e. IE methods utilizing syntactic features only), the semantic features in semantic IE allows for capturing domain-specific text meaning which can lead to a better IE performance (Zhang and El-Gohary 2011). In our IE work, we utilize an ontology together with NLP techniques to conduct semantic IE.

2.3 Automated Reasoning

Automated reasoning is a field of AI that aims at building computing systems that automate the process of inference-making (Portoraro 2011). Both conventional programming languages (e.g. Java programming language) and formal logic could be used to conduct automated reasoning. However, in comparison to conventional programming languages, formal logic could represent complex relations more efficiently. This makes formal logic quite suitable for representing complex regulatory requirements. Several types of formal logics are available, such as propositional logic, predicate logic, modal logic, and description logic. First Order Logic (FOL), as one type of predicate logic, is the most widely-used formal logic for automated deduction purposes. Logic Programming is an important and widespread application of formal logic (Portoraro 2011). Prolog is the most commonly-used logic

programming language. Prolog uses Horn Clauses (HCs). A HC is one of the most restricted forms of FOL, but is very efficient in implementing inference rules due to its restricted syntax (Saint-Dizier 1994). A HC could be represented by a rule that has zero or more antecedents (i.e. bodies in Prolog rules) that are conjoined, and zero or one consequent (i.e. heads in Prolog rules). There are three types of HCs: 1) one or more antecedents and one consequent, 2) zero antecedents and one consequent, and 3) one or more antecedents and zero consequent. Correspondingly, there are three basic logic clauses in Prolog: 1) rules, 2) facts, and 3) queries. We use the rules and facts in Prolog to represent requirement rules and project information, respectively. The reasoning about both the rules and project information, for compliance checking, will be automatically carried out by the underlying theorem prover (i.e. a reasoner) in Prolog.

2.4 Automated Compliance Checking in the Construction Domain

In the existing research efforts in automated compliance checking in the construction domain, several techniques have been utilized for representing building standards and/or specifications to support automated reasoning, such as decision table/trees-based models (Fenves and Garrett 1986), rules in logic programs (Thomson *et al*. 1988), IF/THEN rules in knowledge systems (Delis and Delis 1995), and distributed object system (Han *et al*. 1997). However, the interpretation and encoding of requirements into those computer-processable representation is still conducted manually. To address this gap, we proposed a method to automatically generate requirements in computer-processable representations (i.e. logic clauses) based on textual standards and/or specifications (Zhang and El-Gohary 2013).

3. PROPOSED METHODS

Aligned with our method of automated logic clause generation (Zhang and El-Gohary 2013), we developed two methods to handle sentence complexity: 1) a top-down method: starting from the top level (i.e. full sentence) and proceeding down to identify and process complex sentence components, and 2) bottom-up method: starting from the lowest level (i.e. single terms in a sentence) and proceeding up to identify and process complex sentence components. Complex sentence components are intermediately-processed segments of texts that are: 1) expressed using a variety of natural language structures, and 2) composed of multiple concepts and relations.

3.1 The proposed automated logic clause generation method

In previous work (Zhang and El-Gohary 2013; Salama and El-Gohary 2013), we proposed a method for automatically extracting requirement rules from regulatory text sources and formalizing the extracted information into logic clauses. As shown in Figure 1, we use three main phases of text processing: text classification (TC), information extraction (IE), and information transformation (ITr). In our method, TC recognizes sentences that are relevant (i.e. containing target information). By retrieving the relevant text only, TC allows for filtering off irrelevant sentences prior to IE. This could save computational processing efforts and avoid unnecessary errors in IE.

Subsequently, in our method, semantic IE recognizes instances of target information in the text (i.e. instances of concepts and relations), extracts those instances, and fills those instances into predefined information templates. We use pattern matching-based IE rules for executing the extraction. Pattern matching is a widely-used IE technique. Each pattern matching rule defines the part of text to extract based on a pattern that consists of various text features. Since we use both syntactic and semantic text features in matching patterns, our IE rules are semantic pattern-based. The syntactic features we use include: Part-of-speech (POS) tags, phrasal tags, and gazetteer lists. POS tags are tags indicating words' lexical and functional categories, such as noun, verb, adjective, etc. Phrasal tags are type labels assigned to phrases of a sentence, such as noun phrase, verb phrase, etc. A gazetteer list groups any set of terms based on any commonality possessed by these terms. The semantic features we use are based on the concepts and relations in our ontology.

In our method, ITr utilizes a set of manually-developed semantic mapping (SM) rules and conflict resolution (CR) rules to transform the extracted information instances into Prolog logic clauses. SM rules define how to process the extracted information instances based on their semantic meanings. For example, one semantic

mapping rule (for information elements 'subject' and 'general relation'; detailed meaning of each information element is explained in latter sections) could be "if two 'subject' instances are connected by a 'general relation' instance, then generate a logic clause for each of the 'subject' instance and another logic clause (based on the 'general relation' instance) connecting the two 'subject' instances." According to this semantic mapping rule the logic clauses 'interior_space(Interior_Space), human_occupancy(Human_occupancy), intended_for(Interior Space, Human_occupancy)' will be generated for the part of statement "Interior spaces intended for human occupancy…". CR rules define the methods for resolving each type of extraction conflict among information elements. For example, one conflict resolution rule could be "If there is no 'comparative relation' instance extracted, use the default comparative relation 'greater_than_or_equal'." Prolog logic clauses in our method are composed of concept logic clauses and relation logic clauses. A concept logic clause is in the form of '*element_name*(*Element_name*)', where the *element_name* outside the parenthesis is all lower-cased and the first letter of *element_name* inside the parenthesis is capitalized (in Prolog syntax, names that start with capitalized letter are all representing variables). A relation logic clause is in the form of '*element_name*(*Instance1,Instance2,…*)', where the *element_name* is the name of the relation and the instances inside the parenthesis are the concept instances linked by the relation. How many instances should be in a relation depends on the semantics of the relation. For example, 'provided_by' relation links two concept instances while 'between_and' relation links three concept instances (i.e. one concept instance indicates the subject and the other two indicate boundaries for a range). Each element inside or outside the parenthesis in a logic clause is called a logic clause element. Concept logic clauses and relation logic clauses are used to build rules in Prolog. The reader is referred to (Zhang and El-Gohary 2013) for more technical details of each step.

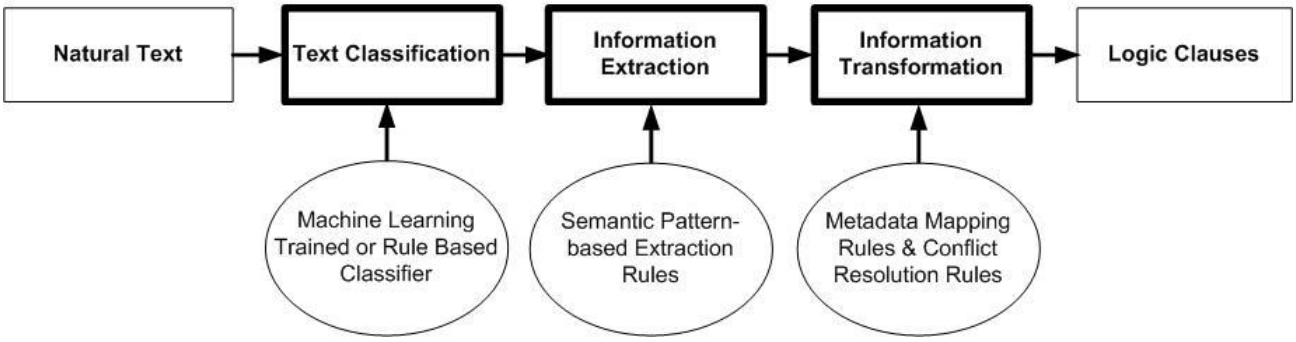


Figure 1 : The proposed automated logic clause generation method

3.2 Top-down method for handling complex sentence components

The top-down method for handling complex sentence components is composed of three main steps: 1) extraction of necessary information elements, 2) extraction of secondary information elements, and 3) information transformation for necessary information elements and secondary information elements.

First, our method extracts the necessary information elements for a specific problem. A necessary information element is an information element that must be defined for this specific type of requirement. For example, for extracting quantitative requirements, we recognized five necessary information elements: 1) 'subject', a thing (e.g. building object, space, etc.) that is subject to a particular regulation or norm; 2) 'compliance checking attribute', a specific characteristic of a 'subject' by which its compliance is assessed; 3) 'comparative relation', a relation used for comparison such as greater_than_or_equal, less_than, or equal_to, etc; 4) 'quantity value', a value, or a range of values, which defines the quantified requirement; and 5) 'quantity unit', the unit measure for the 'quantity value'; or 'quantity reference', a reference to another quantity (which presumably includes a value and a unit).

Second, our method extracts the secondary information elements. Secondary information elements are not necessary for this specific type of requirement, but may exist in defining the requirement. Secondary information elements usually appear in subordinate clauses (i.e. a clause that adds additional information to an independent clause, but which cannot exists on its own) of a sentence or modifiers (i.e. an optional element that depends on

and modifies another element in the same phrase or clause structure). They are more likely to result in complex sentence structures by embedding in or attaching more concepts and relations to a sentence. For example, in the sentence "Courts having windows opening on opposite sides shall not be less than 6 feet in width", the part "having windows opening on opposite sides" is a secondary information element. It appears as a modifier of the main subject of the sentence, and contains several concepts and relations (i.e. 'window', 'opposite side', 'opening on'). We use the same method, namely semantic IE, for extracting both necessary and secondary information elements.

Third, in the ITr step, necessary and secondary information elements are transformed into logic clauses, separately. Transforming secondary information elements requires one extra processing step in comparison to the transformation of necessary information elements: recognizing the concepts and relations contained in the secondary information element. Thus, necessary information elements are transformed first. Secondary information elements are processed to extract the contained concepts and relations. Those concepts and relations from secondary information elements are then transformed into logic clauses, and those logic clauses are combined with the logic clauses transformed from necessary information elements. We call this method top-down, because we start from information extraction on the sentence level, then further extract information on the segment level (i.e. the secondary information elements are extracted in form of segments of a sentence), and end with information extraction on the concept and relation level (i.e. term and phrase level) before transformation.

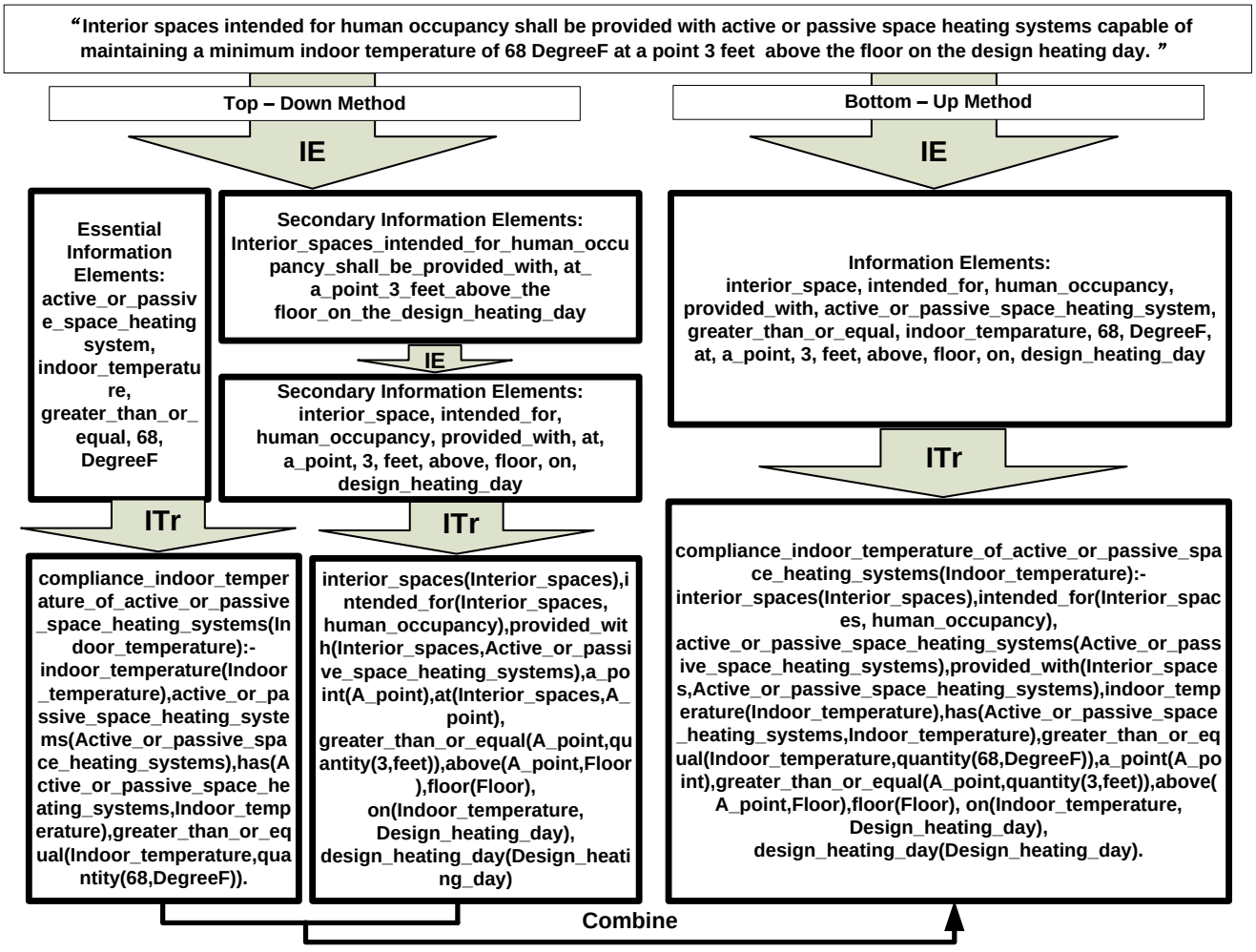
3.3 Bottom-up method for handling complex sentence components

The bottom-up method for handling complex sentence components is composed of two main steps: 1) extraction of necessary information elements and secondary information elements, together, and 2) information transformation through 'consume and generate' mechanism.

First, our method extracts all information elements together, no matter whether they are necessary information elements or secondary information elements. Those information elements are put into a list (i.e. one list for each sentence, we call it information element list), with the order of each information element in the list decided by their order in the original text.

Second, our method transforms all information elements through a new mechanism that we propose – we call it 'consume and generate' mechanism. The 'consume and generate' mechanism implements the SM and CR rules for ITr in a sequential processing. This mechanism is following the heuristics of the 'sliding window' method in computational research (i.e. a sequence of data is processed sequentially, segment by segment, and each segment has a predefined fixed length), and the mechanism of transcription in genetics domain (i.e. a sequence of DNA is transcribed sequentially, segment by segment, and each segment has a length of about 17 base-pair). The 'consume and generate' mechanism is based on processing text segments that match different types of semantic tuples. A type of semantic tuple captures a specific type of relation of concepts. For example, the 'quantity comparison tuple' captures the comparative relation between a 'compliance checking attribute' and a specific quantity (e.g. a quantity is represented by a structured four-tuple consisted of the instances for <'compliance checking attribute', 'comparative relation', 'quantity value', 'quantity unit'/'quantity reference'>, such as in 'less_than_or_equal(Transmission_rate,quantity(1,perm))'); the 'general relation tuple' captures a relation between any two 'subject', such as the 'added to' relation in 'added_to(One_story_addition,Existing_building)'; the 'above' relation captures the relative location relation between two 'subject', such as in above(Ventilators, Eave_or_cornice_vents), etc. Our 'consume and generate' mechanism uses sequential processing - processing segment by segment. Each segment matches a pattern of one type of semantic tuple. However, the segment length is not fixed. The length of each segment is decided by the number of information elements in the corresponding type of semantic tuple. More precisely, it is decided by the span of text covered by the pattern of that type of semantic tuple (i.e. the pattern as represented by a specific sequence of information elements in the information element list). For example, 'one story addition', 'added to', and 'existing building' are the first three elements in the information element list generated for the sentence "A one story addition added to an existing building with a glazing area in excess of 40 percent of the gross area…". They match a pattern for the 'general relation tuple': <'subject', 'general relation', 'subject'>. Correspondingly, a SM rule of the 'general relation tuple' will generate the relation logic clause 'added_to(One_story_addition,Existing_building)' from this tuple. In the 'consume and

generate' mechanism, the length of segment for this part is three (i.e. consuming three elements in the element list). Our 'consume and generate' mechanism allows backward matching. That is, if a segment of text matches the later part of a pattern, then the preceding text is checked for matching of the earlier part of the same pattern. Corresponding logic clauses will be generated if the matching check succeeds. For example, the list of elements generated for the sentence "The unit shall have a living room of not less than 220 square feet of floor area" is: 'unit', 'have', 'living room', 'of', 'not less than', '220', 'square feet', 'of' 'floor area'. Their corresponding information elements are: 'subject', 'general relation', 'subject', 'part of relation', 'comparative relation', 'quantity value', 'quantity unit', 'part of relation', 'compliance checking attribute'. Matching the pattern for 'general relation tuple', the first three elements in the list are consumed and there are three logic clauses generated: 'unit(Unit)', 'living_room(Living_room)', and 'has(Unit, Living_room)'. Then the starting point of the next segment to consume is 'of', which corresponds to 'part of relation' in a pattern for the 'quantity comparison tuple': <'subject', 'part of relation', 'comparative relation', 'quantity value', 'quantity unit', 'compliance checking attribute'>. The five elements starting from 'of' match the later part of the pattern, so 'living room' is checked for the matching of 'subject'. The matching check succeeds, so the following logic clauses are generated: floor_area(Floor_area), has(Living_room, Floor_area), greater_than_or_equal(Floor_area, quantity(220, square_feet)).



** 'DegreeF' is transformed from ' °F' in a preprocessing step for practical reasons*
*** elements to the left of ':-' is head (consequent), elements to the right of ':-' is body (antecedents)*

Figure 2 : A sample sentence processing illustrating top-down method and bottom-up method

4. PRELIMINARY EXPERIMENTS

We tested our top-down and bottom-up methods for handling complex sentence components on processing quantitative requirements in Chapter 12 of the International Building Code 2006 (ICC 2006). In the bottom-up method, since we need to process each word of a sentence, any information other than necessary information elements and secondary information elements (for quantitative requirements) are also automatically processed (i.e. extracted and transformed). Thus, we developed two sets of gold standards, one for each method. The gold standard for the bottom-up method has more information instances than that for the top-down method (Figure 3). But, for both gold standards, there are 57 logic clauses corresponding to 57 requirement sentences.

For TC and IE, we used ANNIE (A Nearly-New Information Extraction System) in GATE (General Architecture for Text Engineering) tools (Cunningham *et al.* 2012) for POS tagging, phrasal tag generation, and gazetteer compiling; and we used JAPE (Java Annotation Patterns Engine) transducer for TC and for writing IE rules. We developed an ontology using the ontology editor in GATE to support our semantic IE. For ITr, we implemented the SM rules and CR rules in Python programming language (v3.2.3). We utilized the "re" module (i.e. regular expression module) in Python for pattern matching, so that the output from the IE step (i.e. text source tagged with information element tags in XML format) could be directly used as input to the ITr step.

For the top-down method, we used five necessary information elements to represent and extract the target information – 'subject', 'compliance checking attribute', 'comparative relation', 'quantity value', and 'quantity unit' or 'quantity reference', as explained in the Proposed Methods section. We used two secondary information elements – 'subject restriction' and 'quantity restriction'. A 'subject restriction' places a constraint on the definition of a 'subject' (e.g. by defining the properties of the 'subject'). Similarly, a 'quantity restriction' places a constraint on the definition of a 'quantity'. A quantity is consisted of either a 'quantity value' and a 'quantity unit', or a 'quantity value' and a 'quantity reference'. For processing (extracting and transforming) concepts and relations inside 'subject restriction' and 'quantity restriction', we utilized the semantics of several specific types of relations, such as 'for_each', 'and', 'or', etc. Seventeen SM rules and 26 CR rules were used.

For the bottom-up method, we used eleven information elements, including the five necessary information elements used in the top-down method in addition to the following information elements: 'semantic clause boundary indicator', 'general relation', 'part of relation', 'reverse part of relation', 'conjunctive term', and 'for each'. We used more information elements than the top-down method, because we need to capture all information (i.e. not only necessary information elements and secondary information elements) in the selected sentences. 'Semantic clause boundary indicator' is a symbol indicating a boundary of two neighboring sub-clauses or complete sets of meanings; this is typically a punctuation like a comma. 'General relation' is a relation between two concepts, such as 'provided by', 'located in', etc. 'Part of relation' is a relation indicating that the concept before the relation is part of or belongs to the concept after the relation; this is mostly indicated by the term 'of'. 'Reverse part of relation' is a relation indicating that the concept after the relation is part of or belongs to the concept before the relation; this is indicated by terms like 'having'. 'Conjunctive term' is the term 'and' or 'or'. Sixty-four and 17 SM and CR rules were used, respectively. The SM rules in the bottom-up method are based on patterns of semantic tuples. The patterns are composed of syntactic and semantic features of the information elements in the corresponding semantic tuples. Each pattern is consisted of not more than six (an empirically-decided number) information elements.

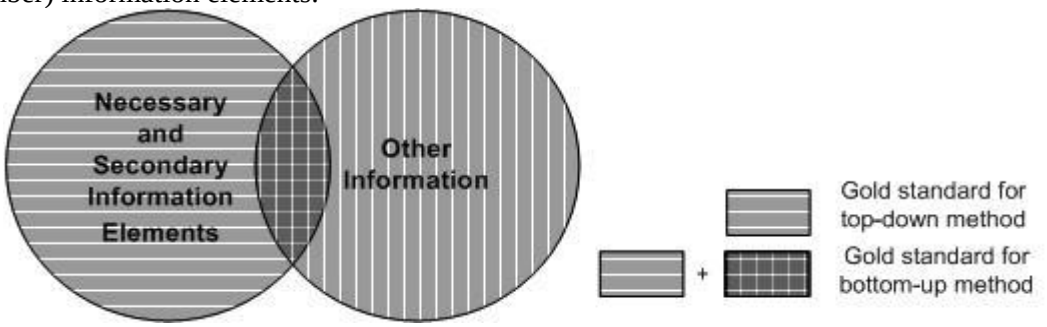


Figure 3: The relation between gold standards for top-down method and bottom-up method

The results were evaluated in terms of precision, recall, and F1 measure, against the two manually-developed gold standards (one for each of the top-down and bottom-up methods). Precision is the correctly-generated logic clause elements divided by the total number of logic clause elements generated. Recall is the correctly-generated logic clause elements divided by the total number of logic clause elements that should be generated. F1 measure is the harmonic mean of precision and recall, that is, F1 = Precision*Recall*2/(Precision+Recall).

5. RESULTS AND DISCUSSION

As shown in Table 1, both the top-down method and the bottom-up method perform well in terms of capturing information elements from complex sentence components, with all precision, recall, and F1 measure values reaching more than 0.92. In addition, although the bottom-up method extracted more information than the top-down method (which might introduce more errors), it outperformed the top-down method in all three measures: total precision, total recall, and total F1 measure. To the best of our knowledge, the closest research studies to this work are those who take a ML-based approach for transforming text into logic clauses (e.g. Raina et al. 2005 and Schoenmackers et al. 2010). Their precision and recall are around 0.80 and 0.35, respectively. But our work is not comparable because: 1) we are taking a different approach (i.e. rule-based approach instead of ML-based approach), and 2) we are addressing a different domain (i.e. construction domain instead of general domain).

Table 1: Preliminary experimental results of the top-down and bottom-up methods.

Performance Measures		**Top-Down Method**	**Bottom-Up Method**
Concept logic clause elements	Number constructed	509	572
	Number correctly-constructed	489	570
	Number in gold standard	530	584
	Precision	0.961	0.997
	Recall	0.923	0.976
	F1 Measure	0.941	0.986
Relation logic clause elements	Number constructed	601	820
	Number correctly-constructed	570	769
	Number in gold standard	615	809
	Precision	0.948	0.938
	Recall	0.927	0.951
	F1 Measure	0.938	0.944
Total	Precision	0.954	0.962
	Recall	0.925	0.961
	F1 Measure	0.939	0.962

In our analysis, we identified four main reasons for the better performance of the bottom-up method: 1) The bottom-up method has a better capability of capturing information carried by complex sentence structures through pattern matching on a small-scale. In the 'consume and generate' mechanism, each segment of text to process contains not more than six concepts/relations; this makes the construction of pattern-based SM rules much easier than matching within a whole sentence. The patterns used for matching on a small-scale tend to be more flexible in capturing the regularities in complex sentence structures. As a result, the chance of missing the information contained in the secondary information elements is smaller; 2) There are less conflicts in extraction using the bottom-up method (17 versus 26 CR rules for the bottom-up and top-down methods, respectively). This is related to the first point, because small segments of the text tend to have less conflicts for the information elements inside; 3) In the 'consume and generate' mechanism, the complete processing of each word in a sentence ensures a higher information coverage. In the 'consume and generate' mechanism, the processing of a sentence cannot finish unless every word in the sentence has been matched by some pattern and processed. This makes the chance of information missing even smaller; 4) For the top-down method, in order to extract secondary information elements, the pattern(s) of secondary information element(s) and the relation between secondary and necessary

information elements need to be decided. This limits the method's ability to incorporate more patterns. For example, we decided that the 'subject restriction' appears after 'subject'. For example, in the sentence "Courts having windows opening on opposite sides shall not be less than 6 feet in width", the 'subject restriction' 'having windows opening on opposite sides' appears after the 'subject' 'court'. But there could be uncommon expressions (in the context of regulatory requirements in construction) for the exact same meaning. For example, "Having windows opening on opposite sides, courts shall not be less than 6 feet in width." The information in the 'secondary information elements' in such uncommonly-expressed sentences would be missed due to the limitation of the top-down method.

Through analysis of the errors for both top-down and bottom-up methods, we found that the errors for concept logic clause elements are caused by: 1) Errors made by the NLP tool used (e.g. due to imperfect precision and recall of POS tagging and ontology-based semantic feature recognition): For example, although 'kitchen' was a concept in the used ontology and successfully extracted most of the cases, there were few cases where the tool failed to extract it; 2) Imperfection of CR rules: For example, in the part of sentence "In one and two family dwellings, beams or girders spaced not less than 4 feet on center…", the CR rules did not separate 'one and two family dwellings' from 'beams' and 'girders' as they are expected to due to unexpected matching of other patterns; 3) Insufficiency of morphological analysis: For example, because any variant of a concept in an ontology is extracted as an instance of the concept, there is a case where some verb form of a concept is incorrectly extracted as an instance, such as in "tested according to…", 'tested' is incorrectly extracted as a concept instance.

The errors for relation logic clause elements are caused by: 1) Errors propagated from errors in concept logic clause elements; For example, if 'yard' fails to be recognized as a 'subject' instance, then the 'subject' instance in the corresponding relation logic clause becomes, consequently, incorrect or missing; and 2) Disambiguation need that is beyond the current capability of the proposed method: For example, we only included disambiguation capability for 'conjunctive term' at the word or phrase level, and it is restricted to the case when the words or phrases connected by the 'conjunctive term' are in equal status (i.e. having the same POS tag or phrasal tag). In more complicated cases of 'conjunctive term' use, such as in "Floor/ceiling assemblies between dwelling units or between a dwelling unit and a public or service area within the structure…", the multiple instances of 'conjunctive term' create more complicated relations among the connected terms than only equal status, thus our method could not process the contained information precisely.

6. CONCLUSIONS AND FUTURE WORK

Existing construction automated compliance checking (ACC) systems require manual effort for extracting requirements from textual regulatory documents (e.g. building codes) and encoding these requirements in a computer-processable format. To address this gap, we developed an automated logic clause generation method to automatically transform regulation text into logic clauses (i.e. rules in Prolog syntax). For handling complex sentence components in the automated logic clause generation method, we proposed two methods - top-down method, which processes text from the sentence level down to identify and process complex sentence components; and bottom-up method, which processes text from the term level up to identify and process complex sentence components. Both top-down and bottom-up methods for handling complex sentence components were tested on processing quantitative requirements in Chapter 12 of the International Building Code 2006 (ICC 2006). Performances were evaluated in terms of precision, recall, and F1 measure, against manually-developed gold standards. Both top-down and bottom-up methods performed well in terms of capturing information elements from complex sentence components, with all precision, recall, and F1 measure values reaching more than 0.92. In addition, although the bottom-up method extracted more information than the top-down method (which might introduce more errors), it outperformed the top-down method in all total measures. Several causes of errors are recognized through analysis, such as errors made by the NLP tool used, imperfection of CR rules, insufficiency of morphological analysis, and disambiguation need beyond the capability of the currently proposed method. In the future, we plan to: 1) add more capabilities to our methods, by leveraging more techniques in state-of-the-art NLP developments (e.g. techniques in complete sentence parsing and semantic relation extraction); 2) test our methods on more construction regulatory text (e.g. environmental protection regulations); and 3) adapt and apply our methods to other types of text in construction (e.g. construction contracts).

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1201170. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Charniak, E. (1997). "Statistical techniques for natural language parsing". *AI Magazine*, 18(4), 33–44.

Cherpas, C. (1992). "Natural language processing, pragmatics, and verbal behavior". *Analysis of Verbal Behavior*, 10,135-147.

Corke, G. (2013). "Solibri Model Checker V8". AECMagazine, Building Information Modelling (BIM) for Architecture, Engineering and Construction. http://aecmag.com/index.php?option=content&task=view&id=527 (5/19/2013, 2013).

Cunningham, H. et al. (2012). "Developing language processing components with GATE version 7 (a user guide)". The University of Sheffield, Department of Computer Science, Sheffield, U.K.

Delis, E.A., and Delis, A. (1995). "Automatic fire-code checking using expert-system technology". *J. Comput. Civ. Eng.*, 9(2), 141-156.

El-Gohary, N.M., and El-Diraby, T.E. (2010). "Domain ontology for processes in infrastructure and construction". *J. of Constr. Eng. Manage.*, 136(7),730-744.

Fenves, S.J., and Garrett, Jr., J.H., (1986). "Knowledge based standards processing". *AI*, 1(1), 3-14.

Han, C.S., Kunz, J.C., and Law, K.H. (1997). "Making automated building code checking a reality". *J. Manage.*, September/October, 22-28.

International Code Council (ICC). (2006). "2006 International Building Code". http://publicecodes.citation.com/icod/ibc/2006f2/index.htm (2/05/2011, 2011).

International Code Council (ICC). (2012). "AEC3". http://www.aec3.com/en/5/5_013_ICC.htm (5/27/2013, 2013).

McCallum, A. and Li, W. (2003). "Named entity recognition with conditional random fields, feature induction and web-enhanced lexicons". *Proc., seventh conf. Natural language learning at HLT-NAACL*, ACL, Stroudsburg, PA, 188-191.

Portoraro, F. (2011). "Automated reasoning". The Stanford Encyclopedia of Philosophy (Summer 2011 Edition), Edward N. Zalta (ed.) http://plato.stanford.edu/archives/sum2011/entries/reasoning-automated/ (5/20/2013, 2013).

Raina, R., Ng, Y.A., and Manning, C.D., (2005). "Robust textual inference via learning and abductive reasoning". *Proc., AAAI 2005*, AAAI Press, Palo Alto, CA, 1099-1105.

Saint-Dizier, P. (1994). "Advanced logic programming for language processing". Publisher: Academic Press, San Diego, CA.

Salama, D., and El-Gohary, N. (2013). "Semantic text classification for supporting automated compliance checking in construction". *J. Comput. Civ. Eng.*, accepted and posted ahead of print.

Schoenmackers, S., Etzioni, O., Weld, D.S., and Davis, J. (2010). "Learning first-order Horn clauses from web text". *Proc., 2010 Conf. Empirical Methods in NLP.*, ACL, Stroudsburg, PA, 1088-1098.

Spencer, A. (1991). "Morphological theory: an introduction to word structure in generative grammar". *No. 2 in Blackwell textbooks in linguistics.*, Blackwell's, Oxford, UK.

Thomson, J., Bird, S., Thomson, D., Marksjo, B., and Sharpe, R. (1988). "Extending Prolog to provide better support for design code expert systems". *Microcomputers in Civ. Eng*., 3(2), 93-109.

Tierney, P.J. (2012). "A qualitative analysis framework using natural language processing and graph theory". *Intl. Review of Research in Open & Distance Learning*, 13(5), 173-189.

Zhang, J., and El-Gohary, N.M. (2013). "Information transformation and automated reasoning for automated compliance checking in construction". *Proc., 2013 ASCE Intl. Workshop Comput. in Civ. Eng.*, ASCE, Reston, VA, 701-708.

Zhang, J., and El-Gohary, N.M. (2011). "Automated information extraction from construction-related regulatory documents for automated compliance checking". *Proc., CIB W78-W102*, CIB, Rotterdam, The Netherlands.