Advances in Cognitive Systems X (20XX) 1-6 Submitted X/20XX; published X/20XX

Acquiring Business Rules: A Challenge for Semantic Parsing

Vinay K. Chaudhri VINAY_CHAUDHRI@YAHOO.COM
Independent Researcher, Sunnyvale, CA, 94087, USA

Jason Freeman
Gautam Shine
Pakapol Supaniratisai
Stanford University, Stanford, CA, 94305, USA

Abstract

There has been a rich history of research on semantic parsing on a wide range of problems some of which include understanding database queries. In this paper, we propose the task of understanding business rules as a new challenge for semantic parsing. Business rules can be usually expressed in a restricted subset of English, and one would hope that semantic parsing would do well on converting such English into logic. We show that an off-the-shelf semantic parser is able to achieve an accuracy of approximately 25% on this task suggesting that this task is a challenge for semantic parsing. We identify several problems that prevent higher performance on this task and outline a research agenda for future research on semantic parsing.

1. Introduction

Semantic parsing is the task of mapping a natural language sentence into a complete, formal meaning representation or logical form [9]. A meaning representation language is a formal unambiguous language that allows for automated inference and processing, such as, first-order predicate logic. Previous work has used semantic parsing with an accuracy in the range of 70-90% in domains such as understanding database queries for Geography [12], and understanding robot commands in a coaching language for Robotic Soccer [3]. Even though early work on semantic parsing focused on providing hand-annotated example mappings between natural language and formal meaning representation, there have been attempts to use un-supervised learning [4, 10]. More recent work has also expanded the applicability of semantic parsing to understand if-then rules of the sort found in conversational assistants [1].

A business rule defines or constrains some aspect of a business, and is intended to assert business structure or to control or influence the behavior of the business. For example, an e-commerce organization may have a business rule that if the order total is greater than $100, the shipping will be free. A large class of business rules can be stated in restricted English. Programming these rules into enterprise systems is of great commercial value. As the natural language used in these rules is circumscribed, we pose the task of converting the business rules stated in English into a pre-specified target representation in logic as a challenge for semantic parsing.

As a specific example, we consider the task of understanding the business rules published by the Internal Revenue Service (IRS). We take an off-the-shelf semantic parser and apply it to process the IRS rules. In spite of being a narrow domain, these rules could be parsed only with an accuracy of 25% suggesting this problem to be a new challenge task for semantic parsing. We do not claim that the work reported here advances the state of the art in semantic parsing. Our contribution lies in an initial exploration of the IRS business rule domain as a new application area for semantic parsing with potential for fruitful near-term results.

We begin by describing the challenge task, and various difficulties that need to be addressed. We then consider our technical approach including training data, grammar engineering, and learning approach. Next, we describe our empirical results, and we conclude with open challenges.

2. IRS Business Rules

Every year, Internal Revenue Service (IRS) in United States receives over 150 million income tax returns. IRS validates these returns through an extensive set of business rules published through its website [7]. Obtaining a formal and executable representation of these rules is of interest for a variety of purposes. IRS could use such a representation to verify the completeness and consistency of the checks it performs. The providers of tax filing software could incorporate such rules to ensure that the filed returns pass the validation checks performed by IRS. Currently, such rules are available only in restricted English. The English used in these rules is restricted in the sense that most of the rules follow an if-then structure, but the conditions used in the rules refer to the input fields in an income tax form that would not be expected in a typical English sentence. In this section, we will provide a logical specification for such rules [11].

If semantic parsing is successful in processing the natural language description of these rules to produce their logical representation, it can be applied to a variety of similar business rules. As the IRS rules change from year to year, a semantic parsing-based solution could be helpful for automatically updating the formal representation of rules. Every state in USA has validation rules similar to the ones used by IRS, and a semantic parsing based solution should generalize to represent those rules as well. Such rules also exist for countries other than USA, and any developed technique would have much broader applicability than the limited context of US tax law.

2.1 Representation of IRS Rules in Logic

We use Datalog syntax [11] for formally representing the IRS Rules. The source document is 96 pages long and contains 1465 rules [7]. All of these rules pertain to the IRS form 1040. Form 1040, along with its variations, is the primary form that individual taxpayers in the USA are required to complete. We illustrate our representation using several examples.

```
Rule statement in English:
   F1040-001-02
      If Form 1040, Line 75 'OverpaidAmt' has a non-zero value and
      Line 79  'EsPenaltyAmt' is greater than Line 75 'OverpaidAmt',
      then Line 78 'OwedAmt' must have a non-zero value.
   Incorrect Data
   Reject
   Active
```

The above example illustrates that each rule has an assigned rule number (`F1040-001-02`), a text description, an error category (`Incorrect Data`), a severity level (`Reject`) and a rule status (`Active`). The rules that are marked as *Deleted* need not be represented, and were omitted from this exercise. We next show the logical representation of this rule.

```
Logical representation of the rule:
  incorrectData(f1040_001_02):-
    value(overpaidAmt,N1) & value(esPenaltyAmt,N2) &
    value(owedAmt,N3) & non_zero(N1) & greater_than(N2,N1) &
   ~non_zero(N3)
```

The head of above rule encodes the error category (`Incorrect Data`) and the rule number (`F1040-001-002`). In representing the rule number, hyphens are replaced by underscores. All the object constant names must begin from a lower case letters. The object constants such as `overpaidAmt` correspond to individual field names in the Form 1040. As all the rules correspond to the Form 1040, and the field names are unique, for the sake of brevity, we have omitted the line numbers from the predicate names that appear in the rules. We do not represent the severity of the rules, and we use the rule status to decide whether we need to formally represent that rule. We have kept the predicate names as close as possible to the natural language words that they correspond to in the English description of the rules. Let us consider another example rule.

```
Rule statement in English:
   F1040-003-01
     If Form 1040, Line 4 checkbox "Head of household" is checked
     (element 'IndividualReturnFilingStatusCd' has the value 4),
     then one of the following fields must have a value: [ Line 4
     'QualifyingHOHNm' and Line 4 'QualifyingHOHSSN' ] , Line 6c
     'ChldWhoLivedWithYouCnt', or Line 6c
     'OtherDependentsListedCnt'.
   Missing Data
   Reject
   Active

Logical representation of the rule:
   missingData(f1040_003_01):-
     value(individualReturnFilingStatusCd4,true) &
   ~(has_value(qualifyingHOHNm)  & has_value(qualifyingHOHSSN)) &
   ~has_value(chldWhoLivedWithYouCnt) &
   ~has_value(otherDependentsListedCnt)
```

The rule above follows a format that is similar to the one considered earlier except that a disjunction of conditions must hold true. In the logical representation, as the disjunction must be negated, it is simplified into a conjunction. Most rules follow a similar structure, but there are several interesting exceptions that we highlight next. There are some rules that have no relationships to the data values in Form 1040. For example, consider the following rule. (From now onwards, we will only include the English statement of the rule, and omit its other properties such as the rule number, etc.)

```
A single PDF file must not exceed 60MB in size.
```

As the above rule refers to the meta properties of files that may be attached to form 1040, and not to the values within the form, we kept such rules outside the scope of our work. Some rules require special notation to be adequately captured. These rules have one or more relations, with each implicitly referring to multiple values (an entire column of values). We consider two examples.

```
If two Schedule C-EZs (Form 1040) are present in the return,
their Social Security Numbers   must not be equal.
```

```
If Form 8697, Part I, Line 10c 'NetAmtOfInterestOwedAmt'
has a non-zero value, then there must be an 'OtherTaxLitCd'
in [ OtherTaxStatement ] in the return with the value
""FROM FORM 8697"" with the corresponding 'OtherTaxAmt'
having a value greater than  zero.
```

The above rules require line-by-line correspondence between two forms, which would require us to extend our notation to allow a reference to the form to which individual values belong.

2.2 Semantic Parsing Challenges in IRS Business Rules

Despite the regularity of the language, the IRS business rules have a considerable level of variation and complexity in lexical items, uses of functional words, entity names, etc. For example, consider the following rule:

```
If Form 1040, Line 8a 'TaxableInterestAmt' is greater than 1500,
then it must be equal to Schedule B (Form 1040), Line 4
'CalculatedTotalTaxableIntAmt' unless Form 8958 is present
in the return.
```

In the above rule, the parser must resolve anaphora due to presence of *it*, and also must handle exceptions due to presence of *except*.

```
If Form 1040, Line 2 checkbox "Married filing jointly" is checked
 (element 'IndividualReturnFilingStatusCd' has the value 2) and
either 'PrimaryDeathDt' or 'SpouseDeathDt' has a value, then
Form 1040 'SurvivingSpouseInd' must be checked.
```

The above rule illustrates the problem of *distributed coordination*. Due to the presence of `either 'PrimaryDeathDt' or 'SpouseDeathDt'`, the parser must associate `has a value` with both of them.

```
If any of the following Form 1040 fields have
a value, then all of these fields must have a value: Line 76b
'RoutingTransitNum', Line 76c 'BankAccountTypeCd',
Line 76d 'DepositorAccountNum'.
```

In the above rule, the parser has to deal with a list of items.

```
Form 1040, Line 6c(2), each 'DependentSSN' provided must
be unique among all the dependent SSNs in Line 6c(2).
```

In the above rule, we will need to iterate over all the `DependentSSN` to compute that the values are unique. The iteration construct introduces additional complexity in the logical form that should be handled. Due to the range of issues to be handled, we believe that IRS business rules offer a regular yet complex new problem domain for semantic parsing.

3. Technical Approach

Early approaches to semantic parsing relied on hand crafting a grammar that could map the parser output to the desired meaning representation. Due to manual engineering of the grammar, such an approach did not scale as the grammar was never able to take into account all possible variations. We, therefore, envision an approach in which we manually create only a minimal grammar that potentially over-generates potential parses, but use machine learning to learn the correct parse. In this approach, the burden of dealing with all possible cases is lifted away from the grammar, and shifted to machine learning. We do recognize that machine learning cannot solve all of the hard problems of language understanding, but at the same time we are driven by the reality that the languge is so rich and complex that it is hard to imagine that all possible variations will be captured by a hand-authored grammar. Therefore, the data-driven approaches have a role to play in creating practical systems. We also believe that the use of machine learning does not imply that hard problems in understanding language will be solved. What is different here is the possibility that the domain exhibits sufficient compositionality [8], that even after one takes into account the effort required to create training examples for the machine learner, the approach will scale much better than manually engineering the grammar. For the rest of this section, we first describe SEMPRE – the semantic parsing system that we used [2], discuss the training data, and then describe our approach for grammar engineering.

3.1 SEMPRE Semantic Parsing System

SEMPRE is a semantic parsing system that performs all stages of the semantic parsing task, including generating parse trees, learning, and executing the resulting representation. The inputs are the grammar rules, an optional lexicon, and the training examples. Each training example includes an utterance-formula pair and optionally a denotation. For the IRS business rules task, the utterance is the business rule, the formula is the representation in logic, and there is no denotation. For the typical application in which SEMPRE has been used, *e.g.,* answering questions over Freebase, the denotation is the result of evaluating the query. For our use case, the denotation would have been the result of executing the extracted business rule against instances of Form 1040. For simplicity, we kept the task of computing denotations outside the scope of our project.

The grammar in SEMPRE has a few built-in categories. For example, the category *$PHRASE* matches spans of arbitrary length, and *$TOKEN* matches one token in the input. The built-in features for learning include rule, span, bi-gram, lemma, and denotation attributes. Rule features indicate whether a particular grammar rule was applied or not to construct a parse. Span features give the length of the matched input for a grammar rule.

For constructing the logical form, lambda functions, lambda dependency-based compositional semantics (DCS), and Java functions are available (both user-defined and built-in such as string concatenation). We made use of lambda functions and Java functions for our parsing task. SEMPRE uses a log-linear model and performs optimization using AdaGrad, a stochastic gradient descent variant that adapts the learning rate to the optimization landscape.

3.2 Training Data

Of the 1465 rules available from IRS, we manually encoded 417 rules into a logical representation. From these encoded rules, we used 244 rules for training, and 173 rules for testing. The source code and the test data used in the project is available on line [5].

The tax rules are of the form *if A then B*, where `A` is a condition, and `B` is a proposition that must hold. Our logical forms express violations of these rules. Therefore, they are of the form `$SENTENCE & ~$SENTENCE`. Therefore, the task to be performed by our semantic parser is to take the English statement of business rules such as:

```
Example input rule to the semantic parser:
     If Form 1040, Line 75 'OverpaidAmt' has a non-zero value
     and Line 79  'EsPenaltyAmt' is greater than Line 75
     'OverpaidAmt', then Line 78 'OwedAmt' must have
     a non-zero value.
```

and produce the output shown below:

```
   value(overpaidAmt,N1) & value(esPenaltyAmt,N2) &
   value(owedAmt,N3) & non_zero(N1) & greater_than(N2,N1) &
  ~non_zero(N3)
```

In consideration to limited time allocated for our project, we dropped some of the rules from the IRS set that were overly long because they took a very long time for the parser to process. It can take 0.5-2 hour to parse a long rule compared to 1-10 seconds needed on a typical example. An example of a long rule follows.

```
If Form 8965, Part III, Line d checkbox 'AllYearInd' is not
checked, then at least one of the following checkboxes in
'MonthIndicatorGrp' must be checked:
Line e 'JanuaryInd' or Line f 'FebruaryInd'
or Line g 'MarchInd' or Line h 'AprilInd'
or Line i 'MayInd' or Line j 'JuneInd'
or Line k 'JulyInd' or Line l 'AugustInd'
or Line m 'SeptemberInd' or Line n 'OctoberInd'
or Line o 'NovemberInd' or Line p 'DecemberInd'.
```

We also adapted the original representation of the rules to be variable free. The original logical forms used variables to store the value of fields encountered in tax forms. This introduces difficulty in grammar engineering but is not central to the semantic parsing task. We modified the logical forms to minimize the number variables. For example, we used the following transformation to eliminate variables `X` and `Y`:

```
Input Rule: If Form 1040, Line 16a 'PensionsAnnuitiesAmt' or
Line 16b 'TotalTaxablePensionsAmt' has a non-zero value,
then both amounts cannot be equal.

original rule body:
non_zero(pensionsAnnuitiesAmt) & non_zero(totalTaxablePensionsAmt)
& value(pensionsAnnuitesAmt,X) & value(totalTaxablePensionsAmt,Y)
& same(X,Y)"

modified rule body:
non_zero(pensionsAnnuitiesAmt) & non_zero(totalTaxablePensionsAmt)
& same(pensionsAnnuitiesAmt, totalTaxablePensionsAmt)"
```

We used the logical form in which all predicates can take entities directly, rather than needing an explicit binding to a variable.

3.3 Grammar Engineering and Models

We now consider examples of several grammar rules we created to support the semantic parsing task. As we will see, we will be flexible with these rules, and often times, our grammar will over-generate. We can also be flexible with our semantics, where the same syntax rules are mapped to multiple semantic representations.

3.3.1 Basic Structures

As our end result is a string representing the logical formula, in the grammar rules shown next, our semantic functions consist of string constants and lambda functions that concatenate strings.

Recall that the tax rules are of the form *if A then B*, where `A` is a condition, and `B` is a proposition that must hold. Our logical forms express violations of these rules. Therefore, they are of the form

`$SENTENCE & ~$SENTENCE`, as dictated by the following rule, which is our most commonly applied top level rule:

```
(rule $ROOT
   (if $SENTENCE (, optional)
   (then optional) (, optional) $SENTENCE .)
(lambda x (lambda y
  (call + (var x)
      (string " & ~") (var y)))))
```

The `$SENTENCE` nonterminal represents a proposition, and is used for both the antecedent and consequent of the conditional.

Next, consider the following rules used to process text of form “A must be greater than B”.

```
(rule $SENTENCE
   ($ENT $MODAL greater than $ENT)
(lambda x
   (lambda y
      (lambda z
         (call + (string "~greater_than(") (var x) (string ",")
            (var z) (string ")"))))))

(rule $ENT
    (Line $NUM ` $NAME ')
        (lambda x (lambda y (var y))))
```

When a text such as `Line 74 'TotalPaymentAmt' must be greater than Line 63 'TotalTaxAmt'` is encountered, the second rule above processes `Line 74 'TotalPaymentAmt'` to produce `TotalPaymentsAmt`; and processes `Line 63 'TotalTaxAmt'` to produce `TotalTaxAmt`; which are then bound to `$ENT` from the first rule to produce `greater_than(A,B)`.

Let us now consider example of a rule that can produce more than one output.

```
(rule $SENTENCE ($ENT $TOBE $REL $ENT)
   (lambda (x (lambda y (lamdbda z (lambda w
        (call + (var y) (var z) (string "(") (var x)
           (string ",") (var w) (string ")") )))))))
```

When the above rule encounters a text fragment such as `'RenewableDieselMixtureCrAmt' must be equal to zero if an amount is entered`, it can generate two different parses: one in in which `equal to` is mapped to a `$REL`, and `zero` is mapped to an `$ENT`; and a second parse in which `equal` is mapped to a `$REL`, and `to zero` is mapped to an `$ENT`. The parser learns that the first of these is the correct parse through training examples.

3.3.2 Distributive Coordination

For some grammatical constructs, however, this method cannot express the correct parse. An example of this is distributive coordination, as we illustrate in the following example:

```
Input rule:
If Form 1040, Line 24 'BusExpnsReservistsAndOthersAmt'
has a non-zero value, then Form 2106 or Form 2106-EZ must be
attached to Line 24.

Rule body:
non_zero(busExpnsReservistsAndOthersAmt) &
~has_value(form2106) &  ~has_value(form2106EZ)
```

When coordinating Form 2106 and Form 2106-EZ, it is not known until later in the derivation that a `has_value` predicate will be applied to each disjunct. To account for this, we use type lifting. This amounts to the coordination rule itself evaluating to a lambda function. When the disjunction is combined with the predicate `must be attached`, the function is applied to the string `has_value`.

The following is an example of an abstraction and application pair using this technique:

Abstraction:

```
(rule $LIFTED_ENT
(($PRE_CONJ optional)
   $ENT $CONJ $ENT)
      (lambda x (lambda c (lambda y
            (lambda f
            (call + (string "(")
            (call + (var f)
            (string "(") (var x) (string ")"))
            (var c)
            (call + (var f)
            (string "(") (var y) (string ")"))
            (string ")")))))))
```

Application:

```
(rule $SENTENCE
   ($LIFTED_ENT $NEG $PRED)
   (lambda x (lambda y (lambda z
      ((var x)
         (call + (var y) (var z)))))))
```

In the abstraction above `x, c, y` are consumed by the nonterminals, whereas `f` is left unspecified. The application then fills in `f` by supplying the concatenation of `y` and `z` (the negation and predicate respectively). This allows us to delay the application of the predicate to the disjuncts.

	# Grammar Rules	Correctness	Oracle	Parsed	Off-Beam
Initial	40	19.4%	19.4%	22.2%	0.0%
Loosened	88	9.3%	23.6%	40.3%	18.1%
Further Loosened	122	8.1%	18.3%	42.3%	45.1%
Distributive Coordination	136	9.2%	15.5%	47.9%	39.4%

Table 1. Effects of loosening the grammar: increased parsing, but lower correctness and oracle accuracy because parse trees fall off the beam.

3.3.3 Logical Form Evaluation

To check if SEMPRE has produced a correct logical form, we do a string comparison between its output and the expected answer. Before performing the comparison, we normalize case and whitespace. We also provide logical simplification, for example, simplify away double negation.

4. Experiments and Results

We had four objectives in performing the experiments reported here: (1) To verify that a minimal grammar can provide a good foundation for the system; (2) To verify that the system can learn correct mappings to the logical form for this domain; (3) To establish a baseline performance of the system; (4) To determine the effect of adding rules to handle distributed coordination.

We begin by first defining two evaluation measures that we used: oracle accuracy and correctness. Oracle accuracy is motivated by the fact that our semantic parser will produce more than one logical form for each input. Oracle accuracy computes the fraction of inputs for which one of the logical forms produced by the semantic parser is correct. Correctness computes the fraction of inputs for which the correct logical form is chosen by the system. Correctness measures the quality of learning by the system. If the system has learned well, the oracle accuracy and correctness will be close to each other. The performance measures considered here are different from traditional precision and recall metrics because for our problem, there is only one correct parse, and for that case, precision and recall coincide.

To verify that a minimal grammar can provide a good foundation for the system, we first wanted to investigate the effect of progressively loosening the grammar. We achieved loosening by adding more rules as doing so enables the grammar to parse a wider class of sentences in the input. Its effects are documented in Table 1. The column labeled as *Parsed* indicates the number of inputs that are successfully parsed by the system. The column labeled *Off-Beam* indicates the number of inputs for which too many parses are generated which need to be dropped because of the parser size limitation. The results in Table 1 show that, as we loosen the grammar by adding additional rules, we can successfully parse a higher percentage of examples, but accuracy goes down. This is because over-generation results in a trade-off due to the finite beam size of the parser. The correct parse may fall off the beam when there are too many candidate parses. This reduces the correctness and oracle accuracy. We observe this behavior in Table 1, where as parsing percentage increases due to grammar loosening (22.2% to 41.7%), more trees fall off the beam (0% to 45.8%). The oracle

	n	Correct	Oracle	Parsed	Off-Beam
Train	243	21.5%	21.5%	40.1%	29.3%
Test	172	24.4%	25.0%	39.0%	26.2%

Table 2. Baseline performance of the semantic parsing system with learning

	n	Correct	Oracle	Parsed	Off-Beam
Train	243	23.6%	23.6%	46.3%	38.8%
Test	172	24.4%	25.0%	41.3%	38.4%

Table 3. Baseline performance of the semantic parsing system with support for distributed coordination

accuracy increases initially as more examples are parsed (middle row), but falls when the grammar is highly flexible (last row). Correctness monotonically decreases because, in this experiment, there is no training and thus more candidates means a lower chance of the correct parse happening to be the top parse. Adding rules, however, does increase the oracle score, meaning that there are some cases that the system could not have gotten correct before but can parse correctly now. In principle, that could outweigh the loss in correctness. Thus, a monotonic decrease in correctness is merely a side effect of structuring the experiment without learning, and correctness may not necessarily go down with additional rules if the system also learned.

Our final grammar over-generates parses (as intended) and the task of learning is to find weights such that the correct parse is scored higher than others. We utilized rule features and span features, and found that each could over-fit the training data by itself. Features from bi-grams, lemmas, and denotation attributes were not found to provide any benefit at all.

The results in Table 2 address our second and third objectives from above: the system can indeed learn to pick out the correct semantic parse, and we establish a baseline performance for the system. From Table 2 we can see that the performance on the test set is as good as the performance on the training set suggesting that the system can indeed learn to identify correct semantic parses. An overall correctness of 24.4% also establishes the baseline performance of the semantic parsing system. The correctness metric is not high enough for the approach to be practically useful, but it does suggest that there is an ample room for further research to improve this performance.

As one possible improvement to the baseline, we consider the effect of adding additional rules to the grammar to handle distributed coordination. The comparison of tables 2 and 3, as well as rows 3 and 4 in table 1 show that the addition of distributed coordination rule did not help much in terms of oracle accuracy. It did, however, increase the number of parses. A possible explanation for this is that distributed coordination is a source of ambiguity, which makes it more likely that the correct parse falls off the beam.

5. Future Work and Conclusions

Despite the restricted domain, this semantic parsing task proved difficult. There are several issues that need to be tackled to increase parsing rate and correctness. Two such issues were mentioned previously: lists and variables in the logical form. Possible approaches include constructing predicates that take a set as an argument (rather than one or two arguments representing a single variable as our unary and binary predicates do) and then disassemble it with the logical form evaluator. Longer-term research needs to be done to minimize manual work. This could be done by using grammar induction so that we do not need to manually author the grammar. Unsupervised approaches should be investigated to reduce the need to provide annotated examples.

Even though the correctness of the off-the-shelf application of a semantic parsing system to the problem of converting business rules into a logical representation is very low, we believe the work reported here is worthy exploratory research [6]. We discuss to what degree it satisfies the criteria for a good exploratory paper: (1) It gives a precise statement of a new learning problem or learning situation that requires a program to convert IRS business rules into a logical representation; (2) it is a new problem for semantic parsing; (3) our baseline results show that the results of applying off-the-shelf semantic parser are fairly low, and that obtaining a good performance requires lot more work; (4) to achieve greater performance, we need to handle longer business rules, lists and variables and distributed coordination; (5) there is long-term research agenda that includes addressing the issues outlined in the previous point, doing grammar induction instead of hand crafting the grammar, and eliminating the need for supervision.

In summary, we believe that business rules, and in particular, IRS business rules, represent a practically important problem which can significantly benefit by semantic parsing. While off-the-shelf solutions do not give high enough performance, but with sustained work, the accuracy could be increased in the range of 70-90% just like other tasks on which semantic parsing has been found to be successful. The results can be used not just by IRS, but also by the providers of different tax software providers.

6. Acknowledgements

This work has been supported by a gift award from Intuit Inc. The authors thank Pat Langley, Mike Genesereth and anonymous reviewers for their valuable feedback on the paper. We also thank Percy Liang and Chris Potts for their encouragement of this project. Finally, we thank Saikat Mukherjee for posing the problem studied in this paper.

References

[1] I Beltagy, Chris Quirk, Nazneen Fatema Rajani, Raymond J Mooney, Karl Pichotta, Subhashini Venugopalan, Lisa Anne Hendricks, Raymond Mooney, Kate Saenko, Marcus Rohrbach, et al. Improved semantic parsers for if-then statements. *Proceedings of the 54th Annual Meeting of the Association of Computational Linguistics*, 2016.

[2] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairsw. *EMNLP*, 2(5):6, 2013.

[3] Mao Chen, Ehsan Foroughi, Fredrik Heintz, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Patrick Riley, Timo Steffens, et al. Users manual: Robocup soccer server manual for soccer server version 7.07 and later, 2003.

[4] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics, 2010.

[5] Semantic Parsing code and Test Data used in the project: Acquiring Business Rules. `https://github.com/JsonFreeman/sempre/tree/nlu-project/data`.

[6] Thomas G. Dietterich. Editorial exploratory research in machine learning. *Machine Learning*, 5(1):5–9, 1990.

[7] Tax Year 2016 Modernized eFile Schemas and Business Rules for Individual Tax Returns and Extensions. `https://goo.gl/QvG4N6`.

[8] Percy Liang and Christopher Potts. Bringing machine learning and compositional semantics together. *Annu. Rev. Linguist.*, 1(1):355–376, 2015.

[9] Raymond J Mooney. Learning for semantic parsing. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 311–324. Springer, 2007.

[10] Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 1–10. Association for Computational Linguistics, 2009.

[11] J Ullman. *Principles of Data and Knowledge-base Systems Vol I and II*. Computer Science Press, 1988.

[12] John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055, 1996.